GOBIERNO DE MÉXICO | EDUCACIÓN SECRETARÍA DE EDUCACIÓN PÚBLICA | TECNOLÓGICO NACIONAL DE MÉXICO | TOLUCA

DEPARTAMENTO ADSCRIPCIÓN
Sistemas y Computación

Programa y Modalidad

B.2 Elaboración de Libro: Programando en Java

Nombre del Docente:
María del Socorro Moreno Figueroa

No dictamen.
AS-2-045/2023

Vo.Bo
Firma y nombre de
Persona(s) Revisora(s)
Fecha:

Nombre Personas Revisoras
Luis Antonio Estrada Manuel
Mauro Sánchez Sánchez
Federico del Razo López

Fecha:31 DE AGOSTO DE 2024

Contribución académica.

Actualmente en el Instituto Tecnológico de Toluca los alumnos de la carrera de Ingeniería en Sistemas Computacionales (Plan ISIC-2010-224) cursan la materia de “Fundamentos de programación” (AED-1285) en el primer semestre. Las competencias específicas están encaminadas al aprendizaje de los conceptos básicos de la programación, aprendiendo a aplicar expresiones aritméticas y lógicas en un lenguaje de programación; asimismo el uso y funcionamiento de estructuras secuenciales y repetitivas, además de la organización de datos mediante arreglos.

Es importante que los alumnos de esta carrera cuenten con un libro de texto que les ayude y facilite la comprensión de un lenguaje de programación, que utilicen la tecnología mediante técnicas y herramientas de programación con la finalidad de que adquieran la capacidad para desarrollar un pensamiento lógico a través de un lenguaje de programación orientado a objetos como lo es el lenguaje de programación java.

De esta manera se podrá dar cumplimiento al perfil profesional del egresado, coadyuvando en los siguientes ámbitos:

- Implementando aplicaciones computacionales para solucionar problemas de diversos contextos, integrando diferentes tecnologías, plataformas o dispositivos.

- Diseña, desarrolla y aplica modelos computacionales para solucionar problemas, mediante la selección y uso de herramientas matemáticas.

- Diseña e implementa interfaces para la automatización de sistemas de hardware y desarrollo del software asociado.

- Coordina y participa en equipos multidisciplinarios para la aplicación de soluciones innovadoras en diferentes contextos.

- Desarrolla y administra software para apoyar la productividad y competitividad de las organizaciones cumpliendo con estándares de calidad.

El libro de texto: “Programando en Java” ayudará al docente que imparta la materia, y facilitará mediante ejercicios resueltos y propuestos la comprensión de los temas, además podrá ser utilizado para otras carreras como Ingeniería en Tecnologías de la Información y Comunicaciones (Fundamentos de programación) e Ingeniería Mecatrónica (Programación avanzada); beneficiando a más de 600 alumnos (de primero, segundo, tercer semestre y alumnos de repetición) del instituto.

ÍNDICE

1. CONCEPTOS BÁSICOS DEL LENGUAJE

Java es un lenguaje de programación de propósito general, se caracteriza por ser totalmente orientado a objetos y la sintaxis del lenguaje es parecida a la del lenguaje C. Su biblioteca es tan extensa que provee funcionalidad para casi todo lo que el programador pueda necesitar.

En este capítulo se abordarán algunos conceptos básicos para comprender la programación en Java, tales como algoritmo, programa, lenguaje de programación, dato, tipos de datos, variable, constante, identificador, etcétera.

También se mostrará la estructura básica de un programa, las partes que lo integran (`import, package, class, main()`), ¿Cómo usar los diferentes tipos de operadores (aritméticos, lógicos y relacionales)?, ¿Qué son las palabras reservadas? ¿Qué son y como se utilizan las entradas y salidas en un programa? Así como el uso de comentarios hasta llegar a la elaboración de un programa sencillo.

1.1 Algoritmo

Todos los días realizamos una serie de pasos o acciones, muchos de manera inconsciente, que nos permite lograr un objetivo, meta, resultado o resolver un problema.

Esta serie de pasos las iniciamos desde que nos despertamos y decidimos levantarnos, como cepillar los dientes, bañarnos, vestirnos, desayunar, salir al trabajo o a la escuela.

Según Cairó Battistutti (2005) “Formalmente definimos un algoritmo como un conjunto de pasos, procedimientos o acciones que nos permiten alcanzar un resultado o resolver un problema” (pág. 1).

Los algoritmos son instrucciones lógicas previas a la codificación y programación, esto quiere decir que para resolver un problema es importante primero realizar un algoritmo para después traducirlo a un lenguaje de programación.

1.1.1 Ejemplo de algoritmo.

Ejemplos de algoritmos son: hacer una receta de cocina, instrucciones para armar una bicicleta, calcular el sueldo de un trabajador, calcular el área de una figura, calcular la edad de una persona, pasos para ver una película, etcétera. Los algoritmos se pueden expresar mediante fórmulas, diagramas de flujo o diagramas N-S y pseudocódigos.

Un algoritmo debe ser:

- **Preciso.** Debe indicar el orden en que se realiza cada paso.
- **Definido.** Si se realiza dos veces, se debe obtener el mismo resultado cada vez.
- **Finito.** Debe tener un fin, esto es, un número determinado de pasos.

Para la resolución de un problema se deben considerar los siguientes pasos:

- **Análisis del problema.** Se requiere una clara definición, donde se explique lo que debe hacer el programa y el resultado o solución deseada, mediante las entradas requeridas, la salida deseada y los cálculos que producirán la salida deseada.
- **Diseño del algoritmo.** ¿Cómo hacer lo planteado en el análisis del problema mediante un algoritmo?
- **Resolución del problema.** Traducir el algoritmo a lenguaje de programación, codificar, compilar y ejecutar con la finalidad de comprobar y eliminar errores que pudieran aparecer.

Ejemplo: Hacer un algoritmo tal que, dado el año de nacimiento de una persona, calcule la edad que tendrá el próximo año. Para la resolución de este problema se usará un pseudocódigo.

Análisis del problema:

Entrada: año de nacimiento, próximo año.

Proceso: calcular edad = próximo año – año de nacimiento.

Salida: imprimir edad.

Algoritmo:

1. Inicio
2. Leer año de nacimiento
3. Leer próximo año
4. Calcular edad = próximo año – año de nacimiento
5. Imprimir edad
6. Fin

1.2 Programa

“Un programa de computadora es un conjunto de instrucciones escrito en forma secuencial en un lenguaje de programación para resolver un problema y alcanzar un resultado específico en una computadora” (Cairó Battistutti, 2022, pág. 25).

Un algoritmo se escribe con palabras en nuestro idioma, en este caso en español, mientras que un programan se realiza en un lenguaje de programación con palabras propias de éste, tales como palabras reservadas las cuales se abordarán en el subtema 1.6.

1.3 Lenguaje de programación

Un lenguaje de programación

> Según Cairó Battistutti (2022)
>
> Es un lenguaje formal compuesto por un conjunto de instrucciones sintácticas y semánticas que permiten que la codificación, el programa, ponga en práctica algoritmos específicos que permiten resolver problemas. Las reglas sintácticas son las encargadas de la sintaxis y las que en definitiva determinan si las instrucciones son válidas. Las reglas semánticas especifican el significado de esas instrucciones (pág. 25).

Por ejemplo: `x = y + 3;`

La sintaxis me indica qué secuencia de caracteres es válida para esta instrucción, ósea que una expresión debe empezar con una variable , que debe llevar el signo igual, si se desea hacer una suma deberá llevar el operador de suma entre dos variables y finalizar la instrucción con punto y coma.

Por otro lado, la semántica nos dice lo que significa esa secuencia de caracteres (para este caso es una expresión).

Este lenguaje formal es entendible por los humanos e interpretable por las computadoras, esto significa que es el medio de comunicación entre los humanos y las computadoras.

1.4 Traductores de lenguaje

Para que un programa escrito en un lenguaje de programación sea comprensible para una computadora es necesario utilizar traductores, son programas que traducen de lenguaje de alto nivel a lenguaje máquina y se clasifican en : compilador e intérprete.

1.4.1 Intérprete

> Según Joyanes Aguilar (2020)
>
> El programa intérprete traduce el programa escrito en el lenguaje de alto nivel, lo analiza y ejecuta directamente, instrucción a instrucción a medida que sea necesario y sin generar ningún código equivalente. Su acción equivale a la de un intérprete humano, que traduce las frases que oye sobre la marcha, sin producir ningún escrito permanente (pág. 26)

Algunos ejemplos de lenguajes intérpretes son Basic, que prácticamente ya no se utiliza, aunque Microsoft comercializa la versión de Visual Basic, Smalltalk, PHP, Perl, Python y Ruby

1.4.2 Compilador

La traducción del programa completo se realiza en una sola operación denominada compilación del programa; es decir, se traducen todas las instrucciones del programa en un solo bloque. El programa compilado y depurado (eliminados los errores del código fuente) se denomina programa ejecutable porque ya se puede ejecutar directamente y cuantas veces se desee; sólo deberá volver a compilarse de nuevo en el caso de que se modifique alguna instrucción del programa. De este modo, el programa ejecutable no necesita del compilador para su ejecución. Los lenguajes típicos más utilizados son: C, C++, Java, Fortran y Cobol. Pascal.

1.5 NetBeans

Para la realización de los programas en Java, se utilizará un entorno de desarrollo integrado (IDE), es un software que contiene todas las herramientas necesarias para facilitar al programador el desarrollo de software.

El IDE es una herramienta que permite editar programas, compilarlos, depurarlos, documentarlos, ejecutarlos, entre otras.

Existen varios IDE, entre los que destacan: NetBeans, Eclipse que son de código abierto, y otros son de paga como BlueJ, y JBuilder entre otros. En este libro se usará NetBeans versión 14.

“NetBeans IDE es un entorno de desarrollo integrado, gratuito y de código abierto para el desarrollo de aplicaciones en los sistemas operativos Windows, Mac, Linux y Solaris” (Oracle, 2024).

“El IDE simplifica el desarrollo de aplicaciones web, corporativas, de escritorio y móviles que utilizan plataformas Java y HTML5. El IDE también ofrece soporte para el desarrollo de aplicaciones PHP y C/C++” (Oracle, 2024).

Este entorno de desarrollo integrado permite escribir, compilar, depurar y ejecutar programas. Existe una variedad de módulos diseñados especialmente para hacerlo extensible. Esta cualidad hace que NetBeans sea lo suficientemente poderoso y escriba aplicaciones Java para escritorio, así como para dispositivos móviles.

Puede descargar el NetBeans IDE en la siguiente liga:

https://netbeans.apache.org/front/main/download/

Una vez instalado el software se tendrá acceso a las funcionalidades que ofrece, de dónde destaca su interfaz amigable.

1.6 Dato

Un dato es una representación simbólica (numérica, alfabética, algorítmica, espacial, etc.) de un atributo o variable cuantitativa o cualitativa. Es un valor o referente que recibe la computadora por diferentes medios. Por ejemplo, la edad de una persona, el nombre de un país, el número atómico de un elemento, el color de un auto, una ecuación aritmética, etcétera.

1.6.1 Tipos de datos

Todas las variables en Java tienen asociado un tipo de dato. Cuando asignamos un valor a una variable, ambos deben ser del mismo tipo. Por ejemplo: `int num = 56;`

Donde la variable `num` está declarada de tipo entero y por lo tanto se le puede asignar un valor entero. Cada tipo de dato ocupa un espacio en la memoria, donde se almacenan los valores correspondientes.

