Tecnológico Nacional de México

Reporte Intermedio del Ejercicio Sabático

ELABORACION DE MATERIALES, RECURSOS O AUXILIARES DIDACTICOS.

Julio César Flores López

Manual de Prácticas de Tópicos Avanzado de Programación

14 de Agosto de 2017 al 13 de Febrero de 2018

Introducción

La materia de Tópicos Avanzado de Programación (TAP) es una continuación de la programación Java y la materia aporta al estudiante conocimientos adicionales a la programación orientada a objetos, incluyendo herramientas como la Interface Gráfica de Usuario (GUI) en particular se empleará Netbeans.

El estilo de los ejercicios es para ser seguido paso a paso. Más que dar una explicación profunda sobre los conceptos se centra en la experiencia de ser maestro de los cursos de la materia TAP impartida en el salón de clases. Este documento está escrito para alumnos que tienen dominio de los conceptos básicos de Java, y docentes que puedan apoyar su curso empleando como guía los ejercicios mostrados.

Los ejercicios están organizados en función del programa oficial de la materia. Estas prácticas, incluyen una breve descripción de cada una de las unidades, en cada tema se muestran los ejercicios en forma individual. En total se muestran las 17 prácticas.

Las prácticas siguen el siguiente formato: Número de práctica, título, objetivo, correlación con temas y subtemas del programa de estudio, introducción, metodología, sugerencias didácticas y equipo necesario, procedimiento y bibliografía preliminar.

OBJETIVO GENERAL

Elaborar un Manual de prácticas que permita a los estudiantes de Tópicos Avanzado de Programación crear aplicaciones computacionales integrando la programación Java por medio del entorno de desarrollo Netbeans, utilizando programación concurrente, acceso a datos, que soporte interfaz gráfica de usuario e incluya programación móvil en Android.

ESQUEMA DE PRÁCTICAS POR UNIDAD

UNIDAD, TEMAS Y SUBTEMAS	NÚMERO, NOMBRE Y OBJETIVO ESPECÍFICO DE CADA PRÁCTICA
UNIDAD 1: Interfaz gráfica de usuario. 1.1Creación de interfaz gráfica para usuarios. 1.2 Tipos de eventos. 1.3 Manejo de eventos. 1.4 Manejo de componentes gráficos de control.	**No. 1. Instalar Interface Gráfica de Usuario Netbeans.** Objetivo. Instalar la herramienta Interface Gráfica de Usuario Netbeans. **No. 2. Crear aplicaciones Gráficas (JFrame).** Objetivo. Crear una primera aplicación en entorno visual. **No. 3. Usando switch y Scanner.** Objetivo. Permitir agregar y validar información proporcionada por el usuario.
UNIDAD 2: Componentes y librerías. 2.1 Definición conceptual de componentes, paquetes / librerías. 2.2 Uso de librerías proporcionadas por el lenguaje. 2.3 Creación de componentes (visuales y no visuales) definidos por el usuario 2.4 Creación y uso de paquetes/librerías definidas por el usuario.	**No. 4. Usando Queue.** Objetivo. Empleo de la librería Queue. **No. 5. Crear la librería personal Círculo.** Objetivo. Crear una librería personal Círculo. **No. 6. Elementos gráficos.** Objetivo. Conocer los elementos comunes en una interface gráfica. **No. 7. La clase Círculo como componente gráfico.** Objetivo. Crear una librería gráfica personal

	Círculo.
UNIDAD 3: Programación concurrente (MultiHilos). 3.1 Concepto de hilo. 3.2 Comparación de un programa de flujo único contra uno de flujo múltiple. 3.3 Creación y control de hilos. 3.4 Sincronización de hilos computación.	**No. 8. Dibujos animados.** Objetivo. Crear una un hilo para mostrar la concurrencia. **No. 9. Simulación de saldos en un banco hipotético.** Objetivo. Crear una un hilo para mostrar la concurrencia y sincronización. **No. 10. Juego animado parte I.** Objetivo. Crear varios hilos para mostrar interacción. **No. 11. Juego animado parte II.** Objetivo. Crear una aplicación con menús de librería y gráficos. **No. 12. Juego animado parte III.** Objetivo. Crear una aplicación con menús de librería y gráficos y tener características de persistencia.
UNIDAD 4: Acceso a datos. 4.1 Introducción. 4.2 Conexión a origen de datos. 4.3 Manipulación de datos. 4.4 Visualización de datos.	**No. 13. Almacenamiento en base de datos.** Objetivo. Conocer las formas de almacenar información.
UNIDAD 5: Programación de dispositivos móviles. 5.1. Introducción a las tecnologías y herramientas móviles. 5.2 Clasificación y aplicaciones de los dispositivos móviles.	**No. 14. Instalar Interface Gráfica de Usuario Android.** Objetivo. Instalar la herramienta Interface Gráfica de Android. **No. 15. Creación y estructura de un proyecto Android.**

5.3 Entorno operativo de las aplicaciones móviles. 5.4 Desarrollo de aplicaciones móviles. 5.5. Aspectos de seguridad.	Objetivo. Crear una primera aplicación Android. **No. 16. Creación y manejo de actividades en un proyecto Android.** Objetivo. Crear una aplicación Android con varias actividades. **No. 17. Uso de unidades y layouts.** Objetivo. Conocer elementos comunes en una interface gráfica.

No. 1. Instalar Interface Gráfica de Usuario Netbeans.

OBJETIVO

Instalar la herramienta Entorno de desarrollo integrado con Netbeans.

TEMARIO

TEMA	NÚM. PRÁCTICA
1.1 Creación de interfaz gráfica para usuarios.	**No. 1. Instalar Interface Gráfica de Usuario Netbeans.**
1.2 Tipos de eventos.	**No. 2. Crear aplicaciones Gráficas (JFrame).**
1.3 Manejo de eventos.	**No. 3. Usando switch y Scanner.**
1.4 Manejo de componentes gráficos de control.	

INTRODUCCIÓN

Java es un lenguaje de programación orientado a objetos ampliamente utilizado. Las aplicaciones Java se ejecutan en Windows, Mac OS X, Linux, Solaris, Android etc.

Una interfaz gráfica para usuarios es una aplicación que proporciona varios servicios para facilitarle al programador el desarrollo de software [1].

El software requerido para instalar el Entorno de Desarrollo Integrado, en inglés Integrated Development Environment (IDE) son: La máquina virtual de Java y el ambiente de desarrollo de software.

Este software es gratuito y se usa bajo licencia "COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0"

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con los requisitos mínimos siguientes [1]:
   - Sistema Operativo: Microsoft Windows XP Professional SP3/Vista SP1/Windows 7 Professional:
   - Procesador: 800MHz Intel Pentium III o mejor.
   - Memoria: 512 MB
   - Espacio en disco: 750 MB libre sugerido.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso al recurso Wiki del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Conectarse a Internet para obtener el software de las páginas oficiales.
3. Investigar las funciones más importantes del GUI de Netbeans.
4. Seguir el procedimiento de instalación.
5. Probar que el software esté funcionando.

Sugerencias didácticas

1. Realizar una investigación documental.
2. Seleccionar a varios participantes para exponer varias funciones de Netbeans.
3. Seleccionar a voluntarios para hacer una demostración del funcionamiento de Netbeans.
4. Llenar el recurso de Wiki con las funciones de Netbeans.

Reporte de los alumnos (resultados)

1. Elaborar una matriz de varios IDE's mostrando ventajas y desventajas.
2. Elaborar una matriz que contenga información de la evolución de las IDE's.
3. Participar en un glosario de términos con acceso en Moodle.

Procedimiento:

1.- Descargar el Software.

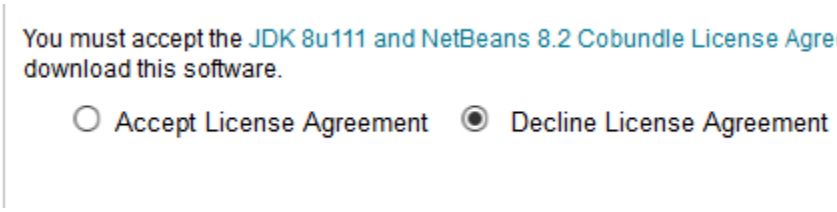
a) En un navegador busque “java jdk download”. Lo cual le mostrará varias ligas, de las cuales le mostrará una que diga “Descargas de Java SE - Oracle” [2] y nos muestra una imagen como:

Descargas de Java SE

Java Platform (JDK) 8u111 / 8u112

NetBeans con JDK 8

b) Aquí seleccionamos Netbeans con JDK 8 y nos lleva a una segunda liga la cual nos muestra las siguientes opciones damos un Clic en “Accept License Agreement”.



c) Al seleccionar vemos que nos muestra varias opciones. Seleccionamos de acuerdo al tipo de equipo en el cual estemos trabajando.

Java SE and NetBeans Cobundle (JDK 8u111 and NB 8.2)		
Product / File Description	**File Size**	**Download**
Linux x86	286.73 MB	jdk-8u111-nb-8_2-linux-i586.sh
Linux x64	282.57 MB	jdk-8u111-nb-8_2-linux-x64.sh
Mac OS X x64	342.99 MB	jdk-8u111-nb-8_2-macosx-x64.dmg
Windows x86	317.21 MB	jdk-8u111-nb-8_2-windows-i586.exe
Windows x64	326.03 MB	jdk-8u111-nb-8_2-windows-x64.exe

d) Si nuestro equipo es Windows de 32 bits seleccionamos la penúltima opción (En caso de no saber si es de 64 o 32 la versión x86 funciona para los dos tipos de procesadores).

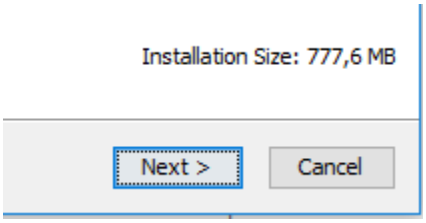
e) Una vez que nuestro equipo ha bajado el software que contiene el Java Development Kit (JDK) y NetBeans (El ambiente de desarrollo) revisamos con el explorador de archivos que efectivamente el tamaño de nuestro archivo sea de 317.21 MB.

2.- Instalación del software.

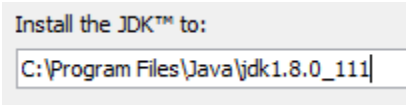
a) Vamos a descargas de nuestro equipo y damos doble clic en el archivo:

b) Nos indica si esta aplicación va a hacer cambios y le indicamos que sí.

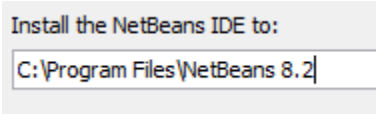
c) No da la bienvenida y le indicamos Next.



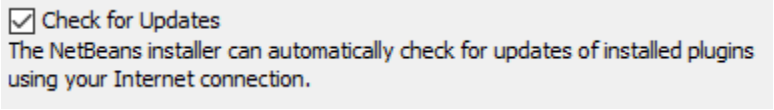
d) Luego nos pregunta donde queremos que el JDK. Damos Next.



e) Enseguida nos pregunta donde instalamos Netbeans. Damos Next.

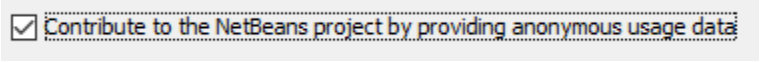


f) La siguiente parte nos pregunta si la instalación nos permita notificarnos en caso de existir actualizaciones lo dejamos seleccionado. Damos Install.

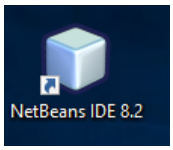


En este momento se lleva a cabo la instalación la cual puede llevarse a cabo en unos minutos.

g) Al finalizar la instalación la aplicación nos pregunta si queremos participar con Netbeans de cómo estamos usando las funciones más importantes. Damos Finish.

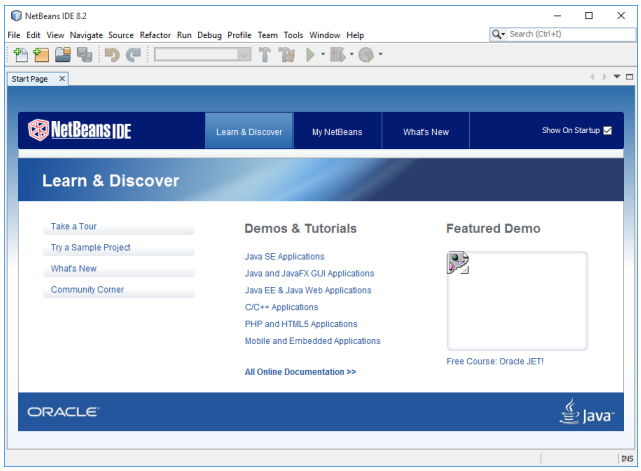


h) Con esto ya se ha instalado y nos deja en el escritorio un icono de la siguiente forma:



j) Damos doble clic sobre este icono y al ejecutar el programa nos muestra un recuadro que incluye lo siguiente.

Y al terminar de cargar nos muestra la interface del programa:



Bibliografía preliminar

[1] Fundación Wikimedia, Inc, «Wikipedia,» 24 Octuble 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado. [Último acceso: 9 Noviembre 2017].

[2] Oracle , «Descargas Java Development Kit,» 22 Noviembre 2017. [En línea]. Available: http://www.oracle.com/technetwork/es/java/javase/downloads/index.html. [Último acceso: 9 Noviembre 2017]

No. 2. Crear aplicaciones Gráficas (JFrame).

OBJETIVO

Crear una primera aplicación en entorno visual. Analizar y comprender el funcionamiento de la librería JFrame y los elementos asociados con la librería Awt y Swing de java creando una Interface Gráfica de Usuario (GUI) básica con Netbeans, usando etiquetas, botones y cajas de texto.

TEMARIO

TEMA	NÚM. PRÁCTICA
1.1 Creación de interfaz gráfica para usuarios. 1.2 Tipos de eventos. 1.3 Manejo de eventos. 1.4 Manejo de componentes gráficos de control.	**No. 1. Instalar Interface Gráfica de Usuario con Netbeans.** **No. 2. Crear aplicaciones Gráficas (JFrame).** **No. 3. Usando switch y Scanner.**

INTRODUCCIÓN

Una GUI es un mecanismo fácil de usar para interactuar con una aplicación. Los componentes GUI o widgets son objetos que interactúan con el usuario a través del mouse o del teclado [1].

Aunque AWT es una librería de Java para crear las interfaces de usuario ya no se es empleada. En su lugar se crearon las librerías Swing y JavaFX y estas a su vez usan AWT [2].

Swing contiene componentes y contenedores. Un componente es un elemento visual independiente tal como un JButton (botón). Y un contenedor (también es un componente) puede agrupar a varios componentes tal como un JFrame (frame).

Se sugiere nombrar los componentes y tener código autodocumentado, anteponiendo en minúsculas el nombre abreviado en español del componente y enseguida la función realizada con la nomenclatura "CamelCase" (se podría traducir como *Mayúsculas/Minúsculas*). Por ejemplo si nuestra etiqueta se denomina "jlabel1" y nos sirve para mostrar el título, entonces el nombre sugerido será "etiTitulo".

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Conectarse a Internet para obtener la documentación de Swing.
3. Investigar algunos componentes de Swing.
4. Seguir el procedimiento de la práctica.
5. Probar que su aplicación este corriendo.

Sugerencias didácticas

1. Realizar una investigación documental de las propiedades de los componentes de Swing.
2. Seleccionar a varios participantes para exponer varias componentes de Swing.
3. Seleccionar a voluntarios para hacer una demostración del funcionamiento de los componentes de Swing.
4. Llenar el recurso de Wiki con las widgets de Swing.

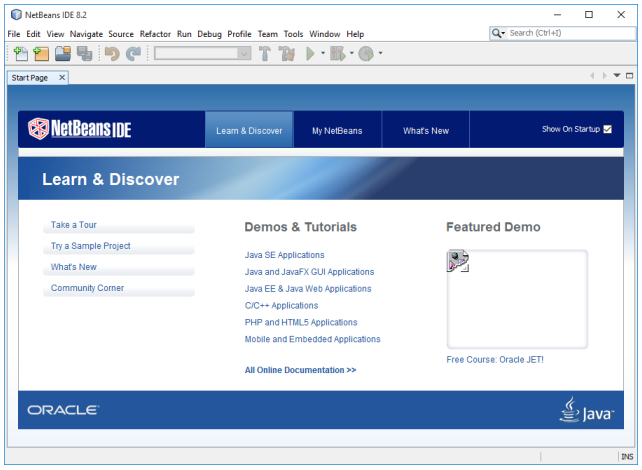
Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo tipo.
2. Crear una aplicación con un widget investigado por el alumno.

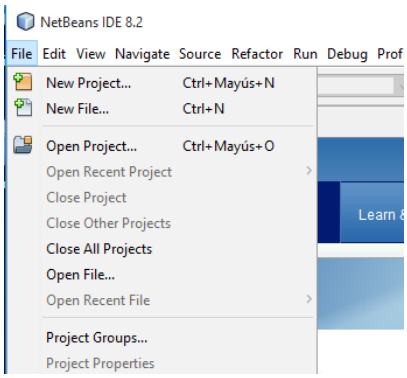
3. Participar en Moodle con el glosario de términos incorporando su widget.

Procedimiento:

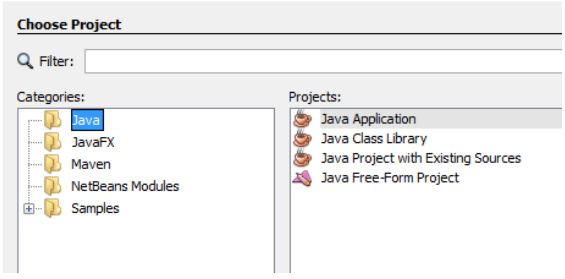
1. Crear un proyecto.
   a. Abrir el programa Netbeans. Una vez ejecutado se muestra lo siguiente.



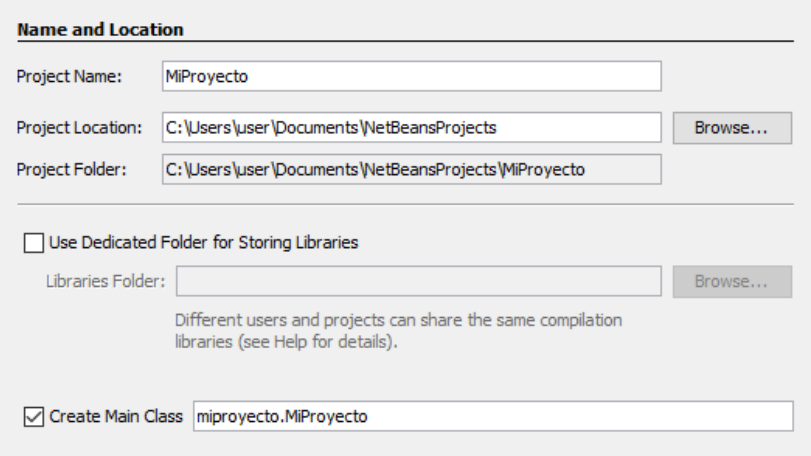
   b. En la parte superior izquierda dar un clic en la opción “File”.



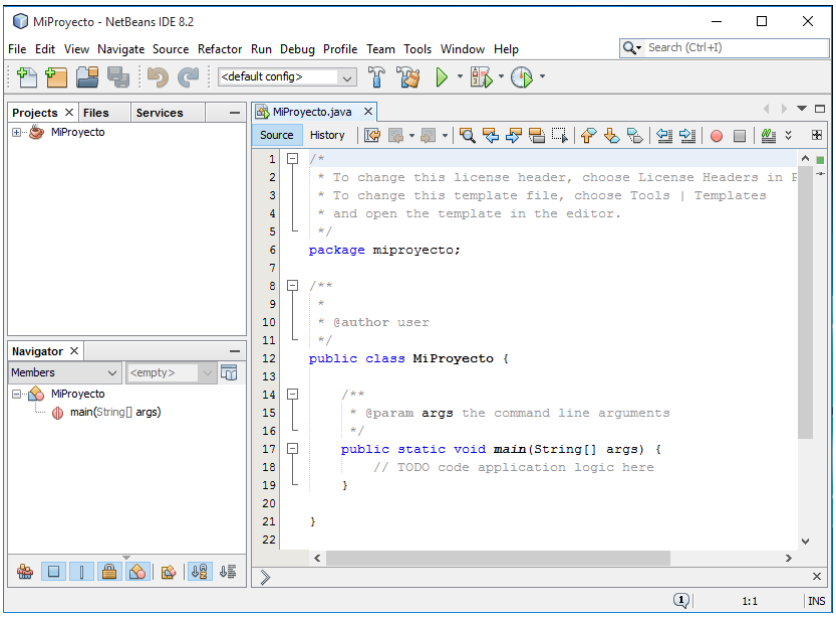
   c. Luego seleccionamos “New Project” y nos lleva a la siguiente pantalla, donde escogemos “Java Application”.



d. Nos lleva a definir el nombre del proyecto y donde lo guardaremos. Por lo pronto solo escribimos en la caja de texto “Proyect Name” escribiremos **MiProyecto** (cabe aclarar que tiene que ser un identificador válido de Java) y dejamos por default los otros valores y damos “Finish”.



e. El proyecto es creado y abierto en el IDE. En donde veremos los siguientes componentes: La ventana de proyectos el cual contiene los elementos del proyecto, El editor del programa fuente llamado MiProyecto.java, la ventana de navegación la cual se usa para navegar rápidamente entre elementos de la clase.



2. Hacemos una prueba de nuestro proyecto incluyendo código básico para mostrar un letrero que diga “Hola Mundo!”.
   a. Sustituimos el renglón que dice:

      // TODO code application logic here

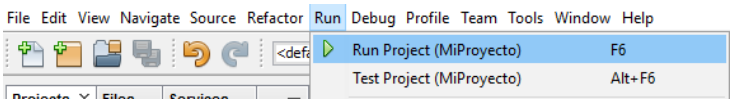
      Por:

      ```
      System.out.println("Hola Mundo!");
      ```

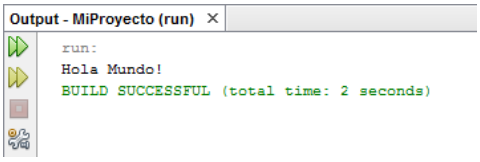
      Resultando en:

```
1  /*
2   * To change this license header, choose License H
3   * To change this template file, choose Tools | Ter
4   * and open the template in the editor.
5   */
6  package miproyecto;
7
8  /**
9   *
10  * @author user
11  */
12 public class MiProyecto {
13
14     /**
15      * @param args the command line arguments
16      */
17     public static void main(String[] args) {
18         System.out.println("Hola Mundo!");
19     }
20
21 }
22
```

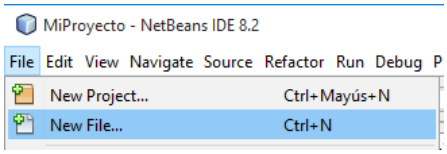
   b. Damos Run > Run Project:



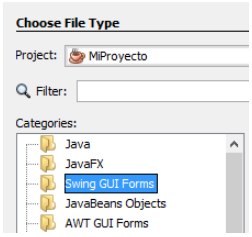
   c. Y el resultado lo vemos en Output:



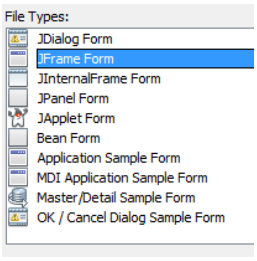
3. Agregar un JFrame.
   a. Escogemos File > New File.



   b. Y esto nos lleva a seleccionar Swing GUI Forms.

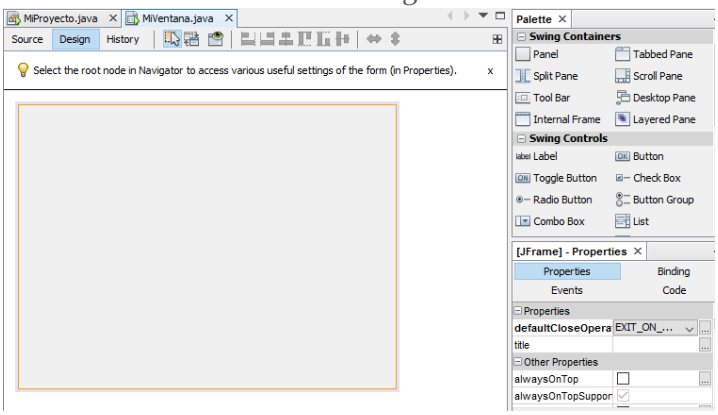


   c. Y aquí seleccionamos un JFrame Form y damos next.

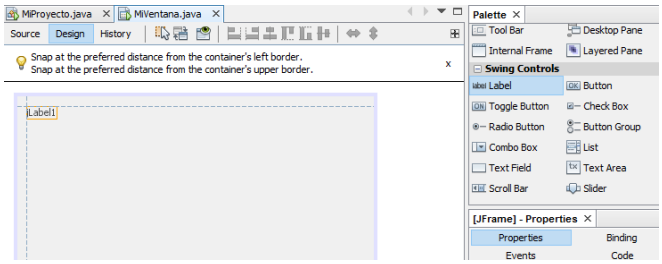


   d. Nuestra clase se llamara “MiVentana” y el paquete “PaqueteFijes”. Y Damos “Finish.”

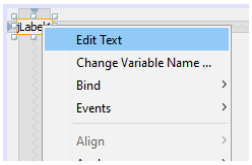
   e. Vemos que nuestra interface ha cambiado y podemos ver un área donde vamos a colocar nuestros elementos swing.



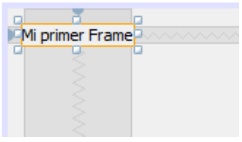
4. Incorporaremos elementos gráficos de control: un Botón, una etiqueta y una caja de texto.
    a. De la ventana “Palette” escogemos un Label y lo posicionaremos en nuestra caja gris que en realidad será nuestra ventana de la aplicación que mostrara los componentes.



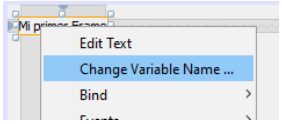
    b. Una vez posicionado nuestro Label (etiqueta) con el botón derecho del ratón damos un clic nos aparece un menú.



    c. Vamos a escoger “Edit Text” y ponemos “Mi primer Frame”. Aparecen unos pequeños marcadores que nos permiten modificar el tamaño visible de nuestro elemento.

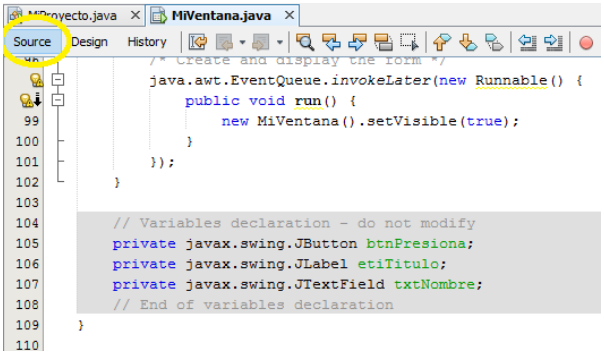


    d. De nuevo damos un clic con el botón derecho. Y le cambiamos el nombre. Por default le asigna jLabel1 y lo cambiamos por etiTitulo



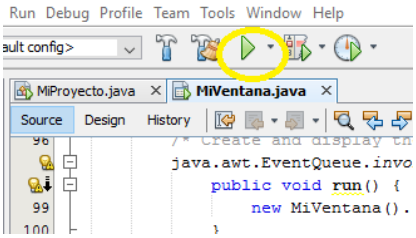
    e. Repetimos los pasos a-d solo escogemos un Button (botón), y hacemos lo mismo con un TextField (texto). Para el paso c el botón tendrá “Presiona”, y el texto “Pon tu nombre”. Cuyos nombres para el paso anterior serían btnPresiona y txtNombre.

f. Observamos que en la parte superior existen dos botones uno indicando "Source" y otro "Design". Al presionar "Source" nos lleva al código fuente generado si navegamos hasta el final vemos los nombres de las variables que hemos cambiado. Y si damos clic a "Design" nos regresa el menú de agregar componentes.

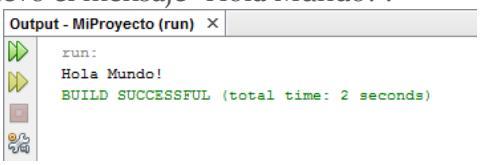


5. Corremos nuestro programa.

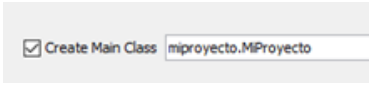
a. En esta ocasión seleccionamos  .



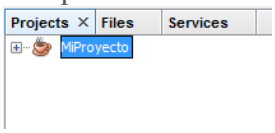
b. Y nos genera de nuevo el mensaje "Hola Mundo!".



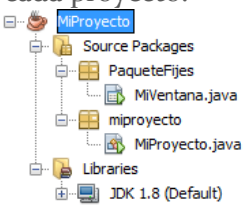
Como nuestro proyecto tiene dos clases una llamada MiProyecto.java y otra clase llamada MiVentana.java al crear el proyecto en el paso 1-d se selecciónó "Create Main Class" se ejecuta las instrucciones de MiProyecto.java y no de MiVentana.java



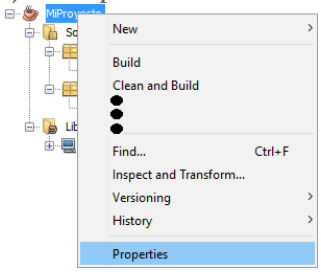
6. Correr nuestro programa con la clase MiVentana.java.
    a. Seleccionar la clase MiVentana.java de MiProyecto.
        i. Nos posicionamos en la ventana de “Projects”



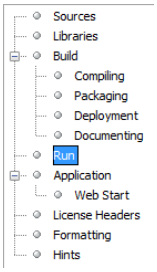
        ii. Originalmente está oculto el contenido del proyecto MiProyecto pero dando un clic en el símbolo nos puede mostrar el contenido de cada proyecto.



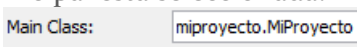
        iii. Damos un clic derecho sobre MiProyecto y aparece un menú en el cual aparece hasta abajo “Properties”. Lo seleccionamos.



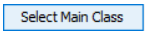
        iv. Nos aparece la opción “Run”

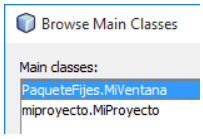


        v. En la cual la clase principal esta seleccionada.



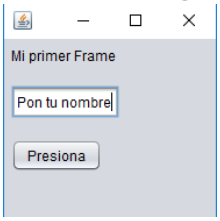
        vi. Para cambiarla damos un clic en

        vii. Y seleccionamos 



        viii. Ahora ya está “PaqueteFijes.MiVentana” y damos

b. Corremos nuestro programa  .
   i. Observamos una disposición como la siguiente.



   ii. Aunque demos un clic en el botón presiona no pasa nada.
   iii. Si podemos cambiar “Pon tu nombre”

Bibliografía preliminar.

[1] H. M. D. P. J. Deitel, Como Programar en Java / 9 Ed., PEARSON, 2012.

[2] H. Schildth, Java - The Complete Reference, 9th Edition, Oracle Press, 2014.

No. 3. Usando switch y Scanner.

OBJETIVO

Permitir agregar y validar información proporcionada por el usuario. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario (GUI) básica con Netbeans, usando un etiquetas, botones y cajas de texto y programando eventos.

TEMARIO

TEMA	NÚM. PRÁCTICA
1.1 Creación de interfaz gráfica para usuarios. 1.2 Tipos de eventos. 1.3 Manejo de eventos. 1.4 Manejo de componentes gráficos de control.	**No. 1. Instalar Interface Gráfica de Usuario con Netbeans.** **No. 2. Crear aplicaciones Gráficas (JFrame).** **No. 3. Usando switch y Scanner.**

INTRODUCCIÓN

Muchos tipos de eventos ocurren cuando el usuario interactúa en una GUI (orientada a eventos) tal como un clic en un botón, escribir texto, etc. La información del evento es guardada en un objeto de AWTEvent [1].

AWT nos permite crear GUIs que están basados en componentes, contenedores, administradores de contenido y eventos. Swing hereda la arquitectura de AWT y la extiende haciendo que indirectamente usemos AWT a través de Swing [2]

La información vertida en las cajas de texto es de tipo String. Si se desea convertir de un tipo String a otro tipo nos va a ayudar la clase Scanner [3]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la documentación de Scanner.
4. Investigar ejemplos tipo relacionados con los botones y las cajas de texto.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de los eventos de los componentes de Swing.
2. Seleccionar a varios participantes para exponer varios eventos de Swing.
3. Seleccionar a voluntarios para hacer una demostración del funcionamiento de los eventos de Swing.
4. Llenar el recurso de Wiki con eventos de Swing.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo tipo.
2. Crear una aplicación con un evento investigado por el alumno.
3. Participar en Moodle con el glosario de términos incorporando su evento.

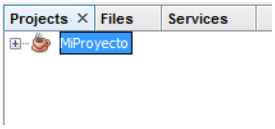
Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente:

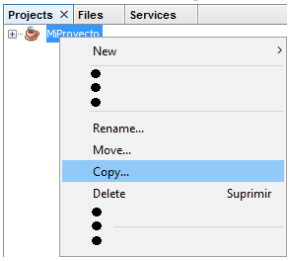
   "Dados tres cantidades enteras mayores que cero. Verificar si con estos tres valores tomados como los valores de la longitud de cada lado de un triángulo respectivamente. Decidir si con estos valores es posible construir un triángulo".

   El algoritmo de solución aquí propuesto es: Primero saber cuál lado es el lado mayor (Si hay otro "mayor", es decir igual, solo se toma el primer mayor). Si la suma de los otros lados es mayor que el lado mayor entonces si se puede formar un triángulo de otra forma no es posible.
   Ejemplos: 5, 1, 1 no cumple. 5,5,5 si cumple. 100,80, 30 si cumple. Etc.
   Para implementar el algoritmo primero preguntaremos si los valores son mayores a cero.

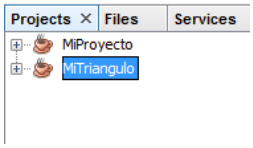
2. Crear un proyecto. Optar por la opción a o b.
   a. Hacer un proyecto como la práctica 2 pasos 1-6.
   b. Crear un proyecto a partir del proyecto de la práctica 2.
      Hacer una copia del proyecto MiProyecto a MiTriangulo.
      i. Seleccionar la ventana de proyectos.



      ii. Con el botón derecho del ratón esconger "copy".



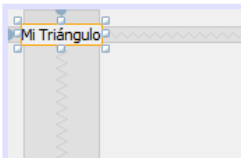
      iii. Nos propone "MiProyecto_1" aquí escribimos "MiTriangulo"
      iv. Seleccionamos el nuevo proyecto MiTriangulo.



      v. Borramos los elementos para dejar limpia la ventana de diseño. Seleccionando cada elemento y presionando la tecla <del> (o <Supr>)
3. Incorporar tres cajas de texto para leer los valores que representan a cada lado del triángulo con sus etiquetas indicado cada lado. Una etiqueta que represente el

resultado ya convertido de los valores de los lados. Un botón de proceso donde muestre en una caja de diálogo si los valores otorgados pueden formar un triángulo.

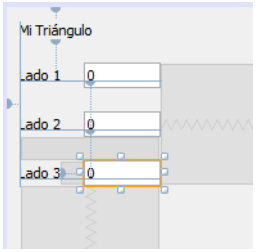
a. Opcional. Agregar una etiqueta que nos informe el título “Mi Triángulo”. Con nombre etiTitulo.



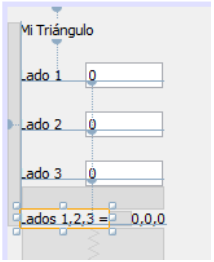
b. Añadir tres etiquetas llamadas etiLado1, etiLado2, etiLado3. Que digan Lado1, Lado2, Lado3.

c. Añadir tres cajas de Texto. Llamadas txtLado1, txtLado2, txtLado3. Con texto “0” (Al cambiar el valor a cero el ancho se reduce, tomar el 

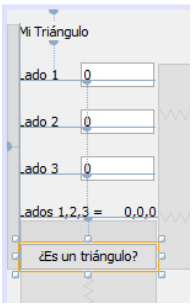
y agrandarlo a unos 60 pixeles)



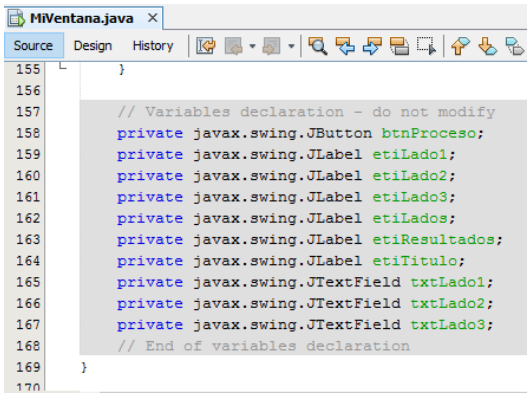
d. Añadir una etiqueta que diga “Lados 1,2,3 =” llamada “etiLados” y otra de resultados parciales. Cada vez que el usuario de un valor en una caja de texto se actualizara estos resultados llamada “etiResultados” y mostrando inicialmente “0,0,0”.



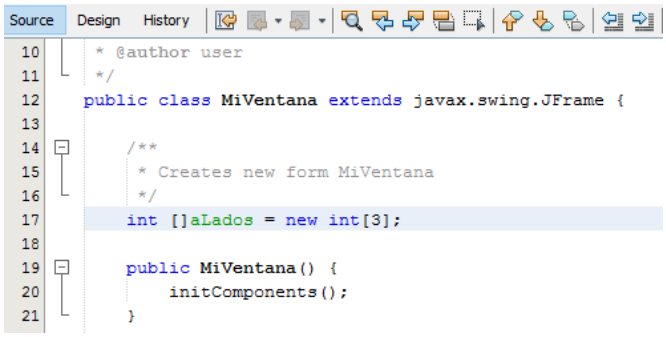
e. Añadir un botón de proceso que muestre “¿Es triangulo?” y llamada btnProceso.



f. Revisamos que nuestras variables muestren los siguientes valores en “source”.



4. Ahora hay que integrar tres variables enteras dentro de la clase “MiVentana”.
Para facilitar la búsqueda del mayor se empleará un arreglo de tres valores enteros.
Incorporaremos la declaración de un arreglo de tres elementos.
int []aLados = new int[3];

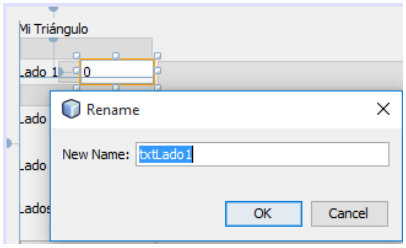


Los elementos del arreglos inician en la posición 0,1,2 para el primero, segundo y tercer entero respectivamente. Se ha creado un arreglo con tres elementos que java por default les ha asignado el valore entero de cero.

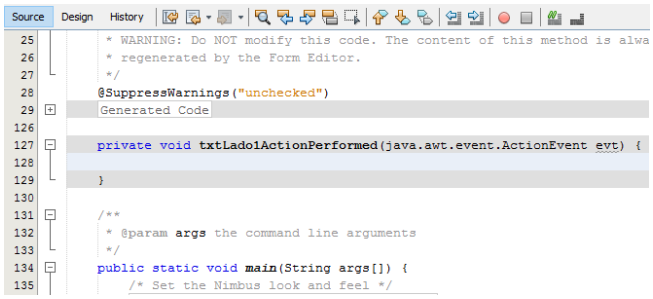
5. Asociar los valores de las cajas de texto con los valores enteros del arreglo. La caja de texto txtLado1 con aLados[0], txtLado2 con aLados[1] y txtLado3 con aLados[2]. La información encontrada en una caja de texto es de tipo String y requerimos

hacer una conversión a tipo entero usando Scanner. La forma que tenemos que hacer es programar un evento de la caja de texto txtLado1 de la siguiente forma:

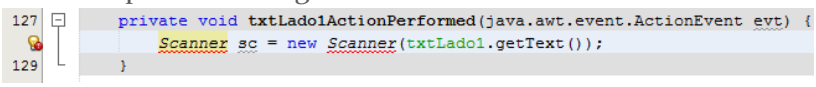
a. No aseguramos que este seleccionado “Design”.
b. Seleccionamos la caja txtLado1 (Nos aseguramos con el botón derecho en “Change Variable Name”) damos “Cancel”.



c. Ahora damos doble clic sobre la caja de texto. Y Automáticamente genera el método asociado al evento de cambio llamado “txtLado1ActionPerformed”.

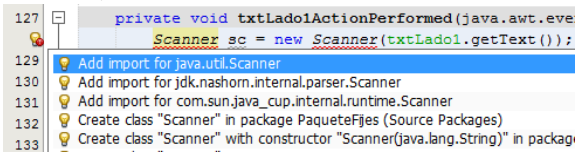


d. En la línea 128 no permite incorporar código el evento ActionPerformed de txtLado1. Incorporamos lo siguiente

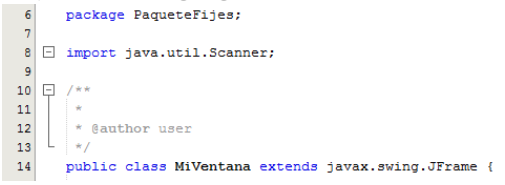


Esto significa: Cuando el programa esté corriendo y el usuario cambie el valor en esta caja de texto, crea un objeto Scanner.

e. Nos aparece un foco rojo le damos un clic.



f. Seleccionamos la primer opción “import java.util.Scanner” con lo cual desaparece el foco rojo y se ha agregado esta instrucción.

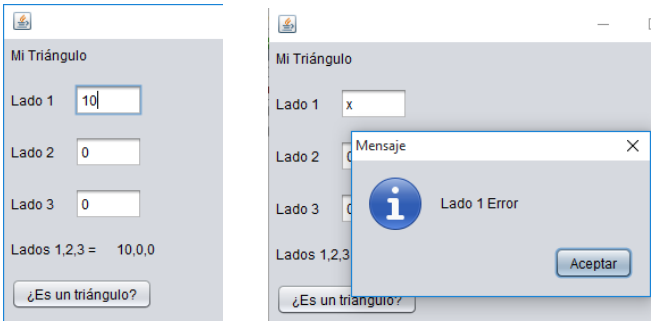


g. Continuamos agregando el siguiente código que básicamente dice: Toma el texto de la caja txtLado1 se pregunta si tiene un entero (el usuario puede equivocarse y dar otros símbolos que no conformen un número válido). Si

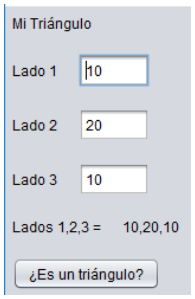
la pregunta es verdadera entonces en la posición “0” se obtiene el entero y luego los tres lados del arreglo son “escritos” y separados por comas en la variable temporal s que posteriormente se asigna a la etiqueta de resultados. En caso que no se tenga un valor entero válido se le muestra un mensaje “Lado 1 error”.

```
130     private void txtLado1ActionPerformed(java.awt.event.ActionEvent evt) {
131         String s;
132         Scanner sc = new Scanner(txtLado1.getText());
133         if (sc.hasNextInt()) {
134             aLados[0] = sc.nextInt();
135             s = String.format("%d,%d,%d", aLados[0], aLados[1], aLados[2]);
136             etiResultados.setText(s);
137         } else {
138             JOptionPane.showMessageDialog(this, "Lado 1 Error");
139         }
140     }
```

h. Corremos lo que llevamos de nuestro programa. Nos damos cuenta que si escribimos 10 y damos <enter> el valor es reflejado en 10,0,0. Y si escribimos una x (valor entero no válido).



i. Podemos repetir el mismo procedimiento para la caja de texto txtLado2 y txtLado3. Solo que la línea equivalente 132 señale la caja 2 y 3. La equivalente 134 sea la 1 y 2. Y el mensaje en línea equivatente 138 sea el lado 2 y 3. En otra práctica incorporaremos el uso de métodos para hacer el mismo procedimiento sin tener que repetir lo mismo. Al correr obtenemos:



6. Programar el algoritmo en el botón “btnProceso”.
   a. En diseño dar doble clic en el botón btnProceso. Nos genera:

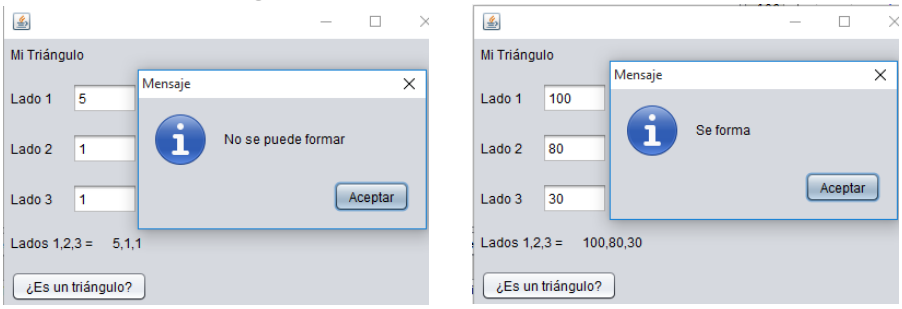
```
181     private void btnProcesoActionPerformed(java.awt.event.ActionEvent evt)
182         // TODO add your handling code here:
183     }
```

b. Escribimos el algoritmo.

```
181    private void btnProcesoActionPerformed(java.awt.event.ActionEvent evt)
182        int suma = aLados[0];
183        int indMayor = 0;
184        boolean menorUno = aLados[0] < 1;
185        for (int ind = 1; ind < 3; ind++) {
186            menorUno |= aLados[0] < 1;
187            suma += aLados[ind];
188            if (aLados[ind] > aLados[indMayor]) {
189                indMayor = ind;
190            }
191        }
192        suma -= aLados[indMayor];
193        if (aLados[indMayor] < suma && !menorUno) {
194            JOptionPane.showMessageDialog(this, "Se forma");
195        } else {
196            JOptionPane.showMessageDialog(this, "No se puede formar");
197        }
198    }
```

El algoritmo dice:
Sumamos todos los valores pero le quitamos el valor del mayor.
Asumimos que el mayor es el primero pero si encontramos que es el segundo o el tercero lo actualizamos.
Verificamos que ningún valor sea menor a uno (cero o negativo).

c. Corremos nuestro programa y verificamos los valores de entrada.



Bibliografía preliminar.

[1] H. M. D. P. J. Deitel, Como Programar en Java / 9 Ed., PEARSON, 2012.

[2] J. Fiessen, Beginning Java 7, New York: Apress, 2011.

[3] Oracle, «Java™ Platform Standard Ed. 8,» 2017. [En línea]. Available: https://docs.oracle.com/javase/8/docs/api/. [Último acceso: 7 12 2017].

No. 4. Usando Queue.

OBJETIVO

Empleo de la librería Queue. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, usando la librería proporcionada por el lenguaje.

TEMARIO

TEMA	NÚM. PRÁCTICA
2.1 Definición conceptual de componentes, paquetes / librerías. 2.2 Uso de librerías proporcionadas por el lenguaje. 2.3 Creación de componentes (visuales y no visuales) definidos por el usuario 2.4 Creación y uso de paquetes/librerías definidas por el usuario.	**No. 4. Usando Queue.** Objetivo. Empleo de la librería Queue. **No. 5. Crear la librería personal Círculo.** Objetivo. Crear una librería personal Círculo. **No. 6. Elementos gráficos.** Objetivo. Conocer los elementos comunes en una interface gráfica. **No. 7. La clase Círculo como componente gráfico.** Objetivo. Crear una librería gráfica personal Círculo.

INTRODUCCIÓN

Los tipos de datos abstractos nos permiten retrasar la implementación hasta saber que operaciones se requieren. Una cola (Queue en inglés) es una línea de espera que crece al agregar elementos a su extremo y se contrae tomando elementos de su frente [1]

Java proporciona una librería genérica LinkedList que implementa la interface Queue [2]

Las colecciones genéricas proporcionan seguridad de tipo en tiempo de compilación [3]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la documentación de LinkedList.
4. Investigar los métodos asociados para la interface Queue.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental del comportamiento de las Colas.
2. Seleccionar a varios participantes para exponer varios usos de Colas.
3. Seleccionar a voluntarios para explicar el comportamiento de las Colas.
4. Llenar el recurso de Wiki con los métodos de las colas.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo tipo. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre.
2. Crear una aplicación por el alumno para usar la clase LinkedList.
3. Participar en Moodle con el glosario de términos incorporando los métodos de la interface Queue.

Procedimiento:

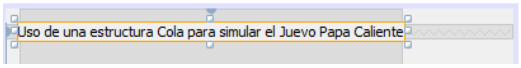
1. El tipo de problema a resolver se propone lo siguiente:

   "El juego infantil la Papa Caliente varios niños se alinean en círculo y se pasan un objeto de vecino a vecino lo más rápido que pueden. En cierto punto, la acción se detiene y el niño que tiene el objeto (la papa) sale. El juego sigue hasta que queda un niño".

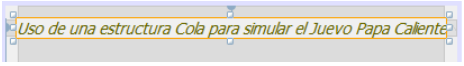
   El algoritmo de solución aquí propuesto es: Simular el círculo usando una cola. Para pasar el objeto simplemente quitamos ese niño de la cola y lo agregamos al final así recorrerán a los demás niños para tomar su turno de nuevo, después de N operaciones el niño que esté al frente se quitará y seguirá otro ciclo.

2. Crear un proyecto llamado "MiJuego" y con JFrame "MiVentana" en el paquete "paketeAyudes" como la práctica 2 pasos 1, 3.
   Nota: En el paso 1-D desactivar la selección automática de la clase.

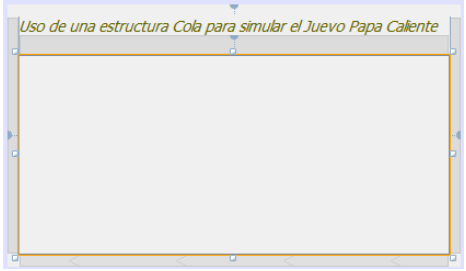
3. En diseño (Design). Podemos poner un título a nuestra Ventana. "Uso de una estructura Cola para simular el Juego Papa Caliente".



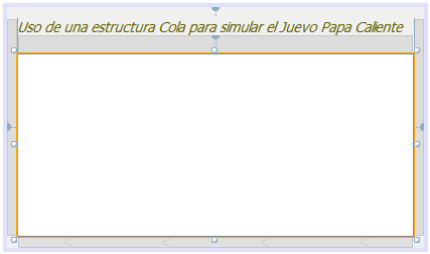
   Podemos cambiar las propiedades Font y Foreground. Hay dos formas de escoger el menú de propiedades una es con el botón derecho aparece el menú de contexto y al fondo aparece "properties", y la otra opción es ir directamente a la ventana de propiedades. Para el Font se escogió Tahoma 14 Italic y para el Foreground el color 102,102,0.



4. Para mostrar los resultados requerimos una ventana de resultados.
   a. Para ello requerimos de la Paleta (Palette) en Swing Containers un "Scroll Pane" que abarque gran parte de nuestra ventana.



b. Seleccionamos de la Paleta (Palette) en Swing Controls un “Text Area” y lo incorporamos dentro del “Scroll Pane”. A la variable “jTextArea1” la renombramos como “taSalida”



5. En código (Source). Creamos un método llamado Actualizar(). Éste método va a ser llamado cada segundo y se mostraran los avances del algoritmo. Lo colocamos después del constructor MiVentana. Adelante incorporaremos el algoritmo en actualizar.

```
17      public MiVentana() {
18          initComponents();
19      }
20
21      private void Actualizar(){
22
23      }
24
25      /**
```

6. Definimos las variables del programa.
   a. Definimos una variable Timer para que nos vaya mostrando cada segundo el avance de la cola. Llamamos al método Actualizar() y creamos el código dentro del constructor MiVentana.
   b. Una constante N que representa el número de veces que un se pasa la “papa”. N=3.
   c. Una variable i para indicar el número de pasos antes de llegar a N.
   d. La estructura cola. Queue <String> cola = new LinkedList<>();
   e. En el constructor agregamos a los “niños”.

```
cola.add("Pablo");
cola.add("Pedro");
cola.add("Javier");
cola.add("Luis");
cola.add("Jorge");
```

f. El código resultante nos queda:

```
18      Timer t;
19      final int N = 3;
20      int i = 0;
21      Queue<String> cola = new LinkedList<>();
22
23      /**
24       * Creates new form MiVentana
25       */
26      public MiVentana() {
27          initComponents();
28          t = new Timer(1000, (ae) -> {
29              Actualizar();
30          });
31          t.start();
32          cola.add("Pablo");
33          cola.add("Pedro");
34          cola.add("Javier");
35          cola.add("Luis");
36          cola.add("Jorge");
37          taSalida.append("\n Inicio" + cola);
38      }
```

Verificamos que el import sea como lo siguiente. Nota: El timer de swing se prefiere cuando se actualizan los componentes gráficos (Gui)

```
8   import java.util.LinkedList;
9   import java.util.Queue;
10  import javax.swing.Timer;
```

7. El resultado de la corrida con los valores determinados es:

Bibliografía Preliminar

[1] A. Drozdek, Data Structures And Algorithms in Java, Thomson, 2010.

[2] Oracle, «Java Documentation Tutorial Queue Implementation,» [En línea]. Available: https://docs.oracle.com/javase/tutorial/collections/implementations/queue.html. [Último acceso: 14 12 2017].

[3] H. M. D. P. J. Deitel, Como Programar en Java / 9 Ed., PEARSON, 2012.

No. 5. Crear la librería personal Círculo.

OBJETIVO

Empleo de la librería personal Círculo. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, usando una librería creada por el usuario.

TEMARIO

TEMA	NÚM. PRÁCTICA
2.1 Definición conceptual de componentes, paquetes / librerías. 2.2 Uso de librerías proporcionadas por el lenguaje. 2.3 Creación de componentes (visuales y no visuales) definidos por el usuario 2.4 Creación y uso de paquetes/librerías definidas por el usuario.	**No. 4. Usando Queue.** Objetivo. Empleo de la librería Queue. **No. 5. Crear la librería personal Círculo.** Objetivo. Crear una librería personal Círculo. **No. 6. Elementos gráficos.** Objetivo. Conocer los elementos comunes en una interface gráfica. **No. 7. La clase Círculo como componente gráfico.** Objetivo. Crear una librería gráfica personal Círculo.

INTRODUCCIÓN

Una clase es un modelo para fabricar objetos. La clase es una representación de entidades del mundo real y lo objetos son manifestaciones específicas de esas entidades a nivel de programación [1].

Una fortaleza del lenguaje Java es su rico contenido de clases que podemos utilizar sin necesidad de "inventar el hilo negro". Estas clases están agrupadas en paquetes y al conjunto se le denomina la librería de Java [2].

Una buena ingeniería de Software en una clase se define datos (atributos) y métodos y se oculta la implementación (encapsulamiento). [2]

Cuando se crea una clase se está formando una biblioteca. Usted mismo o un programador cliente usan su clase posiblemente para crear una librería más grande. [3]

En Java la inicialización de un objeto (instancia de clase) está garantizada ya sea que el programador provea un constructor el cual es invocado en caso contrario se invoca a uno definido por Java. [3]

Todos los objetos tienen un método toString que devuelve una representación de cadena del objeto. [2]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para crear una clase.
4. Investigar “declaración de clases”, los métodos “getter´s” y “setter’s” e “instanciación”.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de la declaración y propiedades de las clases.
2. Seleccionar a varios participantes para exponer la creación e instanciación de clases.
3. Seleccionar a voluntarios para explicar el uso de constructores.

4. Llenar el recurso de Wiki con las definiciones de clase, objeto, constructor.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo tipo. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre.
2. Crear una aplicación por el alumno para usar una clase inventada por el alumno. Entregando información como el paso anterior.
3. Participar en Moodle con el glosario de términos incorporando los conceptos de Clase, Atributo, Método, instanciación, constructor, getter, setter

Procedimiento:

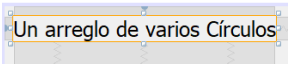
1. El tipo de problema a resolver se propone lo siguiente:

   "Crear la clase Círculo. Pensar en un círculo que pueda ser dibujado en el plano cartesiano. Establecer propiedades como posición, radio. Así como distancia al origen, área y perímetro".

   El algoritmo de solución aquí propuesto es: Crear la clase Circulo agregar los atributos x,y como posición y radio asociados a los métodos getter y setters. Agregar los métodos Origen, Area y Perimetro. Posteriormente crear un arreglo de objetos con varios ejemplos.

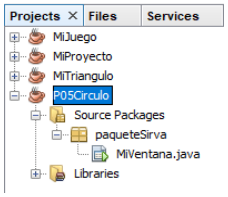
2. Crear un proyecto llamado "P05Circulo" y con JFrame "MiVentana" en el paquete "paketeSirva" como la práctica 2 pasos 1, 3.
   Nota: En el paso 1-D desactivar la selección automática de la clase.

3. En diseño (Design). Podemos poner un título a nuestra Ventana. "Un arreglo de varios Círculos". Podemos cambiar las propiedades Font. Para el Font se escogió Tahoma 24.

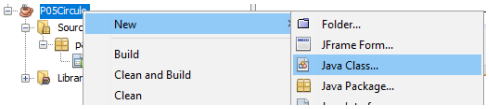


4. Para mostrar los resultados requerimos una ventana de resultados.
   a. Para ello requerimos de la Paleta (Palette) en Swing Containers un "Scroll Pane" que abarque gran parte de nuestra ventana.
   b. Seleccionamos de la Paleta (Palette) en Swing Controls un "Text Area" y lo incorporamos dentro del "Scroll Pane". A la variable "jTextArea1" la renombramos como "taSalida".

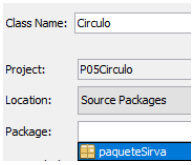
5. Seleccionamos nuestro proyecto.



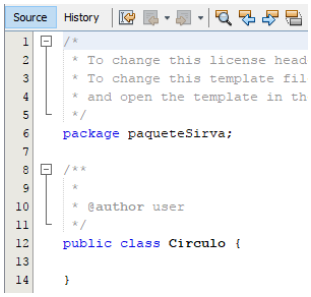
6. Con el ratón sobre el proyecto damos clic derecho donde aparece el botón de contexto seleccionamos “New” -> “Java Class”.



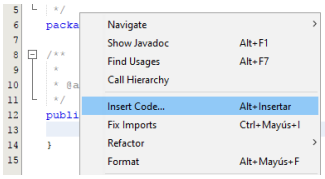
7. En el menú en nombre de la clase escribimos “Circulo” y seleccionamos el paquete “paqueteSirva”



8. Dar “Finish”. Nos muestra una ventana con el esqueleto de la clase para poderla editar.

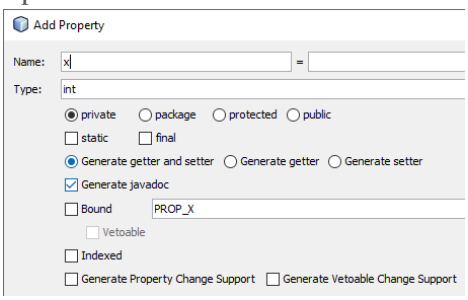


9. Aquí vamos a agregar la posición en función de las variables enteras x,y. Adicionalmente la variable double radio.
    a. En cualquier parte del área de edición con el botón derecho del ratón nos muestra un menú de contexto de donde seleccionaremos “Insert Code”



    b. Y ésta acción nos muestra otro submenú en donde seleccionamos “Add Property”.

c. Escribimos “x” y el tipo “int”.



d. Repetimos los pasos a y b. Ahora escribimos “x” y el tipo “int”.

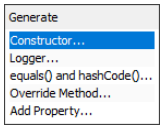
e. Repetimos los pasos a y b. Ahora escribimos “radio” y el tipo “double”.

10. Aunque los comentarios son importantes los vamos a eliminar para centrarnos en el código. Lo que llevamos es lo siguiente.

```
package paqueteSirva;

public class Circulo {

    private int x;
    private int y;
    private double radio;

    public double getRadio() {
        return radio;
    }

    public void setRadio(double radio) {
        this.radio = radio;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }
}
```

11. Agregaremos un constructor para inicializar los atributos x,y,radio.

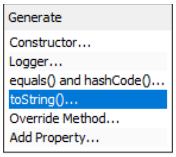
a. En cualquier parte del área de edición con el botón derecho del ratón nos muestra un menú de contexto de donde seleccionaremos “Insert Code”. Y ésta acción nos muestra otro submenú en donde seleccionamos “Constructor”.



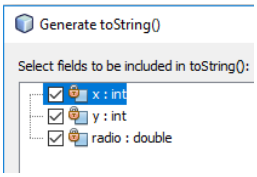
b. Seleccionamos todas las propiedades y damos “Generate”. Se crea un método especial denominado constructor.

```
public Circulo(int x, int y, double radio) {
    this.x = x;
    this.y = y;
    this.radio = radio;
}
```

12. Ahora incorporaremos la reescritura del método toString.
    a. En cualquier parte del área de edición con el botón derecho del ratón nos muestra un menú de contexto de donde seleccionaremos “Insert Code”. Y ésta acción nos muestra otro submenú en donde seleccionamos “toString()”.



    b. Ahora nos permite seleccionar cuales propiedades deseamos mostrar como “texto” es decir un objeto mostrado en forma entendible para el humano.



    c. Seleccionamos todas las propiedades y damos “Generate”. Se agrega el método toString().

```
@Override
public String toString() {
    return "Circulo{" + "x=" + x + ", y=" + y + ", radio=" + radio + '}';
}
```

13. Insertaremos el código para generar el método Perimetro. Para obtener el perímetro de un círculo es Pi * (radio * 2).

```
public double Perimetro() {
    return Math.PI * (2 * radio);
}
```

14. Insertaremos el código para generar el método Área. Para obtener el área de un círculo es Pi * radio ^{2}.

```
public double Area() {
    return Math.PI * radio * radio;
}
```

15. La clase completa es la siguiente:

```
package paqueteSirva;

public class Circulo {

    private int x;
```

```
    private int y;
    private double radio;

    public Circulo(int x, int y, double radio) {
        this.x = x;
        this.y = y;
        this.radio = radio;
    }

    public double Perímetro() {
        return Math.PI * (2 * radio);
    }

    public double Area() {
        return Math.PI * radio * radio;
    }

    public double getRadio() {
        return radio;
    }

    public void setRadio(double radio) {
        this.radio = radio;
    }

    public int getX() {
        return x;
    }

    public void setX(int x) {
        this.x = x;
    }

    public int getY() {
        return y;
    }

    public void setY(int y) {
        this.y = y;
    }

    @Override
    public String toString() {
      return "Circulo{" + "x=" + x + ", y=" + y + ", radio=" + radio + '}';
    }
}
```

16. Crear un arreglo de varios objetos de tipo Circulo.
    a. Cambiamos ahora a la clase “MiVentana” y seleccionamos “Source”.
    b. Aproximadamente en la línea 14 insertamos nuestro arreglo de la siguiente forma:

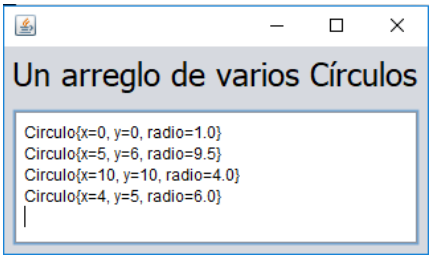
```
Circulo circulos[] = {
    new Circulo(0, 0, 1),
    new Circulo(5, 6, 9.5),
    new Circulo(10, 10, 4),
    new Circulo(4, 5, 6)
};
```

17. Imprimir los resultados usando un “for” extendido. Lo colocamos en el constructor de la clase “MiVentana” de la siguiente forma:

```
public MiVentana() {
    initComponents();
    for (Circulo i : circulos) {
        taSalida.append(i + "\n");
```

```
        }
    }
```

18. Verificar que nuestro programa nos entregue una salida similar a:



Bibliografía Preliminar

[1] J. Fiessen, Beginning Java 7, New York: Apress, 2011.

[2] H. M. D. P. J. Deitel, Como Programar en Java / 9 Ed., PEARSON, 2012.

[3] B. Eckel, Piensa en Java, Madrid: Prentice Hall, 2003.

No. 6. Elementos Gráficos.

OBJETIVO

Conocer los elementos comunes en una interface gráfica. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, usando librerías de gráficos proporcionadas por el lenguaje en una relación de Herencia.

TEMARIO

TEMA	NÚM. PRÁCTICA
2.1 Definición conceptual de componentes, paquetes / librerías. 2.2 Uso de librerías proporcionadas por el lenguaje. 2.3 Creación de componentes (visuales y no visuales) definidos por el usuario 2.4 Creación y uso de paquetes/librerías definidas por el usuario.	**No. 4. Usando Queue.** Objetivo. Empleo de la librería Queue. **No. 5. Crear la librería personal Círculo.** Objetivo. Crear una librería personal Círculo. **No. 6. Elementos gráficos.** Objetivo. Conocer los elementos comunes en una interface gráfica. **No. 7. La clase Círculo como componente gráfico.** Objetivo. Crear una librería gráfica personal Círculo.

INTRODUCCIÓN

Java 2D API es una librería que contiene clases para colores, fuentes. Permite dibujar líneas, arcos, elipses, rectángulos etc. (Shapes). [1]

Java 2D contiene transformaciones de la clase “Affine” se pueden aplicar a formas (Shapes) e imágenes, incluida la traslación, rotación, escalado y corte. [2]

La clase BufferedImage nos permite procesar nuestros datos para manipularlos en forma gráfica y puede permitir a la JVM (máquina virtual de java) usar aceleración por hardware. [2]

La clase "Timer" de la librería "Swing" nos permite realizar una tarea repetidamente. [3]

Nuevas clases se crean por Herencia. Al crear una subclase de una clase padre, se obtiene las declaraciones y los datos que la clase padre define. [4]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para crear dibujos en Java.
4. Investigar uso de Herencia, Timers, BufferedImage, uso de la clase Java 2D y Random.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de la declaración y propiedades de las clases Java 2D.
2. Seleccionar a varios participantes para exponer la creación de formas (Shapes).
3. Seleccionar a voluntarios para explicar el uso de transformaciones sobre formas.
4. Llenar el recurso de Wiki con las definiciones de Herencia, JPanel. Color, Java 2D API, JVM, Affine, Traslacion, Rotación, Stroke, Fill, Random.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.

2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo la creación de un dibujo formado a partir de las formas básicas. Entregando información del paso anterior.
3. Participar en Moodle con el glosario de términos incorporando los conceptos de Herencia, Java 2D API, JVM, Affine, Traslacion, Rotación, corte, Stroke, Fill, JPanel.

Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   "Hacer que una pelota rebote dentro de un área rectangular".

   El algoritmo de solución aquí propuesto es: "Crear la clase Pelota. Pensar en un círculo que pueda ser dibujado en el plano cartesiano. Establecer propiedades como posición, diámetro, color, cambio de posición. Heredar de Ellipse2D y usar la clase Color, dibujarlos en un JPanel. Periódicamente actualizar la posición y dibujar nuestra pelota".

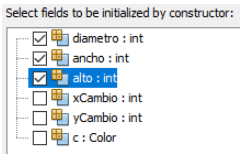
2. Crear un proyecto llamado "Po6Pelota" y con JFrame "MiVentana" en el paquete "paketeMueva".
3. En diseño (Design). Podemos poner un título a nuestra Ventana. "Pelota en movimiento". Podemos cambiar las propiedades Font. Para el Font se escogió Tahoma 24. En el paso 6 continuaremos en diseño agregando un Panel.
4. Para mostrar el comportamiento de la pelota requerimos una versión personalizada de la clase "Ellipse2D.Float". Para ello usamos herencia.
   a. Nos aseguramos que esté seleccionado el proyecto "Po6Pelota".
   b. Con el botón derecho del ratón seleccionamos "New" -> "Java Class".
   c. En el nombre de la clase escribimos "Pelota" y en paquete seleccionamos "paqueteMuevas".
   d. Una vez en el editor borramos los comentarios y extendemos de la clase "Ellipse2D.Float" (aparece un foco rojo damos clic e importamos java.awt.geom.Ellipse2D)

```
package paqueteMuevas;

import java.awt.geom.Ellipse2D;

public class Pelota extends Ellipse2D.Float {

}
```

   e. Incorporamos los atributos siguientes a la clase.

```
int diametro;
int ancho,alto;       //área rectangular ancho por alto
int xCambio, yCambio; //cambio de posición en x y
Color c;
static Random r = new Random();//inicializador aleatorio
```

f. Agregamos un constructor. Con el teclado presionamos <Alt><Ins> nos aparece un menú de contexto y seleccionamos "Constructor" en el nuevo menú seleccionamos los siguientes atributos y damos generar:



g. El constructor se genera y como es de tipo Ellipse2D se inicializa en super con un valor aleatorio dentro del area formada por ancho y alto. El cambio de la x,y toma un valor entre 1 y diámetro y puede ser positivo o negativo.

```
public Pelota(int diametro, int ancho, int alto) {
    super(
            r.nextInt((ancho - 2 * diametro)) + diametro,
            r.nextInt((alto - 2 * diametro)) + diametro,
            diametro, diametro);

    this.diametro = diametro;
    this.ancho = ancho;
    this.alto = alto;

    xCambio = r.nextInt(diametro) + 1;
    yCambio = r.nextInt(diametro) + 1;

    if (r.nextInt(2) > 0) {
        xCambio = -xCambio;
    }
    if (r.nextInt(2) > 0) {
        yCambio = -yCambio;
    }
  c = new Color(r.nextInt(256), r.nextInt(256), r.nextInt(256));
}
```

h. La "Pelota" debe moverse en incrementos de `xCambio`, `yCambio` y permanecer dentro del área permitida.

```
public void mueve() {
    int tx, ty;

    tx = (int) this.x;
    ty = (int) this.y;
    tx += xCambio; // incrementamos la posicion x

    //En caso de salirse de los limites
    if (tx <= 0 || tx >= ancho - diametro) {
        xCambio = -xCambio;
        tx += 2 * xCambio;
    }

    ty += yCambio;
    if (ty <= 0 || ty >= alto - diametro) {
        yCambio = -yCambio;
        ty += 2 * yCambio;
    }

    this.x = tx;
    this.y = ty;
}
```

i. Y finalmente la pelota debe de dibujarse en la imagen buffer.

```
public void dibuja(Graphics2D g) {
```

```
            g.setPaint(Color.WHITE);
            g.draw(this);//Borde blanco
            g.setPaint(c);
            g.fill(this);//Relleno de color aleatorio
        }
```

5. Para el área de dibujo donde se desplazará la pelota usaremos una versión personalizada de la clase "JPanel".
    a. Nos aseguramos que esté seleccionado el proyecto "Po6Pelota".
    b. Con el botón derecho del ratón seleccionamos "New" -> "Java Class".
    c. En el nombre de la clase escribimos "MiPanel" y en paquete seleccionamos "paqueteMuevas".
    d. Una vez en el editor borramos los comentarios y extendemos de la clase "JPanel" e incorporamos los atributos siguientes.

```
package paqueteMuevas;

import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;

public class MiPanel extends JPanel {

    int ancho, alto;
    BufferedImage im = null;
    Graphics2D g = null;
    Pelota pelota;

}
```

    e. Agregamos un constructor. Con el teclado presionamos <Alt><Ins> nos aparece un menú de contexto y seleccionamos "Constructor" en el nuevo menú sin seleccionar nada damos generar.
    f. Dentro del código del constructor agregamos lo siguiente. Como no existe el método `crearImagenyPelota()` lo creamos forma una nueva pelota y una imagen con buffer nueva.

```
public MiPanel() {
  ancho = alto = 100;
  crearImagenyPelota();
}

final void crearImagenyPelota() {
  pelota = new Pelota(10, ancho, alto);
  im = new BufferedImage(ancho, alto, BufferedImage.TYPE_3BYTE_BGR);
  g = (Graphics2D) im.createGraphics();
}
```

    g. Como el panel puede cambiar de tamaño creamos un método para reiniciar con las nuevas dimensiones.

```
    public void setResize() {
        ancho = getWidth();
        alto = getHeight();
        crearImagenyPelota();
    }
```

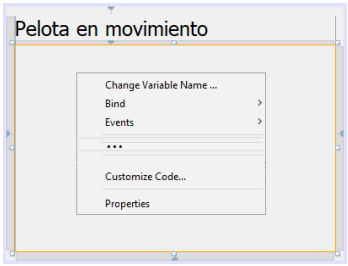
    a. El panel nos sirve como área de dibujo. Limpiamos y dibujamos nuestra pelota y dejamos indicada su nueva posición invocando a `pelota.mueve()`.

```
public void DibujaMueve() {
    g.clearRect(0, 0, ancho, alto);
    pelota.dibuja(g);
    g.drawImage(im, 0, 0, null);
    pelota.mueve();
    repaint();
}
```

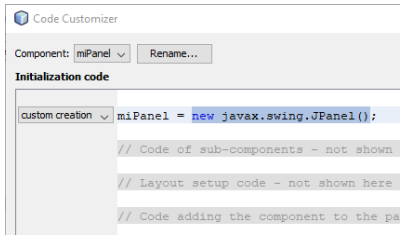
a. Finalmente se lleva a cabo el dibujo mostrado en este panel.

```
@Override
protected void paintComponent(Graphics grphcs) {
    super.paintComponent(grphcs);
    grphcs.drawImage(im, 0, 0, ancho, alto, null);
}
```

6. En diseño (Design) de la clase ventana. Seleccionamos de “Palette” en “Swing Containers” un Panel y lo agregamos debajo del título que abarque la mayor parte del “Frame” y lo renombramos “miPanel”. Temporalmente es un “JPanel” que vamos a sustituir por nuestra versión de la clase “MiPanel” del paso 5.
   a. Con el botón derecho del ratón sobre “miPanel” seleccionar “Customize Code”



   b. En el menú “Code Customizer” seleccionamos “custom creation”



   c. Reemplazamos el código por:

```
miPanel = new javax.swing.JPanel();
->
miPanel = new MiPanel();
```

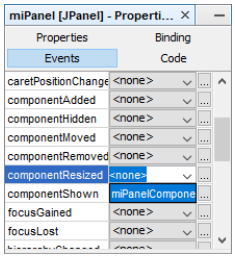
7. Periódicamente actualizar la posición y dibujar nuestra pelota. Ya tenemos la pelota y nuestra área de dibujo (MiPanel) debemos incorporar el panel a nuestra ventana principal.
   a. Seleccionamos “MiVentana” y verificamos que estemos en “Source”.
   b. Agregamos un timer para simular el movimiento dibujar y mover en un período de tiempo de unos 50 milisegundos dentro del constructor.

```
public MiVentana() {
    initComponents();
    Timer timer = new Timer(50, (ae) -> {
        ((MiPanel) miPanel).DibujaMueve();
```

```
        });
        timer.start();
    }
```

c. En el foco rojo damos “Add import javax.swing.Timer”.

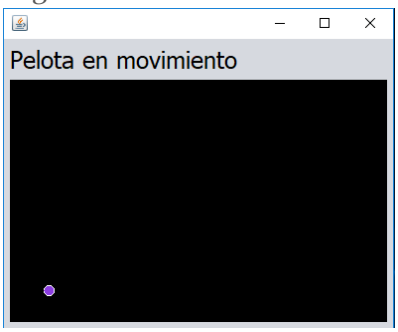
8. Como nuestro frame “MiVentana” se puede cambiar de tamaño hay que prevenir que nuestra pelota no se salga del área de dibujo.
   a. Seleccionamos “MiVentana” y verificamos que estemos en “Design”.
   b. Con el Panel seleccionado nos vamos a la ventana de propiedades. Sólo que ahora escogemos “Events” y buscamos componentResized y donde dice <none> seleccionamos “`miPanelComponentResized`” que inmediatamente nos lleva al código fuente.



   c. En el código fuente dentro del método generado escribimos:

```
private void miPanelComponentResized(java.awt.event.ComponentEvent evt) {
        ((MiPanel) miPanel).setResize();
}
```

9. Verificamos que nuestro programa este corriendo.



Bibliografía Preliminar

[1] H. M. D. P. J. Deitel, Como Programar en Java / 9 Ed., PEARSON, 2012.

[2] A. Davison, Killer Game Programming, Sebastopol: O'Reilly, 2005.

[3] Oracle, «How to Use Swing Timers,» 6 Enero 2018. [En línea]. Available: https://docs.oracle.com/javase/tutorial/uiswing/misc/timer.html.

[4] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

No. 7. Elementos Gráficos.

OBJETIVO

Crear una librería gráfica personal Círculo empleando agregación. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, usando una librería de gráficos definida por el usuario en agregación.

TEMARIO

TEMA	NÚM. PRÁCTICA
2.1 Definición conceptual de componentes, paquetes / librerías. 2.2 Uso de librerías proporcionadas por el lenguaje. 2.3 Creación de componentes (visuales y no visuales) definidos por el usuario 2.4 Creación y uso de paquetes/librerías definidas por el usuario.	**No. 4. Usando Queue.** Objetivo. Empleo de la librería Queue. **No. 5. Crear la librería personal Círculo.** Objetivo. Crear una librería personal Círculo. **No. 6. Elementos gráficos.** Objetivo. Conocer los elementos comunes en una interface gráfica. **No. 7. La clase Círculo como componente gráfico.** Objetivo. Crear una librería gráfica personal Círculo.