En la Tabla 1.1 podrá visualizar los tipos de datos que soporta Java, así como el espacio que ocupan en memoria.

Tabla 1.1. Tipos de datos primitivos en Java.

Enteros	**Tamaño en bytes**
`byte`	1
`short`	2
`int`	4
`long`	8
Flotantes/Decimales	
`float`	4
`double`	8
Caracteres	
`char`	2
Lógicos	
`boolean`	0

1.6.2 Cast

Un cast es una operación especial que nos permite realizar una conversión entre tipos de datos. Por ejemplo, `int` y `double`, el tipo de dato `double` ocupa más espacio de memoria (8 bytes) y no cabe en uno de tipo `int` (4 bytes) como se mostró en la Tabla 1.1.

```
int a;
```

```
double b = 2.5;
a = b;            //Envía un error (tipos incompatibles) a la hora de ejecutar el
                  //programa.
```

Para que no marque error se puede hacer un cast.

```
int a;
double b = 2.5;
a = (int)b;       //No envía error, pero se pierde información.
```

el cast es lo que va entre paréntesis (`int`). Lo que hace es excluir (truncar) la parte decimal de `2.5` y convertirlo en el entero `2`, que podrá ser asignado a la variable. Este tipo de conversión se llama de estrechamiento ya que obliga la asignación de un tipo de dato de mayor tamaño en una variable de menor tamaño, desde luego con pérdida de información.

Sin embargo, un tipo de dato int si cabe en un tipo de dato doble. Por ejemplo:

```
int a = 50;
double b;
b = a;
```

Java permite que a una variable `double` se le asigne un valor `int`, convirtiendo de forma automática el valor entero 50 en el valor `double` 50.0.

Esto es posible porque la variable `double` es de mayor tamaño que el valor `int`, es decir, tiene suficiente espacio para guardarlo. Por lo tanto, este tipo de conversiones y asignaciones automáticas será posible cuando la variable sea de mayor tamaño que el valor asignado.

1.7 Identificador

Es el nombre que se le da a una variable, constante, objeto, clase, o método en Java.

Un identificador se forma con letras, números, carácter de subrayado (_) y signo de pesos ($), y siempre debe comenzar con una letra.

“Las sanas costumbres de la ingeniería de software sugieren que sólo se utilicen letras y si se utilizarán dos palabras, estas deben ir unidas y la primera letra de la segunda palabra debe estar en mayúsculas” (Cairó Battistutti, 2022, pág. 11).

El identificador puede ser de cualquier longitud; Java es sensible a las mayúsculas y minúsculas por lo tanto no es lo mismo el identificador edad que eDaD.

Los identificadores se escriben de la siguiente forma:

Inician por una letra, carácter de subrayado (_), o signo de pesos ($).

Los siguientes caracteres pueden ser letra, números, carácter de subrayado (_), o signo de pesos ($).

Existe diferencia entre mayúsculas y minúsculas.

No hay una longitud máxima para el identificador

Ejemplos:

Habitacion120	numero1	Suma$
primerApellido	a	nombre
Valor_Inicial	AREA	$transaccion
nombre_clase	_variable	LADO_1

Con la finalidad de identificar fácilmente las entidades de un programa, se recomienda escribir los identificadores de la siguiente forma:

1. Identificadores de variables en minúsculas, si tiene dos palabras, la segunda palabra va unida a la primera (sin espacio en blanco) e inicia con mayúscula.
2. Identificadores de constantes en mayúsculas, .
3. Identificadores de métodos en minúsculas, si tiene dos palabras, la segunda palabra va unida a la primera (sin espacio en blanco) e inicia con mayúscula.
4. Identificadores de clase con el primer carácter en mayúsculas.

1.8 Palabras reservadas

“Una palabra reservada es una característica de Java asociada con algún significado especial y no se puede utilizar como nombre de identificador” (Joyanes Aguilar & Zahonero Martínez, 2011, pág. 55).

En la Tabla 1.2 se muestran las palabras reservadas del lenguaje Java, como podrá observar todas ellas están escritas en minúsculas.

Tabla 1.2. Palabras reservadas del lenguaje Java.

abstract	continue	for	new	switch
assert	default	goto	private	synchronized
boolean	do	if	package	this
break	double	implements	protected	throw
byte	else	import	public	throws
case	enum	instanceof	return	transient
catch	extends	int	short	try
char	final	interface	static	void
class	finally	long	strictfp	volatile
const	float	native	super	while

1.9 Variable

Es un dato que puede cambiar su valor durante la ejecución de un programa. Este valor puede ser de cualquier tipo (numérico, carácter o lógico).

Sintaxis general para declarar una variable en java:

```
tipoDato nombreVariable;
```

Ejemplos de las diferentes formas de declarar una variable en Java:

```
int numero;           /*numero es de tipo entero*/
float base, altura;   /*base y altura son de tipo float*/
char letra = 'A';     //letra esta inicializada con la
                      //letra A
```

Note que las variables de un mismo tipo de dato se separan por comas y al final de la declaración lleva punto y coma (;).

1.10 Constante

Es un dato que no cambia durante la ejecución de un programa. En java se utiliza la palabra reservada5 final para declarar una constante. Sintaxis para declarar una constante:

```
final tipoDato nombreConstante = valor;
```

Ejemplos:

```
final double PORCENTAJE = 3.5;
final int COD_POSTAL = 52004;
final char LETRA = 'Z', CALIFICACION = 'A';
```

Note que las constantes de un mismo tipo de dato se separan por comas y al final de la declaración lleva punto y coma (;).

1.11 Tipos de operadores

"Los operadores son elementos que permiten realizar acciones sobre uno o más operandos en una operación, expresión o instrucción. Los operandos, por otra parte, son números, constantes o variables" (Cairó Battistutti, 2022, pág. 15).

Los operadores se dividen en: Aritméticos, relacionales y lógicos.

1.11.1 Operadores aritméticos

Realizan operaciones matemáticas sobre un conjunto de valores. La Tabla 1.3 se muestran los operadores aritméticos del lenguaje Java.

Tabla 1.3. Operadores aritméticos en Java.

OPERACIÓN	OPERADOR	EJEMPLO
Asignación	=	`int b = 2 + 3;`
Suma	+	`s = s + 7;`
Resta	-	`r = r – 5;`
Multiplicación	*	`m = m * 3;`
División	/	`d = d / 5;`
Modulo	%	`m = 4 % 2;`

Es importante mencionar que el operador más (+) también se utiliza en java para concatenar cadenas, esto se podrá observar en la realización de programas.

1.11.2 Operadores simplificados de asignación

Los operadores simplificados de asignación permiten simplificar el código uniendo dos operadores, uno aritmético con el operador de asignación igual (=), con la finalidad de eliminar uno de los operandos. El operando que se elimina a la derecha de la expresión es idéntico a la variable de asignación que se encuentra a la izquierda de la expresión. En la Tabla 1.4 se puede observar el uso de estos operadores.

Tabla 1.4. Operadores simplificados de asignación.

OPERACIÓN	OPERADOR	EJEMPLO	EQUIVALENCIA
Suma	+=	`s += 7;`	`s = s + 7;`
Resta	-=	`r -= 5;`	`r = r – 5;`
Multiplicación	*=	`m *= 3;`	`m = m * 3;`
División	/=	`d /= 15;`	`d = d / 5;`
Modulo	%=	`x =% y;`	`x = x % y`

1.11.3 Operadores de incremento y decremento

Para Cairó Battistutti (2022)

Son operadores unarios que permiten aumentar o disminuir el valor de la variable en una unidad, respectivamente. Estos se pueden utilizar antes (prefijo) o después (sufijo) de una variable. Cuando el operador se utiliza en forma de sufijo entonces

se incrementa o decremento después de ser utilizado en la expresión. Cuando el operador se utiliza en forma de prefijo, este se incrementa o disminuye antes de ser utilizado en la expresión (pág. 18).

En la Tabla 1.5 se muestran los operadores de incremento y decremento. en la Tabla 1.6 se muestra ejemplos de cómo funcionan.

Tabla 1.5. Operadores de incremento y decremento.

Operación	Operador	Descripción
Incremento	++	Aumenta el valor de una variable en una unidad
Decremento	--	Disminuye el valor de una variable en una unidad

En la Tabla 1.6 se muestran algunos ejemplos, con la finalidad de comprender como funcionan estos operadores. Considere que las variables i y k son enteras, mientras que x y w son reales.

Tabla 1.6. Uso de operadores de incremento y decremento.

Ejemplo	Resultado
`i = 5;`	5
`k = i++;`	k = 5, i = 6
`i = 7;`	7
`k = ++i;`	k = 8, i = 8
`x = 3.74;`	3.74
`w = x++ + 2.50;`	w = 6.24, x = 4.74
`i = 5;`	5
`k = i--;`	k = 5, i = 4
`i = 5;`	5
`k = --i;`	k = 4, i = 4

1.11.4 Operadores relacionales

Los operadores relacionales utilizan comparativas entre elementos o valores y así calcular un valor lógico que puede ser falso o verdadero.

Normalmente se usan en las estructuras selectivas (`if`) o estructuras repetitivas (`for`, `while`) que se utilizan para comprobar una condición. En la Tabla 1.7 podrá visualizar los operadores relacionales, así como un ejemplo.

Tabla 1.7. Operadores relacionales en Java.

DESCRIPCIÓN	OPERADOR	EJEMPLO
Menor que	<	`if (a < b)` //si a es menor que b
Mayor que	>	`if (5 > 3)` //si 5 es mayor que 3
Menor o igual que	<=	`if (b <= c)` //si b es menor o igual que c
Mayor o igual que	>=	`if (b >= c)` //si b es mayor o igual que c
Igual que	==	`if(a == c)` //si a es igual que c
Diferente	!=	`if(a != c)` //si a es diferente que c

1.11.5 Operadores lógicos

Estos operadores permiten seleccionar acciones de acuerdo con un criterio. En la Tabla 1.8 podrá visualizar los operadores lógicos.

Tabla 1.8. Operadores lógicos en Java.

NOMBRE	OPERADOR	EJEMPLO
y	&&	`if (a > b) && (a > c)` //si a es mayor que c, y a es mayor que c
no	!	`!(b)` //no b
o	||	`if (a < b) || (a < c)` //si a es menor que c, o a es menor que c

A continuación, en la Tabla 1.9 se presenta la Tabla de verdad de los operadores lógicos.

Tabla 1.9. Tabla de verdad de los operadores lógicos.

A	B	!A	!B	A || B	A && B
Verdadero	Verdadero	Falso	Falso	Verdadero	Verdadero
Verdadero	Falso	Falso	Verdadero	Verdadero	Falso
Falso	Verdadero	Verdadero	Falso	Verdadero	Falso
Falso	Falso	Verdadero	Verdadero	Falso	Falso

1.12 Jerarquía de los operadores

Las expresiones se evalúan de izquierda a derecha, sin embargo, es importante conocer la prioridad de los operadores ya que estos determinan el resultado de una expresión. El operador ( ) es un operador asociativo que tiene la prioridad más alta en cualquier lenguaje de programación. En la Tabla 1.10 se muestra la jerarquía de los diferentes operadores.