INTRODUCCIÓN

Agregación de objetos implica que un objeto tiene o es parte de otro objeto. [1]

El algoritmo DDA (Digital Differential Analyzer) permite tener un comportamiento lineal. Se establece el recorrido a partir de dos puntos. [2]

En el ejercicio anterior empleamos Herencia al establecer que la clase Pelota herede de la clase Ellipse2D.Float. En esta versión emplearemos Agregación.

Pac-Man es un dibujo similar a una pizza sin un pedazo. [3]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para crear dibujos en Java.
4. Investigar uso de Agregación, uso de la clase Java 2D y en especial fillArc.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de la declaración y propiedades de las clases Java 2D, algoritmo DDA.
2. Seleccionar a varios participantes para exponer la creación de otras formas (Shapes), algoritmo DDA.
3. Seleccionar a voluntarios para explicar el uso de Agregación, y de fillArc.
4. Llenar el recurso de Wiki con las definiciones de Agregacion, Frame, DDA, fillArc.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo la creación de un dibujo formado a partir otras formas básicas con

trayectorias lineales usando el algoritmo DDA. Entregando información del paso anterior.

3. Participar en Moodle con el glosario de términos incorporando los conceptos de Agregación, fillArc y nuevas Formas usadas en su programa.

Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   "Hacer una gráfica similar al Pac-Man mostrando la apertura y cierre de la boca en una trayectoria pre-establecida".

   El algoritmo de solución aquí propuesto es: "Crear la clase PacMan, definir parámetros para mostrar un círculo con un pedazo faltante (arco) en función de x,y. Definir una trayectoria con un arreglo de coordenadas y avanzar de coordenada a coordenada siguiendo una línea usando DDA". Establecer propiedades como posición, radio, color amarillo, cambio de posición. Usar fillArc, dibujarlos en un JPanel. Periódicamente actualizar la posición y dibujar nuestro PacMan".

2. Crear un proyecto llamado "P07PacMan" y con JFrame "MiVentana" en el paquete "paketeOriente".
3. En diseño (Design). Podemos poner un título a nuestra Ventana. "Pac-Man en movimiento". Podemos cambiar las propiedades Font. Para el Font se escogió Tahoma 24. En el paso 6 continuaremos en diseño agregando un Panel.
4. Para mostrar el comportamiento lineal de nuestro Pac-Man requerimos crear una clase denominada "DDA".
   a. Nos aseguramos que esté seleccionado el proyecto "P07PacMan".
   b. Con el botón derecho del ratón seleccionamos "New" -> "Java Class".
   c. En el nombre de la clase escribimos "DDA" y en paquete seleccionamos "paketeOriente".
   d. Una vez en el editor borramos los comentarios.

```
package paqueteOriente;

public class DDA {

}
```

   e. Incorporamos los atributos siguientes a la clase.

```
//Parte de x,y se dirige a x2,y2
int x2, y2, sx, sy;
private float x, y, xInc, yInc;
private final float vel;
```

f. Agregamos un constructor. Con el teclado presionamos <Alt><Ins> nos aparece un menú de contexto y seleccionamos "Constructor" en el nuevo menú seleccionamos los atributos `x, y, vel` y damos generar:

```
public DDA(float x, float y, float vel) {
    this.x = x;
    this.y = y;
    this.vel = vel;
}
```

g. El constructor se genera y cambiamos `(float x, float y, float vel)` por `(int x, int y, float vel)`.

h. Incorporamos un método `VeA` que vaya a una nueva coordenada que básicamente llama al método `iniciar`. Agregamos un método `iniciar` que es la construcción del algoritmo DDA.

```
public void VeA(int x3, int y3) {
    iniciar(x2, y2, x3, y3);
}

private void iniciar(int x1, int y1, int x2, int y2) {

    int lenx, leny;
    float deltaVeces;

    lenx = x2 > x1 ? x2 - x1 : x1 - x2;
    leny = y2 > y1 ? y2 - y1 : y1 - y2;
    deltaVeces = lenx > leny ? lenx : leny;

    xInc = (float) lenx / deltaVeces;
    yInc = (float) leny / deltaVeces;

    if (x1 > x2) {
        xInc = -xInc;
        sx = -1;
    } else if (x2 > x1) {
        sx = 1;
    } else {
        sx = 0;
    }
    if (y1 > y2) {
        yInc = -yInc;
        sy = -1;
    } else if (y2 > y1) {
        sy = 1;
    } else {
        sy = 0;
    }

    x = x1;
    y = y1;
    this.x2 = x2;
    this.y2 = y2;

    xInc *= vel;
    yInc *= vel;
}
```

i. El "DDA" debe moverse periódicamente desde `(x1, y1)` hasta `(x2, y2)`. Para ello creamos el método `Actualiza()` y el método privado `yaLlego()`.

```
public boolean Actualiza() {
    if (yaLlego()) {
```

```
            return false;
        } else {
            x += xInc;
            y += yInc;
            return true;
        }
    }

    private boolean yaLlego() {
        boolean p, q;
        p = q = true;
        switch (sx) {
            case 1:
                p = x >= x2;
                break;
            case -1:
                p = x <= x2;
                break;
        }
        switch (sy) {
            case 1:
                q = y >= y2;
                break;
            case -1:
                q = y <= y2;
                break;
        }
        return p && q;
    }
```

j. Agregamos getters. Con el teclado presionamos <Alt><Ins> nos aparece un menú de contexto y seleccionamos "Getter" en el nuevo menú seleccionamos los atributos `sx, sy, x, y` y damos generar:

```
    public int getSx() {
        return sx;
    }

    public int getSy() {
        return sy;
    }

    public float getX() {
        return x;
    }

    public float getY() {
        return y;
    }
```

k. Cambiamos `getX()` y `getX()` para regresar un valor entero.

```
    public int getX() {
        return Math.round(x);
    }

    public int getY() {
        return Math.round(y);
    }
```

5. Para mostrar el Pac-Man requerimos crear una clase denominada "PacMan".
   a. Nos aseguramos que esté seleccionado el proyecto "P07PacMan".
   b. Con el botón derecho del ratón seleccionamos "New" -> "Java Class".
   c. En el nombre de la clase escribimos "PacMan" y en paquete seleccionamos "paqueteOriente".

d. Una vez en el editor borramos los comentarios e incorporamos los atributos siguientes. Nos marca un error que el atributo línea no está inicializado al definir el constructor lo inicializaremos.

```
package paketeOriente;

public class PacMan {
    int radio;
    private final DDA linea;
}
```

a. Agregamos un constructor. Con el teclado presionamos <Alt><Ins> nos aparece un menú de contexto y seleccionamos "Constructor" en el nuevo menú seleccionamos los atributos `radio, linea` y damos generar:

```
public PacMan(int radio, DDA linea) {
    this.radio = radio;
    this.linea = linea;
}
```

b. Modificamos el constructor a la siguiente forma:

```
public PacMan(int radio, int x, int y, float vel) {
    this.radio = radio;
    this.linea = new DDA(x, y, vel);
}
```

c. Incorporamos los métodos `VeA` y `Actualiza` que se basan en el comportamiento de la línea.

```
public void VeA(int x, int y) {
    linea.VeA(x, y);
}

public boolean Actualiza() {
    return linea.Actualiza();
}
```

d. El Pac-Man debe de dibujarse a través del método `Dibuja`. El ángulo de apertura de la "boca" del Pac-Man se calcula en función de la posición de la línea y dependiendo del signo de `x` y `y` si va izquierda o derecha, arriba o abajo.

```
public void Dibuja(Graphics2D g) {
    g.setColor(Color.YELLOW);

    int = linea.getX();
    int y = linea.getY();
    int angulo = (int) (20 * (Math.sin((x + y) * 2 * Math.PI / 50) + 1));// 0and 40

    int sgnx = linea.signoDx();
    int sgny = linea.signoDy();

    //g.drawString("sx=" + sgnx + " sy=" + sgny, 10, 400);
    if (sgnx == 1) {
        g.fillArc(x, y, radio, radio, angulo / 2, 360 - angulo);
    } else if (sgnx == -1) {
        g.fillArc(x, y, radio, radio, 180 + (angulo / 2), 360 - angulo);
    } else if (sgny == 1) {
        g.fillArc(x, y, radio, radio, 270 + (angulo / 2), 360 - angulo);
    } else if (sgny == -1) {
        g.fillArc(x, y, radio, radio, 90 + (angulo / 2), 360 - angulo);
    }
}
```

e. Nos va a marcar dos advertencias con focos amarillos. Damos un clic en cada uno para importar las librerías

```
import java.awt.Color;
import java.awt.Graphics2D;
```

6. Para el área de dibujo donde se desplazará el PacMan usaremos un objeto de la clase “JPanel”.
   a. Seleccionamos “MiVentana” en “Design”.
   b. De la paleta de componentes (Palette) en “Swing Containers” escogemos Panel y lo ubicamos en la mayor parte inferior de nuestra ventana.
   c. Lo renombramos como “miPanel”.
7. Se requiere atributos que señalen un búfer para hacer los dibujos y un tamaño determinado.
   a. Seleccionamos “MiVentana” en “Source” e incorporamos los atributos siguientes.

```
public static final int ANCHO = 640;
public static final int ALTO = 480;
private BufferedImage bimagen;
private Graphics2D g;
private final int FPS = 30;//60;
private final int tiempoEspera = 1000 / FPS;
Timer timer;
PacMan pacman;
private final int[][] coord
        = {{10, 10}, {290, 10}, {290, 180}, {10, 180}, {10, 10}};
int ci;
```

b. Importamos

```
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import javax.swing.Timer;
```

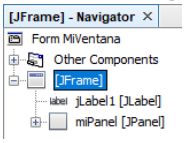
c. En el constructor nos debe quedar de la siguiente forma. (Note que ya está creado).

```
public MiVentana() {
    initComponents();
    ci = 0;
    pacman = new PacMan(50, coord[ci][0], coord[ci][1], 4f);
    ci++;
}
```

a. Creamos los métodos `Actualiza` y `Dibuja`.

```
public void Actualiza() {
    if (!pacman.Actualiza()) {
        if (ci >= coord.length) {
            ci = 0;
        }
        pacman.VeA(coord[ci][0], coord[ci][1]);
        ci++;
    }
}

public void Dibuja(Graphics2D g) {
    g.clearRect(0, 0, MiVentana.ANCHO, MiVentana.ALTO);
    g.drawString("Pac-Man", 150, 120);
    pacman.Dibuja(g);

    Graphics g2 = miPanel.getGraphics();
    g2.drawImage(bimagen, 0, 0, this);
    g2.dispose();
}
```

8. En diseño (Design) de la clase MiVentana. Seleccionamos “JFrame”.
   a. Lo podemos corroborar en la ventana “Navigator”.



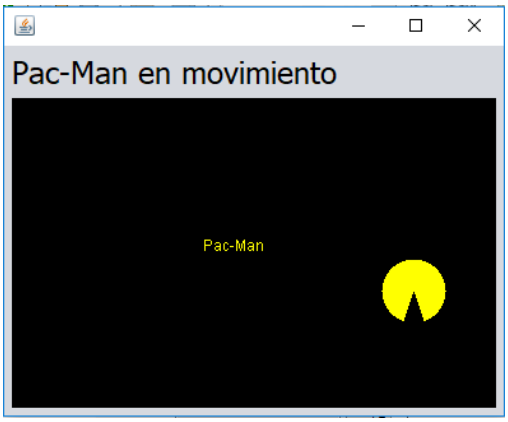
   b. Nos vamos a las propiedades del JFrame y seleccionamos “Events”. Activamos “formWindowActivated” lo cual nos lleva a source. Escribimos el siguiente contenido.

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    bimagen = new BufferedImage(ANCHO, ALTO, BufferedImage.TYPE_INT_RGB);
    g = (Graphics2D) bimagen.getGraphics();
    timer = new Timer(tiempoEspera, (ae) -> {
        Actualiza();
        Dibuja(g);
    });
    timer.start();
}
```

   a. Nos vamos a las propiedades del JFrame y seleccionamos “Events”. Activamos “formWindowClosing” lo cual nos lleva a source. Escribimos el siguiente contenido.

```
private void formWindowClosing(java.awt.event.WindowEvent evt) {
    timer.stop();
}
```

9. Verificamos que nuestro programa este corriendo.



Bibliografía Preliminar.

[1] E. Gamma, R. Helm, R. Johnson y J. Vlissides, Elements of Reusable Object-Oriented Software, Addison-Wesley Professional, 1994.

[2] D. Hearn y M. P. Baker, Gráficos por computadora con OpenGL, Madrid: Pearson Educación S.A., 2006.

[3] I. Fundación Wikimedia, «Wikepedia,» [En línea]. Available: https://es.wikipedia.org/wiki/Pac-Man. [Último acceso: Enero 2018].

No. 8. Dibujos animados.

OBJETIVO

Crear una un hilo para mostrar la concurrencia. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, usando una secuencia de imágenes.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 3: Programación concurrente (MultiHilos). 3.1 Concepto de hilo. 3.2 Comparación de un programa de flujo único contra uno de flujo múltiple. 3.3 Creación y control de hilos. 3.4 Sincronización de hilos computación.	**No. 8. Dibujos animados.** Objetivo. Crear una un hilo para mostrar la concurrencia. **No. 9. Simulación de saldos en un banco hipotético.** Objetivo. Crear una un hilo para mostrar la concurrencia y sincronización. **No. 10. Juego animado parte I.** Objetivo. Crear varios hilos para mostrar interacción. **No. 11. Juego animado parte II.** Objetivo. Crear una aplicación con menús de librería y gráficos. **No. 12. Juego animado parte III.** Objetivo. Crear una aplicación con menús de librería y gráficos y tener características de persistencia.

INTRODUCCIÓN

El modelo de programación en secuencia es realizar una cosa a la vez ejecutando instrucciones secuencialmente de acuerdo a la semántica del lenguaje. Los hilos se ejecutan simultáneamente y asincrónicamente con respecto a si mismos. Comparten un mismo espacio de memoria, es decir, tienen acceso a las variables y objetos. [1]

La clase Timer puede ser considerada obsoleta y con limitaciones. La clase ScheduleThreadPoolExecutor resuelve muchos problemas de la clase Timer. Para animaciones se prefiere el método scheduleWithFixedDelay. [2]

Un Sprite es un objeto gráfico en movimiento, puede representar a un jugador o ser manejado por código. [3]

Parte de código empleado en este ejercicio se tomaron del usuario ForeignGuyMike de YouTube. [4]

Las animaciones Raccoon se han tomado del sitio OpenGameArt. [5]

Estas animaciones por facilidad fueron reducidas de tamaño usando un archivo batch en PHP (Apéndice I) empleando ImageMagick ©.

El programa “SpriteTracer” para almacenar las animaciones en un solo archivo de dibujo ha sido proporcionado por Sprite Database. [6]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para crear Sprites en Java.
4. Investigar uso de ScheduleThreadPoolExecutor, carga y tratamiento de imágenes.

5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de la declaración y propiedades de las clases ScheduleThreadPoolExecutor.
2. Seleccionar a varios participantes para exponer la creación de Sprites.
3. Seleccionar a voluntarios para explicar el uso de Sprites a partir de imágenes.
4. Llenar el recurso de Wiki con las definiciones de Threads, Sprites, Images.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo la creación de un dibujo formado a partir de una secuencia de imágenes. Entregar información del paso anterior.
3. Participar en Moodle con el glosario de términos incorporando los conceptos de Threads ventajas y desventajas.

Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   "Hacer una secuencia gráfica (en inglés clip-art) de cualquier animación tomada de una librería gratuita mostrando estados como reposo, caminar y brincar".

   El algoritmo de solución aquí propuesto es: "Crear la clase Animacion, definir parámetros para mostrar un arreglo de grupos de imágenes las cuales se mostrarán en secuencia de acuerdo al tipo de estado. A través de la clase Tecla se capturará que decisión toma el usuario Reposo, Caminar Izquierda, Caminar Derecha. Dibujar los estados en un JPanel, periódicamente actualizar la posición".

2. Crear un proyecto llamado "Po8Animacion" y con JFrame "MiVentana" en el paquete "paketeAnimes".
3. En diseño (Design). Podemos poner un título a nuestra Ventana. "Dibujos Animados". Podemos cambiar las propiedades Font. Para el Font se escogió Tahoma 24.
4. Agregamos un JPanel con nombre MiPanel. En propiedades (Properties) de MiPanel escogemos minimunSize y establecemos 640 x 480.
5. Nos vamos a "Source" en "MiVentana" y agregamos los siguientes atributos. Y posteriormente importamos las librerías correspondientes.

```
// dimensiones del panel
public static final int ANCHO = 640;
public static final int ALTO = 480;
private BufferedImage bimagen;
private Graphics2D g;
private final int FPS = 30;//60;
private final long tiempoEspera = 1000 / FPS;
private ScheduledExecutorService executor;
```

6. Seleccionamos MiVentana en diseño y en Eventos (Events) vamos a crear los eventos relacionados con `formWindowActivated`, `formWindowClosing`, `formKeyPressed` y `formKeyReleased`. Recordando que en cada caso nos llevará al código volvemos a diseño y repetimos el proceso. No olvidar importar las librerías. Note que hasta el paso 8 crearemos la clase Teclas así que en `formKeyPressed y formKeyReleased` nos va a marcar un error.

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    bimagen = new BufferedImage(
              ANCHO, ALTO, BufferedImage.TYPE_INT_RGB);
    g = (Graphics2D) bimagen.getGraphics();
    executor = Executors.newScheduledThreadPool(1);
    executor.scheduleWithFixedDelay(
              this, 0, tiempoEspera, TimeUnit.MILLISECONDS);
}

private void formWindowClosing(java.awt.event.WindowEvent evt) {
    executor.shutdownNow();
}

private void formKeyPressed(java.awt.event.KeyEvent evt) {
    Teclas.setTecla(evt.getKeyCode(), true);
    evt.consume();
}

private void formKeyReleased(java.awt.event.KeyEvent evt) {
    Teclas.setTecla(evt.getKeyCode(), false);
}
```

7. A la clase MiVentana le indicamos que implemente Runnable. Nos avisará con un foco rojo y ahí le indicamos que implemente todos los métodos abstractos (que en realidad es la generación del método run(). Éste método, por lo general, lo agrega casi al final del código. Run llamara a la actualización y dibujado de la animación.
   a. Implementando Runnable.

```
public class MiVentana extends javax.swing.JFrame implements Runnable {
```

   b. Generando run().

```
@Override
public void run() {
  throw new UnsupportedOperationException("Not supported yet.");
  //To change body of generated methods, choose Tools | Templates.
}
```

   c. Acondicionando run y creando los métodos sólo para separar la lógica de las acciones.

```
private void DibujaEnBuffer() {
    g.clearRect(0, 0, ANCHO, ALTO); //Limpiar el área
}
```

```
    private void dibujaEnPanel() {
        Graphics g2 = miPanel.getGraphics();
        g2.drawImage(bimagen, 0, 0, this);
        g2.dispose();
    }

    @Override
    public void run() {
        Teclas.update();
        DibujaEnBuffer();
        dibujaEnPanel();
    }
```

8. Para capturar la decisión del usuario creamos la clase Tecla.
    a. Nos aseguramos que esté seleccionado el proyecto “Po8Animacion”.
    b. Con el botón derecho del ratón seleccionamos “New” -> “Java Enum”. Es probable que no esté directamente disponible. Si es el caso seleccionamos “Other” y en “Java” aparece “Java Enum”.
    c. En el nombre del Enum escribimos “Teclas” y en paquete seleccionamos “paketeAnimes”.
    d. Una vez en el editor borramos los comentarios.

```
package paketeAnimes;

public enum Teclas {

}
```

    e. Incorporamos los atributos siguientes a la clase.

```
// Mover IZQuierda, DERecha, Brincar ARRiba,
IZQ, DER, ARR; // <- Constantes

private boolean tecla;    // Tecla Actual
private boolean antTecla; // Tecla Anterior
```

    f. Agregamos un método setAnt para guardar el estado previo de una tecla:

```
    private void setAnt () {
        antTecla = tecla;
    }
```

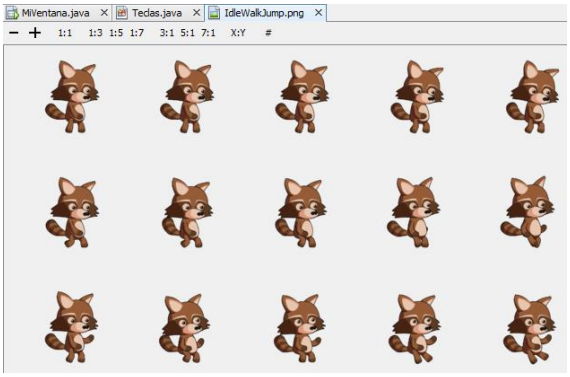
    g. Declaramos el método setTecla que recibe dos valores la tecla y si se ha presionado o ya no está presionada. Importamos `java.awt.event.KeyEvent`.

```
public static void setTecla(int i, boolean b) {
    switch (i) {
        case KeyEvent.VK_LEFT:
            IZQ.tecla = b;
            break;
        case KeyEvent.VK_RIGHT:
            DER.tecla = b;
            break;
        case KeyEvent.VK_UP:
            ARR.tecla = b;
            break;
        default:
            break;
    }
}
```

h. Incorporamos otros dos métodos uno actualiza para cada tecla almacenar su estado anterior y otra para indicar que está presionada.

```
public static void actualiza() {
    for (Teclas ti : values()) {
        ti.setAnt();
    }
}

public boolean estaPresionada() {
    return tecla && !antTecla;
}
```

9. Debemos de crear un archivo de todas las animaciones de Raccoon disminuyendo el tamaño a 128x128 pixeles y posteriormente usando SpriteTracer con once imágenes por renglón y ponerlo dentro del proyecto en el subdirectorio paketeAnimes. En éste caso el nombre del archivo es IdleWalkJump.png.



10. Para mostrar la animación requerimos crear una clase denominada “Animacion”.
    a. Nos aseguramos que esté seleccionado el proyecto “Po8Animacion”.
    b. Con el botón derecho del ratón seleccionamos “New” -> “Java Class”.
    c. En el nombre de la clase escribimos “Animacion” y en paquete seleccionamos “paketeAnimes”.
    d. Una vez en el editor borramos los comentarios e incorporamos los atributos siguientes.

```
private final int REPOSO = 0, CAMINANDO = 1, BRINCANDO = 2;
private final int ANCHO = 128, ALTO = 128;
private final int numCuadros = 11;
private ArrayList<BufferedImage[]> dibujos;
private BufferedImage[] cuadros;

private long tiempoArranque;
private long tiempoEspera;

private boolean vistoUnaVez;
private boolean viendoDerecha = true;

private int accion;
private int cuadroaMostrar;

private static final int MAXDER = MiVentana.ANCHO - 50;
private static final int MAXIZQ = 50;
private static final int MAXABA = MiVentana.ALTO / 3 * 2;

double x, y, dx, dy;
```

```
final double velMovimiento = 1.6;

//Movimiento
boolean izquierda, derecha;
```

a. Agregamos un constructor. Con el teclado presionamos <Alt><Ins> nos aparece un menú de contexto y seleccionamos "Constructor" en el nuevo menú no seleccionamos ningún atributos y damos generar:

```
public Animacion() {
}
```

b. Leemos las imágenes del archivo `"IdleWalkJump.png"`. Y creamos un arreglo de tres animaciones con once dibujos cada una. Modificamos el constructor a la siguiente forma (note que setReposo() no está creado todavía):

```
public Animacion() {
    try {
        BufferedImage spritesheet = ImageIO.read(
                getClass().getResourceAsStream(
                        "/PaketeAnimes/IdleWalkJump.png"
                )
        );

        dibujos = new ArrayList<>();
        for (int i = 0; i < 3; i++) { //Tres animaciones
            BufferedImage[] bi
                    = new BufferedImage[numCuadros];

            for (int j = 0; j < numCuadros; j++) {
                bi[j] = spritesheet.getSubimage(
                        j * ANCHO,
                        i * ALTO,
                        ANCHO,
                        ALTO
                );
            }
            dibujos.add(bi); //Reposo, Caminar, Saltar
        }

    } catch (IOException e) {
        System.out.println("Error al leer dibujos");
    }

    x = MiVentana.ANCHO / 2;
    y = MAXABA;
    setReposo();//Inicia en Reposo
}
```

c. Incorporamos los métodos `setReposo(),setCaminando()` y `setCuadros()` que definen estar en Reposo, estar Caminando y a cual grupo de imágenes (cuadros) corresponden a cada estado.

```
private void setReposo() {
    accion = REPOSO;
    setCuadros(dibujos.get(REPOSO), REPOSO);
    tiempoEspera = 200;
}

private void setCaminando() {
    accion = CAMINANDO;
    setCuadros(dibujos.get(CAMINANDO), CAMINANDO);
    tiempoEspera = 100;
```

```
        }

        private void setCuadros(BufferedImage[] cuadros, int
    accion) {
            this.cuadros = cuadros;
            cuadroaMostrar = 0;
            this.accion = accion;
            vistoUnaVez = false;
        }
```

d. Agregamos el método `actualiza()` el cual define el incremento en x si va a la izquierda o a la derecha. En caso que llegue al borde izquierdo o derecho deja de caminar y pasa al estado de reposo. También calcula si ya ha pasado tiempo suficiente para mostrar el siguiente dibujo (cuadro).

```
public void actualiza() {
    if (izquierda) {
        dx = -velMovimiento;
    } else if (derecha) {
        dx = velMovimiento;
    } else {
        dx = dy = 0;
    }

    x += dx;

    boolean q = false;
    if (x < MAXIZQ) {
        x = MAXIZQ;
        q = true;
    } else if (x > MAXDER) {
        x = MAXDER;
        q = true;
    }
    if (q) {
        izquierda = derecha = false;
        setReposo();
    }
    if (Teclas.IZQ.estaPresionada()) {
        viendoDerecha = false;
        izquierda = true;
        derecha = false;
        setCaminando();
    }
    if (Teclas.DER.estaPresionada()) {
        viendoDerecha = true;
        izquierda = false;
        derecha = true;
        setCaminando();
    }
    //Tiempo en milisegs
    long tiempoHaPasado
            = (System.nanoTime() - tiempoArranque) / 1000000;
    if (tiempoHaPasado > tiempoEspera) {
        cuadroaMostrar++;
        tiempoArranque = System.nanoTime();
    }
    if (cuadroaMostrar == cuadros.length) {
        cuadroaMostrar = 0;
        vistoUnaVez = true;
    }
}
```

e. Finalmente vamos a dibujar el cuadro que corresponda a la animación. Y dependiendo si está viendo a la derecha el cuadro se va a dibujar como está en grabado originalmente. En caso que no esté viendo a la derecha (está viendo a la izquierda), se dibujará invertido.

```
public void dibuja(java.awt.Graphics2D g) {
    if (viendoDerecha) {
        g.drawImage(
                cuadros[cuadroaMostrar],
                (int) (x - ANCHO / 2),
                (int) (y - ALTO / 2),
                null
        );
    } else {
        g.drawImage(
                cuadros[cuadroaMostrar],
                (int) (x - ANCHO / 2 + ANCHO),
                (int) (y - ALTO / 2),
                -ANCHO,
                ALTO,
                null
        );
    }
}
```

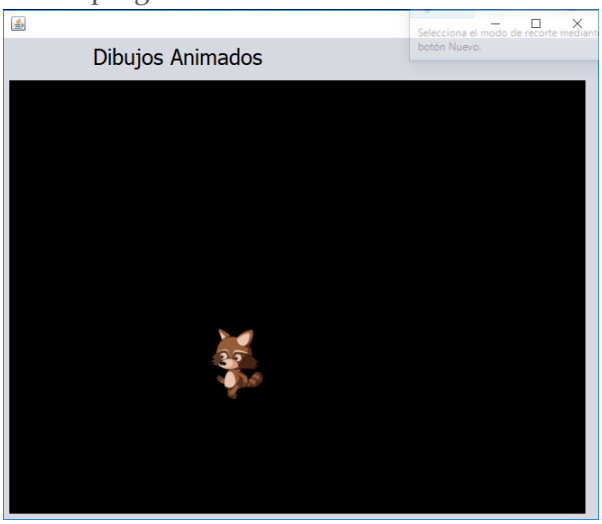
11. Ya creada la clase Animación hay que agregar un atributo en la clase ventana después del atributo `private ScheduledExecutorService executor;`

```
private final Animacion anima;
```

12. En el constructor creamos una instancia de animación y dibujamos en el método `DibujaEnBuffer()` y actualizamos en `run()` de tal suerte que queda:

```
public MiVentana() {
    initComponents();
    anima = new Animacion();
}

private void DibujaEnBuffer() {
    g.clearRect(0, 0, ANCHO, ALTO);
    anima.dibuja(g);
}

@Override
public void run() {
    anima.actualiza();
    Teclas.actualiza();
    DibujaEnBuffer();
    dibujaEnPanel();
}
```

13. Verificamos que nuestro programa este corriendo.



Bibliografía Preliminar.

[1] B. Goetz, T. Peierls, J. Bloch, J. Bowbeer, D. Holmes y D. Lea, Java Concurrency in Practice, Addison Wesley Professional.

[2] S. Oaks y H. Wong, Java Threads, 3rd Edition, O'Reilly Media, 2009.

[3] A. Davison, Killer Game Programming, Sebastopol: O'Reilly, 2005.

[4] ForeignGuyMike, «Java 2D Game Programming Platformer Tutorial,» YouTube, [En línea]. Available: https://www.youtube.com/watch?v=9dzhgsVaiSo. [Último acceso: Nov 2017].

[5] Null-Painter-Error, «OpenGameArt,» Raccoon, [En línea]. Available: https://opengameart.org/content/cute-raccoon-2d-game-sprite-and-animations. [Último acceso: Diciembre 2017].

[6] Grim, «Sprite Database,» [En línea]. Available: http://spritedatabase.net/download. [Último acceso: Diciembre 2017].

Apéndice I.
Archivo Fuente en PHP para conversión de tamaño de varios archivos png.

```php
<?php

$path = realpath('/var/www/html/samba/raccoon');

$objects = new RecursiveIteratorIterator(new
RecursiveDirectoryIterator($path),
RecursiveIteratorIterator::SELF_FIRST);
$i=0;
$sdira = '';
$j=0;
foreach($objects as $name => $object){
    $arch = pathinfo($name);

    if(array_key_exists('extension',$arch)
        && $arch['extension']==='png'
        && substr($arch['filename'],0,1)!=='_'){

        echo "$name\n<br>";
        $dir = $arch['dirname'];
        $sdir = substr($dir, strrpos($dir, '/')+ 1);
        if( $sdir === $sdira ){
            $i++;
        }else{
            $i=0;
            $sdira = $sdir;
        }
        $file = '_'. $sdir . sprintf("%02d", $i);
        $nvo = $dir.'/'.$file.'.png';
        echo "$sdir $nvo\n<br>";

        $im = new Imagick($name);

        //$im->cropImage(580, 580, 200, 70); //790 623
        //$im->setImagePage(0, 0, 0, 0);

        $im->cropThumbnailImage(128,128);
        //$im->SetImageFormat( 'jpeg' );

        $im->writeImage( $nvo );

        /*
        if($j++>10){
            break;
        }*/
    }
}

?>
```

No. 9. Simulación de saldos en un banco hipotético.

OBJETIVO

Crear varios hilos para mostrar la concurrencia sobre un recurso. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, usando varios usuarios haciendo retiros y depósitos a una cuenta de banco.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 3: Programación concurrente (MultiHilos). 3.1 Concepto de hilo. 3.2 Comparación de un programa de flujo único contra uno de flujo múltiple. 3.3 Creación y control de hilos. 3.4 Sincronización de hilos computación.	**No. 8. Dibujos animados.** Objetivo. Crear una un hilo para mostrar la concurrencia. **No. 9. Simulación de saldos en un banco hipotético.** Objetivo. Crear una un hilo para mostrar la concurrencia y sincronización. **No. 10. Juego animado parte I.** Objetivo. Crear varios hilos para mostrar interacción. **No. 11. Juego animado parte II.** Objetivo. Crear una aplicación con menús de librería y gráficos. **No. 12. Juego animado parte III.** Objetivo. Crear una aplicación con menús de librería y gráficos y tener características de persistencia.

INTRODUCCIÓN

Cuando varios hilos (Threads) comparten un mismo objeto y es modificado por ellos se obtienen resultados impredecibles. Esto puede resolverse dando a un hilo a la vez acceso exclusivo al objeto compartido. [1]

ReentrantLock implementa Lock, proporcionando las mismas garantías de exclusión mutua y visibilidad de memoria que sincronizadas. La adquisición de un ReentrantLock tiene la misma semántica de memoria que ingresar un bloque sincronizado, y la liberación de un ReentrantLock tiene la misma semántica de memoria que la salida de un bloque sincronizado. [2]

Para usar colas tipo "Condition" es identificando la condición predicado que un objeto debe esperar. Esta condición predicado es una condición previa que depende del estado del objeto. [2]

La clase usada en este ejercicio CuentaBancaria está basada en un recurso de GitHub. [3]

Una implementación de Colas llamada ConcurrentLinkedQueue permite escrituras y lecturas basadas en accesos simultáneos por varios hilos. [4]

La clase ExecutorService simplifican la programación concurrente manejando los objetos de tipo Thread por nosotros. [4]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.

3. Conectarse a Internet para obtener la información para buscar información en Java para el uso de Hilos, Ejecutores, Locks, Condition.
4. Investigar uso de newFixedThreadPool, bloqueo, sincronización.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de la declaración y propiedades de las clases Locks, Condition, CompletionService.
2. Seleccionar a varios participantes para exponer la creación de Hilos usando la clase Thread, la interface Runnable.
3. Seleccionar a voluntarios para explicar el uso de bloqueo.
4. Llenar el recurso de Wiki con las definiciones de Hilos, ejecutores, bloqueo, semáforos, atomicidad.

Reporte de los alumnos (resultados)

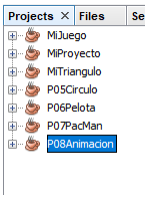
1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo la creación varios hilos para el acceso a un recurso.
3. Participar en Moodle con el glosario de términos incorporando los conceptos de Hilos, bloqueo, sincronización, colas concurrentes.

Procedimiento:

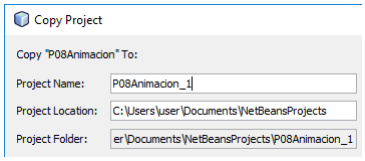
1. El tipo de problema a resolver se propone lo siguiente
   "Simular el comportamiento de una cuenta bancaria donde varios usuarios usen cajeros automáticos haciendo retiros y depósitos en forma simultánea. Sujeto a que el retiro debe tener fondos y esperar a que haya. Iniciar con un saldo de Cero y hacer la misma cantidad de depósitos y retiros para terminar con saldo igual a Cero".

   El algoritmo de solución aquí propuesto es: "Crear una cuenta y sólo permitir un solo acceso a la vez a la cuenta en caso de no contar con fondos establecer una condición de espera hasta que haya. Crear 4 hilos para simular a 4 cajeros automáticos. Se generarán 50 depósitos y 50 retiros. Por cada depósito o retiro el proceso se llevará un tiempo de un segundo y definiremos el monto de cada retiro y depósito en 100. Dibujar los cambios de saldo en un JPanel, periódicamente actualizar el estado de cada cajero, deposito, retiro y esperando fondos".

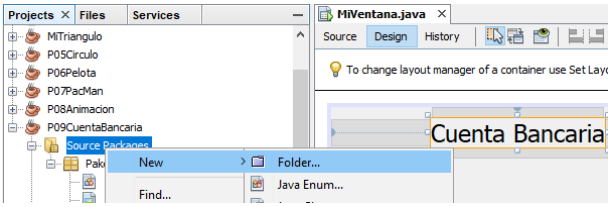
2. Para la creación de este ejercicio partiremos del ejercicio anterior incorporando solo los elementos adicionales que hacen posible la simulación de la cuenta bancaria con acceso a cuatro cajeros.
    a. Nos dirigimos a nuestra ventana de proyectos.



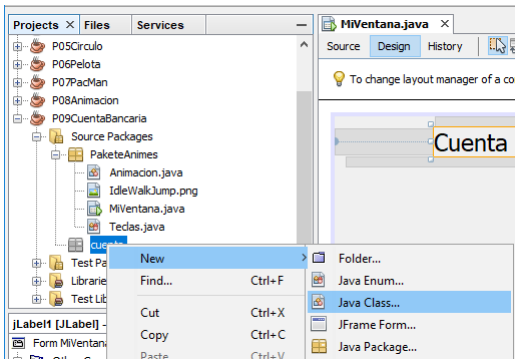
    b. Con el botón derecho nos aparece el menú de contexto donde escogemos copy.



    c. Nos sugiere “P08Animacion_1” pero vamos a escribir “P09CuentaBancaria” y seleccionamos “Copy”. Después que se ha hecho la copia aparece seleccionado nuestro nuevo proyecto.
    d. Verificar que no tenemos ningún archivo abierto para no modificar un archivo que sea de otro proyecto.
3. Abrimos MiVentana y en diseño (Design). Cambiamos el título a nuestra Ventana. “Dibujos Animados” por “Cuenta Bancaria”.
4. En un solo subdirectorio (paquete en Java) incorporaremos las clases requeridas para nuestra cuenta bancaria. Crear el paquete “cuenta”. En nuestro proyecto escogemos con el botón derecho sobre “Source Packages”. Escogemos “New” -> “Folder”. Y en “Folder Name:” escribimos “cuenta” y seleccionamos “Finish”.



5. En el paquete vamos a guardar cada uno de los eventos conforme vayan ocurriendo. Cada evento (deposito, retiro, espera fondos) se almacenara en una estructura que tome un estado “instantáneo” para ello requeriremos la creación de la clase “Info”.
    a. Sobre el paquete “cuenta” con el botón derecho seleccionamos “New” “Java Class”.



b. En nombre de la clase escribimos “Info” y damos “Finish”.

c. Una vez creado el esqueleto de la clase “Info” incorporamos los atributos siguientes:

```
public double saldo;
public String accion;
public double monto;
public int cajero;
```

d. Creamos dos constructores uno sin atributos y otro con todos los atributos:

```
public Info() {
    accion = "_";
}

public Info(double saldo, String accion, double monto, int cajero) {
    this.saldo = saldo;
    this.accion = accion;
    this.monto = monto;
    this.cajero = cajero;
}
```

e. En caso de monitorear una instancia “Info” reescribimos toString. Con el botón derecho sobre la clase o con la combinación de teclas <Alt><Insert> seleccionamos “toString()”. Y por default están seleccionados todos los atributos y damos “Generate”.

```
    @Override
    public String toString() {
        return "Info{" + "saldo=" + saldo + ", accion=" + accion + ",
monto=" + monto + ", cajero=" + cajero + '}';
    }
```

6. De forma similar al paso anterior ahora crearemos la clase “`CuentaBancaria`” la cual contiene cuatro atributos. El “`saldo`”de la cuenta bancaria. Tiene “`bloqueoDeSaldo`” con el cual sólo permitiremos un único acceso a saldo, “`condicionHayFondos`” que bloqueará un retiro cuando no haya fondos suficientes y finalmente para guardar el estado de cada evento la cola concurrente “L”.

   a. La clase y los atributos se muestra a continuación.

```
package cuenta;

public class CuentaBancaria {

    private double saldo;
    private final Lock bloqueoDeSaldo;
```

```
    private final Condition condicionHayFondos;

    public static final ConcurrentLinkedQueue<Info> L = new
ConcurrentLinkedQueue<>();
}
```

b. Aparecen tres focos rojos. En cada uno damos un clic para importar cada clase (nos aparecen dos círculos rojos todavía adelante lo corregiremos).
c. Creamos un constructor por default (sin parámetros). Aquí inicializaremos los atributos faltantes.

```
/**
 * Saldo cero
 */
public CuentaBancaria() {
    saldo = 0;
    bloqueoDeSaldo = new ReentrantLock();
    condicionHayFondos = bloqueoDeSaldo.newCondition();
}
```

d. Importamos "`ReentrantLock`".
e. Creamos el método "`addInfo`" el cual nos almacenará la secuencia de eventos en la cola concurrente "`L`".

```
private void addInfo(String accion, double monto) {
    String nombre = Thread.currentThread().getName();
    int cajero = nombre.charAt(nombre.length() - 1) - '0';
    L.add(new Info(saldo, accion, monto, cajero));
}
```

f. Agregaremos el método "`deposito`" al efectuar un depósito adquirimos o esperamos el "`bloqueoDeSaldo`". Le indicaremos en caso que haya alguien esperando hacer un retiro y no haya fondos le mandaremos un aviso "`signalAll`" para que deje de esperar.

```
public void deposito(double monto, int espera) {
    bloqueoDeSaldo.lock();
    try {
        saldo += monto;

        addInfo("Deposito", monto);
        Thread.sleep(espera);

        condicionHayFondos.signalAll();
    } catch (InterruptedException ex) {
    } finally {
        bloqueoDeSaldo.unlock();
    }
}
```

a. Agregaremos el método "`retiro`" al efectuar un retiro adquirimos o esperamos el "`bloqueoDeSaldo`". Le indicaremos una espera en caso que no haya fondos "`await`".

```
public void retiro(double monto, int espera) {
    bloqueoDeSaldo.lock();
    try {
        while (saldo < monto) {

            addInfo("Espera Fondos", monto);

            condicionHayFondos.await();
```

```
                }
                saldo -= monto;

                addInfo("Retiro", monto);
                Thread.sleep(espera);
            } catch (InterruptedException ex) {
            } finally {
                bloqueoDeSaldo.unlock();
            }
        }
```

b. Para finalizar la clase agregaremos la lectura de saldo.

```
    public double getSaldo() {
        return saldo;
    }
```

7. Haremos ahora la clase "`Operaciones`" con la cual efectuaremos cincuenta retiros y cincuenta depósitos. Cada operación tendrá un monto de 100 y durará un segundo (1000 milisegundos). Esperaremos que se terminen todas las operaciones "`CompletionService`" para finalmente terminar nuestro pool de hilos con "`shutdown`" que simula a los cuatro cajeros "`newFixedThreadPool(4);`". Toda la clase a su vez estará ejecutada en un hilo por eso se reescribe "`run`" e implementa "`Runnable`".

a. Creamos la clase y definimos monto, espera y cuenta bancaria.

```
package cuenta;

public class Operaciones implements Runnable {

   static final int MONTO = 100;
   static final int ESPERA = 1000;//Un segundo
   public static final CuentaBancaria CUENTA = new CuentaBancaria();
}
```

b. Como implementa "" aparece un foco rojo. Le damos clic y le indicamos que implemente todos los métodos abstractos y solo es "".

```
    @Override
    public void run() {
        throw new UnsupportedOperationException("Not supported yet.");
//To change body of generated methods, choose Tools | Templates.
    }
```

c. Dentro de "`run`" creamos los cuatro cajeros "`newFixedThreadPool(4)`" y estaremos pendiente de la terminación de las tareas "`pool`".

```
    @Override
    public void run() {
        ExecutorService threadPool = Executors.newFixedThreadPool(4);

        CompletionService<String> pool = new
ExecutorCompletionService<>(threadPool);

        for (int i = 0; i < 50; i++) {
            pool.submit(() -> {
                CUENTA.retiro(MONTO, ESPERA);
            }, "" + i);
            pool.submit(() -> {
                CUENTA.deposito(MONTO, ESPERA);
            }, "" + i);
        }

        for (int i = 0; i < 50 * 2; i++) {
```

```
            try {
                String result = pool.take().get();
                System.out.println(result);
            } catch (InterruptedException | ExecutionException ex) {
            }
        }

        threadPool.shutdown();
    }
```

d. Importamos Las librerías correspondientes:

```
import java.util.concurrent.CompletionService;
import java.util.concurrent.ExecutionException;
import java.util.concurrent.ExecutorCompletionService;
import java.util.concurrent.ExecutorService;
import java.util.concurrent.Executors;
```

8. Ya tenemos la lógica de nuestro programa solo hay que mostrar el estado de las operaciones bancarias almacenadas en la Cola concurrente "`L`". Para lograr mostrar el avance abrimos la clase "`MiVentana`" en modo "Source" y seguimos los siguientes pasos.

   a. Creamos posiciones para los cuatro cajeros y asociamos los mensajes con cuatro variables de información. Insertamos al final de la clase antes del último cierra paréntesis los siguientes atributos. Nos pide importar "`cuenta.Info`".

```
int POS[][] = {{30, 150}, {180, 150}, {330, 150}, {480, 150}};
Info INFO[] = {new Info(), new Info(), new Info(), new Info()};
boolean generado; //Crear objetos una sola vez
```

   a. Agregamos un nuevo método "`updateCuentaBancaria`" y su función será revisar la Cola "`L`" extraer un aviso y asignarla a una de las cuatro posiciones asociadas con cada cajero. Nos pide importar "`cuenta.CuentaBancaria`".

```
private void updateCuentaBancaria() {
    while (!CuentaBancaria.L.isEmpty()) {
        Info info;

        info = CuentaBancaria.L.remove();
        info.cajero--;//Porque Threads Name inicia con 1
        INFO[info.cajero] = info;
        System.out.println(info.toString());//Monitoreo de info
    }
}
```

   b. Incorporamos el método "`dibujaCuentaBancaria`" con el cual dibujaremos continuamente la información vertida para cada "cajero" en una posición, color y tipo de letra determinados. Importamos las librerías de "`Font`", "`Color`" y "`cuenta.Operaciones`".

```
    private void dibujaCuentaBancaria(Graphics2D g) {
        int x, y;

        g.setFont(new Font("TimesRoman", Font.PLAIN, 20));
        g.setColor(Color.white);
        for (Info info : INFO) {
            x = POS[info.cajero][0];
```

```
                y = POS[info.cajero][1] + 20;
                g.drawString(info.accion, x, y);
            }

            g.setColor(Color.green);
            for (int i = 0; i < 4; i++) {
                x = POS[i][0];
                y = POS[i][1];
                g.drawString("Cajero " + (i + 1), x, y);
            }

            g.setFont(new Font("TimesRoman", Font.PLAIN, 30));
            g.setColor(Color.yellow);
            g.drawString("Saldo: " + Operaciones.CUENTA.getSaldo(),
250, 50);
        }
```

c. Agregamos dentro del método "`run`" y dentro del método "`DibujaEnBuffer`" las instrucciones "`updateCuentaBancaria`" y "`dibujaCuentaBancaria`" respectivamente.

```
@Override
public void run() {
    updateCuentaBancaria();
    anima.update();
    Teclas.update();
    DibujaEnBuffer();
    dibujaEnPanel();
}

private void DibujaEnBuffer() {
    g.clearRect(0, 0, ANCHO, ALTO);
    anima.draw(g);
    dibujaCuentaBancaria(g);
}
```

b. Solo nos falta crear un hilo (Thread) para iniciar la clase "`Operaciones`". Para ello en el evento "`formWindowActivated`" agregamos la construcción del hilo y lo arrancamos. Para no crear los objetos cada vez que se oculte y muestre la ventana se incluye "`!generado`" así solo se crearan una sola vez.

```
private void formWindowActivated(java.awt.event.WindowEvent evt)
{
    if (!generado) {
        generado = true;

        bimagen = new BufferedImage(
                ANCHO, ALTO, BufferedImage.TYPE_INT_RGB);
        g = (Graphics2D) bimagen.getGraphics();

        executor = Executors.newScheduledThreadPool(1);
        executor.scheduleWithFixedDelay(
                this, 0, tiempoEspera, TimeUnit.MILLISECONDS);

        Thread t = new Thread(new Operaciones());
        t.start();
    }
}
```

9. Verificamos que nuestro programa este corriendo.

Bibliografía Preliminar.

[1] H. M. D. P. J. Deitel, Como Programar en Java / 9 Ed., PEARSON, 2012.

[2] B. Goetz, T. Peierls, J. Bowbeer, D. Holmes y D. Lea, Java Concurrency in Practice, Addison Wesley Profesional, 2006.

[3] tacksoo, «GitHubGist,» 2012. [En línea]. Available: https://gist.github.com/tacksoo/4728939. [Último acceso: Diciembre 2017].

[4] B. Eckel, Piensa en Java, Madrid: Prentice Hall, 2003.

No. 10. Juego animado parte I.

OBJETIVO

Crear varios hilos para mostrar la interacción con el usuario. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, permitiendo que el usuario seleccione de un menú.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 3: Programación concurrente (MultiHilos). 3.1 Concepto de hilo. 3.2 Comparación de un programa de flujo único contra uno de flujo múltiple. 3.3 Creación y control de hilos. 3.4 Sincronización de hilos computación.	**No. 8. Dibujos animados.** Objetivo. Crear una un hilo para mostrar la concurrencia. **No. 9. Simulación de saldos en un banco hipotético.** Objetivo. Crear una un hilo para mostrar la concurrencia y sincronización. **No. 10. Juego animado parte I.** Objetivo. Crear varios hilos para mostrar interacción. **No. 11. Juego animado parte II.** Objetivo. Crear una aplicación con menús de librería y gráficos. **No. 12. Juego animado parte III.** Objetivo. Crear una aplicación con menús de librería y gráficos y tener características de persistencia.

INTRODUCCIÓN

La información de tipo de tiempo de ejecución (RTTI) le permite descubrir y usar información de tipo al ejecutar un programa. RTTI "tradicional", que supone que tiene todos los tipos disponibles en tiempo de compilación, y el mecanismo de reflexión, que le permite descubrir y utilizar información de clase únicamente en tiempo de ejecución. [1]
Hay dos métodos reflexivos para crear instancias de clases: `Constructor.newInstance()` y `Class.newInstance()`. [2]
A veces es importante que algunas clases tengan exactamente una instancia (Patrón Singleton). Si creamos más de una instancia, tendremos problemas como el comportamiento incorrecto del programa, el uso excesivo de recursos o resultados inconsistentes. [3]
Las clases son la mejor forma para lograr Modularidad. Sin embargo, para lograr de manera más efectiva las metas de abstracción y encapsulamiento para agrupar grupos de objetos se obtiene usando paquetes (packages). [4]

Material y equipo necesario.
El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:
Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para buscar información sobre Singleton, paquetes en Java.
4. Investigar uso de Modularidad, Abstracción, encapsulamiento.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de la declaración y propiedades de paquetes.
2. Seleccionar a varios participantes para exponer propiedades de Singleton, Clases en paquetes.
3. Seleccionar a voluntarios para explicar el uso de Modularidad, Abstracción y Encapsulamiento.
4. Llenar el recurso de Wiki con las definiciones de Modularidad, Abstracción y Encapsulamiento, Singleton, Patrones y Características de paquetes.

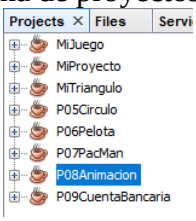
Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo el uso de varios paquetes y una clase Singleton.
3. Participar en Moodle con el glosario de términos incorporando los conceptos de Modularidad, Abstracción y Encapsulamiento, Paquetes.

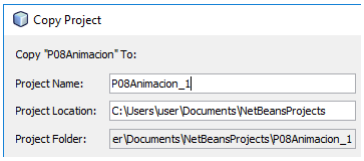
Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   "Crear un juego similar a "Space Invaders". Crear varios alienígenas y un cañón láser para eliminarlos hasta donde sea posible".

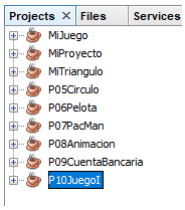
   El algoritmo de solución aquí propuesto es: "Crear tres etapas para elaborar el proyecto. En esta primera parte, definir paquetes para agrupar clases por funcionalidad. Crear una clase Singleton para administrar varias clases, tales como menú, nivel uno, nivel dos etc. Crear la clase Menu y la clase Nivel Uno. Dibujar los cambios de selección en el menú. Ampliar la clase Teclas."
2. Para la creación de este ejercicio partiremos del ejercicio número 8. Incorporaremos solo los elementos adicionales que hacen posible la clase Singleton (AdminNiveles).
   a. Nos dirigimos a nuestra ventana de proyectos.



   b. Con el botón derecho nos aparece el menú de contexto donde escogemos copy.



   c. Nos sugiere "P08Animacion_1" pero vamos a escribir "P10JuegoI" y seleccionamos "Copy". Después que se ha hecho la copia aparece seleccionado nuestro nuevo proyecto.
   d. Verificar que no tenemos ningún archivo abierto para no modificar un archivo que sea de otro proyecto.
   e. Seleccionamos el proyecto "P10JuegoI".



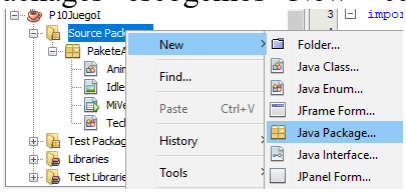
3. Abrimos MiVentana y en diseño (Design). Cambiamos el título a nuestra Ventana. “Dibujos Animados” por “Juego Parte I”.
4. Con la finalidad de generar objetos una sola vez cuando se activa la ventana de nuestro JFrame en el evento “formWindowActivated” hacemos lo siguiente.
   a. Agregamos a la clase “MiVentana” el atributo booleano “`generado`”.

```
// dimensiones del panel
public static final int ANCHO = 640;
public static final int ALTO = 480;
private BufferedImage bimagen;
private Graphics2D g;
private final int FPS = 30;//60;
private final long tiempoEspera = 1000 / FPS;
private ScheduledExecutorService executor;
private final Animacion anima;
private boolean generado;
```

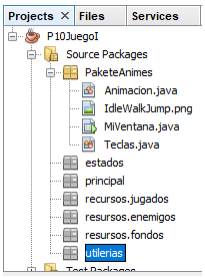
   b. Modificamos el código de “`formWindowActivated`” para no crear de Nuevo los objetos del evento.

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
        if (!generado) {
            generado = true;

            bimagen = new BufferedImage(
                    ANCHO, ALTO, BufferedImage.TYPE_INT_RGB);
            g = (Graphics2D) bimagen.getGraphics();

            executor = Executors.newScheduledThreadPool(1);
            executor.scheduleWithFixedDelay(
                    this, 0, tiempoEspera, TimeUnit.MILLISECONDS);
        }
}
```

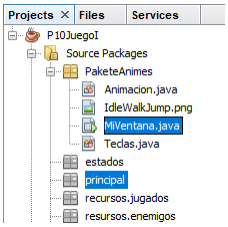
5. Todo nuestro código ha quedado dentro de un solo paquete “paqueteAnimes”. Vamos a incorporar varios paquetes para agrupar las clases por su funcionalidad. La clase “MiVentana” la guardaremos en un paquete “principal”. En el paquete “niveles” agregaremos los niveles de juego, donde menú es el nivel cero. Se definirá la lógica de los actores principales en “entidades”. Para guardar imágenes, agregaremos los paquetes “recursos.fondos”, “recursos.jugador”, “recursos.enemigos”. Un paquete “utilerias” para incorporar funciones de propósito general.
   a. No posicionamos en la ventana de “Proyectos” y con el botón derecho del ratón sobre “Source Packages” escogemos “New” “Java Package”.



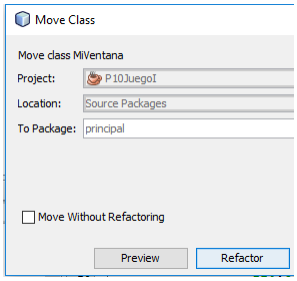
   b. Escribimos “principal” y damos “Finish”.
   c. Repetimos el procedimiento para “estados”, “entidades”, “recursos.fondos”, “recursos.jugador”, “recursos.enemigos”, “utilerías”.



d. Ahora con el cursor del ratón vamos a seleccionar MiVentana.java y la vamos a "arrastrar" (dar clic y mantener presionado moviendo el ratón) hasta el paquete "principal".

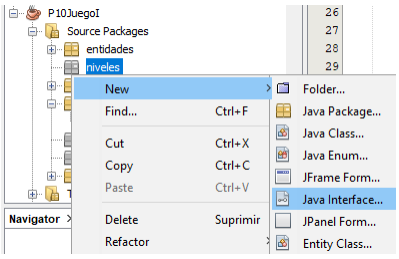


e. Nos aparece un menú donde seleccionamos Refactor.



f. Ahora movemos la clase "Teclas.java" al paquete "utilerias".
g. Es el turno de mover "Animacion.java" a al paquete "entidades".
h. Nos queda el archivo "IdleWalkJump.png" lo movemos al paquete "recursos.player". Antes de moverlo puede verificar que el programa esté corriendo. Ya después de movido no correrá. Para que corra de nuevo hay que modificar en el constructor de animación `"/PaketeAnimes/IdleWalkJump.png"` por `"/recursos/jugador/IdleWalkJump.png"`
i. Borramos el paquete PaketeAnimes. Con el botón derecho sobre cada uno dar "Delete" y confirmamos el borrado.

6. Supondremos que vamos a tener varios niveles iniciando con el nivel cero o menú, luego nivel uno de juego. Cada nivel va a tener dos métodos: Actualiza y Dibuja. Crearemos una interface que contenga estos métodos llamada Niveles. Crearemos la clase "Niv0Menu" y "NivIJuego" y otra clase enumerada que guarde todos los dos niveles y finalmente una clase que administre los niveles.
   a. Seleccionamos el paquete "niveles" e incorporamos una interface llamada Niveles y damos "Finish".



b. Dentro de la interface agregamos los métodos "`Actualiza`" y "`Dibuja`".

```
package niveles;

public interface Niveles {

    void Actualiza();

    void Dibuja(java.awt.Graphics2D g);
}
```

c. Ahora agregamos la clase NivoMenu que implemente la interface anterior.

d. Con el botón derecho sobre Niveles agregamos la clase "NivoMenu". Luego de generada agregamos "implements Niveles" con lo cual nos pedirá implementar los métodos abstractos. Le borramos la línea generada de "throw" y nos queda.

```
package niveles;

import java.awt.Graphics2D;

public class Niv0Menu implements Niveles {

    @Override
    public void Actualiza() {

    }

    @Override
    public void Dibuja(Graphics2D g) {

    }
}
```

e. Hacemos lo mismo con NivIJuego.

```
package niveles;

import java.awt.Graphics2D;

public class NivIJuego implements Niveles {

    @Override
    public void Actualiza() {

    }

    @Override
    public void Dibuja(Graphics2D g) {

    }
}
```

f. Incorporamos una clase que describa todos los estados posibles asociados con cada nivel y guarde una referencia a cada clase. Y usando reflexión que pueda crear una nueva instancia de cada clase guardada.

```
package niveles;

public enum Estados {
    MENU(Niv0Menu.class), JUEGO(NivIJuego.class);
```

```
    private final Class<Niveles> nivelClass;

    private Estados(Class nivelClass) {
        this.nivelClass = nivelClass;
    }

    public Niveles newInstancia() {
        try {
            return nivelClass.newInstance();
        } catch (InstantiationException | IllegalAccessException ex) {
        }
        return null;
    }
}
```

g. Para administrar los niveles se requiere una clase Singleton llamada "AdminNiveles" que con ayuda de una clase interna "Aux" nos permita cambiar de un nivel a otro y que inicie en Menu. Esta será a su vez una clase Proxy para invocar a "`Actualiza`" y a "`Dibuja`".

```
package niveles;

import java.awt.Graphics2D;

public final class AdminNiveles {

    private static class Aux {

        static final AdminNiveles INSTANCIA = new AdminNiveles();

        static Estados nivelActual;
        static Niveles instanciaActual;
    }

    public static AdminNiveles getInstance() {
        return Aux.INSTANCIA;
    }

    private AdminNiveles() {
        setEstado(Estados.MENU);
    }

    public void setEstado(Estados estado) {
        Aux.nivelActual = estado;
        Aux.instanciaActual = estado.newInstancia();
    }

    public void Actualiza() {
        try {
            Aux.instanciaActual.Actualiza();
        } catch (Exception e) {
        }
    }

    public void Dibuja(Graphics2D g) {
        try {
            Aux.instanciaActual.Dibuja(g);
        } catch (Exception e) {
        }
    }
}
```

7. Vamos a eliminar la animación "~~`anima`~~" que aparece al arrancar el programa de la clase MiVentana y reemplazarla por "`adminNiveles`".
   a. Borrar el atributo "`private final Animacion anima`" de la clase MiVentana e incorporar la administración de niveles a cargo de la referencia "`private AdminNiveles adminNiveles = AdminNiveles.getInstance()`";

```
        // dimensiones del panel
```

```
public static final int ANCHO = 640;
public static final int ALTO = 480;
private BufferedImage bimagen;
private Graphics2D g;
private final int FPS = 30;//60;
private final long tiempoEspera = 1000 / FPS;
private ScheduledExecutorService executor;
private final Animacion anima;
private boolean generado;
static AdminNiveles adminNiveles = AdminNiveles.getInstance();
```

b. Nos va marcar varios errores la ausencia de "`private final Animacion anima`" por lo que borramos las tres líneas donde se haga referencia a ella. La borramos del constructor, de "`DibujaEnBuffer`" y en "`run`". En forma opcional eliminamos el import de la animación.

c. Ahora que hemos eliminado la actualización y dibujo de "~~`anima`~~" vamos a llamar a "`adminNiveles`" para actualizarlo y dibujarlo (básicamente donde estaba "~~`anima`~~" se reemplaza por "`adminNiveles`".

```
private void DibujaEnBuffer() {
    g.clearRect(0, 0, ANCHO, ALTO);
    adminNiveles.Dibuja(g);
}

@Override
public void run() {
    adminNiveles.Actualiza();
    Teclas.actualiza();
    DibujaEnBuffer();
    dibujaEnPanel();
}
```

8. En la clase Teclas sólo están definidas `IZQ, DER, ARR` vamos a incorporar `ABA, ENT, ESC, SPC` representan flecha abajo, Enter, Esc, Space. Y modificamos "`setTecla`" para responder a la variable enumerada de acuerdo a la tecla correspondiente.

   a. Los enumerados nos quedan:

```
IZQ, DER, ARR, ABA, ENT, ESC, SPC;
```

   b. Y la clase "`setTecla`" agregamos los nuevos casos.

```
public static void setTecla(int i, boolean b) {
    switch (i) {
        case KeyEvent.VK_LEFT:
            IZQ.tecla = b;
            break;
        case KeyEvent.VK_RIGHT:
            DER.tecla = b;
            break;
        case KeyEvent.VK_UP:
            ARR.tecla = b;
            break;
        case KeyEvent.VK_DOWN:
            ABA.tecla = b;
            break;
        case KeyEvent.VK_ENTER:
            ENT.tecla = b;
            break;
        case KeyEvent.VK_ESCAPE:
            ESC.tecla = b;
            break;
        case KeyEvent.VK_SPACE:
            SPC.tecla = b;
            break;
        default:
```

```
                break;
            }
        }
```

9. Como Menu (“Niv0Menu”) va a ser lo primero que se muestre. El usuario seleccionara dando <Enter> entre tres opciones iniciar el “Juego Nivel I”, “Juego Nivel II” o “Salir”.
    a. Abrimos la clase “Niv0Menu” y agregamos los siguientes atributos. Importamos las librerías correspondientes.

```
private int seleccion = 0;
private final String[] opciones = {
    "Nivel I",
    "Nivel II",
    "Salir"
};
private final Color tituloColor;
private final Font tituloFuente;
private final Font font;
```

    b. Los errores de inicialización de variables se corregirán cuando agreguemos un constructor sin parámetros y les asignemos un valor.

```
public Niv0Menu() {
    tituloColor = Color.RED;
    tituloFuente = new Font(
            "Century Gothic",
            Font.PLAIN,
            60);

    font = new Font("Arial", Font.PLAIN, 50);
}
```

    c. El método “Actualiza” permite verifica que tecla ha presionado el teclado y actuar en consecuencia.

```
@Override
public void Actualiza() {
    if (Teclas.ENT.estaPresionada()) {
        switch (seleccion) {
            case 0:
                AdminNiveles.getInstance().setEstado(N1JUEGO);
                break;
            case 1:
                break;
            case 2:
                System.exit(0);
                break;
        }
    } else if (Teclas.ARR.estaPresionada()) {
        seleccion--;
        if (seleccion < 0) {
            seleccion = opciones.length - 1;
        }
    } else if (Teclas.ABA.estaPresionada()) {
        seleccion++;
        if (seleccion >= opciones.length) {
            seleccion = 0;
        }
    }
}
```

    d. Ahora para “Dibuja” de acuerdo a la selección con las flechas arriba y abajo se muestra en color blanco el “avance” de las opciones.

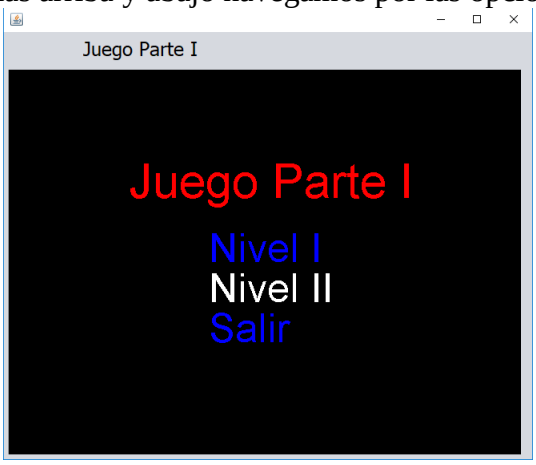
```
@Override
public void Dibuja(Graphics2D g) {
    g.setColor(tituloColor);
    g.setFont(tituloFuente);
    g.drawString("Juego Parte I", 150, 160);
```

```
            // dibujar las Opciones
            g.setFont(font);
            for (int i = 0; i < opciones.length; i++) {
                if (i == seleccion) {
                    g.setColor(Color.WHITE);
                } else {
                    g.setColor(Color.BLUE);
                }
                g.drawString(opciones[i], 250, 240 + i * 50);
            }
        }
```

10. La clase de Juego ("`NivIJuego`") vamos a introducir la animación y cuando el usuario seleccione ESC nos regresamos al Menu ("`Niv0Menu`").

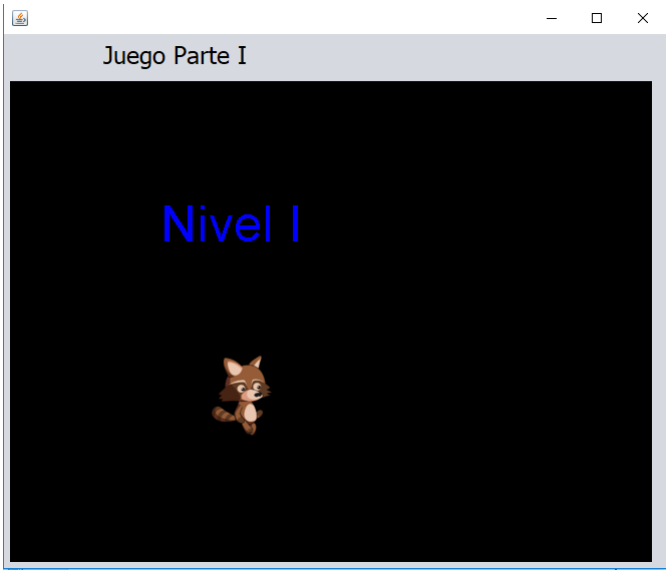
```
package niveles;

import entidades.Animacion;
import java.awt.Graphics2D;
import static niveles.Estados.N0MENU;
import utilerias.Teclas;

public class NivIJuego implements Niveles {

    Animacion anima = new Animacion();

    @Override
    public void Actualiza() {
        anima.actualiza();
        if (Teclas.ESC.estaPresionada()) {
            AdminNiveles.getInstance().setEstado(N0MENU);
        }
    }

    @Override
    public void Dibuja(Graphics2D g) {
        g.drawString("Nivel I", 150, 160);
        anima.dibuja(g);
    }
}
```

11. Verificamos que nuestro programa este corriendo.
    a. Con las flechas arriba y abajo navegamos por las opciones.



    b. Y al dar <Enter> si estamos en "Salir" termina nuestro programa.

c. Pero si damos <Enter> en “Nivel I”. Nos muestra:



d. Al dar <Esc> nos regresará al Menú de nuevo.

Bibliografía Preliminar.

[1] B. Eckel, Piensa en Java, Madrid: Prentice Hall, 2003.

[2] Oracle, «The Java™ Tutorials - Creating New Class Instances,» [En línea]. Available: https://docs.oracle.com/javase/tutorial/reflect/member/ctorInstance.html. [Último acceso: Diciembre 2017].

[3] D. Singh, Java Design Patterns, Kindle Edition, 2016.

[4] S. McConnell, Code Complete, Redmon, WA: Microsoft Press, 2004.

No. 11. Juego animado parte II.

OBJETIVO

Mostrar la interacción con el usuario para controlar una animación. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, permitiendo que el usuario seleccione de un menú con gráficos de fondo.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 3: Programación concurrente (MultiHilos). 3.1 Concepto de hilo. 3.2 Comparación de un programa de flujo único contra uno de flujo múltiple. 3.3 Creación y control de hilos. 3.4 Sincronización de hilos computación.	**No. 8. Dibujos animados.** Objetivo. Crear una un hilo para mostrar la concurrencia. **No. 9. Simulación de saldos en un banco hipotético.** Objetivo. Crear una un hilo para mostrar la concurrencia y sincronización. **No. 10. Juego animado parte I.** Objetivo. Crear varios hilos para mostrar interacción. **No. 11. Juego animado parte II.** Objetivo. Crear una aplicación con menús de librería y gráficos. **No. 12. Juego animado parte III.** Objetivo. Crear una aplicación con menús de librería y gráficos y tener características de persistencia.

INTRODUCCIÓN

Antes de la versión 8 de Java las librerías asociadas con fechas y horas no eran muy intuitivas, su uso un tanto engañoso y su formateo no funcionaba con varios hilos. A partir de Java 8 se presenta una nueva API de fechas y horas para abordar estos problemas. [1]
La diferencia entre una imagen regular y un Sprite es que un Sprite encapsulará los datos de la imagen y los métodos necesarios para manipularla. [2]
Un tipo enumerado es un tipo cuyos valores legales consisten en un conjunto fijo de constantes. Antes de la aparición de los tipos enumerados se usaban constantes numéricas y se tenía el inconveniente de poderse mezclar números de dos grupos diferentes. [3]
Las animaciones para este juego la mayor parte son tomadas de MillionthVector. [4]

Material y equipo necesario.
El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:
Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para buscar información sobre Tipos enumerados, El método drawImage de Graphics2D, ImageIO.
4. Investigar uso Tipos enumerados.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de las funciones de Graphics2D e ImageIO.
2. Seleccionar a varios participantes para exponer propiedades de Tipos enumerados.
3. Seleccionar a voluntarios para explicar el uso de Tipos enumerados.
4. Llenar el recurso de Wiki con las definiciones de Tipos enumerados.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo el uso de varios paquetes y una clase que incluya al uso de los Tipos Enumerados.
3. Participar en Moodle con el glosario de términos incorporando Tipos Enumerados.

Procedimiento:

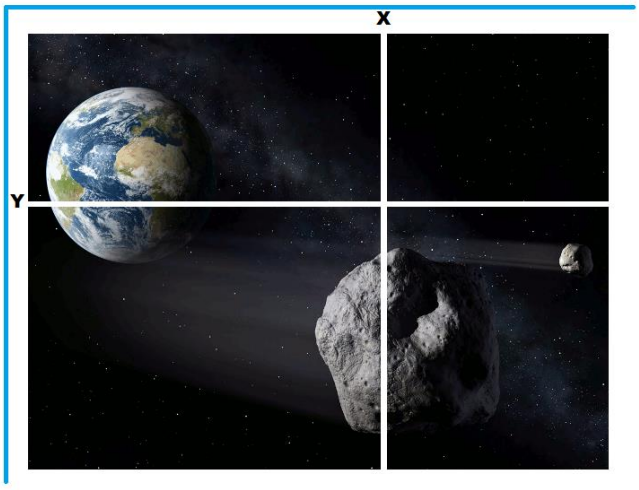
1. El tipo de problema a resolver se propone lo siguiente
   "Crear un juego similar a "Space Invaders". Crear varios alienígenas y un cañón láser para eliminarlos hasta donde sea posible".

   El algoritmo de solución aquí propuesto es: "Crear tres etapas para elaborar el proyecto. En esta segunda parte, definir la imagen de fondo tanto para el menú como para el primer nivel. Crear los Sprites para los enemigos".

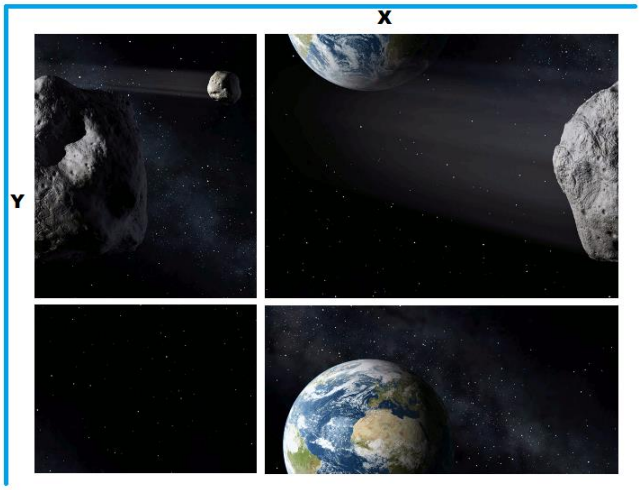
2. Partir del Proyecto "P10JuegoI" hacer una copia y la llamaremos "P11JuegoII". Se recomienda cerrar toda clase abierta en el editor para no correr el riesgo de editar un proyecto anterior. Seleccionar el nuevo proyecto y abrir en el paquete "principal" la clase "MiVentana" para renombrar la etiqueta principal a "Juego Parte II".

3. Crear una clase llamada "`ImagenDeFondo`" en el paquete entidades. Una imagen la podemos "desplazar" dibujándola en cuatro partes. A partir de un punto (x,y) vamos mostrando las cuatro partes dando la impresión de desplazar el fondo.

   a. Una imagen en cuatro porciones.



b. Se realizarían el dibujo de estas cuatro porciones en función del desplazamiento gradual de (x,y) con incrementos en cada variable.



c. La clase "`ImagenDeFondo`" contendrá una imagen leída de un archivo gráfico en su constructor. Un método que nos dé oportunidad de posicionar (x,y). Establecer la velocidad de cambio y poderlo actualizar. Y finalmente dibujar las cuatro porciones "Movidas".

```
package entidades;

import java.awt.*;
import java.awt.image.*;
import java.io.IOException;
import javax.imageio.ImageIO;
import principal.MiVentana;

public class ImagenDeFondo {

    private BufferedImage imagen;

    private double x;
    private double y;
    private double incx;
    private double incy;

    private double moveScale;

    public ImagenDeFondo(String s, double ms) {
        try {
            imagen = ImageIO.read(getClass().getResourceAsStream(s));
            moveScale = ms;
        } catch (IOException e) {
            System.out.println("Error al leer:" + s);
        }
    }

    public void setPosicion(double x, double y) {
        this.x = (x * moveScale) % MiVentana.ANCHO;
        this.y = (y * moveScale) % MiVentana.ALTO;
    }

    public void setVelocidad(double dx, double dy) {
        this.incx = dx;
        this.incy = dy;
    }

    public void Actualiza() {
```

```
            x = (x + incx) % MiVentana.ANCHO;
            y = (y + incy) % MiVentana.ALTO;
        }

        public void Dibuja(Graphics2D g) {
            int ix, iy, sumax, sumay;

            sumax = ix = (int) x;
            sumay = iy = (int) y;

            sumax += ix < 0 ? +MiVentana.ANCHO : -MiVentana.ANCHO;
            sumay += iy < 0 ? +MiVentana.ALTO : -MiVentana.ALTO;

            g.drawImage(imagen, ix, iy, null);

            g.drawImage(imagen, sumax, iy, null);

            g.drawImage(imagen, ix, sumay, null);

            g.drawImage(imagen, sumax, sumay, null);
        }
    }
```

4. Separar las funciones de la clase “Animacion”. La clase Animacion hace varias cosas: 1- Lee de un archivo gráfico una serie de pequeñas fotos. 2- Hace un arreglo (principal) de arreglos de fotos (animacion). 3-Selecciona cual animación se va a mostrar. 4-Toma el tiempo para mostrar la siguiente imagen con una versión anterior. 5-Revisa si una tecla en particular se ha presionado. 6-Y la lógica para permanecer en reposo y avanzar.

    a. Crear una clase que solo realice las funciones 3 y 4 con la API 8. La clase se llamará “Animar” en el paquete “entidades”. Y que contenga los siguientes atributos. Y un constructor que señale que no se ha mostrado.

```
public class Animar {

    private BufferedImage[] cuadros;//Fotos de animación
    private int cuadroaMostrar;

    private Instant tiempoArranque;
    private Duration duracion;

    private int yaMostrado;//Si ya mostro las fotos
    private boolean nuevaMuestra;

    public Animar() {
        yaMostrado = 0;
    }
}
```

    b. Importamos las clases requeridas.

```
import java.awt.image.BufferedImage;
import java.time.Duration;
import java.time.Instant;
```

    c. Un método que acepte una secuencia y la duración entre fotos.