Tabla 1.10. Jerarquía de operadores

Jerarquía	Operador
mayor	( )
	*, /, %
	+, -
	>,<, >=, <=
	==, !=
	&&
menor	||

A continuación, se presenta un ejercicio en la Figura 1.1 puede ver la forma en que se resuelve de acuerdo con la jerarquía de los operadores:

```
9 + 7 * 8 - 36 / 5
```

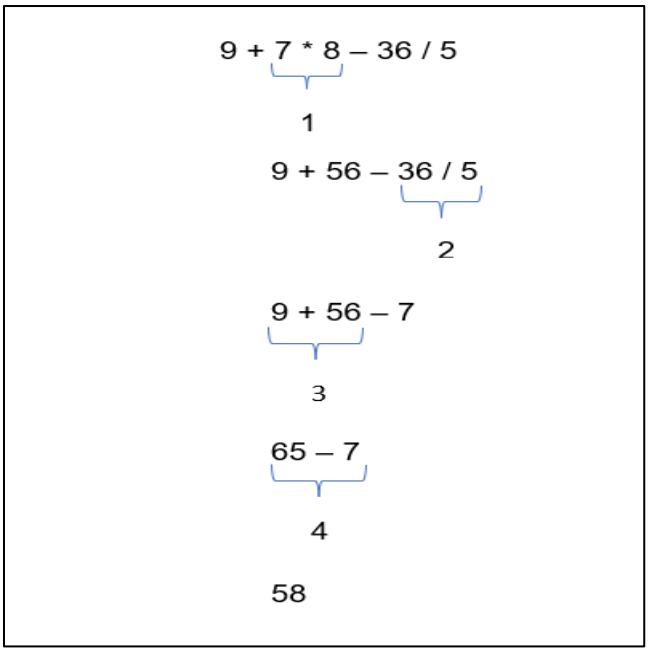


Figura 1.1. Jerarquía de operadores.

Paso 1. Multiplicación de 7 * 8.
Paso 2. División de 36 / 5, valores enteros lo que genera un resultado entero (7).
Paso 3. Suma de 9 + 56
Paso 4. Resta de 65 – 7

En la Figura 1.2 se muestra el resultado mediante la codificación en Java, así como la ejecución. En la línea 7 se puede observar el uso del operador + de tres formas diferentes:

1. Dentro de las comillas dobles ("9+7*8-36/5 = ") en color verde como un carácter.
2. Después del cierre de las comillas dobles en color negro, concatenando (uniendo) una cadena con una expresión.
3. Dentro de la expresión (9+7*8-36/5) haciendo la función de sumar.

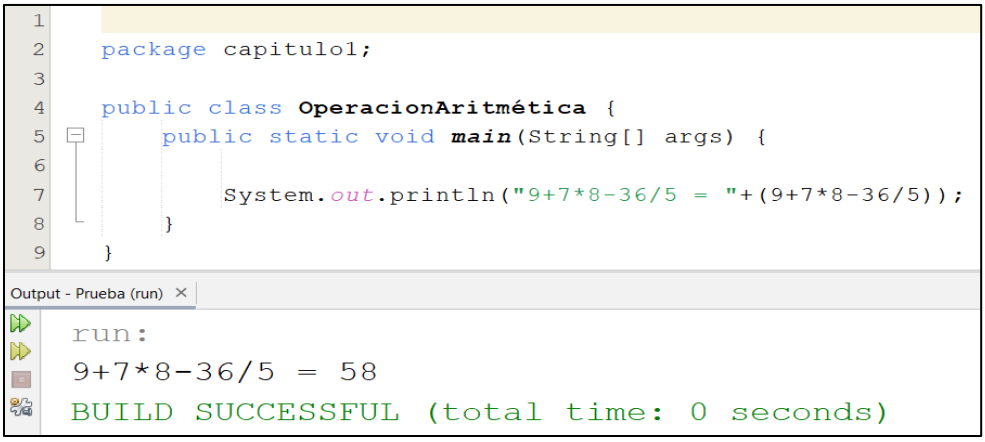


Figura 1.2. Codificación y ejecución del ejemplo.

A veces es necesario realizar otras operaciones más especiales que las operaciones básicas, por ejemplo, calcular la raíz cuadrada o cúbica de un número, el valor absoluto, el arco tangente de un número, el logaritmo natural, etcétera. Estas operaciones especiales se llaman funciones o métodos, y se encuentran en la clase `Math` del paquete `java.lang`. A continuación, se presenta la sintaxis de cómo utilizar estos métodos.

```
Math.nombre_método (argumento);
```

Ejemplos:

```
Math.pow (2, 4);
Math.sqrt (x);
Math.max (3.7, 2.5);
```

En la tabla 1.11 se muestran los principales métodos de la clase `Math`.

Tabla 1.11. Principales métodos de la clase Math.

FUNCIÓN	DEVUELVE
`sqrt (x)`	Raíz cuadrada de x (x >= 0)
`pow (x, y)`	Potencia de x^y
`exp (x)`	e^x
`log (x)`	Logaritmo natural
`round (x)`	Entero más próximo a x
`ceil(x)`	Entero más pequeño >= x
`floor(x)`	Entero más largo >= x
`abs(x)`	Valor absoluto de x
`max(x, y)`	Valor mayor de x e y
`min(x, y)`	Valor menor de x e y
`sin(x)`	Seno de x
`cos(x)`	Coseno de x
`tan(x)`	Tangente de x

`asin(x)`	Arco seno de x
`atan2(y, x)`	Arco cuya tangente es y/x
`toRadianes(x)`	Convierte x grados a radianes
`toDegrees(x)`	Convierte x radianes a grados

1.13 Definición de estructura secuencial

La estructura secuencial es un conjunto de instrucciones o sentencias que se ejecutan una a continuación de la otra hasta llegar al final.

En este capítulo se iniciará aprendiendo a hacer programas con estructura secuencial.

1.14 Estructura básica de un programa

Los programas en Java se componen de una o más clases y obligatoriamente el método `main()` de la clase principal. Pueden incluir:

- Declaraciones `import`.
- Declaraciones de clases (`class`).
- El método `main()`.
- Métodos definidos por el usuario dentro de las clases
- Comentarios del programa

1.14.1 Declaración `import`

Los paquetes (`package`) son contenedores de clases e interfaces que tienen elementos y funcionalidades comunes, permiten crear clases con el mismo nombre siempre y cuando estén en paquetes diferentes.

Por defecto muchas clases que vienen incluidas en java pertenecen a un paquete determinado. por ejemplo, la clase `String` pertenece al paquete `java.lang`, la clase `BufferedReader` pertenece al paquete java.io.

Básicamente es el ordenamiento de las clases en grupos funcionales.

“En java, las clases se agrupan en paquetes (`package`) que definen utilidades o grupos temáticos, y que se encuentran en directorios del disco con su mismo nombre, Para incorporar y utilizar las clases de un paquete en un programa se utiliza la declaración `import`” (Joyanes Aguilar & Zahonero Martínez, 2011, pág. 48).

Para utilizar una clase de un paquete específico es necesario importarla, esto se hace por medio de una declaración `import`.

Sintaxis de la declaración `import`

```
import nombrePaquete.NombreClase;
```

El identificador `NombreClase` es el nombre de una clase del paquete, por convención todos los nombres de clase comienzan con una letra mayúscula, y la primera letra de cada palabra en el nombre de la clase debe de ir en mayúscula (por ejemplo, `NombreClase`). Es posible incorporar varias clases con una secuencia de sentencias `import`; por ejemplo:

```
import java.util.Scanner.
import java.util.Calendar.
import java.util.Random;
```

En la primera instrucción se está importando la clase `Scanner` que se encuentra en el paquete `java.util`.

En la segunda instrucción se está importando la clase `Calendar` que se encuentra en el paquete `java.util`.

En la tercera instrucción se está importando la clase `Random` que se encuentra en el paquete `java.util`.

Como todas las clases importadas están en el mismo paquete también es posible incorporar todas las clases públicas del paquete, sustituyendo el nombre de la clase por un asterisco, como se muestra a continuación:

```
import java.util.*;
```

1.14.2 Declaración de Clase (`class`)

“Una clase es un tipo definido por el usuario. Las clases son los bloques de construcción fundamentales de los programas orientados a objetos” (Joyanes Aguilar & Zahonero Martínez, 2014, pág. 474).

Sánchez Sánchez (2015) define “La clase es considerada como un conjunto de objetos o bien el “molde” con el que los objetos serán creados” (pág. 5).

Los programas en Java deben tener al menos una clase que debe definir, llamada clase principal y que incluye el método `main()`, y si es necesario otros método y variables. Para declararla es opcional empezar con una palabra reservada (indica el acceso: `public`, `private` o `protected`), seguida por la palabra clave `class` (indicador de clase), nombre de la clase y miembros de la clase (variables y métodos) encerrados en llaves ({ }), como lo puede ver en la Figura 1.3 donde la clase Potencia es la clase principal.

```java
package capitulo1;

public class Potencia {

    public static void main(String[] args){
        int r, e;
        int n, p;

        n = 7;
        p = e = 5;
        r = 1;
        for ( ; p>0; p--)
            r = r * n;
        System.out.println("Potencia de " + n + "^" + e + " = " + r);
    }
}
```

Figura 1.3. Ejemplo de una declaración de clase (Potencia).

1.14.3 Método `main()`

Los programas en Java tienen un método `main()`, es la parte donde inicia la ejecución de un programa Java. Su estructura es la siguiente:

```
public static void main (String [ ] args)
{
     bloque de instrucciones
}
```

Un programa debe tener al menos un método `main()`. Los métodos pueden realizar tareas y devolver información

El argumento del método `main(String []args)` es decir lo que va dentro de los paréntesis, es una parte requerida en la declaración del método `main()` y es un arreglo de objetos de la clase `String`.

1.14.4 Métodos definidos por el usuario

Todos los programas se construyen a partir de una o más clases compuestas por una serie de variables y métodos que se integran para crear una aplicación; todos los métodos contienen una o más sentencias de Java, generalmente creadas para realizar una única

tarea, cómo imprimir en pantalla, escribir un archivo o cambiar el color de la pantalla; es posible declarar un número casi ilimitado de métodos en una clase de Java.

“Los métodos definidos por el usuario se invocan, dentro de la clase donde se definieron, por su nombre y los parámetros opcionales que pudieran tener; después de que él método se invoca, el código asociado se ejecuta y, a continuación. se retorna al método llamador” (Joyanes Aguilar & Zahonero Martínez, 2011, pág. 51)

1.14.5 Comentarios.

Los comentarios son notas que se añaden en un programa, que proporciona información ignorada por el compilador, su uso es opcional, aunque recomendable. La finalidad de los comentarios es describir cómo funciona el código y facilitar la comprensión. Existen dos formas de escribir los comentarios:

- **Dos barras diagonales seguidas //:** Para incorporar una línea de texto.

  ```
  //Esto es un comentario escrito en una sola línea
  ```

- **Con los caracteres /* */:** Para insertar más de una línea de texto.

  ```
  /* Esto es un comentario que ocupa
  más de una línea de texto */
  ```

1.15 Entrada y salida de datos por consola

> Según Sznajdleder (2016)
>
> Llamamos “consola” al conjunto compuesto por la pantalla (en modo texto) y el teclado de la computadora donde se ejecutará nuestro programa. Así, cuando hablemos de “ingreso de datos por consola” nos estaremos refiriendo al teclado y cuando hablemos de “mostrar datos por consola” nos referimos a la pantalla (siempre en modo texto) (pág. 4).

La clase `System` se encuentra en el paquete `java.lang`. No es necesario importarla para poder hacer uso de ella. Esta clase nos permite trabajar con los diferentes canales de datos de nuestro programa. Entre las funciones proporcionadas se encuentran los flujos de entrada y la salida estándar.

`System.in` para entrada por teclado.

`System.out` para salida por pantalla.