```
    public void setFrames(BufferedImage[] cuadros, Duration duracion) {
        this.cuadros = cuadros;
        cuadroaMostrar = 0;
        tiempoArranque = Instant.now();
        yaMostrado = 0;
        this.duracion = duracion;
    }
```

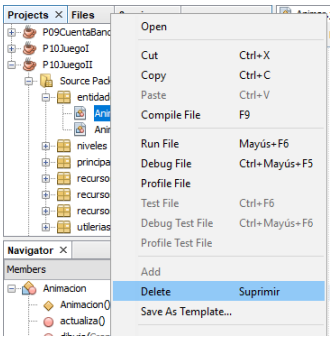
    d. Otro método para tomar el tiempo entre cuadros y si ya se debe mostrar el siguiente cuadro por haber pasado el tiempo necesario.

```
    public void Actualiza() {

        if (duracion.isNegative()) {
            return;
        }

        Instant ahora = Instant.now();
        Duration transcurrido = Duration.between(tiempoArranque,
ahora);

        if (transcurrido.compareTo(duracion) > 0) {
            cuadroaMostrar++;
            tiempoArranque = ahora;
        }
        nuevaMuestra = false;
        if (cuadroaMostrar == cuadros.length) {
            cuadroaMostrar = 0;
            yaMostrado++;
            nuevaMuestra = true;
        }
    }
```

e. Otros métodos auxiliares que nos regresen un valor como si ya paso un ciclo con "`isNuevaMuestra`" que no regrese la foto a mostrar con "`getImagen`" y si solo si ya pasado un solo ciclo en "`yaMostradoRst`" y finalmente saber cuántas veces se ha mostrado en "`getNumMostrado`".

```
    public boolean isNuevaMuestra() {
        return nuevaMuestra;
    }

    public BufferedImage getImagen() {
        return cuadros[cuadroaMostrar];
    }

    public boolean yaMostradoRst() {
        if (yaMostrado > 0) {
            yaMostrado = 0;
            return true;
        }
        return false;
    }

    public int getNumMostrado() {
        return yaMostrado;
    }
```

f. Como la clase "`Animacion`" sirve solo para un tipo de animación y hemos tomado la función común en la clase "`Animar`" para las demás animaciones podemos prescindir de ella borrándola.



g. Al borrarla nos genera un error en la clase NivIJuego.

5. Crear una instancia de la imagen de fondo en la clase `NivIJuego`.

a. Iniciaremos editando `NivIJuego` para corregir el error ocasionado por la eliminación de la clase "`Animacion`". En esa clase borramos las líneas donde encontremos `Animacion` y `anima`.

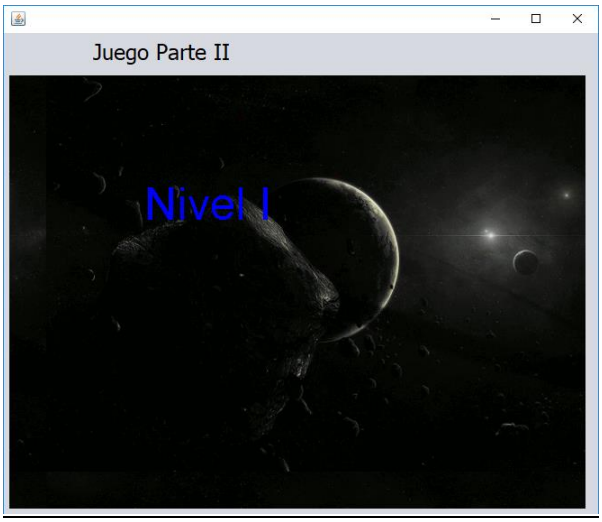
```
public class NivIJuego implements Niveles {

    @Override
    public void Actualiza() {
        if (Teclas.ESC.estaPresionada()) {
            AdminNiveles.getInstance().setEstado(N0MENU);
        }
    }

    @Override
    public void Dibuja(Graphics2D g) {
        g.drawString("Nivel I", 150, 160);
    }
}
```

b. Agregaremos un atributo de tipo "`ImagenDeFondo`" donde lo inicializaremos en un constructor haciendo referencia a la imagen espacio.gif. Lo actualizamos y dibujamos. Tomando en cuenta que debe ser la primer instrucción de dibuja ya que de lo contrario los dibujos previos dibujados serian borrados.

```
package niveles;

import entidades.ImagenDeFondo;
import java.awt.Graphics2D;
import static niveles.Estados.N0MENU;
import utilerias.Teclas;

public class NivIJuego implements Niveles {
    private final ImagenDeFondo bg;

    public NivIJuego() {
        bg = new ImagenDeFondo("/recursos/fondos/espacio.gif", 0.1);
        bg.setVelocidad(0.3, -0.3);
    }

    @Override
    public void Actualiza() {
        bg.Actualiza();
        if (Teclas.ESC.estaPresionada()) {
            AdminNiveles.getInstance().setEstado(N0MENU);
        }
    }

    @Override
    public void Dibuja(Graphics2D g) {
        bg.Dibuja(g);
        g.drawString("Nivel I", 150, 160);
    }
}
```

6. Actualizamos los letreros de "Juego Parte I" a "Juego Parte II"
   a. Abrimos en "Design" del paquete "principal" la clase MiVentana y la etiqueta la modificamos.
   b. Abrimos del paquete "niveles" la clase "`Niv0Menu`" y en el método "`Dibuja`" modificamos "`g.drawString("Juego Parte I", 150, 160)`" por "`g.drawString("Juego Parte II", 150, 160)`"

7. Corremos el programa ya debe aparecer “Juego Parte II”. Y seleccionamos “Nivel I” dando <Enter>. Asi vemos “desplazándose” el fondo.



8. Crear una instancia de la imagen de fondo en la clase `Niv0Menu`. Editamos “`Niv0Menu`” y vamos a efectuar los mismos pasos que el paso anterior inciso b.
    a. Agregar un atributo “`private final ImagenDeFondo bg`” a la clase.
    b. Luego inicializarlo en el constructor solo que ahora su valor será:

```
bg = new ImagenDeFondo("/recursos/fondos/asteroides.gif", 1);
bg.setVelocidad(-0.1, 0);
```

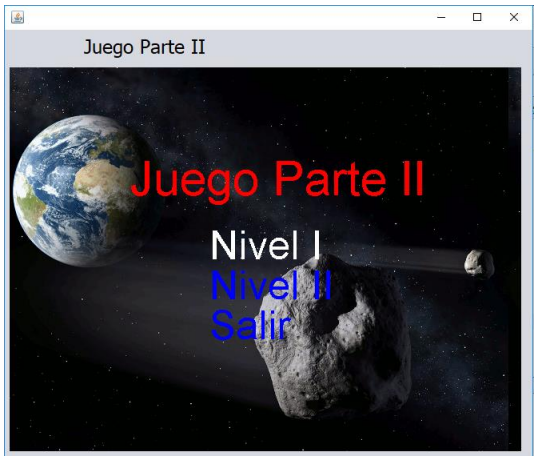
    c. Agregamos la línea a “Actualiza”.

```
bg.Actualiza();
```

    d. Y la primera línea a “Dibuja”.

```
bg.Dibuja(g);
```

9. Corremos el programa ya debe aparecer “Juego Parte II” con fondo desplazándose.

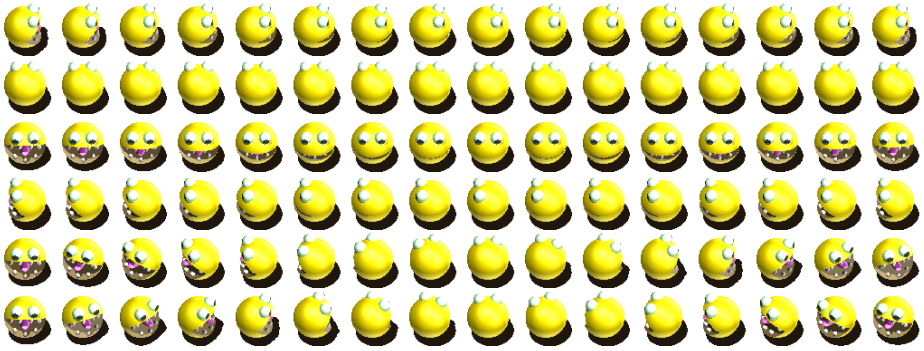


10. Los personajes que incorporan este Juego van a tener característica globales como posición, velocidad, si vive o no, como se va a actualizar y dibujar. Para efecto de "colisiones" incorporar un área definida por un rectángulo y verificar si continúa vivo o muere. Para este propósito se crea la clase abstracta "`Entidad`".

```
package entidades;

import java.awt.Graphics2D;
import java.awt.Rectangle;

/**
 *
 * @author Servidor
 */
public abstract class Entidad {

    private boolean estavivo;
    private double x, y;
    private double vel, velx;

    // animación
    protected Animar animar;

    public Entidad() {
        animar = new Animar();
    }

    public boolean estaVivo() {
        return estavivo;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }

    final public void setVivo(boolean alive) {
        this.estavivo = alive;
    }

    final public void setX(double x) {
        this.x = x;
    }

    public void incX(double i) {
        this.x += i;
    }
```

```
    final public void setY(double y) {
        this.y = y;
    }

    public void incY(double i) {
        this.y += i;
    }

    final public void setVel(double vel) {
        this.vel = vel;
    }

    final public void setVelx(double velx) {
        this.velx = velx;
    }


    public double getVelx() {
        return velx;
    }

    public void incVel(double i) {
        this.vel += i;
    }

    public double getVel() {
        return vel;
    }

    public abstract void Dibuja(Graphics2D g);

    public abstract void Actualiza();

    public abstract Rectangle getRectangulo();

    public boolean Murieron(Entidad e) {
        if (estaVivo() && e.estaVivo()) {
            Rectangle r1 = getRectangulo();
            Rectangle r2 = e.getRectangulo();
            if (r1.intersects(r2)) {
                setVivo(false);
                e.setVivo(false);
                return true;
            }
        }
        return false;
    }
}
```

11. Creación de Sprite enemigo. Para este enemigo en particular se escogió un personaje similar al Pac-Man (comelón). Este personaje van a ser creados en la parte superior y avanzaran de izquierda a derecha y viceversa. Al llegar al límite derecho o izquierdo bajaran una porción del camino hasta alcanzar el suelo donde se comerá a una nave que está disparando siempre y cuando no se encuentren con una bala ya que explotarán.
    a. Creación de este personaje se baja en la serie de fotogramas siguientes.



b. Se crea una clase abstracta "`ComelonBasico`" derivada de "`Entidad` y tenga un "`enum Act`" para guardar los pasos a dar y lograr avanzar a la derecha tomando las primeras 16 fotos, luego, las siguientes 16 de avance arriba no las usamos pero las dejamos, después, las 16 avance hacia abajo, las 16 siguientes avance a la izquierda. Y a partir de aquí cada cuatro fotos permiten hacer un giro hasta completar 16 con una rotación en sentido de las manecillas del reloj y otro giro con las siguientes 16 fotos en contra de las manecillas del reloj. Se cuenta con un constructor estático para la lectura de las fotos.

```
package entidades;

import java.awt.image.BufferedImage;
import java.io.IOException;
import javax.imageio.ImageIO;

public abstract class ComelonBasico extends Entidad {

    static final int ALTOANCHO = 64;  //Tamaño de la imagen
    static final int NIMGXRNGL = 16;  //Imagenes por renglon

    public enum Act {
        DER(16, 50), ARR(16, 50), ABA(16, 50), IZQ(16, 50),
        SW(4, 40), WN(4, 40), NE(4, 40), ES(4, 40),
        SE(4, 40), EN(4, 40), NW(4, 40), WS(4, 40);

        private final int retrazo;
        private final BufferedImage[] bi;

        Act(int n, int retrazo) {
            bi = new BufferedImage[n];
            this.retrazo = retrazo;
        }

        public int getRetrazo() {
            return retrazo;
        }

        public BufferedImage[] getBi() {
            return bi;
        }
    }

    static {
        try {
            BufferedImage spritesheet = ImageIO.read(
                    ComelonBasico.class.getResourceAsStream(
                            "/recursos/enemigos/comelon.png"));
            int i = 0;
            int k = 0;
            for (Act act : Act.values()) {
```

```
                    BufferedImage[] bi = act.getBi();
                    for (int j = 0; j < bi.length; j++) {
                        bi[j] = spritesheet.getSubimage(
                                k * ALTOANCHO,
                                (i / NIMGXRNGL) * ALTOANCHO,
                                ALTOANCHO, ALTOANCHO
                        );
                        k++;
                    }
                    i += bi.length;
                    k %= NIMGXRNGL;
                }
            } catch (IOException e) {
                System.out.println("Error al leer dibujos");
            }
        }
    }
```

c. Ahora creamos una clase "`Comelon`" que deriva de "`ComelonBasico`". Ésta clase va a contar con una secuencia de pasos basados en las actividades definidas en "`enum Act`" cada secuencia de pasos va a estar definida en una lista de tipo "`ArrayList`" establecidas en el inicializador "`static`". Un objeto de ésta clase va a estar un una posición (x,y) y por default va a tener una actividad de "`enum Act`" por ello se define en el constructor. Contará con dos métodos uno para establecer la secuencia de actividades y otro para actualizarla. Y como la clase no es abstracta se requiere definir los métodos abstractos de la clase "`Entidad`".

```
package entidades;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.time.Duration;
import java.util.ArrayList;
import java.util.List;
import java.util.Random;

public class Comelon extends ComelonBasico {

    static final List<Act> ROTDI = new ArrayList<>();
    static final List<Act> ROTID = new ArrayList<>();
    static final List<Act> ROTIA = new ArrayList<>();
    static final List<Act> ROTDA = new ArrayList<>();
    static final List<Act> ROTAD = new ArrayList<>();
    static final List<Act> ROTAI = new ArrayList<>();

    static {
        ROTAD.add(Act.SE);
        ROTAD.add(Act.DER);

        ROTAI.add(Act.SW);
        ROTAI.add(Act.IZQ);

        ROTIA.add(Act.WS);
        ROTIA.add(Act.ABA);

        ROTDA.add(Act.ES);
        ROTDA.add(Act.ABA);

        ROTDI.add(Act.ES);
        ROTDI.add(Act.SW);
        ROTDI.add(Act.IZQ);

        ROTID.add(Act.WS);
        ROTID.add(Act.SE);
        ROTID.add(Act.DER);
    }
```

```
    private Act accion;
    List<Act> secuencia;
    int numactiv;
    boolean der;
    static final Random RND = new Random();

    public Comelon(int x, int y, Act accion) {
        setX(x);
        setY(y);
        setActividad(accion);
        setVel(0.7);
    }

    private void setSecuencia(List<Act> secuencia) {
        this.secuencia = secuencia;
        numactiv = 0;
    }

    private void actualizaSecuencia() {
        if (secuencia != null && animar.yaMostradoRst()) {
            if (numactiv < secuencia.size()) {
                setActividad(secuencia.get(numactiv++));
            } else {
                secuencia = null;
                numactiv = 0;
            }
        }
    }

    private void setActividad(Act accion) {
        this.accion = accion;
        animar.setFrames(accion.getBi(),
                Duration.ofMillis(accion.getRetrazo() /*+ rand*/));
    }

    @Override
    public void Dibuja(Graphics2D g) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void Actualiza() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public Rectangle getRectangulo() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

d. Ahora vamos a definir varios métodos. Para ubicar el centro de nuestra animación. Definir el área de colisión basada en un rectángulo. Si el objeto está “vivo” se podrá actualizar y dibujar. Cuando se actualice de debe actualizar la animación y la secuencia y dependiendo de cuál acción se lleve a cabo se actuará en consecuencia.

```
    public int getXCentro() {
        return (int) getX() + ALTOANCHO / 2;
    }

    public int getYCentro() {
        return (int) getY() + ALTOANCHO / 2;
    }
```

```
@Override
public Rectangle getRectangulo() {
    return new Rectangle((int) getX() + 10, (int) getY() + 10,
            ALTOANCHO - 20, ALTOANCHO - 20);
}

@Override
public void Actualiza() {
    if (estaVivo()) {
        animar.Actualiza();
        actualizaSecuencia();

        switch (accion) {
            case ABA:
                incY(getVel());
                if (animar.getNumMostrado() > 2) {
                    incVel(0.2);
                    if (der) {
                        setSecuencia(ROTAI);
                    } else {
                        setSecuencia(ROTAD);
                    }
                }
                break;
            case DER:
                if (animar.isNuevaMuestra() && RND.nextInt(10) == 0) {
                    //Bombas.Disparo(this);
                }
                incX(getVel());
                if (getXCentro() > MiVentana.ANCHO - ALTOANCHO / 2) {
                    setSecuencia(ROTDA);
                    der = true;
                }
                break;
            case IZQ:
                incX(-getVel());
                if (getXCentro() < ALTOANCHO / 2) {
                    setSecuencia(ROTIA);
                    der = false;
                }
        }
    }
}

@Override
public void Dibuja(Graphics2D g) {
    if (estaVivo()) {
        g.drawImage(
                animar.getImagen(),
                (int) getX(),
                (int) getY(),
                null
        );
    }
}
```

e. Ahora agregamos una clase que englobe hasta 8 comelones en ella se van a posicionar en la parte superior de la pantalla nueve comelones y tendrá el los métodos Dibuja y Actualiza para actualizar a los 8 comelones.

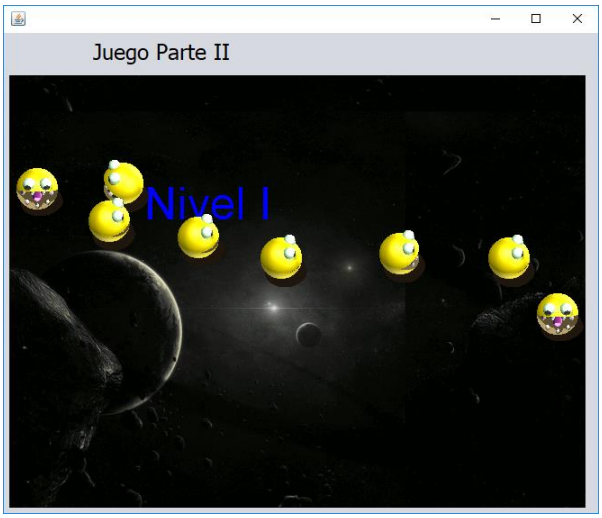
```
package entidades;

import static entidades.ComelonBasico.Act.DER;
import java.awt.Graphics2D;

public class Comelones {

    public static Comelon comelon[];
    final int NC = 8;
```

```
    public Comelones() {
        comelon = new Comelon[NC];

        for (int i = 0; i < NC; i++) {
            (comelon[i] = new Comelon(80 * i, 20, DER)).setVivo(true);
        }
    }

    public void Dibuja(Graphics2D g) {
        for (int i = 0; i < NC; i++) {
            comelon[i].Dibuja(g);
        }
    }

    public void Actualiza() {
        for (int i = 0; i < NC; i++) {
            comelon[i].Actualiza();
        }
    }
}
```

12. En la clase "`NivIJuego`" creamos un atributo de tipo Comelones. Y en el constructor lo instanciamos y se actualiza y dibuja. Hay que considerar en el método "`Dibuja`", de comelones sólo debe ser invocado posterior a la línea donde se dibuja el fondo "`bg.Dibuja(g);`". Y ésta clase nos queda:

```
package niveles;

import entidades.Comelones;
import entidades.ImagenDeFondo;
import java.awt.Graphics2D;
import static niveles.Estados.N0MENU;
import utilerias.Teclas;

public class NivIJuego implements Niveles {

    private final ImagenDeFondo bg;
    private final Comelones comelones;

    public NivIJuego() {
        bg = new ImagenDeFondo("/recursos/fondos/espacio.gif", 0.1);
        bg.setVelocidad(0.3, -0.3);

        comelones = new Comelones();
    }

    @Override
    public void Actualiza() {
        bg.Actualiza();
        comelones.Actualiza();

        if (Teclas.ESC.estaPresionada()) {
            AdminNiveles.getInstance().setEstado(N0MENU);
        }
    }

    @Override
    public void Dibuja(Graphics2D g) {
        bg.Dibuja(g);
        g.drawString("Nivel I", 150, 160);
        comelones.Dibuja(g);
    }
}
```

13. Comprobamos el programa con el siguiente resultado en la opción de “Nivel I”.



Bibliografía Preliminar.

[1] R. G. Urma, M. Fusco y A. Mycroft, Java 8 in Action, Shelter Island, NY: Manning Publications Co., 2015.

[2] J. S. Harbour, Beginning Java SE 6 Grame Programming., Boston: Course Technology, 2012.

[3] J. Bloch, Effective Java™, Santa Clara: Addison-Wesley, 2008.

[4] MillionthVector, «Free Sprites,» [En línea]. Available: http://millionthvector.blogspot.mx/p/free-sprites.html. [Último acceso: Dic 2017].

No. 12. Juego animado parte III.

OBJETIVO

Mostrar la interacción con el usuario para controlar una animación. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, permitiendo que el usuario seleccione de un menú con gráficos de fondo y controle una animación y tener ligera persistencia.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 3: Programación concurrente (MultiHilos). 3.1 Concepto de hilo. 3.2 Comparación de un programa de flujo único contra uno de flujo múltiple. 3.3 Creación y control de hilos. 3.4 Sincronización de hilos computación.	**No. 8. Dibujos animados.** Objetivo. Crear una un hilo para mostrar la concurrencia. **No. 9. Simulación de saldos en un banco hipotético.** Objetivo. Crear una un hilo para mostrar la concurrencia y sincronización. **No. 10. Juego animado parte I.** Objetivo. Crear varios hilos para mostrar interacción. **No. 11. Juego animado parte II.** Objetivo. Crear una aplicación con menús de librería y gráficos. **No. 12. Juego animado parte III.** Objetivo. Crear una aplicación con menús de librería y gráficos y tener características de persistencia.

INTRODUCCIÓN

En un programa se crean objetos y existen mientras se usen pero dejan de existir cuando el programa termina. La forma de persistir un objeto cuando el programa termina es implementando la interface Serializable. [1]
Implementando Serializable es la forma más simple de persistencia. Selectivamente se pueden excluir miembros de una clase empleando la palabra reservada transient. [2]
Antes de java 7 para abrir y cerrar recursos era muy verboso y propenso a errores. El uso de la instrucción try-with-resources automáticamente cierra los recursos cuando se sale del bloque try. [3]

Material y equipo necesario.
El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:
Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para buscar información sobre Serialización, el bloque try-with-resources.
4. Investigar tipos de Persistencia y el bloque try-with-resources.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de las funciones writeObject, readObject, el bloque try-with-resources, la interface Serializable.
2. Seleccionar a varios participantes para exponer Persistencia.
3. Seleccionar a voluntarios para explicar el uso de Persistencia usando serialización y el bloque try-with-resources.
4. Llenar el recurso de Wiki con las definiciones de Persistencia y serialización.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Hacer un programa similar al punto número 1. Crear un segundo Nivel de dificultad.
3. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo el uso de Serialización y una clase que incluya Persistencia
4. Participar en Moodle con el glosario de términos incorporando Persistencia, Selializable, el bloque try-with-resources.

Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   "Crear un juego similar a "Space Invaders". Crear varios alienígenas y un cañón láser para eliminarlos hasta donde sea posible".

   El algoritmo de solución aquí propuesto es: "Crear tres etapas para elaborar el proyecto. En esta tercera parte, persistir el nombre del jugador. Crear los Sprites el jugador, las balas y las bombas.".

2. Partir del Proyecto "P11JuegoII" hacer una copia y la llamaremos "P12JuegoIII". Se recomienda cerrar toda clase abierta en el editor para no correr el riesgo de editar un proyecto anterior. Seleccionar el nuevo proyecto y abrir en el paquete "principal" la clase "MiVentana" para renombrar la etiqueta principal a "Juego Parte III".

3. Crear una clase llamada "`NombreJugador`" en el paquete "`entidades`" que tenga dos atributos uno llamado nombre y otro archivo la clase debe implementar "`Serializable`" la clase va a poder salvar y leer el nombre en un archivo empleando el bloque try-with-resources.
   a. Crear la clase con un constructor por default y un constructor que acepte un nombre sólo si no está vacío. El archivo será "`c:/tmp/JuegoNombre.ser`".

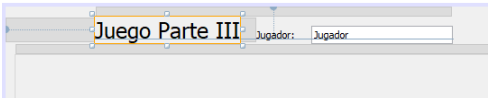
```
package entidades;

import java.io.Serializable;

public class NombreJugador implements Serializable {

    private String nombre = "Jugador";
    transient final String archivo = "c:/tmp/JuegoNombre.ser";

    public NombreJugador() {
    }

    public NombreJugador(String nombre) {
        if (nombre != null && !nombre.isEmpty()) {
            this.nombre = nombre;
        }
    }
```

```
}
```

b. Crear un método para salvar y otro para leer empleando el bloque try-with-resources y un getter para el nombre (No olvide incorporar los imports).

```
    public boolean Salvar() {
        try (FileOutputStream fileOut = new FileOutputStream(archivo);
                ObjectOutputStream out = new
ObjectOutputStream(fileOut)) {
            out.writeObject(this);
        } catch (IOException ex) {
            return false;
        }
        return true;
    }

    public boolean Leer() {
        NombreJugador nj;
        try (FileInputStream fileIn = new FileInputStream(archivo);
                ObjectInputStream in = new ObjectInputStream(fileIn)) {
            nj = (NombreJugador) in.readObject();
            this.nombre = nj.nombre;
        } catch (IOException | ClassNotFoundException i) {
            return false;
        }
        return true;
    }

    public String getNombre() {
        return nombre;
    }
```

4. Una vez creada la clase vamos a incorporar en diseño en la clase principal a la derecha de la etiqueta “Juego Parte II” una etiqueta que diga “Jugador:” y un TextField llamado “tfJugador” con el texto “Jugador”. Salvar el texto al darle <Enter> y que se recupere al iniciar el programa.
   a. Los componentes puede tener una configuración siguiente.



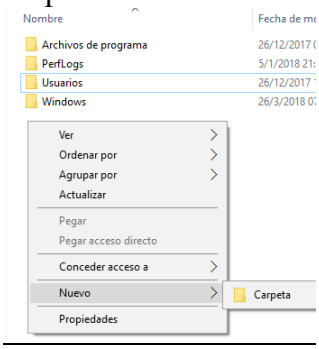
   b. Programar la caja de texto “tfJugador” dando doble clic. El evento “`tfJugadorActionPerformed`” se generara al escribir sobre la caja de texto y finalizar con la tecla <Enter>. Incorporar las librerías “`NombreJugador`” y “`JOptionPane`”.

```
private void tfJugadorActionPerformed(java.awt.event.ActionEvent evt) {
    NombreJugador nj = new NombreJugador(tfJugador.getText());
    if (nj.Salvar()) {
        JOptionPane.showMessageDialog(this, "Nombre guardado: " + nj.getNombre());
    } else {
        JOptionPane.showMessageDialog(this, "No se guardó: " + nj.getNombre());
    }
}
```

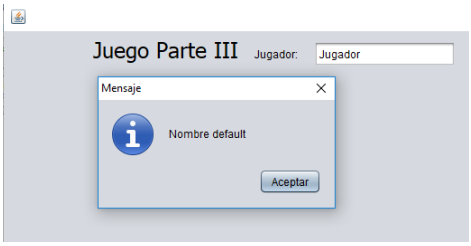
c. Agregar el siguiente fragmento de código en el evento "`formWindowActivated`" que nos servirá cuando se inicie el programa.

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    if (!generado) {
        generado = true;

        NombreJugador nj = new NombreJugador();
        if (nj.Leer()) {
            JOptionPane.showMessageDialog(this, "Recuperado : " + nj.getNombre());
        } else {
            JOptionPane.showMessageDialog(this, "Nombre default");
        }
        tfJugador.setText(nj.getNombre());

        bimagen = new BufferedImage(
                ANCHO, ALTO, BufferedImage.TYPE_INT_RGB);
        g = (Graphics2D) bimagen.getGraphics();

        executor = Executors.newScheduledThreadPool(1);
        executor.scheduleWithFixedDelay(
                this, 0, tiempoEspera, TimeUnit.MILLISECONDS);
    }
}
```

5. Comprobar que está funcionando la persistencia del nombre.
   a. Verificar que exista el subdirectorio "c:tmp". Si no existe crearlo con el explorador y escribimos "tmp"



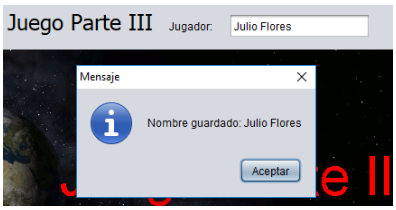
   b. Correr el programa y la primera vez obtenemos algo como:



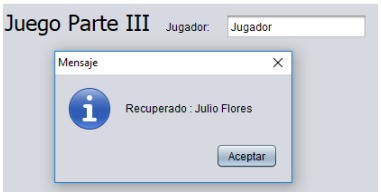
   c. Luego modificamos "Jugador" y escribimos nuestro nombre y damos <Enter>


Juego Parte III
Jugador:
Julio Flores

d. Nos debe mostrar un mensaje:



e. Terminamos el juego y volvemos a arrancarlo. Y debe de cargar el nombre que escribimos la última vez.



f. Para evitar que continuamente nos esté mostrando que ha recuperado el nombre podemos prescindir del "`if`" el evento "`formWindowActivated`" y nos queda el código.

```
private void formWindowActivated(java.awt.event.WindowEvent evt) {
    if (!generado) {
        generado = true;

        NombreJugador nj = new NombreJugador();
        nj.Leer();
        tfJugador.setText(nj.getNombre());
        this.requestFocus();

        bimagen = new BufferedImage(
                ANCHO, ALTO, BufferedImage.TYPE_INT_RGB);
        g = (Graphics2D) bimagen.getGraphics();

        executor = Executors.newScheduledThreadPool(1);
        executor.scheduleWithFixedDelay(
                this, 0, tiempoEspera, TimeUnit.MILLISECONDS);
    }
}
```

a. Lo mismo hacemos en el método "`tfJugadorActionPerformed`".

```
private void tfJugadorActionPerformed(java.awt.event.ActionEvent evt) {
    NombreJugador nj = new NombreJugador(tfJugador.getText());
    nj.Salvar();
    this.requestFocus();
}
```

6. Crear la clase Nave la cual va a derivar de la clase "`Entidad`" tomará de una serie de fotos sus características visibles. Si está viva se actualizará dentro de los límites de la franja visible inferior. Tendrá un área rectangular con propósitos de colisión y de la punta de la nave tendremos un punto a partir de ahí hacer disparos. La nave tendrá un bamboleo hacia la izquierda y derecha.

a. La nave leerá el archivo `"/recursos/jugador/nave.png"` siguiente. Cada foto tiene un tamaño de 48 de ancho por 76 pixeles de alto y como se observa son 12.

b. Hacer una clase con un constructor que lea esta imagen y la guarde en un objeto animar. Como animar se encuentra en Entidad se debe derivar de ésta clase.

```
package entidades;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.time.Duration;
import javax.imageio.ImageIO;
import principal.MiVentana;

public class Nave extends Entidad {

    final static int ANCHO = 48;
    final static int ALTO = 76;
    final static int N = 12;

    public Nave() {
        BufferedImage[] bi = new BufferedImage[N];
        try {
            BufferedImage spritesheet = ImageIO.read(
                    getClass().getResourceAsStream(
                            "/recursos/jugador/nave.png"
                    )
            );
            int k = 0;
            for (int i = 0; i < 2; i++) {
                for (int j = 0; j < 6; j++) {
                    bi[k] = spritesheet.getSubimage(
                            j * ANCHO, i * ALTO, ANCHO, ALTO);
                    k++;
                }
            }
        } catch (IOException e) {
            System.out.println("Error al leer dibujos");
        }
        animar.setFrames(bi, Duration.ofMillis(300));
        setX(MiVentana.ANCHO / 2);
        setY(MiVentana.ALTO - 100);
        setVel(0);
        setVivo(true);
    }

    @Override
    public void Dibuja(Graphics2D g) {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    @Override
    public void Actualiza() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
```

```
    @Override
    public Rectangle getRectangulo() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
}
```

c. Si está la Nave “viva” se actualiza animar y se verifica que esté dentro de cierto rango en la parte inferior. Para dibujar la Nave solo se dibuja la siguiente imagen guardada en animar. Y se define un área rectangular que engloba a la nave. Crearemos un método para obtener un punto de la punta de la nave. La nave puede irse a la izquierda y a la derecha.

```
@Override
public void Dibuja(Graphics2D g) {
    if (estaVivo()) {
        g.drawImage(
                animar.getImagen(),
                (int) getX(),
                (int) getY(),
                null
        );
    }
}

@Override
public void Actualiza() {
    int x;
    if (estaVivo()) {
        animar.Actualiza();

        x = getTop().x;
        if (x < 100) {
            if (getVel() < 0) {
                setVel(0);
            }
        } else if (x > MiVentana.ANCHO - 100) {
            if (getVel() > 0) {
                setVel(0);
            }
        }
        setX(getX() + getVel());
    }
}

@Override
public Rectangle getRectangulo() {
    return new Rectangle((int) getX() + 10, (int) getY() + 10,
            ANCHO - 20, ALTO - 20);
}

public Point getTop() {
    Point p;
    Rectangle r = getRectangulo();
    int x;
    x = r.x + r.width / 2;
    p = new Point(x, r.y);
    return p;
}

public void setIzq() {
    incVel(-0.1);
}

public void setDer() {
    incVel(0.1);
}
```

7. Añadir en la clase "`NivIJuego`" un objeto de tipo "`Nave`".
   a. Como referencia podemos poner "abajo" de "`Comelones`" en cada sección (declaración, constructor, actualiza, dibuja) incorporaremos una función similar.

```
private final Nave nave; //Línea en declaración

nave = new Nave(); //Incorporado al constructor

nave.Actualiza();  //Va en el método actualiza

nave.Dibuja(g);  //Va en el método dibuja
```

   b. Modificar el "`if`" del método actualiza para maniobrar la nave de izquierda a derecha.

```
if (Teclas.ESC.estaPresionada()) {
    AdminNiveles.getInstance().setEstado(N0MENU);
} else if (Teclas.IZQ.estaPresionada()) {
    nave.setIzq();
} else if (Teclas.DER.estaPresionada()) {
    nave.setDer();
}
```

8. Verificamos lo que llevamos y al correr nuestro programa vemos lo siguiente. La Nave puede ir de izquierda a derecha presionando la tecla correspondiente.

9. Ahora implementaremos que nuestra nave pueda disparar. Para simular una bala diseñaremos un pequeño rectángulo de 1 pixel de alto por 1 pixel de ancho. Y que derive de entidad. Como no tiene animación no se considerará anima. La bala arrancará desde la nave y desaparecerá cuando toque un "comelon" o una "bomba" en una "explosión".

```
package entidades;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import principal.MiVentana;
```

```
public class Bala extends Entidad {

    public Bala(int x, int y, double vel) {
        setX(x);
        setY(y);
        setVel(vel);
    }

    @Override
    public void Dibuja(Graphics2D g) {
        if (estaVivo()) {
            g.setColor(Color.YELLOW);
            g.draw(getRectangulo());
        }
    }

    @Override
    public void Actualiza() {
        if (estaVivo()) {
            incY(-getVel());
            incX(getVelx());
            if (getY() < 0 || getX() < 0 || getX() > MiVentana.ANCHO) {
                setVivo(false);
            }
        }
    }

    @Override
    public Rectangle getRectangulo() {
        return new Rectangle(
                (int) getX(), (int) getY(), 1, 1);
    }
}
```

10. La nave va a poder disparar hasta 5 balas y es por ello que crearemos la clase “Balas” en donde la nave escogerá una bala disponible.

```
package entidades;

import java.awt.Graphics2D;
import java.awt.Point;
import principal.MiVentana;

public class Balas {

    public static Bala[] balas;
    static final int NB = 5;

    public Balas() {
        balas = new Bala[NB];

        for (int i = 0; i < NB; i++) {
            balas[i] = new Bala(80 * i, MiVentana.ALTO - 20, 2);
        }
    }

    public void Actualiza() {
        for (int i = 0; i < NB; i++) {
            balas[i].Actualiza();
        }
    }

    public void Dibuja(Graphics2D g) {
        for (int i = 0; i < NB; i++) {
            balas[i].Dibuja(g);
        }
    }
```

```
    public boolean Disparo(Nave nave) {
        Point p;
        boolean q = false;
        if (nave.estaVivo()) {
            p = nave.getTop();
            for (Bala bala : balas) {
                if (!bala.estaVivo()) {
                    bala.setX(p.x);
                    bala.setY(p.y);
                    bala.setVelx(nave.getVel());
                    bala.setVivo(true);
                    q = true;
                    break;
                }
            }
        }
        return q;
    }
}
```

11. Haremos un procedimiento similar al paso 7. Añadir en la clase "`NivIJuego`" un objeto de tipo "`Balas`".
    a. Como referencia podemos poner "abajo" de "`Nave`" en cada sección (declaración, constructor, actualiza, dibuja) incorporaremos una función similar.

```
private final Balas balas; //Línea en declaración

balas = new Balas(); //Incorporado al constructor

balas.Actualiza();  //Va en el método actualiza

balas.Dibuja(g);  //Va en el método dibuja
```

    b. Modificar el "`if`" del método actualiza para que la nave pueda disparar.

```
if (Teclas.ESC.estaPresionada()) {
    AdminNiveles.getInstance().setEstado(N0MENU);
} else if (Teclas.SPC.estaPresionada()) {
    balas.Disparo(nave);
} else if (Teclas.IZQ.estaPresionada()) {
    nave.setIzq();
} else if (Teclas.DER.estaPresionada()) {
    nave.setDer();
}
```

12. Corremos el programa y podemos disparar hasta 5 balas.

13. Ahora debemos hacer explotar a un comelón cuando "choque" con una bala. Es decir, cuando el área definida por el rectángulo de un comelón se intersecta con el rectángulo de la bala.
    a. Definir una clase "`Explosion`" basada en un dibujo siguiente. Donde tenemos cuatro renglones de ocho fotos en total tenemos un máximo de 4x8 fotos de 64x64 pixeles.

    b. La clase debe tener la referencia a un arreglo de cada foto de la imagen anterior definida en el constructor y mientras esté "viva" se va a actualizar y dibujar y por supuesto enmarcará el área del rectángulo. El objeto anima se encargará de mostrar las 32 fotos. Cuando se muestre la "explosión" una sola vez dejará de estar "viva".