1.15.1 Salida (`System.out`)

El objeto `out` definido en la clase `System` se asocia con el flujo de salida, que dirige los datos a consola y permite visualizarlos en la pantalla de la computadora; por ejemplo:

```
System.out.println (“ Buenos días ”);
```

En pantalla se visualizará lo que está entre comillas dobles (“ “): Buenos días

Se puede imprimir en pantalla utilizando cualquiera de los siguientes métodos:

- `print()`
- `println()`
- `printf()`

Cuando se utilizan los métodos `print()`, `println()` o `printf()` para salida, los datos se proporcionan con un argumento entre paréntesis y la sentencia termina con un punto y coma (;) por ejemplo:

```
print (“ Esto es una cadena “);
```
Imprime en pantalla: Esto es una cadena

```
println (“ Hola, “ + “estoy usando el operador + para concatenar dos cadenas”);
```
Imprime en pantalla: Hola, estoy usando el operador + para concatenar dos cadenas

```
print (“La suma es “ + (5+6));
```
Imprime en pantalla: La suma es 11

El primer operador + se utiliza para concatenar (unir) la cadena con el resultado de la operación aritmética que es 11, el segundo operador + se utiliza para hacer la suma.

Se pueden usar más de dos concatenaciones, todas las que sean necesarias para la resolución del problema.

En la Figura 1.4 se puede apreciar el uso del método `println()`:

```
1
2   package capitulo1;
3
4   public class ImprimeConPrintln {
5       public static void main(String[] args) {
6           double area = 15.3;
7           System.out.println("Esto es una prueba");
8           System.out.println("El área es: " + area);
9       }
10  }
```

Figura 1.4. Uso del método `println()`.

En la línea 7 se imprime una cadena.
En la línea 8 se imprime una cadena y el contenido de la variable área, usando el operador + para unirlos.

En la Figura 1.5 se muestra la ejecución del programa anterior, visualizando la salida en consola.

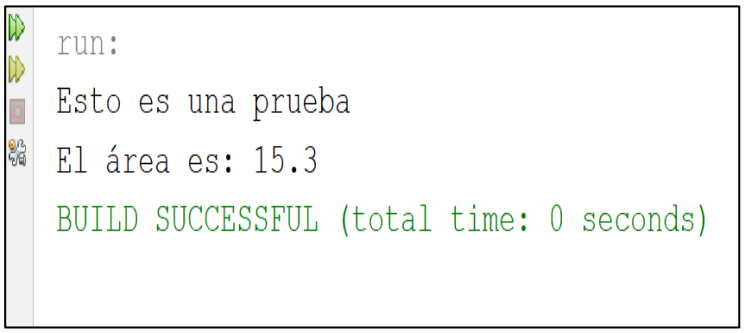


Figura 1.5. Ejecución del programa de la Figura 3

La diferencia entre el método `println()` y `print()` es que con `println()` la siguiente salida salta a una nueva línea, mientras que con `print()` la siguiente salida se sitúa en la misma línea. En la Figura 1.6 se muestra ejecución del programa anterior, utilizando el método `print()` en lugar de `println()`.

```
run:
Esto es una pruebaEl área es: 15.3BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 1.6. Uso del método `print()`.

El método `printf()` se puede utilizar para imprimir en pantalla en un formato específico. Este método tiene un primer argumento cadena, conocida como especificador de formato y el segundo argumento es el valor de la salida en el formato establecido. El tamaño de la cadena depende básicamente del número de parámetros que se incluyan en la misma. La sintaxis básica es:

```
printf( " % tipo_dato ", valor a imprimir);
```

% .- Indica que en esa posición se escribirán los datos que aparecen después de las comillas.

`tipo_dato`.- Se utiliza para expresar el tipo de dato: `s` para cadena de caracteres, `d` para entero, `f` para real; `E`, `e` para número real con notación científica, `g` número real se representa con notación científica si el número es muy grande o pequeño; `X`, `x` número entero en base hexadecimal, `c` carácter Unicode.

En caso de imprimir un valor real, se puede incluir el número de decimales que se desean visualizar, así como el ancho de la impresión, por ejemplo "%6.2f" imprime un campo de 6 dígitos de los cuales 2 de ellos serán utilizados para los decimales.

En la Figura 1.7 se muestra un ejemplo del uso del método `printf()`.

```
1  package capitulo1;
2
3  public class ImprimeConPrintf {
4      public static void main(String[] args) {
5          double x = 10000.0 / 3.0;
6          System.out.printf("%9.3f\n", x);
7      }
8  }
```

Figura 1.7. Uso del método `printf()`.

En la línea 5 se hace la declaración de la variable x de tipo `double` y además se le asigna una expresión (`1000.0 / 3.0`).

Al ejecutarse la línea 6 se visualiza x con una anchura de 9 caracteres y una precisión de 3 caracteres (un espacio en blanco a la izquierda y 8 caracteres), además se le agrego una secuencia de escape (`\n`) con la finalidad de pasar a una nueva línea al terminar la impresión, como se puede observar en la Figura 1.8.

```
run:
 3333.333
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 1.8. Ejecución del programa usando el método `printf()`.

La diagonal inversa (\) se conoce como carácter de escape. Cuando aparece una barra diagonal inversa en una cadena de caracteres, Java la combina con el siguiente carácter para formar una secuencia de escape. La secuencia de escape `\n` representa el carácter de una nueva línea. El carácter de nueva línea hace que el cursor de salida de la pantalla se desplace al inicio de la siguiente línea en la pantalla.

En la Tabla 1.12 se listan varias secuencias de escapes comunes, con descripciones de cómo afectan la manera de mostrar caracteres en la pantalla.

Tabla 1.12. Secuencias de escape comunes.

Secuencia de escape	Descripción
\n	Coloca el cursor de la pantalla al inicio de la siguiente línea.
\t	Desplaza el cursor de la pantalla hasta la siguiente posición de tabulación.
\r	Coloca el cursor de la pantalla al inicio de la línea actual; no avanza a la siguiente línea.
\\	Se usa para imprimir un carácter de barra diagonal inversa.
\”	Se usa para imprimir un carácter de comilla doble.
\’	Se usa para imprimir un carácter de comilla simple.

En la Figura 1.9 se pueden apreciar más ejercicios usando el método `printf()`.

```
1   package capitulo1;
2
3   public class UsaPrintf {
4       public static void main(String[] args) {
5           double y = 2.35178,n = 258.2681;
6           int z = 195;
7
8           System.out.printf("x = %.2f\n", 12.6395);
9           System.out.printf("y = %.3f\n", y);
10          System.out.printf("y = %+.3f\n", y);
11          System.out.printf("z = %d\n", z);
12          System.out.printf("z = %+d\n", z);
13          System.out.printf("y = %.2f z = %d\n", y, z);
14          System.out.printf("El cuadrado de %.2f es %.2f\n", n, n * n);
15          System.out.printf("y =%+15.2f\n", y);
16          System.out.printf("y =%+015.2f\n", y);
17          System.out.printf("y =%-15.2f\n", y);
18      }
19  }
```

Figura 1.9. Uso del método `printf()` con datos enteros y reales.

Línea 5 y 6. Declaración de las variables, dos reales y una entera.

Línea 8. `("x = %.2f\n", 12.6395);` imprime el número indicado después de la coma con una precisión de 2 decimales.

Línea 9. `("y = %.3f\n", y);` imprime el contenido de la variable y que es de tipo `double`, con una precisión de 3 decimales.

Línea 10. `("y = %+.3f\n", y);` imprime el contenido de la variable y que es de tipo double como un valor positivo, con una precisión de 3 decimales .

Línea 11. `("z = %d\n", z);` imprime el contenido de la variable z que es de tipo entera (195).

Línea 12. `("z = %+d\n", z);` imprime el contenido de la variable z que es de tipo entera como un valor positivo (+), quedando de la siguiente forma +195.

Línea 13. `("y = %.2f z = %d\n", y, z);` imprime dos variables `(y, z)` de diferente tipo. Cada una debe llevar su formato.

Línea 14. `("El cuadrado de %.2f es %.2f\n", n, n * n);` imprime la variable n que es de tipo `double` y el resultado de la multiplicación (n * n). Tanto la variable como la operación aritmética llevan su formato.

Línea 15. `("y =%+15.2f\n", y);` imprime la variable y que es de tipo double como número positivo (+), con un formato de 15 espacios justificados a la derecha, esto significa que primero imprimirá 10 espacios en blanco y después +2.35 (que ocupan 5 espacios más), con una precisión de 2 decimales.

Línea 16. `("y =%+015.2f\n", y);` esta instrucción es muy parecida a la de la línea 15 ya que también tiene un formato de 15 espacios, la diferencia es que se le agregó un cero antes del 15, por lo tanto, imprime primero el operador +, seguido de 10 ceros (por `015`) y por último la variable y que es de tipo `double` con una precisión de 2 decimales, quedando de la siguiente forma. +00000000002.35. Lo que hace es llenar con ceros los espacios en blanco.

Línea 17. `("y =%-15.2f\n", y);` ahora se sustituye el operador + por el operador -, esto permite justificar la impresión a la izquierda.

A continuación, se muestra en la Figura 1.10 la ejecución del programa para que pueda verificar los resultados.

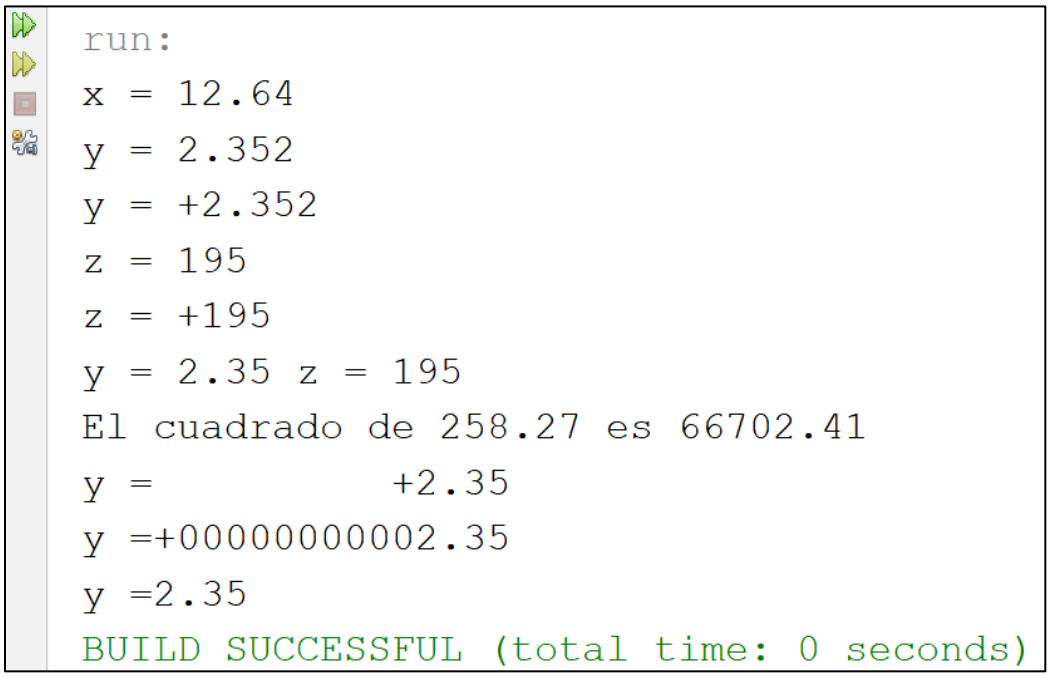


Figura 1.10. Ejecución del programa UsaPrintf.

1.15.2 Entrada

En la versión 5.0, Java incluyó una clase para simplificar la entrada de datos por el teclado. Esta clase se llama `Scanner` y se conecta a `System.in`.

“Para leer la entrada de la consola, se debe construir primero un objeto de la clase `Scanner`, pasando simplemente el objeto `System.in` al constructor `Scanner`” (Joyanes Aguilar & Zahonero Martínez, 2014, pág. 613).

Ejemplo:

```
Scanner leer = new Scanner (System.in);
```

En este ejemplo el objeto de la clase `Scanner` se llama `leer`, luego se coloca el signo = y la palabra reservada `new` para crear el objeto de tipo `Scanner` (leer), luego sigue el constructor de la clase `Scanner` que se llama igual que la clase (`Scanner`), le siguen los paréntesis y dentro de ellos la clase `System.in`, que es la encargada de permitir el ingreso de los datos. Se termina la declaración con punto y coma (`;`).

En la Figura 1.11 se muestran mediante un programa, ejemplos de cómo se introduce diferentes tipos de datos usando la clase `Scanner` y diferentes métodos tales como `next()`, `nextLine()`,`nextByte()` y `nextFloat()`. El lenguaje Java como tal no cuenta con un tipo de dato cadena, para resolver el problema utiliza la clase `String`.

```
1  package capitulo1;
2
3  import java.util.Scanner;
4
5  public class EntradaDatos {
6      public static void main(String[] args) {
7          Scanner leer = new Scanner(System.in);
8          String nombre, apellidos;
9          byte edad;
10         float calificacion;
11
12         System.out.print("Introduce tus apellidos: ");
13         apellidos = leer.nextLine();
14         System.out.print("Introduce tu nombre: ");
15         nombre = leer.next();
16         System.out.print("Introduce tu edad: ");
17         edad = leer.nextByte();
18         System.out.print("Introduce calificación: ");
19         calificacion = leer.nextFloat();
20         System.out.println("Te llamas "+nombre+" "+apellidos);
21         System.out.println("tienes "+edad+" años ");
22         System.out.println("y tu calificación es: "+calificacion);
23     }
24 }
```

Figura 1.11. Entrada de datos con `Scanner` y el uso de diferentes métodos.

En la línea 3 se importa la clase `Scanner` que se encuentra en el paquete `java.util`.
En la línea 7 se encuentra el objeto leer de la clase `Scanner`.
De la línea 8 a la 10 declaración de variables.
En la línea 13 se lee una línea de entrada, esto es una cadena con espacios es blanco, con la finalidad de introducir dos apellidos separados por un espacio en blanco.
En la línea 15 se lee una palabra, ósea una cadena sin espacios en blanco.
En la línea 17 se lee un tipo de dato `byte` para introducir la edad.
En la línea 19 se lee un tipo de dato `float` para introducir una calificación con punto decimal.

La Figura 1.12 muestra la ejecución del programa anterior.

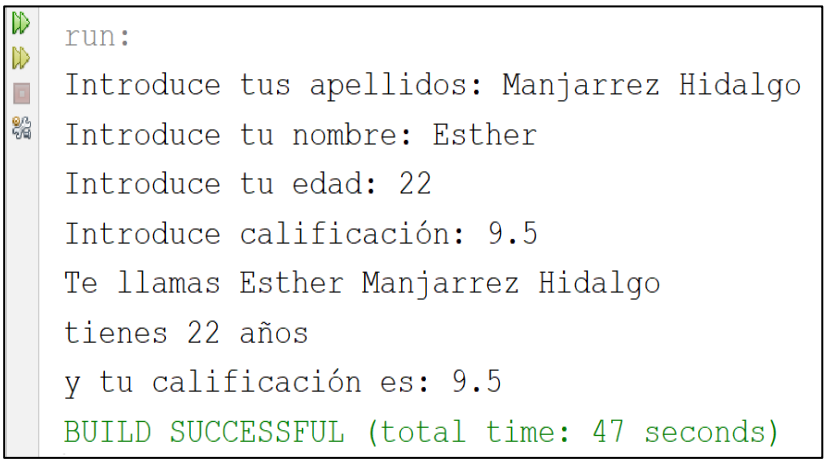


Figura 1.12. Ejecución del programa EntradaDatos.

1.16 Programas resueltos

Ejercicio 1. Dado como datos dos números enteros, hacer un programa que calcule la suma de los números e imprima el resultado.

Entrada: Dos números enteros, numero1 y número2.

Proceso: Operación aritmética (suma = numero1 + numero2).

Salida: Impresión de resultado (suma).

En la Figura 1.13 se muestra el programa codificado y mediante comentarios se hacen las acotaciones necesarias para su comprensión.

```java
package capitulo1;

//Incorporar esta librería para utilizar la clase Scanner
import java.util.Scanner;

public class Suma {
    public static void main(String[] args) {
        int numero1,numero2,resultado;
        //Declara e instancia el objeto lee que le permitirá leer los datos
        Scanner lee = new Scanner(System.in);

        System.out.println("Ingresa el primer número entero");
        //Asi puede leer un dato entero: nextInt()
        numero1 = lee.nextInt();
        System.out.println("Ingresa el segundo número entero");
        numero2 = lee.nextInt();
        //Operación aritmética: suma
        resultado = numero1 + numero2;
        //Salida de datos: Imprime resultado
        System.out.println("La suma de los 2 números enteros es..."+resultado);
    }
}
```

Figura 1.13. Ejercicio 1 codificado.

Explicación:

Todos los programas en Java tienen un paquete (`package`), una clase (`class`) y un método principal (`main()`).

Línea 4. Para poder leer los datos y utilizar la clase `Scanner` es necesario incorporar (importar) la librería `java.util.Scanner.`

Línea 8. Las variables `numero1, numero2` y `resultado` se declararon de tipo `int`, con la finalidad de que puedan tomar valores enteros.

Línea 10. Para leer los datos también se debe declarar e instanciar un objeto de la clase `Scanner`. La parte izquierda del operador = se utiliza para la declaración y la derecha para la instanciación. La variable objeto se llama `lee`.

Líneas 14 y 16. Para leer los datos enteros debe utilizar el método `nextInt()` acompañado del objeto de tipo `Scanner` (`lee`) de la siguiente forma `lee.nextInt();` Observe que el resultado de la lectura se debe asignar a una variable de tipo entera.

Línea 20. Para imprimir el resultado se utiliza la instrucción `System.out.println()`, dentro de los paréntesis se escribe lo que se desea imprimir.

En la Figura 1.14 se puede apreciar la ejecución del programa.

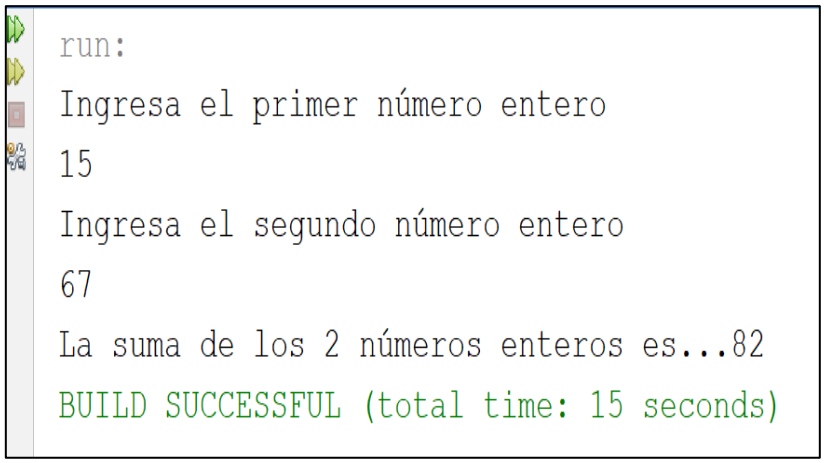


Figura 1.14. Ejecución del programa Ejercicio 1.

Ejercicio 2. Hacer un programa en Java tal que, dado el precio de un artículo y la cantidad de dinero entregada por el cliente, calcule e imprima el cambio que se debe entregar al cliente.

Entrada: Precio del artículo (precio) y cantidad entregada por el cliente (cantidad).

Proceso: Operación aritmética (total = cantidad - precio).

Salida: Impresión de resultado (total).

En la Figura 1.15 se muestra el Ejercicio 2 resuelto.

Explicación:

Línea 4. Para poder leer los datos y utilizar la clase `Scanner` es necesario incorporar (importar) la librería `java.util.Scanner`.

Línea 8. Las variables `precio, cantidad y total` se declararon de tipo `double`, con la finalidad de que puedan tomar valores con punto decimal.

Línea 9. Para leer los datos también se debe declarar e instanciar un objeto de la clase `Scanner`. La parte izquierda del operador = se utiliza para la declaración y la derecha para la instanciación. El nombre de la variable objeto se llama obj.

Línea 12. Para leer los datos debe utilizar la instrucción `obj.nextDouble();` Observe que el resultado de la lectura se debe asignar a una variable de tipo `double`.

```java
package capitulo1;

import java.util.Scanner;

public class Calculo {
    public static void main(String[] args) {
        double precio, cantidad,total;
        Scanner obj = new Scanner(System.in);

        System.out.print("Precio del artículo... ");
        precio = obj.nextDouble();
        System.out.print("Cantidad entregada por el cliente... ");
        cantidad = obj.nextDouble();
        total = cantidad - precio;
        System.out.printf("Cambio al cliente %8.2f\n",total);
    }
}
```

Figura 1.15. Programa Ejercicio 2 codificado,

Línea 16. Para imprimir el resultado se utiliza la instrucción `System.out.printf()` para darle formato a la salida, dentro de los paréntesis se escribe lo que se desea imprimir, en este caso con 8 caracteres, de los cuales 2 son de precisión (`%8.2f`). Además, se agrega la secuencia de escape `\n` para que después de imprimir el resultado dé un salto de línea y sea clara la respuesta. Haga la prueba quitando la secuencia de escape y vea la diferencia.

En la Figura 1.16 se puede apreciar la ejecución del programa.

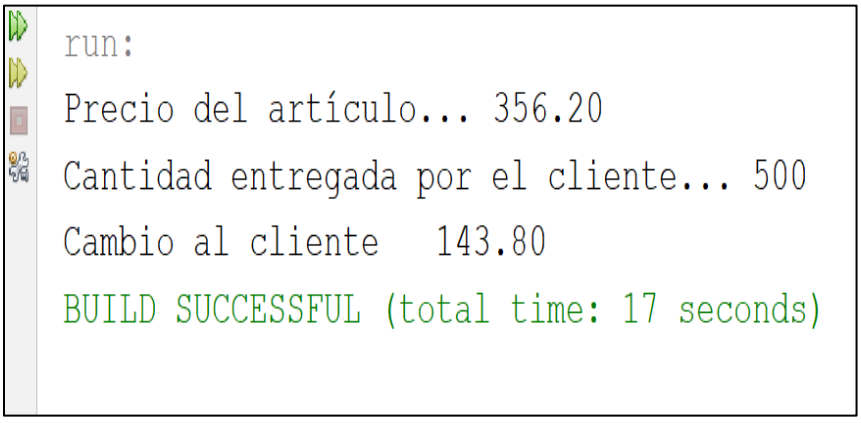


Figura 1.16. Ejecución del programa Ejercicio 2.

Ejercicio 3. Hacer un programa en java tal que, dado como datos el radio y la altura de un cilindro, calcule e imprima el área y su volumen.