```
package entidades;

import static entidades.Explosiones.ALTOANCHO;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.time.Duration;

public class Explosion extends Entidad {

public static final int EALTOANCHO = 64;

    Explosion(BufferedImage[] bi, int x, int y) {
        animar.setFrames(bi, Duration.ofMillis(100));
        setX(x);
        setY(y);
    }

    @Override
    public void Dibuja(Graphics2D g) {
        if (estaVivo()) {
            g.drawImage(
                    animar.getImagen(),
                    (int) getX(),
                    (int) getY(),
                    null
            );
        }
    }

    @Override
    public void Actualiza() {
        if (estaVivo()) {
            animar.Actualiza();
            if (animar.yaMostradoRst()) {
                setVivo(false);
            }
```

```
            }
        }

        @Override
        public Rectangle getRectangulo() {
            return new Rectangle((int) getX() + 10, (int) getY() + 10,
                    EALTOANCHO - 20, EALTOANCHO - 20);
        }
    }
```

c. Ahora vamos a definir un conjunto de explosiones. El número va a estar en función de cuantas balas tenemos y que la nave explote. El constructor define la lectura de un arreglo de "`BufferedImage`" los cuales se construyen 6 explosiones (representan hasta 5 balas y una explosión más por la posibilidad de explosión de la nave). Para "explotar" se toma la siguiente explosión disponible (muerta).

```
package entidades;

import static entidades.Balas.NB;
import static entidades.Explosion.EALTOANCHO;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.IOException;
import javax.imageio.ImageIO;

public class Explosiones {

    final int N = 4;
    final int R = 8;
    final int MAX = N * R;

    public static Explosion[] explosiones;
    static final int NE = NB + 1;

    public Explosiones() {
        BufferedImage[] bi = new BufferedImage[MAX];
        try {
            BufferedImage spritesheet = ImageIO.read(
                    getClass().getResourceAsStream(
                            "/recursos/jugador/explosion.png"
                    )
            );
            int k = 0;
            for (int i = 0; i < N; i++) {
                for (int j = 0; j < R; j++) {
                    bi[k++] = spritesheet.getSubimage(
                            j * EALTOANCHO, i * EALTOANCHO,
                            EALTOANCHO, EALTOANCHO
                    );
                }
            }
        } catch (IOException e) {
            System.out.println("Error al leer dibujos");
        }

        explosiones = new Explosion[NE];
        for (int i = 0; i < NE; i++) {
            explosiones[i] = new Explosion(bi, 80 * i, 20);
        }
    }

    public void Actualiza() {
        for (int i = 0; i < NE; i++) {
            explosiones[i].Actualiza();
        }
    }
```

```
    public void Dibuja(Graphics2D g) {
        for (int i = 0; i < NE; i++) {
            explosiones[i].Dibuja(g);
        }
    }

    public static boolean Explota(Entidad entidad) {
        int x, y;
        boolean q = false;
        x = (int) entidad.getX();
        y = (int) entidad.getY();
        for (Explosion explosion : explosiones) {
            if (!explosion.estaVivo()) {
                explosion.setX(x);
                explosion.setY(y);
                explosion.setVivo(true);
                q = true;
                break;
            }
        }
        return q;
    }
}
```

12. Añadir en la clase "`NivIJuego`" un objeto de tipo "`Explosiones`". Verificar si una bala ha alcanzado a un comelón que en dicho momento lo hacemos explotar.
    a. Como referencia podemos poner "abajo" de "`Balas`" en cada sección (declaración, constructor, actualiza, dibuja) incorporaremos una función similar.

```
private final Explosiones explosiones; //Línea en declaración

explosiones = new Explosiones(); //Incorporado al constructor

explosiones.Actualiza();  //Va en el método actualiza

explosiones.Dibuja(g);  //Va en el método dibuja
```

    b. Cuando la nave sea alcanzada por un comelón explotará. Esto significa fin del juego. Para controlar este caso vamos a incluir una variable booleana llamada "`finjuego`". Como lo vamos a mostrar con una fuente grande definimos "`font`". Los agregamos en la sección de declaración o atributos. Y la declaración del "`font`" lo incluimos en el constructor.

```
private boolean finjuego //Línea en declaración
private final Font font; //Línea en declaración

font = new Font("Arial", Font.PLAIN, 50); //En el constructor
```

    c. Si una bala alcanza a un comelón lo haremos explotar. Si una comelón alcanza a la nave indicamos la finalización del juego al mismo tiempo que explotamos la nave. Agregamos en el método "Actualiza" lo siguiente.

```
        for (Bala bala : Balas.balas) {
            for (Comelon comelon : Comelones.comelon) {
                if (bala.Murieron(comelon)) {
                    Explosiones.Explota(comelon);
                }
            }
```

```
}
for (Comelon comelon : Comelones.comelon) {
    if (nave.Murieron(comelon)) {
        finjuego = true;
        Explosiones.Explota(nave);
    }
}
```

13. En caso de haber terminado el juego, le mostramos al jugador con un mensaje. Para ello insertamos la siguiente instrucción en el método “Dibuja”. atributos.

```
if (finjuego) {
    g.setFont(font);
    g.drawString("Oops! <Dar Esc>", 150, 200);
}
```

14. Corremos el juego y vemos que una bala hace explotar un “Comelón”.

15. Sin embargo también un comelón puede alcanzar la nave. En caso de un comelón alcance el nivel de la nave, se comerá a la nave y explotará.
    a. Antes de alcanzar la nave.

b. Después de ser “comida” la nave.

16. Podemos optar por un segundo nivel donde los “comelones” lancen bombas. O en este mismo nivel agregar una opción para que los comelones se “defiendan”. Optaremos por esta segunda opción y dejaremos como ejercicio crear un nivel de mayor dificultad.
    a. Crearemos la clase “`Bomba`” a partir del siguiente dibujo con nueve fotos de 32 pixeles de ancho por 55 pixeles de alto.

    b. La clase sigue la lógica de construcción de los objetos anteriores.

```
package entidades;

import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.time.Duration;
import principal.MiVentana;

public class Bomba extends Entidad {

    final static int BANCHO = 32;
    final static int BALTO = 55;

    public Bomba(BufferedImage[] bi, int x, int y) {
        animar.setFrames(bi, Duration.ofMillis(100));
        setX(x);
        setY(y);
        setVel(0.5);
    }

    @Override
    public void Dibuja(Graphics2D g) {
        if (estaVivo()) {
            g.drawImage(
                    animar.getImagen(),
```

```
                    (int) getX(),
                    (int) getY(),
                    null
                );
            }
        }

        @Override
        public void Actualiza() {
            if (estaVivo()) {
                animar.Actualiza();
                setY(getY() + getVel());
                if (getY() > MiVentana.ALTO) {
                    setVivo(false);
                }
            }
        }

        @Override
        public Rectangle getRectangulo() {
            return new Rectangle((int) getX() + 10, (int) getY() + 10,
                    BANCHO - 20, BALTO - 20);
        }
    }
```

c. Ahora crearemos un grupo de bombas disponibles para lanzarse desde la posición de los comelones.

```
package entidades;

import static entidades.Bomba.BALTO;
import static entidades.Bomba.BANCHO;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.IOException;
import javax.imageio.ImageIO;

public class Bombas {

    final static int N = 9;

    public static Bomba[] bombas;
    static final int NB = 8;

    public Bombas() {
        BufferedImage[] bi = new BufferedImage[N];
        try {
            BufferedImage spritesheet = ImageIO.read(
                    getClass().getResourceAsStream(
                            "/recursos/enemigos/bomba.png"
                    )
            );
            for (int j = 0; j < N; j++) {
                bi[j] = spritesheet.getSubimage(j * BANCHO, 0,
                        BANCHO, BALTO
                );
            }
        } catch (IOException e) {
            System.out.println("Error al leer bombas");
        }
        bombas = new Bomba[NB];
        for (int i = 0; i < NB; i++) {
            bombas[i] = new Bomba(bi, 80 * i, 20);
        }
    }

    public void Actualiza() {
```

```
        for (int i = 0; i < NB; i++) {
            bombas[i].Actualiza();
        }
    }

    public void Dibuja(Graphics2D g) {
        for (int i = 0; i < NB; i++) {
            bombas[i].Dibuja(g);
        }
    }

    public static boolean Lanzar(Comelon comelon) {
        int x, y;
        boolean q = false;
        x = comelon.getXCentro();
        y = comelon.getYCentro();
        for (Bomba bomba : bombas) {
            if (!bomba.estaVivo()) {
                bomba.setX(x);
                bomba.setY(y);
                bomba.setVivo(true);
                q = true;
                break;
            }
        }
        return q;
    }
}
```

12. Añadir en la clase "`NivIJuego`" un objeto de tipo "`Bombas`". Como referencia podemos poner "abajo" de "`Explosiones`" en cada sección (declaración, constructor, actualiza, dibuja) incorporaremos una función similar.

```
private final Bombas bombas; //Línea en declaración

bombas = new Bombas(); //Incorporado al constructor

bombas.Actualiza();  //Va en el método actualiza

bombas.Dibuja(g);  //Va en el método dibuja
```

13. Quien lanza las "`Bombas`" es el comelón por ello vamos a incluir que lance una bomba. La vamos a lanzar con una probabilidad de 1/10 cuando inicie la animación y vaya hacia la derecha.

    a. Buscamos cuando la animación vaya a la derecha "**`case`** `DER`".

```
case DER:
    if (animar.isNuevaMuestra() && RND.nextInt(10) == 0) {
        //Bombas.Disparo(this);
    }
```

    b. Quitamos el comentario y sustituimos por lo siguiente.

```
case DER:
    if (animar.isNuevaMuestra() && RND.nextInt(10) == 0) {
        Bombas.Lanzar(this);
    }
```

14. Si corremos el programa vemos que no hay interacción entre las bombas y las balas y la nave.

15. Para incorporar la interacción agregamos lo siguiente en la clase "`NivIJuego`" en el método actualizar. Ahora si una bala alcanza una bomba explota. Si una bomba alcanza a la nave explota.

```
for (Bala bala : Balas.balas) {
    for (Comelon comelon : Comelones.comelon) {
        if (bala.Murieron(comelon)) {
            Explosiones.Explota(comelon);
        }
    }
    for (Bomba bomba : Bombas.bombas) {
        if (bala.Murieron(bomba)) {
            Explosiones.Explota(bomba);
        }
    }
}
for (Comelon comelon : Comelones.comelon) {
    if (nave.Murieron(comelon)) {
        finjuego = true;
        Explosiones.Explota(nave);
    }
}
for (Bomba bomba : Bombas.bombas) {
    if (nave.Murieron(bomba)) {
        finjuego = true;
        Explosiones.Explota(nave);
    }
}
```

16. Ahora ya es sensible la nave a las bombas. Y las bombas a las balas ☺.

Bibliografía Preliminar.

[1] B. Eckel, Piensa en Java, Madrid: Prentice Hall, 2003.

[2] H. Schildth, Java - The Complete Reference, 9th Edition, Oracle Press, 2014.

[3] R. M. Reese y J. L. Reese, java 7 New Features Cookbook, Birmingham - Mumbai: Packt Publishing, 2012.

No. 13. Almacenamiento en base de datos.

OBJETIVO

Mostrar la interacción con el usuario para almacenar información en una base de datos. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Netbeans, permitiendo que el usuario agregue instrucciones para manipular la información procesada.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 4: Acceso a datos. 4.1 Introducción. 4.2 Conexión a origen de datos. 4.3 Manipulación de datos. 4.4 Visualización de datos.	**No. 13. Almacenamiento en base de datos.** Objetivo. Conocer las formas de almacenar información.

Introducción

Java brinda soporte para persistencia ligera (ver práctica 12), y mecanismos como JDBC e Hibernate brindan un soporte más sofisticado para almacenar y recuperar información de objetos en bases de datos. [1]

JDBC es un estándar para conectarse directamente a un manejador de base de datos usando SQL pero es propenso a contener un código oneroso. JPA es un estándar cuya tecnología permite mapear objetos en código y tablas de base de datos (Hibernate es un proveedor de JPA). Finalmente JPA usa JDBC y SQL. [2]

Un controlador (driver) JDBC permite que las aplicaciones Java se conecten a una base de datos en un RDBMS (base de datos relacional) particular como ejemplo los manejadores de datos H2, Mysql, Excel, SQL server, Informix, Postgres, etc. (ver apéndice I) y le permite manipular esa base de datos usando la API JDBC. [3]

Hay RDDBMS en puro Java sin tener que hacer una instalación de software nativo y se puede incorporar fácilmente en una aplicación Java. [4]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Netbeans instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:
Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para buscar información sobre JDBC, el estándar JPA.
4. Investigar ejemplos usando JDBC y JPA.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de las funciones JDBC y JPA.
2. Seleccionar a varios participantes para exponer ejemplos JDBC y JPA.
3. Seleccionar a voluntarios para explicar el uso de
4. Llenar el recurso de Wiki con las definiciones de JDBC y JPA.

Reporte de los alumnos (resultados)

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo el uso de JDBC.
3. Participar en Moodle con el glosario de términos incorporando JDBC y JPA

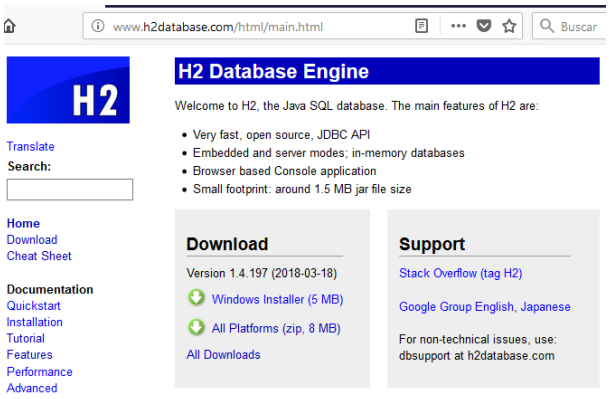
Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   “Hacer un programa para crear una tabla, insertar registros, actualizarlos y borrarlos en un manejador de base de datos usando JDBC”.

   El algoritmo de solución aquí propuesto es: “Hacer una interface gráfica a través de Netbeans donde se proporcione una serie de instrucciones SQL sean

procesadas usando expresiones regulares y enviadas en secuencia a la base de datos".

2. Primeramente requerimos seleccionar cual manejador de base de datos requerimos emplear. La ventaja de emplear el driver JDBC es que sólo se programa una sola vez. Difiere cada solo cuando se hace una conexión a cada base de datos. También difiere en que las instrucciones SQL tienen algunas variantes particulares a cada manejador de base de datos. El software de instalación del manejador de base de datos puede requerir una instalación adicional pudiendo a llegar a ser un proceso largo. La base de datos escogida por la simplicidad de su incorporación a un proyecto con Netbeans es la base de datos H2. El proceso de instalación se muestra en los pasos siguientes.
    a. Ir a la página "H2 Database Engine". Cabe hacer la aclaración sobre la versión del manejador de base de datos. Esta puede variar de tiempo en tiempo.
    La liga es: http://www.h2database.com/html/main.html

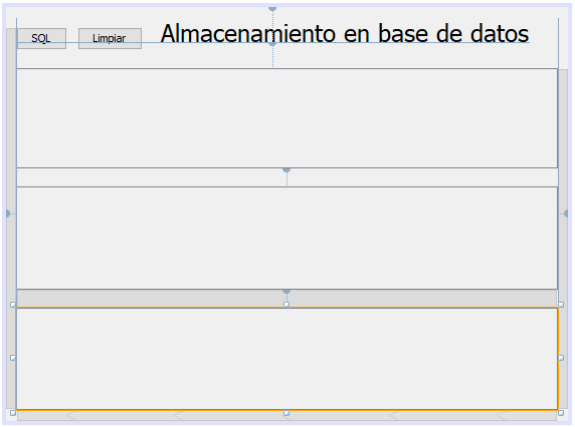


    b. Bajar el software en la liga

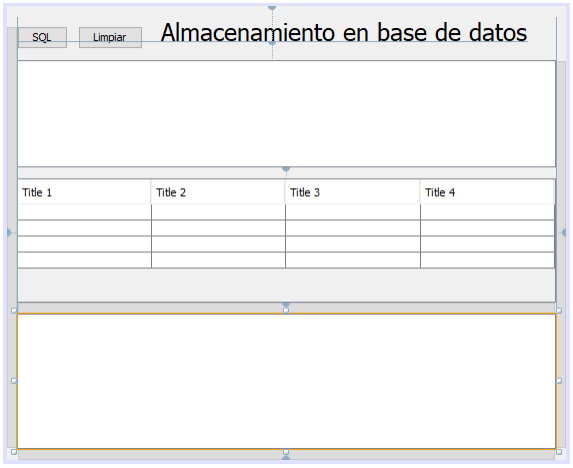
    c. Cuando se bajó este programa fue el archivo "h2-2018-03-18.zip"
    d. Se descomprime y en la carpeta "h2-2018-03-18.zip\h2\bin" se encuentra el archivo "h2-1.4.197.jar".
    e. Por facilidad para incorporarlo a un proyecto de Netbeans se hace una copia al subdirectorio "C:\tmp". Ahí lo dejamos por el momento.

3. Ahora se hace un proyecto en Netbeans denominado "P13BaseDeDatos" sin clase Main y con un JFrame de nombre "MiVentana" en el paquete "paketeGuarde".

4. Vamos a crear una interface gráfica. Se mostrará el resultado de consultas SQL proporcionadas por el usuario.
    a. Incorporamos dos botones de tipo JButton uno con texto “SQL” y con nombre de la variable “btnSql” y el otro con texto “Limpiar” y con nombre “btnLimpiar”. Agregamos una etiqueta con texto “Almacenamiento en base de datos” con un Font grande de tamaño 24. Incorporamos 3 JScrollPane. Estos elementos pueden aparecer como la siguiente muestra.



    b. Dentro del JScrollPane superior agregamos un JTextArea con nombre “taSql”.
    c. Hacemos una operación similar con el JScollPane inferior. Incorporamos un “JTextArea” con nombre “taSalida”.
    d. Para el JScrollPane intermedio escogemos un “JTable”. Lo nombramos “tblDatos” y se agrega dentro de él. Se muestra a continuación.



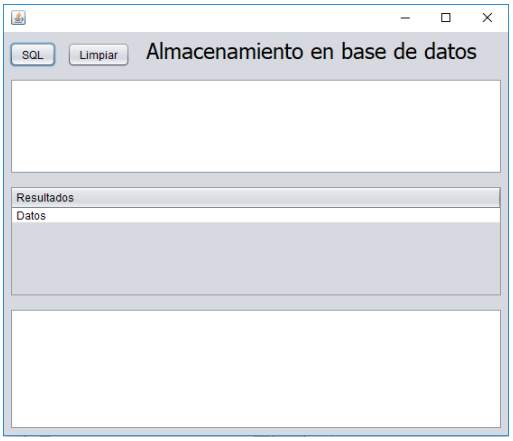
5. Los componentes gráficos de tipo “JTable” funcionan en base a un arreglo de contiene una matriz de datos de tipo “Object” y para mostrar los encabezados se

basan en un arreglo de tipo String. Mostraremos una asignación constante para que no tenga esa vista generada por default.

a. Modificamos el constructor incorporando nuestros datos y encabezado con el siguiente código agregado al constructor.

```
public MiVentana() {
    initComponents();
    tblDatos.setModel(new javax.swing.table.DefaultTableModel(
            new Object[][]{
                {"Datos"}
            },
            new String[]{
                "Resultados"
            }
    ));
}
```

b. El resultado al correr nuestro programa se ve como lo siguiente. (Seleccionando como main “paketeGuarde.Miventana”).



6. Existen dentro de Java en JDBC varias clases relacionadas con la conexión a una base de datos. Otras clases relacionadas con consultas (SELECT, UPDATE, etc). Y otras clases con información relativa a las columnas de las tablas. Vamos a crear una clase abstracta que nos cierta funcionalidad integrada en varios métodos y nos permita ser la clase base para administrar una colección de manejadores de base de datos.
   a. Creamos un paquete denominado “libbds”.
   b. Dentro de él creamos una clase abstracta denominada “`Consulta`” con los siguientes atributos y sin constructor, con métodos “`conecta`” para establecer una sesión, además el método “`Cierra`” para finalizar la sesión. Para el caso partículas de una base de datos H2 se reescribe el método “`conecta`”.

```
/*
 * ConsultaDBS.java
 *
 * Created on 8 de junio de 2006, 11:47
 */
package libbds;
```

```
import java.sql.*;

/**
 *
 * @author jflores
 */
public abstract class Consulta {

    public final int MAXROWS = 50;

    protected Connection coneccion = null;  // sesión
    protected Statement instruccion = null; // instrucción SQL
    protected ResultSet rs = null;          // tabla de resultados
    protected ResultSetMetaData md = null;  // nombre y tipo de columnas

    private int numcols = 0;
    private int numrows = 0;
    private int numupdates = 0;
    private boolean fuequery = false;

    protected void conecta(String conectaStr) throws SQLException {
        coneccion = DriverManager.getConnection(conectaStr);
        instruccion = coneccion.createStatement
           (ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
    }

    public void Cierra() throws Exception {
        if (coneccion != null) {
            coneccion.close();
        }
    }
}
```

c. De la clase localizamos algunos objetos de la librería de java JDBC. Vemos un objeto "`Connection coneccion`" especifica una sesión con una base de datos específica. Un objeto de tipo "`Statement instruccion`" permite darle una instrucción SQL a la base de datos. "`ResultSet rs`" nos da una tabla de resultados. Un objeto de tipo "`ResultSetMetaData md`" permite obtener información de la tabla de resultados sobre el nombre y tipo de las columnas.

d. En la clase también hay otros objetos auxiliares. Uno de ellos "`conectaStr`" de tipo "`String`" tiene un formato URI en particular como ejemplo si se tiene una conexión a una base de datos MySql sería como:

```
jdbc:mysql://192.168.1.74:3306/BD?user=sa&password=xxx
```

Otro ejemplo de conexión a una base de datos Sybase:

```
jdbc:sybase:Tds://192.168.1.74:5000/BDTEC?Schema=dbo&user=sa&password=xxx
```

e. Aunque la base de datos H2 no tiene habilitado conexiones de éste tipo hay una variante de conexión.

```
DriverManager.getConnection("jdbc:h2:C:\\tmp\\prue", "sa", "xxx");
```

7. Ahora crearemos la clase "ConsultaH2" en el paquete "libbds" derivada de "Consulta" donde definiremos como conectarse.

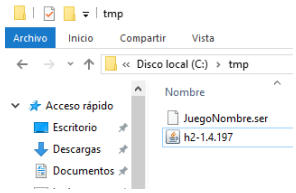
```
/*
 * ConsultaMySql.java
 *
 * Created on 8 de junio de 2006, 11:55
 */
package libbds;

import java.sql.DriverManager;
import java.sql.ResultSet;
import java.sql.SQLException;

/**
 *
 * @author jflores
 */
public final class ConsultaH2 extends Consulta {

    /**
     * Creates a new instance of ConsultaH2
     *
     * @param donde
     * @throws java.lang.Exception
     */
    public ConsultaH2(String donde) throws Exception {
        Class.forName("org.h2.Driver");
        String url = "jdbc:h2:" + donde;    //"jdbc:h2:~/test", "sa",""
        conecta(url);
    }

    @Override
    protected void conecta(String url) throws SQLException {
                                            //url user password
        coneccion = DriverManager.getConnection(url, "sa", "");
        instruccion = coneccion.createStatement
          (ResultSet.TYPE_SCROLL_SENSITIVE, ResultSet.CONCUR_UPDATABLE);
    }
}
```

8. Nos regresamos a la clase MiVentana y creamos un objeto de la clase "Consulta" lo agregamos como atributos y lo inicializamos en el constructor de tipo "ConsultaH2" de la siguiente forma. Está inmerso en un bloque try-catch para revisar si existe algún problema de conexión. Si es el caso se mostrará el tipo de error y se dará por concluida la aplicación.

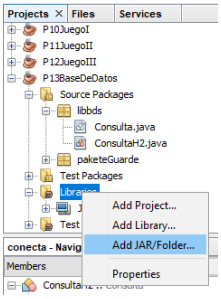
```
    Consulta h2;

    public MiVentana() {
        initComponents();
        tblDatos.setModel(new javax.swing.table.DefaultTableModel(
                new Object[][]{
                    {"Datos"}
                },
                new String[]{
                    "Resultados"
                }
        ));
        try {
            h2 = new ConsultaH2("C:\\tmp\\prue");
        } catch (Exception ex) {
            System.out.println(ex);
            System.exit(0);
```

```
        }
    }
```

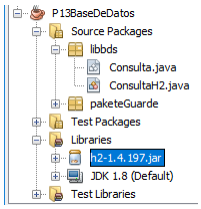
9. Si corremos el programa nos indica un error “java.lang.ClassNotFoundException: org.h2.Driver”. Para resolver éste problema hay que incluir el archivo obtenido del paso 2.
    a. El archivo lo localizamos en “C:\tmp\ h2-1.4.197.jar”.



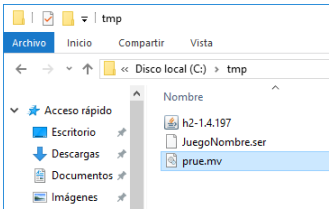
    b. En el proyecto incorporaremos éste archivo que es la librería del driver y el manejador de la base de datos al mismo tiempo. Abrimos la sección “libraries” y con el botón derecho usamos la opción “Add JAR/Folder”.



    c. Nos vamos al subdirectorio “C:\tmp” y localizamos el archivo “h2-1.4.197.jar” y damos “Abrir”. Ahora ya tenemos incorporada la librería.



10. Corremos nuestro programa ahora no nos indica ningún problema de conexión y ha creado la base de datos “prue” localizada como archivo en “C:\tmp\ prue.mv”.



11. Vamos a continuar con la interface gráfica. En MiVentana en “Design” se generan eventos para la continuación de las actividades.
    a. Se recomienda cerrar la conexión a la base de datos al terminar el programa. en propiedades del JFrame escogemos el evento windowClosing lo generamos y escribimos en él lo siguiente.

```
private void formWindowClosing(java.awt.event.WindowEvent evt) {
    try {
        h2.Cierra();
    } catch (Exception ex) {
        System.out.println(ex);
    }
}
```

    b. En “Design” seleccionamos el botón “Limpiar” y damos doble click y nos lleva al código donde programamos las siguientes dos líneas.

```
    private void btnLimpiarActionPerformed
                  (java.awt.event.ActionEvent evt) {
        taSql.setText("");
        taSalida.setText("");
    }
```

    c. Para que la tabla sea “limpiada” movemos del constructor de MiVentana la asignación del modelo éste método.

```
private void btnLimpiarActionPerformed
              (java.awt.event.ActionEvent evt) {
    taSql.setText("");
    taSalida.setText("");
    tblDatos.setModel(new javax.swing.table.DefaultTableModel(
            new Object[][]{
                {"Datos"}
            },
            new String[]{
                "Resultados"
            }
    ));
}
```

    d. Y en el constructor llamamos a éste evento de la siguiente manera.

```
public MiVentana() {
    initComponents();
    btnLimpiarActionPerformed(null);
    try {
        h2 = new ConsultaH2("C:\\tmp\\prue");
    } catch (Exception ex) {
        System.out.println(ex);
        System.exit(0);
    }
}
```

12. Una instrucción SQL la podemos dar en varios renglones para poder “visualizar” secciones de la instrucción y poderla dar en varios renglones. Lo anterior es apto para el humano pero para el manejador de base de datos requiere que las instrucciones SQL sólo contenga caracteres permitidos.

Requerimos de una clase "`Parser`" para que nos regrese una lista de instrucciones limpias de saltos de línea y del separador ";". La crearemos en el paquete "`paketeSalve`". Incorporar las funciones previas en un método "`getInstrucciones`". En la búsqueda de lograr distinguir entre varias instrucciones se establece el separador de instrucción ";".
Usa dos expresiones regulares la primera encuentra en una variable tipo "`String`" una instrucción de todas las letras hasta un ";" y la otra expresión para cada instrucción reemplaza tabuladores y saltos de línea por un solo espacio.
Todas las líneas son agrupadas en una lista de instrucciones de tipo "`String`" llamada "`result`".

```
public class Parser {

    private static final Pattern PINST, PESP;

    static {
        PINST = Pattern.compile("(?s)(?m)\\s*([^;]+)\\s*;*\\s*");
        PESP = Pattern.compile("(?s)(?m)([^\\s]+)\\s*");
    }

    public static List<String> getInstrucciones(String data) {
        StringBuilder sb = new StringBuilder();
        List<String> result = new ArrayList<>();

        Matcher matcher = PINST.matcher(data);

        while (matcher.find()) {
            result.add(matcher.group(1));
        }

        for (int i = 0; i < result.size(); i++) {
            sb.setLength(0);
            matcher = PESP.matcher(result.get(i));
            while (matcher.find()) {
                sb.append(matcher.group(1)).append(' ');
            }
            if (sb.length() > 0) {
                sb.setLength(sb.length() - 1);
            }
            result.set(i, sb.toString());
        }

        return result;
    }
}
```

13. La funcionalidad de la clase "`Consulta`" no está cubierta todavía. Los manejadores de datos tienen tiempo determinado para una sesión. Si la instrucción no está sintácticamente correcta. Si la sintaxis es correcta tal vez una tabla o un campo no exista. La función más relevante para nuestro proyecto es la consulta. La consulta puede ser un SELECT, UPDATE, DELETE etc. Para lograr una consulta debe cumplir con los requisitos mencionados. Si se cumplió con los requisitos anteriores y se un SELECT entonces nos regresará un conjunto de resultados, si no, será una actualización de los datos. El método fundamental es "`Consulta(String query)`" para lograr obtener los resultados deseados los adicionales son getters de las variables generadas actualizadas con excepción de "`getNRows`". Éste último

método obtiene indirectamente el número de filas (rows) usando los métodos de "`ResultSet`" para posicionar al final de los resultados y saber cuántos son.

```
public void Consulta(String query) throws SQLException //usa rs y numcol
{
    fuequery = instruccion.execute(query);
    if (fuequery) //Regresa true si es SELECT
    {
        rs = instruccion.getResultSet();
        md = rs.getMetaData();

        numrows = getNRows();
        numcols = md.getColumnCount();
        numupdates = 0;
    } else {
        numrows = 0;
        numcols = 0;
        numupdates = instruccion.getUpdateCount();
    }
}

private int getNRows() {
    int nrows = 0;
    try {
        if (rs.last()) {
            nrows = rs.getRow();
            rs.beforeFirst();
        }
    } catch (SQLException ex) {
    }
    return nrows;
}

public boolean isFuequery() {
    return fuequery;
}

public int getNumupdates() {
    return numupdates;
}

public int getNumcols() {
    return numcols;
}

public int getNumrows() {
    return numrows;
}
```

14. Pero requerimos explotar la información en búsqueda de compatibilidad con el componente JTable. En el paso 5 no dimos cuenta mostramos el cambio de la información mostrada por default en la JTable. Requerimos una matriz tipo "`Object[][]`" y un arreglo de "`String[]`". Generaremos dos métodos adicionales para cumplir con los requisitos de JTable para la clase "`Consulta`". Agregamos el método "`getData`" y el método "`getHeader`". Como información, en el método "`getData`" se restringe el número de renglones a desplegar en "`MAXROWS = 50`".

```
public Object[][] getData() throws SQLException {
    Object[][] data = null;
    int rows, nrow = 0;
    Object[] row;

    if (fuequery) {
```

```
            rows = numrows;
            rows = rows > MAXROWS ? MAXROWS : rows;
            data = new Object[rows][numcols];
            while (rs.next()) {
                row = data[nrow];
                for (int i = 0; i < numcols; i++) {
                    row[i] = rs.getObject(i + 1);
                }
                nrow++;
            }
        }
        return data;
    }

    public String[] getHeader() throws SQLException {
        String header[] = new String[getNumcols()];
        for (int i = 1; i <= getNumcols(); i++) {
            header[i - 1] = md.getColumnName(i);
        }
        return header;
    }
```

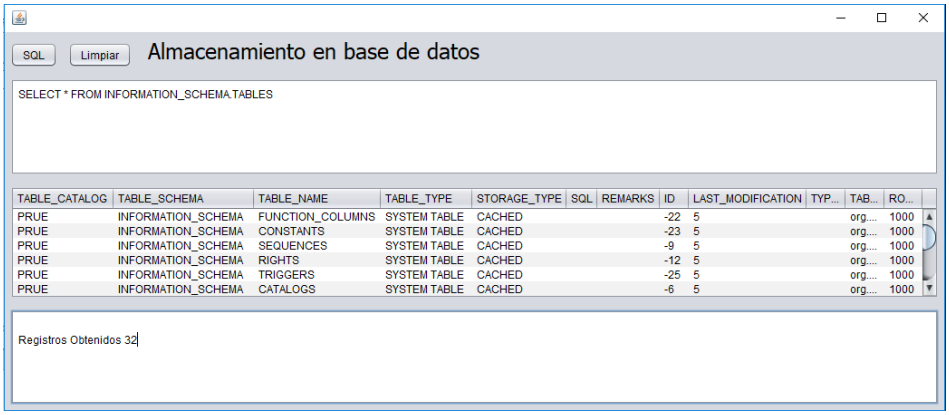
15. En MiVentana en “Design” falta la generación de un evento para procesar las instrucciones SQL. Seleccionamos el botón “SQL” damos doble clic y nos lleva al código. Tenemos la opción de seleccionar o no el texto del componente llamado “`taSql`” y lo tomamos para procesarlo. En caso que haya sido la instrucción SELECT se obtendrán los datos y el encabezado para la tabla “`tblDatos`”

```
    private void btnSqlActionPerformed(java.awt.event.ActionEvent evt) {
        List<String> inst;

        if (taSql.getSelectedText() != null) {
            inst = Parser.getInstrucciones(taSql.getSelectedText());
        } else {
            inst = Parser.getInstrucciones(taSql.getText());
        }

        try {
            for (int i = 0; i < inst.size(); i++) {
                h2.Consulta(inst.get(i));
                if (h2.isFuequery()) {
                    tblDatos.setModel(
                            new javax.swing.table.DefaultTableModel(
                                    h2.getData(), h2.getHeader()));
                    taSalida.append("\nRegistros Obtenidos " +
                            h2.getNumrows() + "\n");
                } else {
                    taSalida.append("\nRegistros Actualizados " +
                            h2.getNumupdates() + "\n");
                }
            }
        } catch (SQLException ex) {
            taSalida.append("\n" + ex + "\n");
        } catch (Exception ex) {
            taSalida.append("\n" + ex + "\n");
        }
    }
```

16. Ya estamos listos para nuestras primeras instrucciones SQL. Para probar nuestro programa crearemos una tabla.
    a. Una instrucción propietaria de H2 para listar las tablas es:

```
SELECT * FROM INFORMATION_SCHEMA.TABLES
```

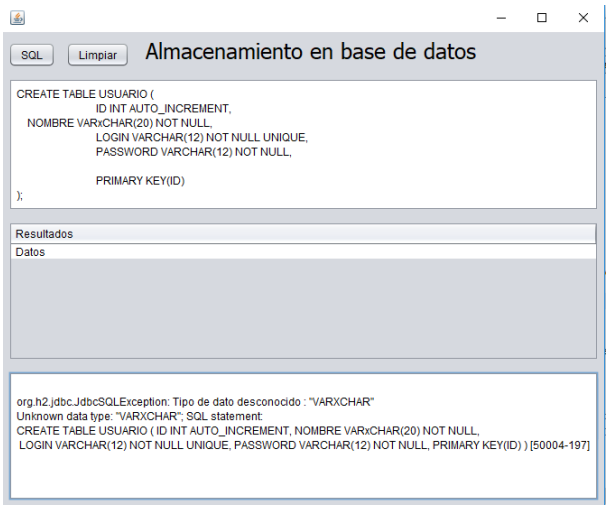
    b. Obtenemos un resultado mostrado al correr y dar clic en el botón “SQL”.



    c. Si se desea crear una tabla a propósito tiene un error.

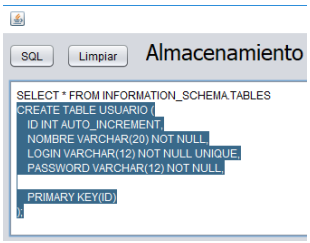
```
CREATE TABLE USUARIO (
    ID INT AUTO_INCREMENT,
    NOMBRE VARxCHAR(20) NOT NULL,
    LOGIN VARCHAR(12) NOT NULL UNIQUE,
    PASSWORD VARCHAR(12) NOT NULL,

    PRIMARY KEY(ID)
);
```

    d. Ahora el resultado al dar clic en el botón “SQL” es el mostrado, y la aplicación nos da el tipo de error.



    e. Una vez corregido el error nos muestra “Registros Actualizados 0”.

f. Si damos “Limpiar” se presenta el borrado de las tres secciones.
g. Podemos incluir la instrucción del paso 16-a y de nuevo la instrucción de creación de la tabla del paso 16-c siguiente forma. Marcamos la creación de la tabla la ejecutamos (dando clic en el botón SQL). Como la tabla ya está creada nos lo hace saber. Así mismo seleccionamos la primer instrucción y en el último renglón nos muestra la existencia de la tabla usuario.



17. Ahora vamos a insertar registros en nuestra tabla.
    a. Podemos incluir todos las instrucciones y ejecutarlas.

```
insert into usuario values (null, 'Julio Flores','jflores','xxxxx');
insert into usuario values (null, 'Carmen
Salinas','csalinas','aaaaa');
insert into usuario values (null, 'Angelica
Maria','amaria','bbbbb');
insert into usuario values (null, 'Chucho el Roto','croto','ccccc');
select * from usuario;
insert into usuario values (null, 'Julio
Iglesias','jiglesias','ddddd');
insert into usuario values (null, 'Agustin Lara','alara','eeeee');
```

    b. Podemos borrar elementos e insertarlos de nuevo y los actualizamos.

```
select * from usuario where nombre like '%t%';
delete usuario where nombre like '%t%';
insert into usuario values (null, 'Chucho el Roto','croto','ccccc');
insert into usuario values (null, 'Agustin Lara','alara','eeeee');
select * from usuario;

update usuario set password='zzzz' where nombre like '%t%';
select * from usuario where nombre like '%t%';
select * from usuario
```

18. Ejercicio finalizado ☺.

Bibliografía Preliminar

[1] B. Eckel, Piensa en Java, Madrid: Prentice Hall, 2003.

[2] M. D, «StackOverflow,» [En línea]. Available: https://stackoverflow.com/questions/11881548/jpa-or-jdbc-how-are-they-different. [Último acceso: Dic 2017].

[3] H. M. D. P. J. Deitel, Como Programar en Java / 9 Ed., PEARSON, 2012.

[4] H2, «H2 Database,» [En línea]. Available: http://www.h2database.com/html/main.html. [Último acceso: Dic 2017].

Apéndice I.

Clases derivadas de consulta para diferentes manejadores de base de datos en reemplazo de la clase `ConsultaH2` del paso 7. *(Para algunos manejadores de base de datos se requiere el software y el JConnector específico obtenido del mismo proveedor).*

1. Para un manejador de base de datos MySql.

```
/*
 * ConsultaMySql.java
 *
 * Created on 8 de junio de 2006, 11:55
 */
package libbds;

/**
 *
 * @author jflores
 */
public class ConsultaMySql extends Consulta {

    /**
     * Creates a new instance of ConsultaMySql
     */
    private ConsultaMySql() {
    }

    public ConsultaMySql(String ip, String user, String passwd) throws Exception {
        Class.forName("com.mysql.jdbc.Driver").newInstance();
        String basededatos = "";
        String Url = ip + "/" + basededatos;  //localhost/Cafe
        String Res = "jdbc:mysql://" + Url
                + "?user=" + user + "&password=" + passwd;
        super.conecta(Res);
        //jdbc:mysql://192.168.1.74:3306/BD?user=sa&password=xxx
    }
}
```

2. Para un manejador de base de datos Sybase.

```
    /*
     * ConsultaMySql.java
     *
     * Created on 8 de junio de 2006, 11:55
     */
    package libbds;

    import java.sql.Driver;
    import java.sql.DriverManager;

    /**
     *
     * @author jflores
     */
    public class ConsultaSybase extends Consulta {

        /**
         * Creates a new instance of ConsultaMySql
         *
         * @param ip
         * @param user
```

```
         * @param passwd
         * @throws java.lang.Exception
         */
        public ConsultaSybase(String ip, String user, String passwd) throws Exception {
            DriverManager.registerDriver(
                    (Driver) Class.forName(
                            "com.sybase.jdbc4.jdbc.SybDriver").newInstance());
            String Url = ip + ":5000/BDTEC?Schema=dbo&";
            String Res = "jdbc:sybase:Tds:" + Url
                    + "user=" + user + "&password=" + passwd;
            super.conecta(Res);
            //jdbc:sybase:Tds:192.168.1.74:5000/BDTEC?Schema=dbo&user=sa&password=xxx
        }
    }
```

3. Para un manejador de base de datos Microsoft Access.

```
/*
 * ConsultaMDB.java
 *
 * Created on 23 de enero de 2006, 15:47
 */
package libbds;

/**
 *
 * @author jflores
 */
public class ConsultaMDB extends Consulta {

    /**
     * Creates a new instance of ConsultaMDB
     *
     * @param bd
     * @param u
     * @param p
     * @throws java.lang.Exception
     */
    public ConsultaMDB(String bd, String u, String p) throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String strcon = "jdbc:odbc:Driver={Microsoft Access Driver (*.mdb)};"
                + "Dbq=" + bd + ";"
                + "Uid=" + u + ";"
                + "Pwd=" + p + ";";
        super.conecta(strcon);
    }
}
```

4. Para un manejador de base de datos Dbase.

```
/*
 * ConsultaDBF.java
 *
 * Created on 8 de junio de 2006, 11:50
 */
package libbds;

/**
 *
 * @author jflores
 */
public class ConsultaDBF extends Consulta {

    public ConsultaDBF(String sitiobase) throws Exception {
        String strcon = "jdbc:odbc:Driver={Microsoft dBase Driver (*.dbf)};Dbq="
                + sitiobase;
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        super.conecta(strcon);
```

```
        }
    }
```

5. Para un manejador de base de datos Excel.

```
/*
 * ConsultaXLS.java
 *
 * Created on 23 de enero de 2006, 15:47
 */
package libbds;

/**
 *
 * @author jflores
 */
public class ConsultaXLS extends Consulta {

    /**
     * Creates a new instance of ConsultaMDB
     *
     * @param bd
     * @param u
     * @param p
     * @throws java.lang.Exception
     */
    public ConsultaXLS(String bd, String u, String p) throws Exception {
        Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
        String strcon = "jdbc:odbc:Driver={Microsoft Excel Driver (*.xls)};"
                + "Dbq=" + bd + ";"
                + "Uid=" + u + ";"
                + "Pwd=" + p + ";";
        super.conecta(strcon);
    }
}
```

No. 14. Instalar Interface Gráfica de Usuario Android.

OBJETIVO

Instalar la herramienta Interface Gráfica de Android.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 5: Programación de dispositivos móviles. 5.1. Introducción a las tecnologías y herramientas móviles. 5.2 Clasificación y aplicaciones de los dispositivos móviles. 5.3 Entorno operativo de las aplicaciones móviles. 5.4 Desarrollo de aplicaciones móviles. 5.5. Aspectos de seguridad.	**No. 14. Instalar Interface Gráfica de Usuario Android.** Objetivo. Instalar la herramienta Interface Gráfica de Android. **No. 15. Creación y estructura de un proyecto Android.** Objetivo. Crear una primera aplicación Android. **No. 16. Creación y manejo de actividades en un proyecto Android.** Objetivo. Crear una aplicación Android con varias actividades. **No. 17. Uso de unidades y layouts.** Objetivo. Conocer elementos comunes en una interface gráfica.

Introducción

Java es un lenguaje de programación orientado a objetos ampliamente utilizado. Las aplicaciones Java se ejecutan en Android, Windows, Mac OS X, Linux, Solaris etc.

Una interfaz gráfica para usuarios es una aplicación que proporciona varios servicios para facilitarle al programador el desarrollo de software [1].

El software requerido para instalar el Entorno de Desarrollo Integrado, en inglés Integrated Development Environment (IDE) es Android Studio y JDK (Ambiente de desarrollo de Java).

Android Studio es gratuito y se usa bajo licencia de Google. Java también es software gratuito y se usa bajo licencia "COMMON DEVELOPMENT AND DISTRIBUTION LICENSE (CDDL) Version 1.0"

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con los requisitos mínimos siguientes: [2]
   - Sistema Operativo: Microsoft Windows® 7/8/10
   - Procesador 64 bits, soporte en Intel VT-x, EM64T y XD. En AMD AMD-V y SSSE3
   - Memoria: 4 GB mínimo.
   - Espacio en disco: 4 GB libre sugerido.
   - Resolución de video 1280 x 800 mínimo.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso al recurso Wiki del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Conectarse a Internet para obtener el software de las páginas oficiales.
3. Seguir el procedimiento de la práctica.
4. Probar que el software esté funcionando.

Sugerencias didácticas

1. Realizar una investigación documental de los dispositivos Android.
2. Seleccionar a varios participantes para demostrar el funcionamiento de Android Studio.
3. Llenar el recurso de Wiki con las funciones más importantes de Android.

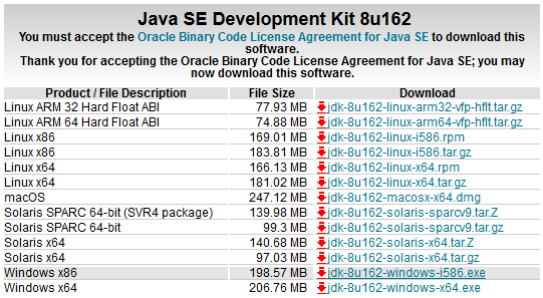
Reporte de los alumnos (resultados)

1. Elaborar ventajas de usar Netbeans con Android, Eclipse con Android, MIT App Inventor, Xamarin, Basic4android, Appcelerator Titanium, ItelliJ IDEA, Scripting Layer For Android, AppInventor. Elaborar una matriz de software para el desarrollo de aplicaciones en Android mostrando ventajas y desventajas.
2. Participar en Moodle en el glosario de términos incorporando software de desarrollo para Android.

Procedimiento:

1. Descargar e instalar el Software Java.
   a. Verificar tener instalado el JDK (Java Development Kit). Si es así ir al paso 2. En caso contrario seguir las instrucciones siguientes.
   b. En un navegador busque "java jdk download". Lo cual le mostrará varias ligas, de las cuales le mostrará una que diga "Descargas de Java SE - Oracle" y nos muestra una imagen como:

   c. Se selecciona Java Download. Damos aceptar la licencia.



Java SE Development Kit 8u162

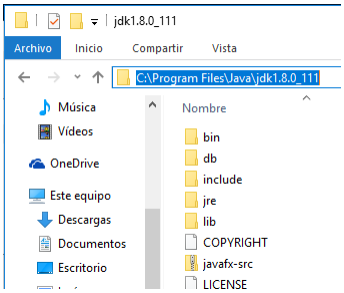
You must accept the Oracle Binary Code License Agreement for Java SE to download this software.

Thank you for accepting the Oracle Binary Code License Agreement for Java SE; you may now download this software.

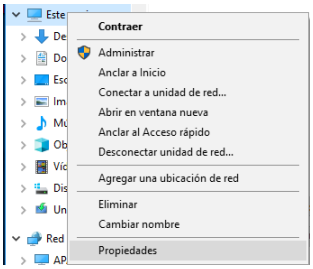
Product / File Description	File Size	Download
Linux ARM 32 Hard Float ABI	77.93 MB	jdk-8u162-linux-arm32-vfp-hflt.tar.gz
Linux ARM 64 Hard Float ABI	74.88 MB	jdk-8u162-linux-arm64-vfp-hflt.tar.gz
Linux x86	169.01 MB	jdk-8u162-linux-i586.rpm
Linux x86	183.81 MB	jdk-8u162-linux-i586.tar.gz
Linux x64	166.13 MB	jdk-8u162-linux-x64.rpm
Linux x64	181.02 MB	jdk-8u162-linux-x64.tar.gz
macOS	247.12 MB	jdk-8u162-macosx-x64.dmg
Solaris SPARC 64-bit (SVR4 package)	139.98 MB	jdk-8u162-solaris-sparcv9.tar.Z
Solaris SPARC 64-bit	99.3 MB	jdk-8u162-solaris-sparcv9.tar.gz
Solaris x64	140.68 MB	jdk-8u162-solaris-x64.tar.Z
Solaris x64	97.03 MB	jdk-8u162-solaris-x64.tar.gz
Windows x86	198.57 MB	jdk-8u162-windows-i586.exe
Windows x64	206.76 MB	jdk-8u162-windows-x64.exe

   d. Descargamos el software (verificar si su equipo soporta 32 o 64 bits) y lo ejecutamos y damos "Next" hasta que termine la instalación.

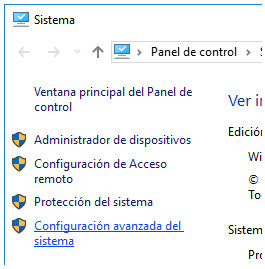
e. Android Studio requiere Java, y para poder localizar el JDK se requiere dar de alta una variable de entorno “JAVA_HOME”. Localizamos en donde está instalado el JDK. Usamos el explorador de archivos por lo general lo encontramos en “Archivos de programa” o “Program Files”. Realizamos una copia de la trayectoria.



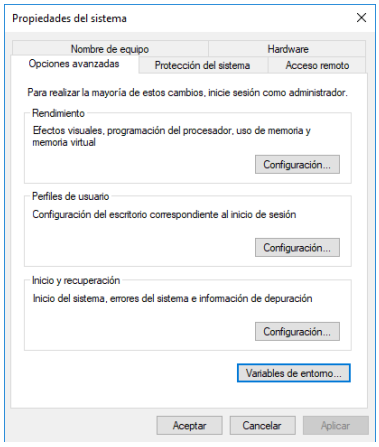
f. Una vez terminada vamos a dar de alta la variable de entorno “JAVA_HOME”. Usamos el explorador de archivos. Encontramos el icono “Este equipo” y con el botón derecho seleccionamos propiedades.



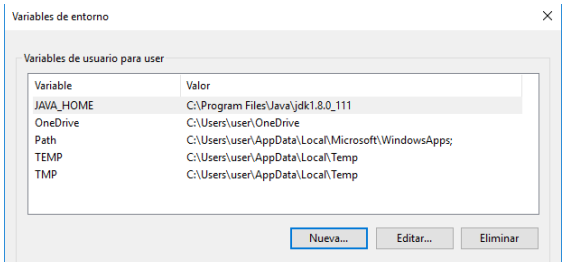
g. Una vez abierto propiedades. Seleccionamos “Configuración avanzada del sistema”.



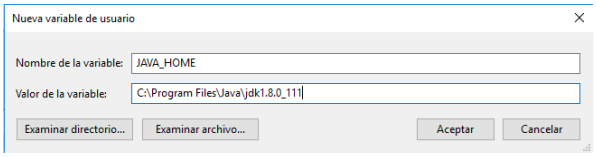
h. Y luego en la siguiente pantalla. Seleccionamos variables de entorno.



i. En la parte superior en donde dice: “Variables de usuarios para …” seleccionamos “Nueva”.

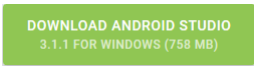


j. Del paso 1-e escribimos lo siguiente.

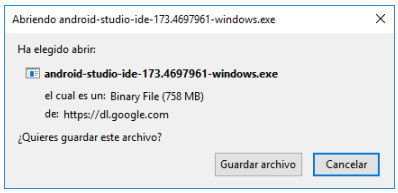


k. Y finalmente damos “Aceptar” y en la pantalla previa “Aceptar”. Y de nuevo “Aceptar”.

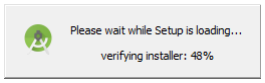
2. Descargar e instalar el software Android Studio.
   a. En un navegador busque “android studio download”. Nos va a llevar a la liga: https://developer.android.com/studio/index.html. Y la abrimos.
   b. Damos clic en la liga siguiente.



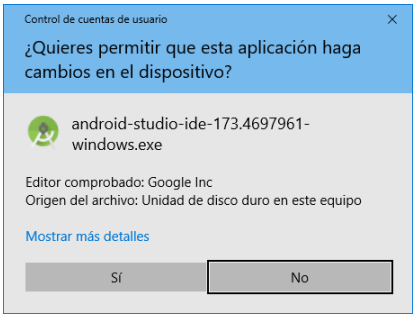
c. Al bajar el software obtenemos un archivo similar a “android-studio-ide-173.4697961-windows.exe”.



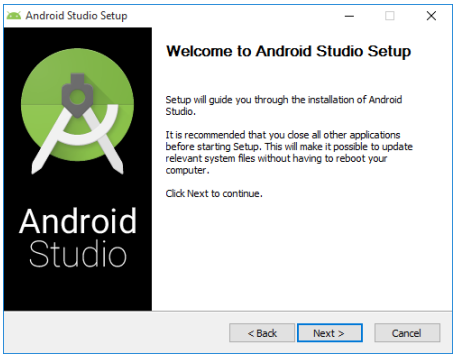
d. Lo ejecutamos. Se auto verifica.



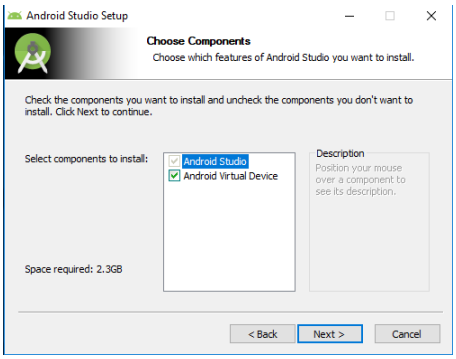
e. Nos pide permiso para ser instalado. Le indicamos que sí.



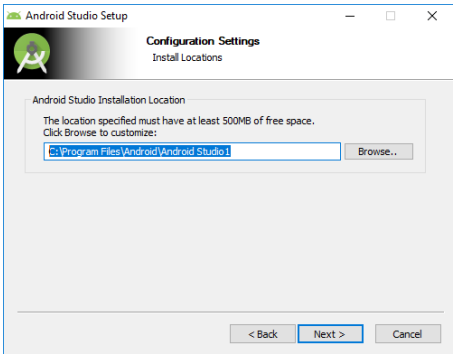
f. Luego se muestra la pantalla inicial. Damos “Next”.



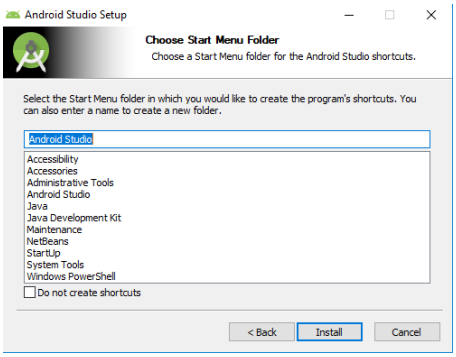
g. Luego que instale los componentes seleccionados por default y damos “Next”.



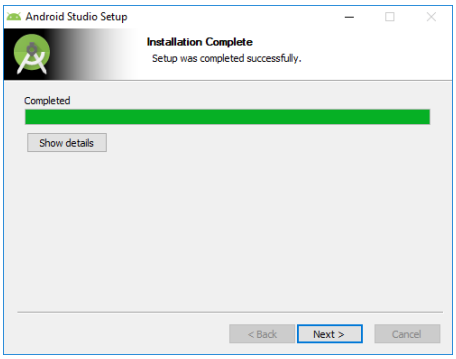
h. Luego nos muestra donde queremos instalar la aplicación. Damos “Next”.



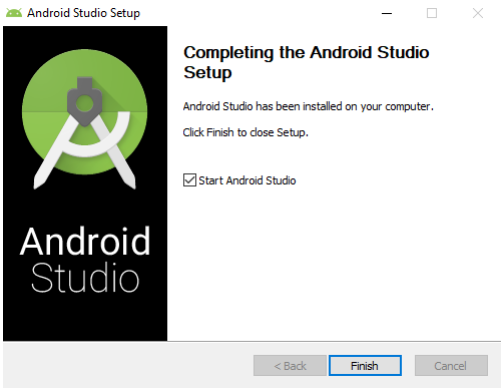
i. luego “Install”. Esperamos unos momentos a que la instalación termine.



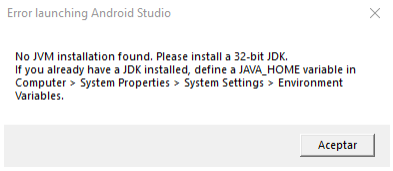
j. Nos muestra la instalación completa. Damos “Next”.



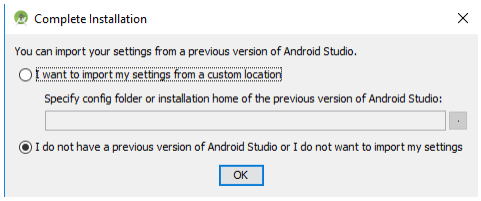
k. Al finalizar la instalación nos muestra una última pantalla. En este momento no pregunta si deseamos ejecutar Android Studio. Dejamos seleccionado la ejecución de Android Studio y damos “Finish”.



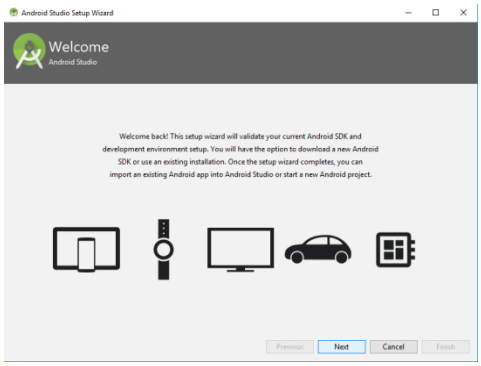
l. Al dar finalizar es posible que Android Studio no encuentre Java. Si nos da el siguiente error hacer el procedimiento del paso 1-e al 1-k.



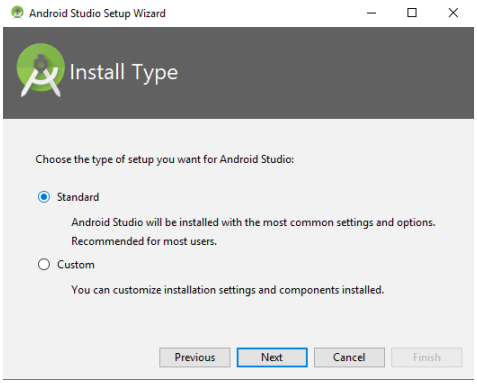
m. Lanzamos Android Studio. Y le indicamos no importar nada. Damos “OK”.



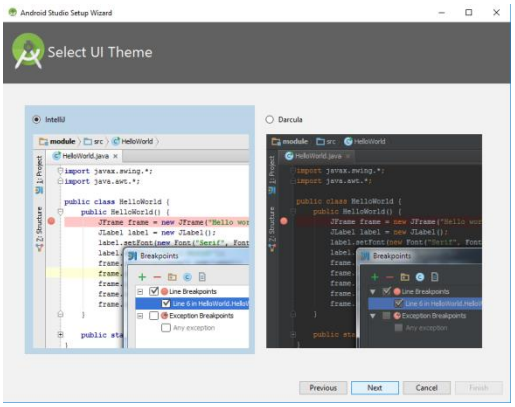
n. Nos muestra una pantalla donde se está cargando Android Studio. Luego sigue configurar el ambiente de desarrollo. Damos “Next”.



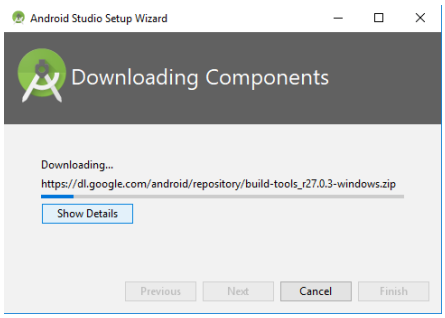
o. En las siguiente pantalla escogemos el valor default “Standard”.



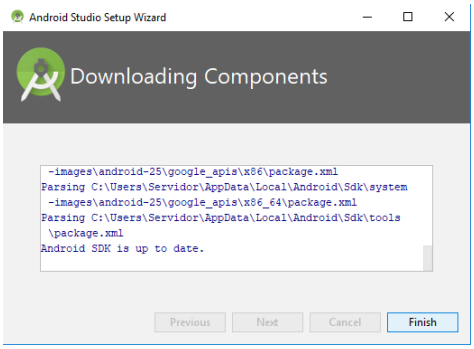
p. Escogemos “IntelliJ”.



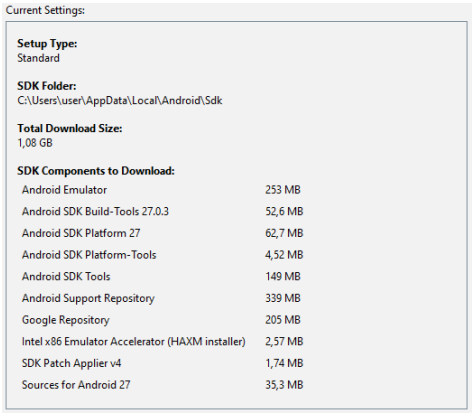
q. En seguida se pueden bajar algunos componentes adicionales.



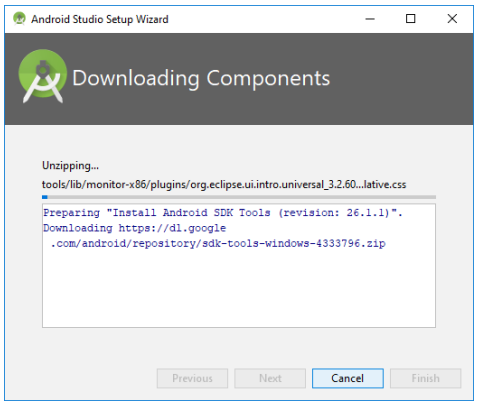
r. Se terminan de bajar y nos indica el SDK ha sido actualizado.



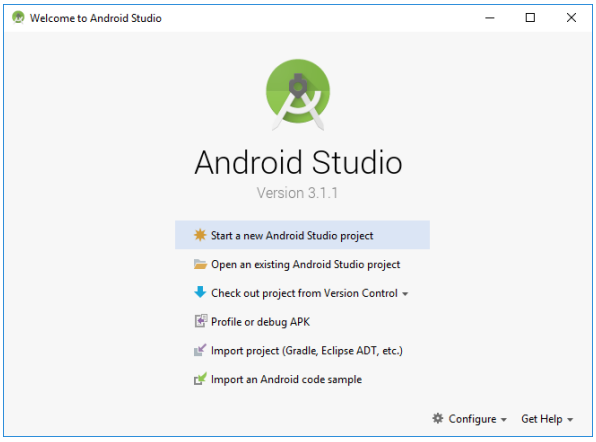
s. Ahora se muestra el resumen de las opciones escogidas. Damos “Finish”



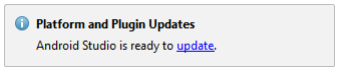
t. Baja otros archivos adicionales de acuerdo a nuestra selección.



u. Nos muestra ahora la pantalla principal de Android Studio.



3. La actualización de Android Studio puede invocarse cuando arrancamos nuestra IDE y/o abrimos un proyecto en la parte inferior derecha nos aparece un mensaje siguiente.



a. Antes de actualizar debemos de prever el tiempo puede ser considerable.
b. Podemos optar por que nos avise luego. Abrimos “Updates” y escogemos “Remind Me Later”.

4. Ejercicio finalizado ☺.

Bibliografía Preliminar

[1] Fundación Wikimedia, Inc, «Wikipedia,» 24 Octuble 2017. [En línea]. Available: https://es.wikipedia.org/wiki/Entorno_de_desarrollo_integrado. [Último acceso: 9 Noviembre 2017].

[2] Google, «Android Studio,» [En línea]. Available: https://developer.android.com/studio/index.html#Requirements. [Último acceso: 01 2018].

No. 15. Creación y estructura de un proyecto Android.

Objetivo

Crear una primera aplicación en entorno visual. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Android, permitiendo que el usuario cree una primera aplicación Android.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 5: Programación de dispositivos móviles. 5.1. Introducción a las tecnologías y herramientas móviles. 5.2 Clasificación y aplicaciones de los dispositivos móviles. 5.3 Entorno operativo de las aplicaciones móviles. 5.4 Desarrollo de aplicaciones móviles. 5.5. Aspectos de seguridad.	**No. 14. Instalar Interface Gráfica de Usuario Android.** Objetivo. Instalar la herramienta Interface Gráfica de Android. **No. 15. Creación y estructura de un proyecto Android.** Objetivo. Crear una primera aplicación Android. **No. 16. Creación y manejo de actividades en un proyecto Android.** Objetivo. Crear una aplicación Android con varias actividades. **No. 17. Uso de unidades y layouts.** Objetivo. Conocer elementos comunes en una interface gráfica.

Introducción

Una aplicación Android contiene pantallas, de manera similar en una ventana de Windows. Las pantallas en Android se llaman “Activities” o actividades. Las pantallas contienen “Layouts” o disposición y “widget” o controles. Los controles activan cierta lógica llamada “actions” o acciones. Los controles tienen propiedades como text, width, heigth. [1]

Una App o aplicación de Android consiste en código de Java, algunos documentos XML y otra información. [2]

Un “layout” define un grupo de objetos y su disposición en la pantalla. Un “layout” está hecho de definiciones en un archivo XML. Cada definición se emplea para crear un objeto a mostrar en la pantalla, como un botón o un texto. [3]

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Android Studio instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para buscar información sobre las aplicaciones Android.
4. Investigar ejemplos básicos en Android.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas

1. Realizar una investigación documental de la estructura de una “App Android”.
2. Seleccionar a varios participantes para exponer ejemplos de los componentes de una App Android.
3. Seleccionar a voluntarios para explicar el uso de Android Studio.
4. Llenar el recurso de Wiki con las definiciones de “Activities”, “Intents”, “Layouts”, “widgets”, “XML”.

Reporte de los alumnos (resultados)

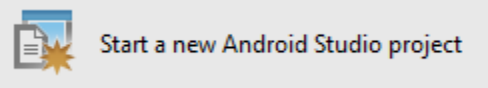
1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo los componentes básicos.
3. Participar en Moodle con el glosario de términos incorporando Android, Kernel, Linux, SDK, nombres de versiones Android, IDE, Apps, Dispositivos,

Procedimiento:

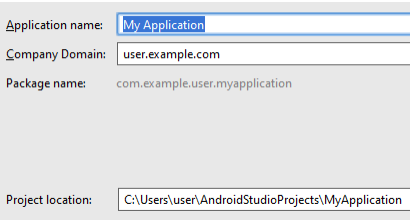
1. El tipo de problema a resolver se propone lo siguiente
   "Crear una primera aplicación Android mostrando el texto 'Hola Mundo' ".

   El algoritmo de solución aquí propuesto es: "Hacer una interface gráfica a través de Android Studio donde crea el texto se construye y ejecuta la aplicación".

2. Iniciar Android Studio. Al iniciar el programa se muestra una pantalla de inicio donde se selecciona "Start a new Android Studio Project".



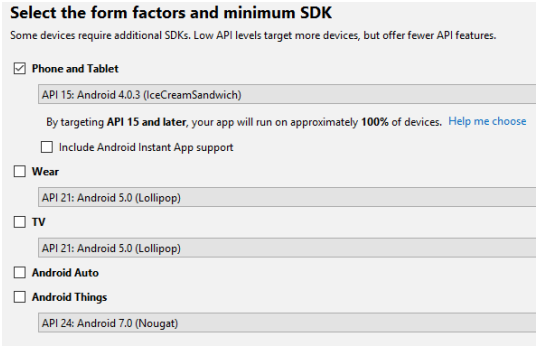
3. Una vez dado el inicio de un proyecto de Android se nos pide información inicial para nuestro proyecto.



   a. El nombre de nuestro proyecto en Android. En "Application name" escribiremos "Mi Primer Proyecto".
   b. El siguiente punto escribimos un dominio de nuestro lugar de trabajo y si no podemos inventar uno o dejar el preexistente. En éste caso pondremos "ith.mx".
   c. Y en "Project location" define el nombre del subdirectorio (de trabajos Android) en donde se crearán los archivos necesarios para el proyecto. Se crea en ese subdirectorio uno nuevo como subdirectorio del proyecto (puede ser el nombre del inciso a sin espacios). Damos "Next".

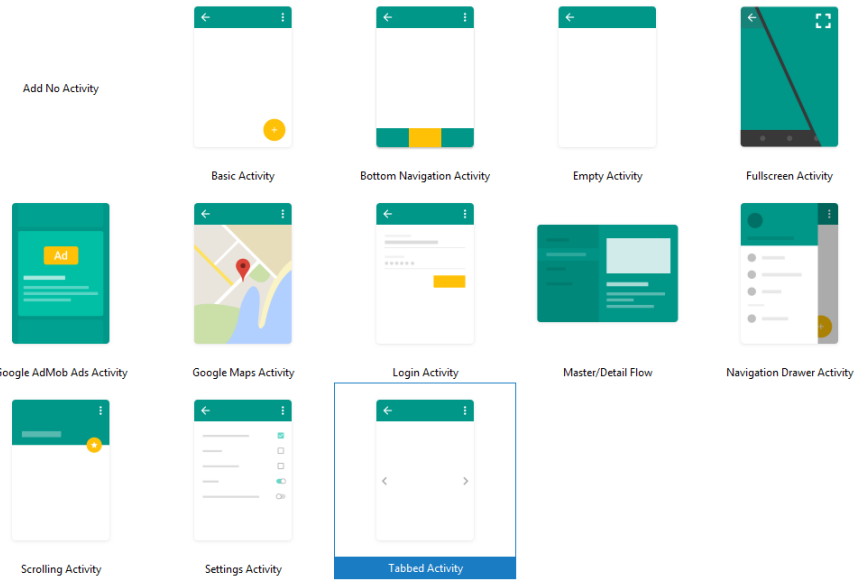
Ejemplo: **C:/Sabatico/AndrPracticas/***MiPrimerProyecto*

4. Enseguida nos pide el SDK mínimo a usar.



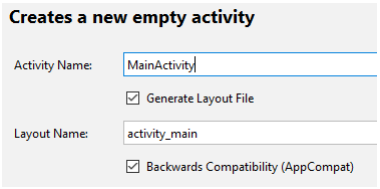
   a. Nos muestra el mercado disponible es 100%. Esto es, de todos los teléfonos existentes nuestra aplicación se va a poder instalar en la mayoría. Cuando damos "Help me choose" nos muestra una tabla en donde el compromiso va a estar en abarcar mayor mercado pero nuestra aplicación va a tener menos funcionalidad y viceversa.
   b. Dejaremos usar el mínimo SDK como "IceCreamSandwich".
   c. Seleccionamos "Next".

5. Nos presenta un menú para seleccionar el tipo de actividad a usar.



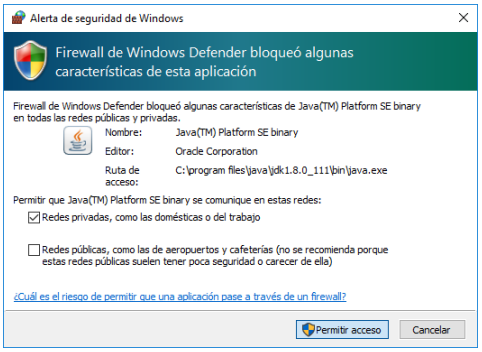
   a. Damos "Next". Es decir, aceptamos la opción default "Empty Activity".

6. Enseguida nos muestra las opciones para nuestra pantalla.

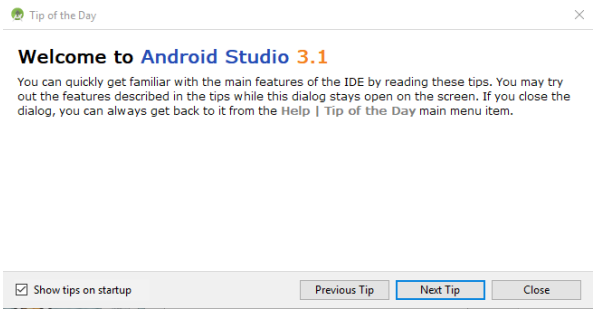


   a. Seleccionamos “Finish”. Significa: aceptar los valores por default.
   b. Si seleccionamos otra actividad (pantalla), damos “previous” para corregir nuestra selección.

7. El firewall de Windows nos puede prevenir de un acceso a redes privadas de Java. Damos “Permitir acceso”.



8. Ahora se nos presenta la interface de Android Studio con el “Tip of the Day” o consejo del día. Éste pequeño aviso nos puede ser útil para el manejo de Android Studio. Lo cerramos.

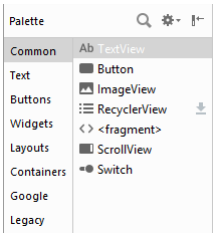


9. Nos aparece la interface con dos archivos uno llamado “`MainActivity.java`” con el siguiente contenido. El contenido es muy similar al siguiente.

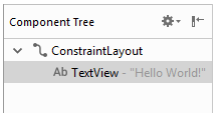
```
package mx.ith.miprimerproyecto;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
```

```
public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
    }
}
```

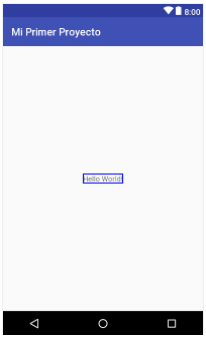
10. Damos un clic en la pestaña "`activity_main.xml`". Y nos muestra varia áreas.
    a. Un área donde están algunos componentes a escoger. Por default nos enseña los componentes comunes.



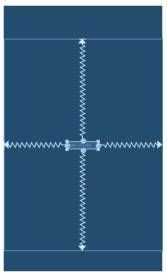
    b. Otra área se encuentra el árbol de los componentes agregados.



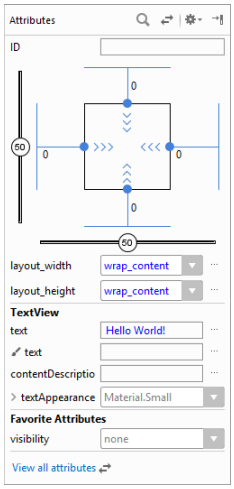
    c. Un área más nos muestra cómo se visualizarán los componentes agregados.



d. Hay otra área muy parecida a la anterior y nos muestra donde un componente puede ser desplazado.



e. Existe un área adicional donde un componente puede ser modificado a través de sus atributos.



f. Cambiaremos el texto "`Hello World!`" por "`Hola Mundo!`".

g. En la parte inferior hay dos pestañas 

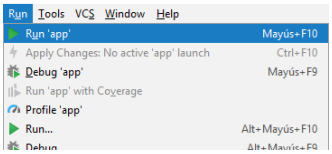
. Podemos cambiar ya sea en diseño o en texto "**`Hola Mundo!`**" por cualquier otro texto y generaremos el mismo resultado. Esto es tanto en diseño como en texto se modifica el mismo archivo "`activity_main.xml`".

h. Si escogemos "Text" obtenemos el archivo "`activity_main.xml`" generado.

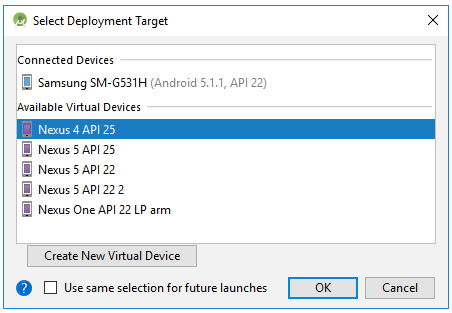
```
<?xml version="1.0" encoding="utf-8"?>
<android.support.constraint.ConstraintLayout
xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    tools:context=".MainActivity">
```

```
    <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="Hola Mundo!"
        app:layout_constraintBottom_toBottomOf="parent"
        app:layout_constraintLeft_toLeftOf="parent"
        app:layout_constraintRight_toRightOf="parent"
        app:layout_constraintTop_toTopOf="parent" />