Entrada: valores de radio y altura.

Proceso: Dos operaciones aritméticas:

$$volumen = \pi * radio^2 * altura$$

$$area = 2 * \pi * radio * altura$$

Salida: Impresión de resultados (*volumen y area*).

En la Figura 1.17 se muestra el Ejercicio 3 resuelto.

```
2   package capitulo1;
3
4   import java.util.Scanner;
5
6   public class AreaVolumenCilindro {
7       public static void main(String[] args) {
8
9           double radio, altura, volumen, area;
10          Scanner obj = new Scanner(System.in);
11
12          System.out.println("Introduce el radio del cilindro");
13          radio = obj.nextDouble();
14          System.out.println("Introduce la altura del cilindro");
15          altura = obj.nextDouble();
16          volumen = Math.PI * radio * radio* altura;
17          area = 2 * Math.PI * radio * altura;
18          System.out.printf("El volumen del cilindro es: %8.2f\n",volumen);
19          System.out.printf("El área del cilindro es: %8.2f\n",area);
20      }
21  }
```

Figura 1.17. Programa Ejercicio 3 codificado,

Explicación:

Línea 4. Para poder leer los datos y utilizar la clase `Scanner` es necesario incorporar (importar) la librería `java.util.Scanner.`

Línea 9. Las variables `radio, altura, volumen` y `total` se declararon de tipo `double`, con la finalidad de que puedan tomar valores con punto decimal.

Línea 10. Para leer los datos también se debe declarar e instanciar un objeto de la clase `Scanner`. La parte izquierda del operador = se utiliza para la declaración y la derecha para la instanciación. El nombre de la variable objeto se llama `obj`.

Líneas 13 y 15. Para leer los datos debe utilizar la instrucción `obj.nextDouble();` Observe que el resultado de la lectura se debe asignar a una variable de tipo `double`.

Líneas 16 y 17. Se utilizo la clase `Math` de Java para obtener el valor preciso de `Pi` mediante la instrucción `Math.PI`.

Líneas 18 y 19. Para imprimir el resultado se utiliza la instrucción `System.out.println()`. Pruebe imprimiendo con `System.out.print()`, y observe la diferencia al imprimir los resultados.

En la Figura 1.18 se puede apreciar la ejecución del programa.

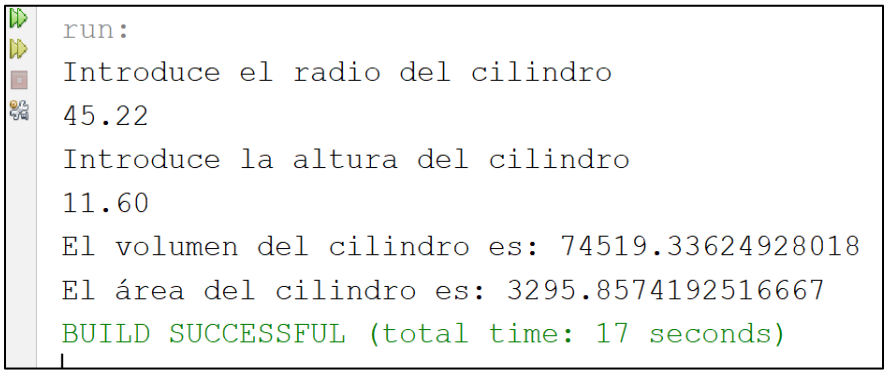


Figura 1.18. Ejecución del programa Ejercicio 3.

En la Figura 1.19 se muestra la diferencia de imprimir los resultados con las siguientes instrucciones:

```
System.out.printf ("El volumen del cilindro es: %8.2f\n",volumen);
System.out.printf ("El área del cilindro es: %8.2f\n",area);
```

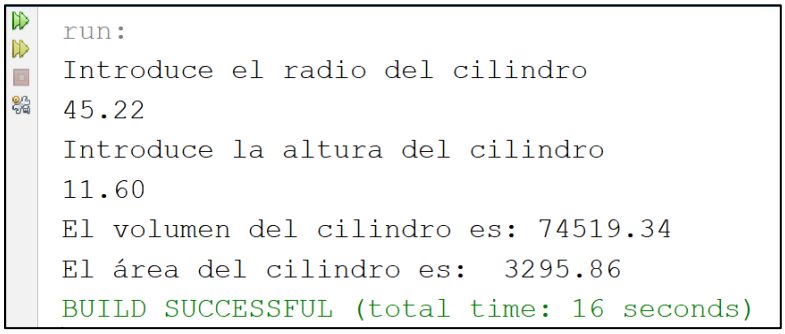


Figura 1.19. Impresión de resultados usando el método `printf()`.

Ejercicio 4. Hacer un programa en Java tal que, dado los tres lados de un triángulo calcule e imprima su área, utilizando las siguientes fórmulas;

$$sp = (lado1 + lado2 + lado\ 3)/2$$

$$area = \sqrt{sp * (sp - lado1) * (sp - lado2) * (sp - lado3)}$$

Entrada: valores de `lado1, lado2 y lado3.`

Proceso: Dos operaciones aritméticas: Calculo de semiperímetro y área

$$sp = (lado1 + lado2 + lado\ 3)/2$$

$$area = \sqrt{sp * (sp - lado1) * (sp - lado2) * (sp - lado3)}$$

Salida: Impresión de resultado (*area*).

En la Figura 1.20 se muestra el Ejercicio 4 resuelto.

```
2   package capitulo1;
3
4   import java.util.Scanner;
5
6   public class AreaTriangulo {
7       public static void main(String[] args) {
8
9           double lado1,lado2,lado3,sp,area;
10          Scanner o = new Scanner(System.in);
11
12          System.out.print("Ingrese el primer lado del triángulo ");
13          lado1 = o.nextDouble();
14          System.out.print("Ingrese el segundo lado del triángulo ");
15          lado2 = o.nextDouble();
16          System.out.print("Ingrese el tercer lado del triángulo ");
17          lado3 = o.nextDouble();
18          sp =(lado1 + lado2 + lado3)/2;
19          area = Math.sqrt(sp*(sp - lado1)*(sp - lado2)*(sp - lado3));
20          System.out.printf("El área del triángulo es:%.2f\n",area);
21      }
22  }
```

Figura 1.20. Programa Ejercicio 4 codificado.

Explicación:

Línea 4. Para poder leer los datos y utilizar la clase `Scanner` es necesario incorporar (importar) la librería `java.util.Scanner`.

Línea 2. Las variables `lado1, lado2, lado3, sp` y `area` se declararon de tipo `double`, con la finalidad de que puedan tomar valores con punto decimal.

Línea 3. Para leer los datos también se debe declarar e instanciar un objeto de la clase `Scanner`. La parte izquierda del operador = se utiliza para la declaración y la derecha para la instanciación. El nombre de la variable objeto se llama `o`.

Líneas 13, 15 y 17. Para leer los datos debe utilizar la instrucción `o.nextDouble();` observe que el resultado de la lectura se debe asignar a una variable de tipo double.

Línea 19. Se utilizo la clase `Math` de Java para obtener la raíz cuadrada mediante la instrucción `Math.sqrt()`.

Línea 20. Para imprimir el resultado se utiliza la instrucción `System.out.printf()` para darle formato a la salida, en este caso se especifica que el resultado tendrá 2 decimales (`%.2f`). Además, se agrega la secuencia de escape `\n` para que después de imprimir el resultado dé un salto de línea y sea clara la respuesta.

En la Figura 1.21 se puede apreciar la ejecución del programa.

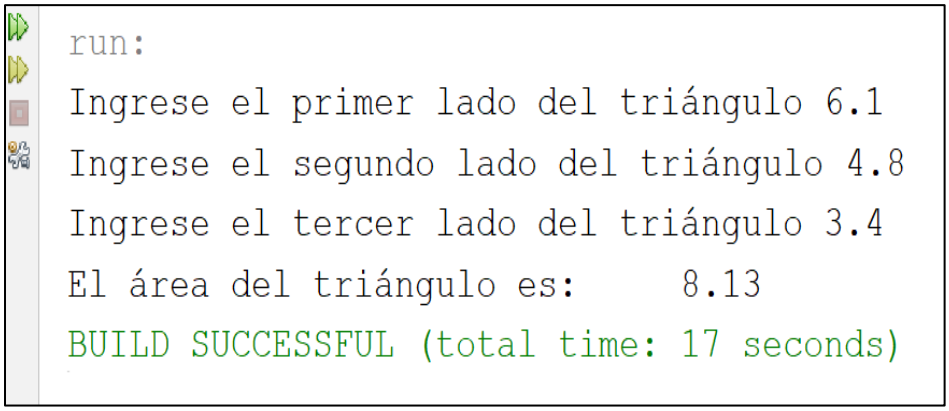


Figura 1.21. Ejecución del programa Ejercicio 4.

Ejercicio 5. Sean `a, b,` y `c` constantes de tipo double con valores `a = 88.0, b = 3.5, c = -5.2`; determina mediante un programa en Java el valor de las siguientes expresiones aritméticas. Obtén el resultado de cada expresión con un máximo de cuatro decimales.

```
a * (b % c)
2 * b + 3 * (a - c)
```

Entrada: Como `a, b` y `c` son constantes se asignarán sus valores en la codificación.

Proceso: Dos operaciones aritméticas:

$r1 = a * (b \,\%\, c)$

$r2 = 2 * b + 3 * (a - c)$

Salida: Impresión de resultados (*r1 y r2*).

En la Figura 1.22 se muestra el Ejercicio 5 resuelto.

```
2   package capitulo1;
3
4   public class OperacionesAritméticas {
5       public static void main(String[] args) {
6           final double a = 88.0, b = 3.5,c = -5.2;
7           double r1,r2;
8           r1 = a * (b % c);
9           r2 = 2 * b + 3 *(a - c);
10          System.out.print("a * (b % c) = ");
11          System.out.printf("%.4f\n",r1);
12          System.out.print("2 * b + 3 *(a - c) = ");
13          System.out.printf("%.4f\n",r2);
14      }
15  }
```

Figura 1.22. Programa Ejercicio 5 codificado.

Explicación:

Línea 6. Como no se introducen datos por teclado, no hay necesidad de importar la clase `Scanner`. Se declaran 3 constantes de tipo `double` (`a, b y c`) utilizando la palabra reservada `final` y asignando su valor 88.0, 3.5 y -5.2 respectivamente.

Línea 7. Se declaran dos variables tipo `double` (`r1 y r2`) para guardar el resultado de las operaciones aritméticas.

Líneas 10 y 12. El método `print()` se utilizó para imprimir las expresiones aritméticas.

Líneas 11 y 13. El método `printf()` se utilizó para imprimir los resultados (`r1 y r2`) con 4 decimales.

En la Figura 1.23 se puede apreciar la ejecución del programa.

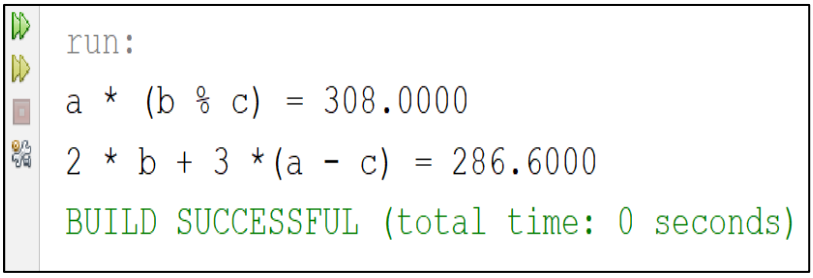


Figura 1.23. Ejecución del programa Ejercicio 5.

1.17 Programas propuestos.

1. Hacer un programa en Java que convierta una cantidad expresada en pesos a:
    a. Dólares
    b. Libras
    c. Euros

La salida debe tener dos decimales.

2. Dado como datos dos números reales, hacer un programa en Java que calcule la suma, resta, multiplicación y división de dichos números.
3. Una persona desea saber cuánto tiene que pagar por un producto que tiene el 16% de IVA. Hacer un programa en Java que calcule e imprima el precio total a pagar.
4. Una persona quiere comprar un auto y la agencia le solicita un enganche del 25% del valor total del auto. Hacer un programa que calcule e imprima el enganche que tiene que pagar por el auto.
5. Hacer un programa que solicite al usuario su edad actual y muestre la que tendrá el próximo año.
6. Dado como datos 3 valores enteros, hacer un programa que los imprima de forma inversa, por ejemplo, si ingresa 4, 5, 6 deberá imprimir 6, 5, 4.
7. Dado el año actual y el año de nacimiento de una persona, hacer un programa que calcule e imprima su edad.
8. Dada la estatura de 3 personas, hacer un programa que calcule e imprima el promedio de dichas estaturas.
9. Dado como dato el radio de una esfera, hacer un programa que calcule e imprima el área y su volumen.
10. Una tienda ofrece el 15% de descuento sobre total de la compra a sus clientes. Hacer un programa que imprima la cantidad de la compra sin descuento, calcule e imprima la cantidad que se le descontará al cliente por su compra, así como el total a pagar.
11. Hacer un programa que calcule e imprima el número de segundos que hay en un determinado número de días.
12. Hacer un programa que calcule e imprima el sueldo semanal de un trabajador con base al número de horas trabajadas y el precio por hora.
13. Dado el precio y el número de artículos. Hacer un programa que calcule e imprima la cantidad a pagar.

14. Dados los catetos de un triángulo. Hacer un programa que calcule e imprima la hipotenusa del triángulo.
15. Una persona desea saber cuánto gasta mensualmente en la compra de gasolina para su auto, si carga gasolina cada semana. Calcular e imprimir el gasto mensual, así como el promedio de lo que gasta mensualmente.
16. Un alumno desea saber su calificación final en la materia de Programación. Dicha calificación se compone de los siguientes porcentajes:
    a. 50% del promedio de dos calificaciones parciales.
    b. 30% de la calificación del examen final.
    c. 20% de la calificación de tareas.

    Hacer un programa que calcule e imprima su calificación final.

17. Dado un número entero de tres dígitos hacer un programa que calcule e imprima cuántas unidades, decenas y centenas tiene dicho número. Ejemplo: el número 456 tiene 4 centenas, 5 decenas y 6 unidades.
18. Un ciclista sale de Toluca a las h horas, m minutos y s segundos. El tiempo de viaje hasta llegar a Metepec es de s1 segundos. Hacer un programa que calcule e imprima la hora de llegada Metepec.
19. Un auto recorre 1580 kilómetros. Hacer un programa que calcule e imprima:
    a. El equivalente a metros.
    b. El equivalente a centímetros.
20. Dado un número entero hacer un programa que calcule e imprima lo siguiente:
    a. El número.
    b. Cuadrado del número.
    c. Cubo del número.
    d. Raíz cuadrada del número.

2. ESTRUCTURAS SELECTIVAS

Las estructuras selectivas se utilizan en los programas para regular las acciones que se puedan presentar en la ejecución del código. Mediante estas estructuras se pueden tomar decisiones de acuerdo con criterios lógicos establecidos en el código del programa (Gómez Jiménez & Moreno Nuñez, 2019, pág. 24)

Se utilizan cuando en la solución de un problema debemos tomar una decisión lógica, es decir tenemos que optar por una alternativa entre varias posibles. De ahí su nombre: estructuras selectivas.

En programación, un problema de decisión es aquel en el que las respuestas que tenemos que encontrar son sí o no, verdadero o falso.

Las estructuras selectivas que se utilizan para la toma de decisiones lógicas se pueden clasificar en:

- Estructura selectiva simple
- Estructura selectiva doble
- Estructura selectiva múltiple
- Estructura selectiva anidada o en cascada

2.1 Estructura selectiva simple

“La estructura selectiva simple permite que el flujo del programa siga por un camino especifico si se cumple una condición o un conjunto de condiciones. Si al evaluar la condición (o condiciones) el resultado es verdadero, entonces se ejecutan ciertas instrucciones. Después se continua con la secuencia normal del programa” (Cairó Battistutti, 2005, pág. 54). Sintaxis de la estructura selectiva simple:

```
if( condición){
      Bloque de instrucciones
}
```

Lo anterior se lee de la siguiente forma:

Si (`if`) la condición es verdadera realiza el bloque de instrucciones.

Por ejemplo, si solicitamos la edad de una persona y queremos saber si es mayor de edad, la instrucción que evalúa esa condición es la siguiente:

```
if(edad >= 18){
      System.out.println(“ Eres mayor de edad”);
}
```

En la Figura 2.1 se muestra el programa completo codificado en Java utilizando la estructura selectiva simple.

```java
package capitulo2;

import java.util.Scanner;

public class SeleccionSimple {
    public static void main(String[] args) {
        byte edad;
        Scanner o = new Scanner(System.in);
        System.out.println("Introduce la edad de una persona");
        edad = o.nextByte();

        if(edad >= 18){
            System.out.println("Eres mayor de edad");
        }

    }
}
```

Figura 2.1. Ejemplo de uso de la estructura selectiva simple.

En la Figura 2.2 se puede observar la ejecución del programa, al introducir la edad (en este caso 19 años) la instrucción `if()` evalúa la condición y como el resultado es verdadero imprime el mensaje “Eres mayor de edad”.

```
run:
Introduce la edad de una persona
19
Eres mayor de edad
BUILD SUCCESSFUL (total time: 3 seconds)
```

Figura 2.2. Ejecución del programa con estructura selectiva simple.

En caso de que se introduzca una edad menor de 18 el programa no hace nada ya que la estructura selectiva simple se utiliza para cuando se tiene una sola opción y el resultado de la condición sea verdadera.

2.2 Estructura selectiva doble

La estructura selectiva doble permite que el flujo del programa se divida en 2 caminos diferentes en el punto de la toma de decisión. Si al evaluar la condición o condiciones el resultado es verdadero, entonces se sigue por un camino específico y se ejecutan ciertas operaciones. En el caso de que el resultado sea falso entonces se sigue por otro camino y

se ejecutan otras instrucciones. En cualquiera de los casos, luego de ejecutarse las instrucciones indicadas, se continúa con la secuencia normal del programa.

Sintaxis de estructura selectiva doble:

```
if( condición){
      Bloque de instrucciones 1
}
else{
      Bloque de instrucciones 2
}
```

Esta estructura se lee:

Si (`if`) la condición es verdadera realiza el bloque de instrucciones 1; si no, en caso contrario o de otra forma (`else`) realiza el bloque de instrucciones 2.

Para explicar esta estructura se utilizará el programa anterior, complementando con un mensaje de “No eres mayor de edad” en caso de que la edad sea menor de 18. En la Figura 2.3 se muestra cómo se modifica la estructura selectiva simple convirtiéndose en estructura selectiva doble.

```
11
12              if(edad >= 18){
13                  System.out.println("Eres mayor de edad");
14              }
15              else{
16                  System.out.println("No eres mayor de edad");
17              }
```

Figura 2.3. Estructura selectiva doble.

Ahora si se introduce una edad mayor o igual a 18(condición verdadera) imprimirá el mensaje “Eres mayor de edad”, en caso contrario (condición falsa) imprimirá el mensaje “No eres mayor de edad”, como se muestra en la Figura 2.4.

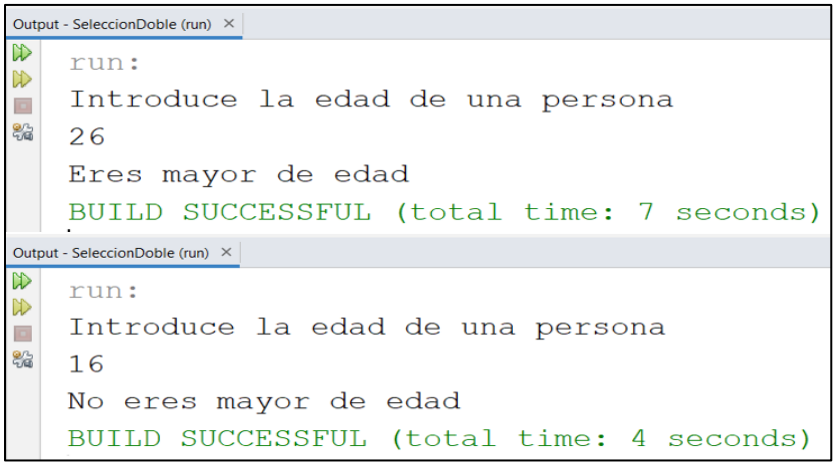


Figura 2.4. Ejecución de la estructura selectiva doble.

2.3 Estructura selectiva múltiple `switch`

La estructura selectiva múltiple `switch` permite que el flujo del programa se divida por varias ramas (más de dos) en el punto de la toma de decisiones, de acuerdo con un elemento (que puede ser una variable o una expresión) llamada selector. De tal forma que, sí el selector toma el valor 1 se ejecuta la acción 1, si se toma el valor 2 se ejecuta la acción 2, si se toma el valor 3 se ejecuta la acción 3, y así sucesivamente hasta N. También puede contener la instrucción default (de otra forma), la cual puede ser utilizada para el valor N o para enviar un mensaje cuando no exista el valor introducido.

La estructura `switch` puede reemplazar a la estructura `if`, para simplificar el proceso de decisión. y ofrecer mayor claridad al código.

Sintaxis de estructura selectiva múltiple:

```
switch(selector){
case valor_1:
     acción_1;
     break;
case valor_2:
     acción_2;
     break;
case valor_3:
     acción_3;
     break;
case valor_N:
```

```
            acción_N;
            break;
      default:
            último caso o mensaje de no existir la opción introducida.
}
```

Como se puede observar la estructura `switch` utiliza la instrucción `break` (romper), esta sirve para salir de las opciones (`case`) una vez realizadas las acciones correspondientes y no entrar a los demás casos (`case`). Para el caso N, y si no utiliza la instrucción `default`(de otra forma), puede omitir la instrucción `break` ya que es el último caso (`case`). Si utiliza la instrucción `default` (de otra forma), tampoco es necesario usar la instrucción `break` (romper).

En la Figura 2.5 se muestra un fragmento de programa, utilizando la estructura múltiple `switch`.

```
11              System.out.print("Seleccione una opción (1..3): ");
12              opcion = lee.nextInt();
13              switch(opcion){
14                  case 1:
15                      System.out.println("Seleccionó la opción 1");
16                      break;
17                  case 2:
18                      System.out.println("Seleccionó la opción 2");
19                      break;
20                  case 3:
21                      System.out.println("Seleccionó la opción 3");
22                      break;
23              }
```

Figura 2.5. Uso de la estructura múltiple `switch`.

En la Figura 2.6 se puede apreciar la ejecución del programa con los diferentes valores que puede tomar (del 1 al 3), así como ¿Qué sucede si se introduce un valor no válido? (en este caso el 4).