</android.support.constraint.ConstraintLayout>
```

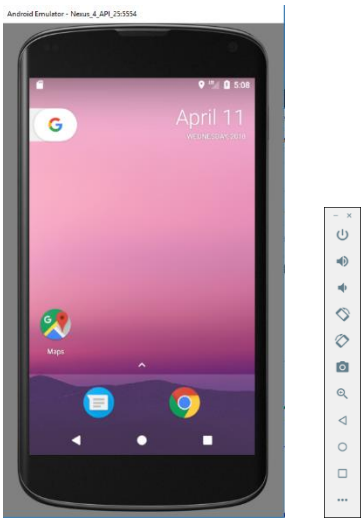
11. Ahora corremos el programa. En la parte superior seleccionamos “Run” y de nuevo “Run” o con una combinación de teclas sería <Mayus><F10>



12. Al correr nos indica si deseamos correrlo en un emulador (Usamos dispositivos Nexus creamos un ‘Create New Virtual Device’ con ‘Nougat’ u ‘Oreo’ y nos puede llevar tiempo la descarga de la imagen) o en un dispositivo (Se usó celular Samsung en modo debug conectado a una USB y debemos dar ‘OK’ en el celular en el diálogo ‘Allow USB Debugging’).



13. Escogeremos el primer dispositivo creado y escogemos “OK”. El dispositivo hace la emulación de un teléfono. Una vez “prendido” se carga en él la aplicación.



14. Y nos muestra nuestro “Hola Mundo”.

15. Ejercicio finalizado ☺.

Bibliografía Preliminar

[1] M. L. Murphy, The Busy Coder's Guide to Android Development, CommonsWare, LLC, 2008.

[2] B. Burd, Java® Programming for Android™ Developers For Dummies®, Hoboken, New Jersey: John Wiley & Sons, Inc., 2014.

[3] B. Phillips y B. Hardy, Android Programming: The Big Nerd Ranch Guide, Indianapolis, IN 46240 USA: Pearson Technology Group, 2013.

No. 16. Creación y manejo de actividades en un proyecto Android.

Objetivo

Crear una primera aplicación en entorno visual. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Android, permitiendo que el usuario cree una aplicación Android con dos "Activities".

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 5: Programación de dispositivos móviles. 5.1. Introducción a las tecnologías y herramientas móviles. 5.2 Clasificación y aplicaciones de los dispositivos móviles. 5.3 Entorno operativo de las aplicaciones móviles. 5.4 Desarrollo de aplicaciones móviles. 5.5. Aspectos de seguridad.	**No. 14. Instalar Interface Gráfica de Usuario Android.** Objetivo. Instalar la herramienta Interface Gráfica de Android. **No. 15. Creación y estructura de un proyecto Android.** Objetivo. Crear una primera aplicación Android. **No. 16. Creación y manejo de actividades en un proyecto Android.** Objetivo. Crear una aplicación Android con varias actividades. **No. 17. Uso de unidades y layouts.** Objetivo. Conocer elementos comunes en una interface gráfica.

Introducción

Al crear una aplicación Android se llevan a cabo en varios pasos: (1)

1. Instalación de Android Studio y su configuración.
2. Construir una aplicación (app) más elaborada.
3. Correr la App en el emulador.
4. Cambiar la app con unos cambios adicionales del paso 2 y correrla de nuevo.

Los controles comunes de Android son: (2)

1. Los controles de Texto:
    a. TextView. Muestra texto, no se puede editar.
    b. EditText. Muestra texto y se puede modificar.
    c. AutoCompleteTextView. Texto editable con sugerencias.
    d. MultiAutoCompleteTextView. Texto editable con sugerencias para varias palabras.
2. Los controles de botón.
    a. Button. Se programa un evento con código de Java.
    b. ImageButton. Un botón (button) con imagen.
    c. ToggleButton. Un botón con dos estados prendido o apagado.
    d. CheckBox. Es como un ToggleButon pero se muestra con un "palomita".
3. Los controles ImageView. Es un área para mostrar una imagen.
4. Los controles de Date y Time. Muestran la fecha y la hora.
5. El control MapView. Junto con los servicios de "Google Play services" sirve para mostrar un mapa.

Las intenciones (Intents) encapsulan una solicitud, hecha a Android, para que alguna actividad u otro receptor haga algo. (3)

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Android Studio instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:

Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para buscar información sobre los controles comunes en las aplicaciones Android.

4. Investigar ejemplos básicos en Android usando controles comunes.
5. Seguir el procedimiento de la práctica.
6. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas.

1. Realizar una investigación documental de controles comunes en una “App Android”.
2. Seleccionar a varios participantes para exponer ejemplos de los controles comunes.
3. Seleccionar a voluntarios para explicar el uso de controles comunes.
4. Llenar el recurso de Wiki con las funciones de “Text”, “Button”, “Activities”, “Button”, “ImageView”.

Reporte de los alumnos (resultados).

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo los controles comunes.
3. Participar en Moodle con el glosario de términos incorporando: controles, eventos, el objeto “R”, funciones más importantes de “Android Software Stack”, Servicios, “Intents”.

Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   “Crear una primera aplicación con una ventana principal e invoque a otra ventana”.

   El algoritmo de solución aquí propuesto es: “Hacer una interface gráfica a través de Android Studio donde crea una “activity” principal con un control se invoca a otra “activity” secundaria en donde se encuentra otro control para regresar a la pantalla principal”.

2. Iniciar Android Studio. Al iniciar el programa se muestra una pantalla de inicio donde se selecciona “Start a new Android Studio Project” o en su defecto en el menú seleccionar: “File”, “New”, “New Project”.

3. Una vez dado el inicio de un proyecto de Android se nos pide información inicial para nuestro proyecto.

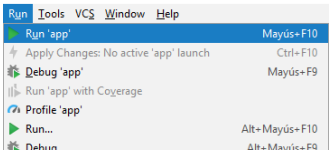
   a. El nombre de nuestro proyecto en Android. En “Application name” escribiremos “App16Intents”.
   b. El siguiente punto escribimos un dominio de nuestro lugar de trabajo y si no estamos liberando nuestra app no es importante y podemos inventar uno o dejar el preexistente. En éste caso pondremos “ith.mx”.
   c. Y en “Project location” define el nombre del subdirectorio (de trabajos Android) en donde se crearán los archivos necesarios para el proyecto. Se crea en ese subdirectorio uno nuevo como subdirectorio del proyecto (puede ser el nombre del inciso a sin espacios). Escribimos **C:/Sabatico/AndrPracticas/***App16Intents* y damos “Next”.

4. Enseguida nos pide el SDK mínimo a usar.
   a. Por el momento no escogemos otro que no sea “Phone and Tablet”.
   b. Dejaremos usar el mínimo SDK como API 15: “IceCreamSandwich”.
   c. Seleccionamos “Next”.

5. Nos presenta un menú para seleccionar el tipo de actividad a usar.
   a. Damos “Next”. Es decir, aceptamos la opción default “Empty Activity”. Nos genera menos código y nos facilita agregar código.

6. Enseguida nos muestra las opciones para nuestra pantalla.
   a. Seleccionamos “Finish”. Significa: aceptar los valores por default. No nos preocupemos por los nombres, ya que los podemos cambiar después.
   b. Ahora Android Studio hace su trabajo.
   c. En el editor aparecen disponibles “`MainActivity`.java” y “`activity_main`.xml”.

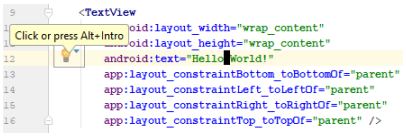
7. Si corremos el programa. En la parte superior seleccionamos “Run” y de nuevo “Run” o con una combinación de teclas sería <Mayus><F10>



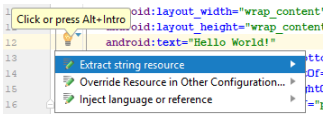
8. Y nos muestra el título la aplicación con título “App16Intents” y con el texto “Hello World!”.

9. Vamos a modificar "activity_main.xml". Al texto creado por default le asociaremos a un identificador constante.
   a. Seleccionamos "activity_main.xml" y lo abrimos en texto.

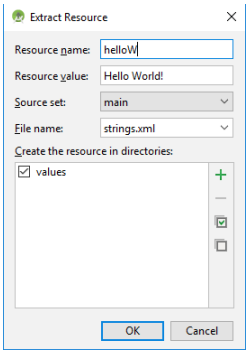
b. En la línea donde está "`android:text="Hello World!"`" damos clic sobre cualquier parte del texto `"Hello World!"`



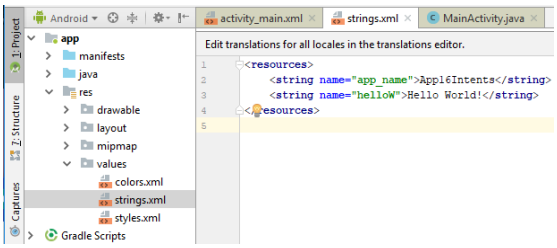
c. Se nos presenta un foco con la invitación a abrirlo. Lo abrimos con cualquiera de las opciones presentadas “clic” o “<Alt><Enter>”.



d. Seleccionamos la primer opción “Extract string resource”. Usamos el alias `"helloW"`



e. Estas constantes junto con otros recursos se guardan en varios archivos en un apartado “res” de recursos y en él está “values” que a su vez contiene “strings.xml” donde se almacenan los valores.

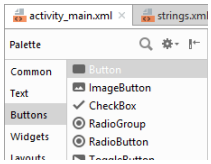


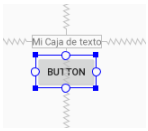
f. Y en archivo fuente "activity_main.xml" queda modificada la línea "`android:text="Hello World!"`" cambia a "`android:text="@string/hellow"`"

10. Modificar el texto por default y personalizarlo.
    a. En el archivo fuente "activity_main.xml" en la línea: "`android:text="@string/hellow"`"

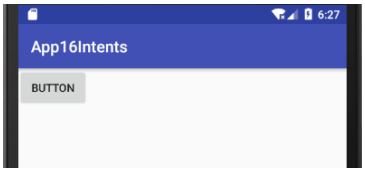
La cambiamos a:
"`android:text="Mi Caja de texto"`"
Hacemos un procedimiento similar al punto 9. Para que nuestro texto quede: "`android:text="@string/mi_caja_de_texto"`".

11. Como ya la constante previa "`hellow`" ya no es necesaria. Es opcional borrar la siguiente línea del archivo "strings.xml":
    "`<string name="helloW">Hello World!</string>`"
12. Agregar un botón. Observe el texto del archivo "activity_main.xml" al principio y al terminar los incisos c y d.
    a. En diseño en "activity_main.xml" de la paleta de componentes en "Buttons" escogemos "Button" y lo arrastramos debajo de "Mi Caja de Texto".

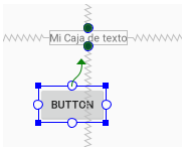




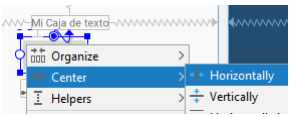
    b. Si corremos nuestro programa nos muestra el botón en la posición (0,0) esto es aparece en la esquina superior izquierda. Y no en la posición aplicada en diseño.



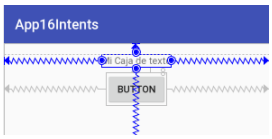
    c. Para que el botón se muestra debajo de nuestro texto "Mi Caja de Texto" debemos editar el botón. En diseño lo seleccionamos y conectamos la parte superior con la parte de debajo de nuestro texto.



    d. Luego vamos a centrarlo horizontalmente. Con el botón seleccionado dando un clic derecho sobre el ratón le indicamos el centrado.



e. Ahora llevamos el texto hacia la parte superior justo debajo del título de nuestro proyecto.



f. Podemos observar: Cada vez, al realizar un cambio en diseño, se refleja los cambios en el archivo "activity_main.xml". Lo corremos y ahora aparece el botón aparece centrado y debajo de nuestro texto y el texto en la parte superior. Así obtenemos el siguiente resultado.

13. Cada elemento agregado tiene mayor cantidad de propiedades que las mostradas por default. Una vez seleccionado un elemento podemos ver sus propiedades. Para ver mayor o menor cantidad de propiedades hacemos lo siguiente:

    En la parte derecha abajo aparece un letrero "View all attributes". Lo presionamos y nos muestra todos los atributos. Para regresarnos posiblemente usemos la barra de deslizamiento (scroll) para ver en la parte inferior "View fewer attributes".

    Nota. Al cambiar el valor de un atributo su cambio se refleja cuando damos <Enter>. O también cuando seleccionamos otra propiedad.

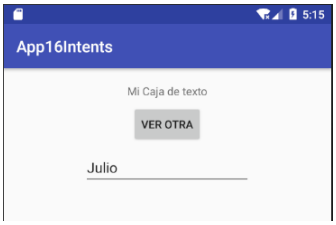
14. Es recomendable asociar un elemento con un nombre representativo. Cada elemento tiene una propiedad de identificación denominada "ID". Cada elemento agregado tiene un nombre por default. Para que sea útil para el programador se le asigna un nombre ligado a la función del elemento.
    Nota. Hay ocasiones, al cambiar id, se nos muestre un letrero informado actualizar las referencias en el campo "Java R" le indicamos "Yes".

15. Cambiaremos el "ID" del botón. La función prevista para el botón es "Cuando se le dé clic va a mostrar la otra actividad y le va a enviar información".
    a. Nos aseguramos que esté seleccionado el botón. En diseño en "activity_main" damos clic en "BUTTON".
    b. En propiedades su "ID" tiene asignada el valor por default "button". Lo cambiamos a "btnVerOtra".
    c. En propiedad text dice "Button" lo cambiamos a "VerOtra".

16. Agregamos un "Plain Text". Va estar centrado horizontalmente y pegado a la parte superior similar al punto 12-e.

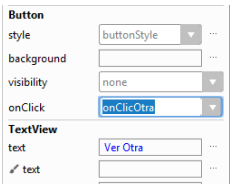
a. Seleccionamos el TextView. Observe el nombre desplegado por default dice “Name”. Lo cambiamos a “Julio”.
b. Modificamos el atributo “id”. Por default muestra “editText”. Lo cambiamos a “txtNombre”.
c. Cambiamos la propiedad “text” en lugar de “Name” escribimos “Nombre”.

17. Corremos nuestro programa. Nos permite escribir y modificar “Julio”. Y el botón “VER OTRA” no hace ninguna función todavía.



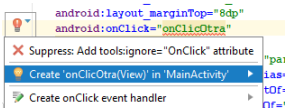
18. Programaremos el botón “VER OTRA”. Su función servirá para mostrar otra ventana y enviarle información.

a. Seleccionamos en diseño en “activity_main.xml” el botón “VER OTRA”.
b. Cambiamos la propiedad propiedad OnClick por default “” por el valor “onClicOtra”.



c. Sin embargo “onClicOtra” no está definido. Abrimos “activity_main.xml” en modo texto. Observar que del lado derecho aparece una marca en forma de línea roja y del lado izquierdo aparece un foco rojo.

d. En el foco rojo le indicamos generar el evento. (En caso de no aparecer el foco rojo dar un clic en la marca roja).



e. Una vez indicado la creación del método abrimos "MainActivity.java". Ya está generado el método "`onClicOtra`".

```
package mx.ith.app16intents;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class MainActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_otra);
    }

    public void onClicOtra (View view) {
    }
}
```

f. En la clase agregamos el atributo "txtMiNombre" el cual hace referencia a "txtNombre". Lo inicializamos en "`onCreate`". Para definir una librería como "`EditText`" nos pide que presionemos <Alt><Enter>.

```
    android.widget.EditText txtMiNombre;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);
        txtMiNombre = findViewById(R.id.txtNombre);
    }
```

g. Agregamos el código siguiente a "`onClicOtra`". Todavía no existe "`OtraActivity.class`" y por lo tanto nos marca un error.

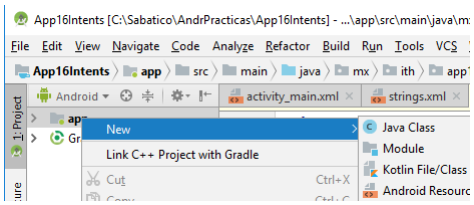
```
public void (android.view.View view) {
                                          //No existe OtraActivity
    Intent intent = new Intent(MainActivity.this, OtraActivity.class);
    Bundle bundle= new Bundle();
    bundle.putString("miNombre", txtMiNombre.getText().toString());
    intent.putExtras(bundle);
    startActivityForResult(intent, 1);
}
```

h. En la clase vamos a agregar un método "`onActivityResult`". Nos sirve cuando la otra actividad regresa a la actividad principal. En la clase con el botón derecho seleccionamos "Generate" luego "Override Methods" y vamos hasta el bloque "android.support.v4.pp.FragmentActivity" y seleccionamos el método. Y escribimos dentro de él lo siguiente.

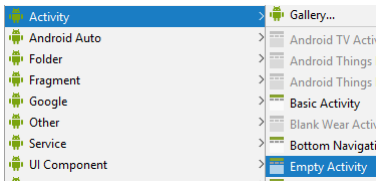
```
@Override
protected void onActivityResult(int requestCode, int resultCode, Intent
data) {
    super.onActivityResult(requestCode, resultCode, data);
    if (requestCode == 1) {
        if(resultCode == android.app.Activity.RESULT_OK){
```

```
                String s = data.getStringExtra("miNombre");
                android.util.Log.e("X", "regreso "+ s );
                txtMiNombre.setText(s);
            }else
            if (resultCode == android.app.Activity.RESULT_CANCELED) {
                android.widget.Toast.makeText(this, "Cancelado",
                android.widget.Toast.LENGTH_LONG).show();
            }
        }
    }
```

19. Agregamos una nueva actividad a nuestro proyecto.
    a. En nuestro proyecto con el botón derecho seleccionamos “New”.

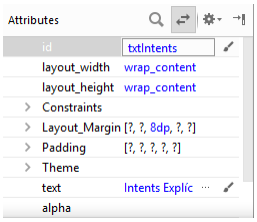


    b. Luego buscamos “Activity” y “Empty activity”.

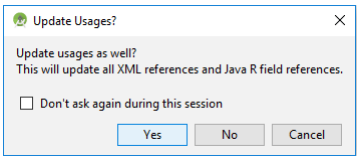


    c. Nos presenta un menú con nombre de la actividad por default “Main2Activity” lo cambiamos con nombre “OtraActivity”. Y dejamos los valores por default. Vemos el nombre del layout como “activity_otra” y seleccionado “Backwards Compatibility (AppCompat)”. Escogemos “Finish”.

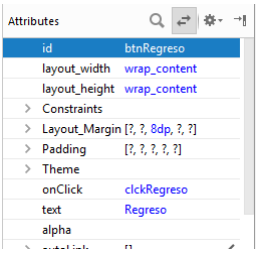
20. En la nueva actividad agregamos un TextView. Va estar centrado horizontalmente y pegado a la parte superior similar al punto 12-e. En los pasos siguientes cambiaremos sus propiedades texto con “Intents explícitos” y su id con “txtIntents”.
    a. Seleccionamos el TextView.
    b. Modificamos apropiadamente los atributos id y text.



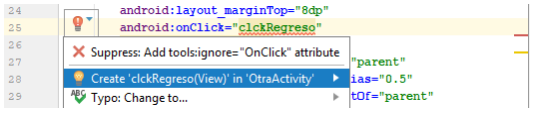
c. Al cambiar id nos muestra un letrero informado actualizar las referencias en el campo “Java R” le indicamos “Yes”.



21. Agregamos un botón. Lo colocamos centrado horizontalmente debajo del texto “Intents Explícitos” similar los pasos 12-c y 12-d.
    a. Cambiamos la propiedad text por default “Button” por el valor “Regreso”. Y la propiedad OnClick por default “” por el valor “clckRegreso”. Y id por “btnRegreso”.



    b. Sin embargo “clckRegreso” no está definido. Abrimos “activity_otra.xml” en modo texto. Observar que del lado derecho aparece una marca en forma de línea roja y del lado izquierdo aparece un foco rojo. En el foco rojo le indicamos generar el evento. (En caso de no aparecer el foco rojo dar un clic la marca roja).



    c. Una vez indicado la creación del método abrimos “OtraActivity.java”. Ya está generado el método “`clckRegreso`”.

```
package mx.ith.app16intents;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.view.View;

public class OtraActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_otra);
    }

    public void clckRegreso(View view) {
    }
}
```

d. En la clase agregamos el atributo "*miTexto*" y lo inicializamos.

```
EditText miTexto;

@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_otra);
    miTexto = findViewById(R.id.txtNombre);
}
```

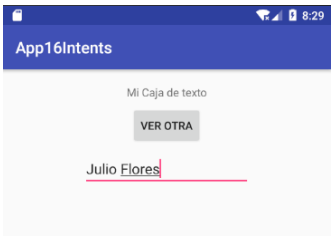
e. Ahora programamos el botón de regreso.

```
public void clckRegreso(View view) {
    Intent intent = new Intent();
    intent.putExtra("miNombre", miTexto.getText().toString());
    setResult( android.app.Activity.RESULT_OK, intent);
    finish();
}
```

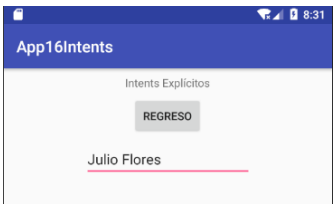
f. Si ahora corremos el programa resulta en la falta de información actualizada de nuestra caja de texto "`miTexto`". Para resolverlo reprogramamos el método "`onResume`" de la siguiente forma.

```
@Override
public void onResume() {
    super.onResume();

    String s= getIntent().getExtras().getString("miNombre");
    miTexto.setText(s);
}
```

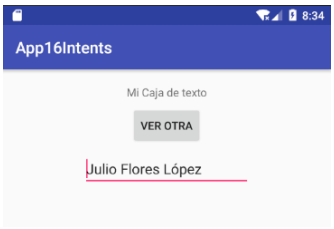
22. Ahora ya podemos enviar y recibir información de una caja de texto de una actividad a otra actividad.
    a. Capturamos un nombre en la pantalla principal.



    b. Éste nombre se recibe en la otra pantalla.



c. Si lo modificamos de nuevo y nos regresamos. Obtenemos en principal los cambios.



23. Ejercicio finalizado ☺.

Bibliografía Preliminar

1. **Griffiths, Dawn y Griffiths, David.** *Head First Android Development.* 1005 Gravenstein Highway North, Sebastopol, CA 95472 : O'Reilly, 2015.
2. **MacLean, Dave, Komatineni, Satya y Allen, Grant.** *Pro Android 5.* New York : Apress, 2015.
3. **Murphy, Mark L.** *The Busy Coder's Guide to Android Development.* United States of America : CommonsWare, LLC., © 2008-2017.

No. 17. Uso de unidades y layouts.

Objetivo

Crear una primera aplicación en entorno visual. Analizar y comprender el funcionamiento de una Interface Gráfica de Usuario con Android, permitiendo que el usuario cree una aplicación Android para programar elementos tipo “TabLayout” y “ListView”.

TEMARIO

TEMA	NÚM. PRÁCTICA
UNIDAD 5: Programación de dispositivos móviles. 5.1. Introducción a las tecnologías y herramientas móviles. 5.2 Clasificación y aplicaciones de los dispositivos móviles. 5.3 Entorno operativo de las aplicaciones móviles. 5.4 Desarrollo de aplicaciones móviles. 5.5. Aspectos de seguridad.	**No. 14. Instalar Interface Gráfica de Usuario Android.** Objetivo. Instalar la herramienta Interface Gráfica de Android. **No. 15. Creación y estructura de un proyecto Android.** Objetivo. Crear una primera aplicación Android. **No. 16. Creación y manejo de actividades en un proyecto Android.** Objetivo. Crear una aplicación Android con varias actividades. **No. 17. Uso de unidades y layouts.** Objetivo. Conocer elementos comunes en una interface gráfica.

Introducción

Cada instancia de Activity tiene un ciclo de vida. Durante este ciclo de vida, una actividad transita entre varios estados: al crearse, en ejecución, en pausa y detenido. Para cada transición, existen métodos para notificar la actividad del cambio en su estado. Estos métodos también denominados "callbacks" son: `onCreate()`, `onStart()`, `onResume()`, `onPause()`, `onStop()`, y `onDestroy()`. (1)

Los controles de lista se utilizan para mostrar colecciones de datos. Pero en lugar de usar un solo tipo de control para administrar tanto la pantalla como los datos, Android separa estas dos responsabilidades en controles de lista y adaptadores. Los controles de lista son: ListView, GridView, Spinner y Gallery. (2)

Un adaptador es una instancia de una clase que implementa la interfaz "Adapter". Ejemplo: Si vamos a utilizar una instancia de ArrayAdapter <T>, es un adaptador que sabe cómo trabajar con datos en una matriz (o una ArrayList) de objetos de tipo T. (3)

Los fragmentos (Fragments) de diseño son una forma poderosa de reutilizar Actividades simplemente recargando una porción de la interface gráfica en la Actividad cargada actualmente. (4)

Material y equipo necesario.

El material necesario para la práctica es el siguiente:

1. Tener una computadora con Android Studio instalado.
2. Tener acceso a Internet.
3. Este manual de prácticas.
4. Utilizar hojas y lápiz.
5. Tener acceso a los recursos del Moodle.

Metodología:
Las prácticas están pensadas para emplear los algoritmos de la materia Tópicos Avanzado de Programación en una computadora y hacer programas tipo.

1. Trabajo individual.
2. Analizar el problema tipo propuesto. Proponer un algoritmo de solución.
3. Conectarse a Internet para obtener la información para buscar información sobre el ciclo de vida de las "Activities", "List Views" y "Adapters".
4. Obtener ejemplos básicos en Android usando configurando los "Callbacks".
5. Obtener ejemplos básicos en Android usando "List Views".
6. Seguir el procedimiento de la práctica.
7. Probar que su aplicación este corriendo y otorgando resultados correctos.

Sugerencias didácticas.

1. Realizar una investigación documental de controles comunes en una "App Android" empleando "List Views" o "RecyclerViews" con "Fragments".
2. Seleccionar a varios participantes para exponer ejemplos de los controles comunes en "List Views".
3. Seleccionar a voluntarios para explicar el ciclo de vida de "Activities".
4. Seleccionar a voluntarios para explicar "Fragments".
5. Seleccionar a voluntarios para explicar "Toast".
6. Llenar el recurso de Wiki con las funciones de "RecyclerView", "List Views", de "Adapters", de los "Callbacks" en Activities.

Reporte de los alumnos (resultados).

1. Hacer el programa con el ejemplo de la práctica. Imprimir las pantallas, así como su programa fuente. Entregar con su nombre, materia, fecha, nombre del maestro.
2. Crear una aplicación por el alumno para usar una ejemplo inventada por el alumno incluyendo "RecyclerViews".
3. Participar en Moodle con el glosario de términos incorporando: "RecyclerView", "Activities", "Adapter", "Callback", "Layout", "Widget", "Fragment", "Toast".

Procedimiento:

1. El tipo de problema a resolver se propone lo siguiente
   "Crear una primera aplicación con una ventana principal con una lista de datos como elemento base para mostrar un diálogo para cada elemento de la lista".

   El algoritmo de solución aquí propuesto es: "Hacer una interface gráfica donde crea una pantalla principal con elementos responsivos a eventos del ratón y programados para mostrar diálogos informativos". Se escogió hacerlo con una lista de libros.

2. Iniciar Android Studio. Al iniciar el programa se muestra una pantalla de inicio donde se selecciona "Start a new Android Studio Project" o en su defecto en el menú seleccionar: "File", "New", "New Project".

3. Una vez dado el inicio de un proyecto de Android se nos pide información inicial para nuestro proyecto. El nombre de nuestro proyecto escribiremos "App17Fragment".

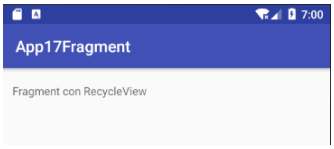
4. Enseguida nos pide el SDK mínimo a usar.
   a. Por el momento no escogemos otro que no sea "Phone and Tablet".
   b. Dejaremos usar el mínimo SDK como API 15: "IceCreamSandwich".

c. Seleccionamos “Next”.

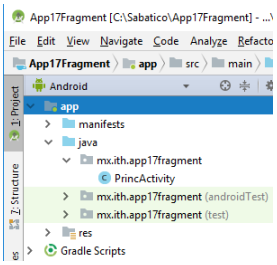
5. Nos presenta un menú para seleccionar el tipo de actividad a usar.
   a. Damos “Next”. Es decir, aceptamos la opción default “Empty Activity”. Nos genera menos código y nos facilita agregar código.

6. Enseguida nos muestra las opciones para nuestra pantalla.
   a. En “Activity Main” escribimos “PrincActivity” y seleccionamos “Finish”.
   b. Ahora Android Studio hace su trabajo.
   c. En el editor aparecen disponibles “`PrincActivity`.java” y “`activity_princ`.xml”.

7. Vamos a modificar "activity_princ.xml".
   a. Nos aseguramos estar en diseño (Design).
   b. El texto “TextView” indicando “Hello Wold!” lo moveremos al tope y a la izquierda y lo cambiamos a “Fragment con RecycleView”. Lo corremos:



8. Agregaremos un “fragment” con “RecycleView” a nuestro proyecto.
   a. Nos aseguramos seleccionar nuestro proyecto.



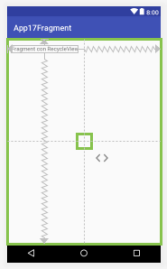
   b. Seleccionamos “File” -> “New” -> “Fragment” -> Fragment (List).

   c. En ese momento nos indica el siguiente menú del cual en “Object Kind” escribimos “Libro”. Y las demás opciones las dejamos por default. Y damos finish.

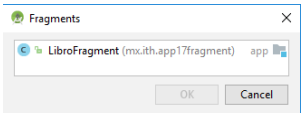
   d. Observamos que se ha generado varias clases adicionales a la clase “PrincActivity”. Estas son: “DummyContent”, “LibroFragment” y “MyLibroRecyclerViewAdapter.

9. Con el propósito de ser congruente con nuestro libro. Renombraremos los paquetes y los nombres de las clases.
    a. Nos posicionamos en el paquete “dummy” y con el botón derecho seleccionamos “Refactor” -> “Rename” o <Shift><F6>. En la ventana de diálogo escribimos “libro”.
    b. Con la clase “DummyContent” la modificamos con “refactor” a “GeneraLibros”. Nos aseguramos que la palabra esté seleccionada.
    c. Abrimos en el editor la clase GeneraLibros y al fondo renombramos usando refactor la clase “DummyItem” por “Libro”.
    d. Cambiamos de la misma forma “createDummyItem” por “crearLibro”. Seleccionada la palabra “createDummyItem” y luego <Shift><F6>.

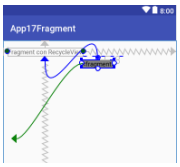
10. Al correr de nuevo nuestro programa no encontramos nada nuevo. Necesitamos incorporar el fragmento a nuestra actividad “PrincActivity”. Vamos a modificar "activity_princ.xml".
    a. En diseño abrimos “activity_princ.xml”.
    b. De la paleta escogemos “Containers” y luego “<fragment>”. Lo arrastramos y lo ubicamos en el centro.



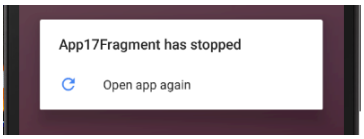
    c. En ese momento nos muestra un diálogo para escoger el fragmento que hemos creado “LibroFragment”.



    d. Lo seleccionamos y damos “OK”. Ahora ubicamos nuestro fragment abajo del texto y alineado a la parte inferior del texto y alineado a la izquierda.



11. Si corremos nuestro programa no funciona.
    a. Nos muestra un error en el emulador.



    b. Y en los mensajes de "Run" nos dice:

```
E/AndroidRuntime: FATAL EXCEPTION: main
…
Caused by: java.lang.RuntimeException: mx.ith.app17fragment.PrincActivity@43f7663 must implement
OnListFragmentInteractionListener
        at mx.ith.app17fragment.LibroFragment.onAttach(LibroFragment.java:81)
…
```

    c. Lo cual nos lleva al código donde explica la actividad debe implementar: "`OnListFragmentInteractionListener`".

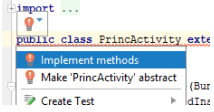
```
@Override
public void onAttach(Context context) {
    super.onAttach(context);
    if (context instanceof OnListFragmentInteractionListener) {
        mListener = (OnListFragmentInteractionListener) context;
    } else {
        throw new RuntimeException(context.toString() +
        " must implement OnListFragmentInteractionListener");
    }
}
```

12. Implementado "`OnListFragmentInteractionListener`".

    a. Abrimos "PrincActivity.java".

```
package mx.ith.app17fragment;

import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;

public class PrincActivity extends AppCompatActivity {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_princ);
    }
}
```

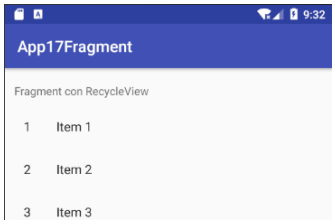
    a. La clase le indicamos que implemente la interface. Al escribir la implementación de la interface nos indica con un foco rojo las opciones para corregir el la agregación de la interface.



b. Optamos por la primer opción de implementar los métodos. Y finalmente la clase queda como sigue.

```
public class PrincActivity extends AppCompatActivity
    implements LibroFragment.OnListFragmentInteractionListener {

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_princ);
    }

    @Override
    public void onListFragmentInteraction(GeneraLibros.Libro item) {

    }
}
```

13. Al correr nuestro programa nos muestra una pantalla similar a la siguiente:



14. Pero la información no tiene nada relacionada con libros. Vamos a modificar la clase “Libros” para tener dos campos el título y el autor.
    a. Abrimos la clase “GeneraLibros”
    b. Vamos a cambiar el atributo “content” por “nombre” y “details” por “autor”. El atributo “id” lo dejamos intacto. Recordemos primero seleccionar y luego presionamos <Shift>F6 y escribimos el nuevo valor.
    c. En el constructor cambiamos los valores en forma similar al paso anterior.
    d. Creamos los getters de los tres atributos. La clase queda como sigue:

```
public static class Libro {
    final String id;
    final String nombre;
    final String autor;

    Libro(String id, String nombre, String autor) {
        this.id = id;
        this.nombre = nombre;
        this.autor = autor;
    }

    public String getId() {
        return id;
    }

    public String getNombre() {
        return nombre;
    }

    public String getAutor() {
        return autor;
    }
```

```
    @Override
    public String toString() {
        return nombre;
    }
}
```

15. Vamos a hacer alguno cambios a la clase “GeneraLibros”.
    a. Como vamos a tener una colección de libros cambiamos “ITEMS” por “LIBROS”.

```
public static final List<Libro> LIBROS = new ArrayList<>();
```

    b. Comentamos (o borramos) algunas líneas innecesarias para nuestro propósito. Se muestran en orden de aparición de arriba abajo. Éstas son:

```
//private static final Map<String, Libro> ITEM_MAP = new HashMap<>();

//private static final int COUNT = 25;
…
/* Add some sample items.
for (int i = 1; i <= COUNT; i++) {
    addItem(createDummyItem(i));
}*/
…
/*private static void addItem(Libro item) {
    ITEMS.add(item);
    //ITEM_MAP.put(item.id, item);
}

private static Libro createDummyItem(int position) {
    return new Libro(String.valueOf(position), "Item " + position,
makeDetails(position));
}*/
```

    c. Creamos los libros de muestra con las siguientes instrucciones.

```
static String[][] infoLibros = {
        {"De (casi) todo se aprende","Paula Gonu"},
        {"El Príncipe de la Niebla","Carlos Ruiz Zafón"},
        {"Detective Conan II n° 89","Meitantei Conan II"},
        {"Leiva. Toquemos juntos hasta que la muerte nos joda","Wilma
Lorenzo"},
        {"Las hijas del Capitán","María Dueñas"},
        {"Donde fuimos invencibles","María Oruña"},
        {"Patria","Fernando Aramburu Irigoyen"},
        {"Morder la manzana","Leticia Dolera"},
        {"El legado de los espías","John le Carré"},
        {"Las almas de Brandon","César Brandon Ndjocu"},
        {"El fuego invisible","Javier Sierra"},
        {"Maestros de la costura","Shine | CR TVE"},
        {"A comer se aprende","Álvaro Vargas"}};

static {
    int id=1;
    for( String[] si:infoLibros){
        Libro libro = new Libro(String.valueOf(id++), si[0], si[1]);
        LIBROS.add(libro);
    }
}
```

16. Los cambios van a afectar a las demás clases. Vamos a corregir "MyLibroRecyclerAcapter".
    a. Localizamos abajo la clase "`ViewHolder`" y cambiamos el atributo ítem por libro.

```
public class ViewHolder extends RecyclerView.ViewHolder {
    final View mView;
    final TextView mIdView;
    final TextView mContentView;
    public Libro libro;
```

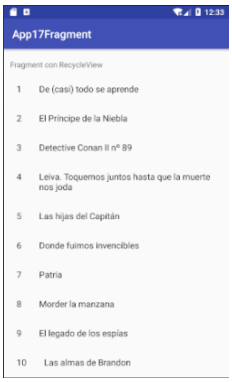
    b. Cambiamos en el método "`onBindViewHolder`" las primeras líneas por:

```
@Override
public void onBindViewHolder(@NonNull final ViewHolder holder, int
position) {
    holder.libro = mValues.get(position);
    holder.mIdView.setText(holder.libro.getId());
    holder.mContentView.setText(holder.libro.getNombre());
```

17. Otra clase a corregir es "LibroFragment". Modificamos "ITEMS" por "LIBROS".

```
@Override
public View onCreateView(@NonNull LayoutInflater inflater, ViewGroup container,
Bundle savedInstanceState) {
    View view = inflater.inflate(R.layout.fragment_libro_list, container, false);

    // Set the adapter
    if (view instanceof RecyclerView) {
        Context context = view.getContext();
        RecyclerView recyclerView = (RecyclerView) view;
        if (mColumnCount <= 1) {
            recyclerView.setLayoutManager(new LinearLayoutManager(context));
        } else {
            recyclerView.setLayoutManager(new GridLayoutManager(context,
mColumnCount));
        }
        recyclerView.setAdapter(new MyLibroRecyclerViewAdapter
                (GeneraLibros.LIBROS, mListener));
    }
    return view;
}
```

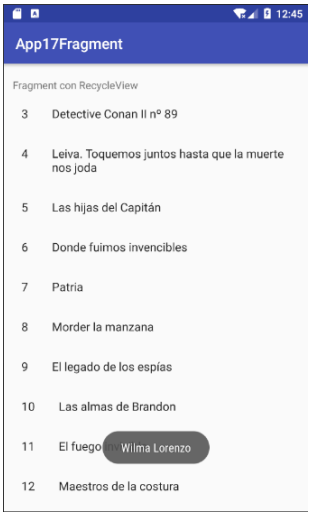
18. Ejecutamos nuestro programa y obtenemos un resultado siguiente:



19. Ahora le vamos a agregar una función. Ésta función consiste en que cuando le demos clic en un elemento de la lista nos muestre el autor. Modificaremos la clase "PrincActivity".
    a. Abrimos la clase "PrincActivity.java".
    b. Crearemos un mensaje tipo "Toast" en el método "".
    c. El método modificado se muestra a continuación:

```
@Override
public void onListFragmentInteraction(GeneraLibros.Libro libro) {
    Toast.makeText(this, libro.getAutor(),
            Toast.LENGTH_SHORT).show();
}
```

20. Al correr el programa si se selecciona un nombre de libro "*Leiva. Toquemos juntos hasta que la muerte nos joda*" nos muestra:



21. Ejercicio finalizado ☺.

Bibliografía Preliminar

1. **Google.** Activities. *The Activity Life Cycle.* [En línea] https://developer.android.com/guide/components/activities/activity-lifecycle.html.
2. **MacLean, Dave, Komatineni, Satya y Allen, Grant.** *Pro Android 5.* New York : Apress, 2015.
3. **Phillips, Bill y Hardy, Brian.** *Android Programming: The Big Nerd Ranch Guide.* Indianapolis, IN 46240 USA : Pearson Technology Group, 2013.
4. **Deutsch, Roger.** *Launch Your Android App.* s.l. : Kindle Edition , © 2016.