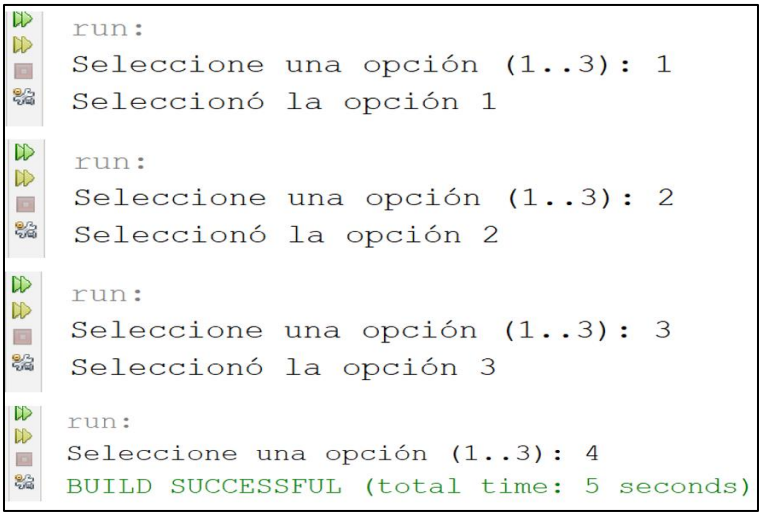


Figura 2.6. Ejecución de estructura múltiple (`switch`)

Se agregará la instrucción `default` para mayor claridad del programa, lo cual podrá visualizar en la Figura 2.7 y en la Figura 2.8 se mostrará la ejecución para el caso de que se introduzca un valor distinto como el número 4.

```
11              System.out.print("Seleccione una opción (1..3): ");
12              opcion = lee.nextInt();
13              switch(opcion){
14                  case 1:
15                      System.out.println("Seleccionó la opción 1");
16                      break;
17                  case 2:
18                      System.out.println("Seleccionó la opción 2");
19                      break;
20                  case 3:
21                      System.out.println("Seleccionó la opción 3");
22                      break;
23                  default:
24                      System.out.println("¡Esa opción no existe!");
25              }
```

Figura 2.7. Uso de instrucción `default`.

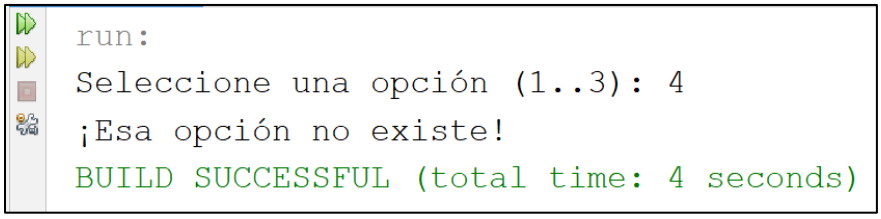


Figura 2.8. Ejecución del programa con el uso de instrucción `default`.

La estructura selectiva múltiple `switch` es muy flexible por lo que permite aplicarse de diferentes formas. En la figura 2.9 muestra un fragmento de programa en el que se pide que se introduzca el número de mes (del 1 al 12) y de acuerdo con el número que se ingrese el mensaje podría ser “Este mes tiene 30 días”, “Este mes tiene 31 días” o “Este mes tiene 28 o 29 días”.

```
11              System.out.print("Introduce un número de mes (1..12) ");
12              numMes = obj.nextInt();
13              switch(numMes){
14                  case 1,3,5,7,8,10,12:
15                      System.out.println("Este mes tiene 31 días");
16                      break;
17                  case 2:
18                      System.out.println("Este mes tiene 28 o 29 días");
19                      break;
20                  case 4,6,9,11:
21                      System.out.println("Este mes tiene 30 días");
22                      break;
23                  default:
24                      System.out.println("No existe ese número de mes");
25              }
```

Figura 2.9. Fragmento de programa usando la estructura `switch` y `default`.

Observe que si el valor del selector (numMes) es 1, 3, 5,7,8,10 o 12 se realizará la misma acción, esto es "Este mes tiene 31 días", si el valor del selector (numMes) es 2 se realiza la acción "Este mes tiene 28 o 29 días" y si el valor del selector (NumMes) es 4, 6, 9 o 11 se realizará la misma acción ósea "Este mes tiene 30 días". En caso de que se ingrese un valor diferente a los permitidos (del 1 al 12) el mensaje será "No existe ese número de mes".

En la Figura 2.10 se muestra la ejecución del programa con diferentes valores de entrada incluyendo un valor incorrecto.

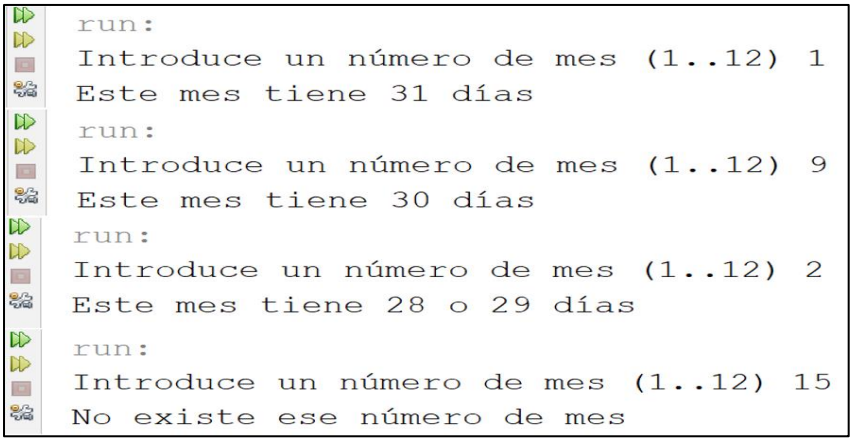


Figura 2.10. Ejecución del programa con diferentes valores.

También se pueden utilizar cadenas como selector en la instrucción `switch`. En la Figura 2.11 se muestra un fragmento de programa plasmando el uso de la instrucción switch con cadena, solicitando que se ingrese el día de la semana (lunes, martes, miércoles, jueves, viernes, sábado o domingo), dependiendo del día ingresado se enviará un mensaje que puede ser: "Inicio de semana", "Mediados de semana", "Inicia fin de semana", "Fin de semana" o "Ese día no existe" para el caso de introducir una cadena que no sea un día de la semana. Como Java es sensible a las mayúsculas y minúsculas se agregó el método `toUpperCase()`, este método convierte la cadena ingresada en mayúsculas.

```
11              System.out.print("Introduce un día de la semana: ");
12              diaSemana = s.next();
13              switch(diaSemana.toUpperCase()){
14                  case "LUNES":
15                      System.out.println("Inicio de semana");
16                      break;
17                  case "MARTES","MIERCOLES","JUEVES":
18                      System.out.println("Mediados de semana");
19                      break;
20                  case "VIERNES":
21                      System.out.println("Inicia fin de semana");
22                      break;
23                  case "SABADO","DOMINGO":
24                      System.out.println("Fin de semana");
25                      break;
26                  default:
27                      System.out.println("Ese día no existe");
28              }
```

Figura 2.11. Uso de `switch` con cadena y método `toUpperCase()`.

En la Figura 2.12 se muestra la ejecución del programa con diferentes valores.

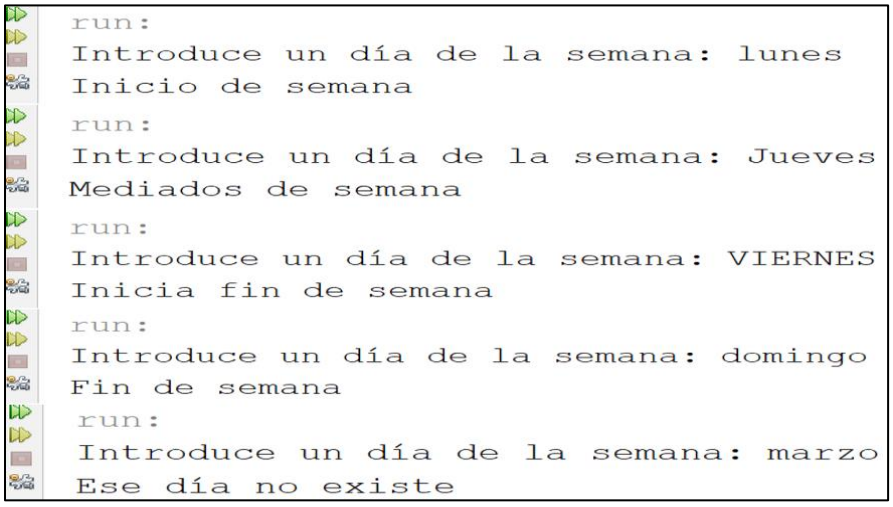


Figura 2.12. Ejecución del programa con diferentes valores.

2.4 Estructura selectiva anidada o en cascada

Existen numerosas situaciones en el desarrollo de la solución de problemas en el que luego de tomar una decisión y seguir un camino, es necesario tomar otra decisión. Luego de evaluar esa condición se toma otro camino, y nuevamente podemos tener que tomar otra decisión, este proceso podría repetirse varias veces, para lo cual estaríamos aplicando la estructura selectiva anidada o en cascada.

Dicho de otra forma, una sentencia `if` es anidada cuando alguno de sus caminos, sin importar si es verdadero o falso, es también una sentencia `if`.

Sintaxis:

```
if(condición 1){
     sentencia 1
}
else if(condición 2){
     sentencia 2
}
else if(condición 3){
     sentencia 3
}
else if(condición 4){
     sentencia 4
}
else{
     sentencia 5
     }
```

Suponga que tiene 3 valores enteros a = 20, b = 15 y c = 10, y desea imprimir el número mayor de los tres. Para determinar que a (20) es el mayor, es necesario cumplir con dos condiciones:

1. a > b y,
2. a > c

Tal y como se muestra en el siguiente fragmento de programa

```
if(a>b){
   if(a>c){
      System.out.println("El número mayor es "+a);
   }
}
```

Evalúa que a > b, si es verdadero, entra al siguiente if para evaluar que a > c, y si es verdadero imprime “El número mayor es 20”.

En las Figuras 2.13 y 2.14 se presentará dos formas de hacer el programa anterior, utilizando if anidados o en cascada.

```
2  package capitulo2;
3  public class Anidado1 {
4      public static void main(String[] args) {
5          int a = 26, b = 116, c = 37;
6
7          if(a>b){
8              if(a>c){
9                  System.out.println("El número mayor es "+a);
10             }
11             else{
12                 System.out.println("El número mayor es "+c);
13             }
14         }else{
15             if(b>c){
16                 System.out.println("El número mayor es "+b);
17             }
18             else{
19                 System.out.println("El número mayor es "+c);
20             }
21         }
22     }
23 }
```

Figura 2.13. Forma 1 de usar if anidados o en cascada.

```
2   package capitulo2;
3   public class Anidado2 {
4       public static void main(String[] args) {
5           int a = 26, b = 116, c = 37;
6
7           if((a>b)&&(a>c)){
8               System.out.println("El número mayor es "+a);
9           }
10          else if(b>c){
11              System.out.println("El número mayor es "+b);
12          }
13          else{
14              System.out.println("El número mayor es "+c);
15          }
16      }
17
18  }
```

Figura 2.14. Forma 2 de utilizar `if` anidado o en cascada.

2.5 Programas resueltos

Ejercicio 1. Dado como dato el sueldo de un trabajador, aplíquele un aumento del 12% si su sueldo es inferior a $1200.00. Calcule e imprima el nuevo sueldo del trabajador.

Entrada: Sueldo del trabajador (`sueldo`)

Proceso: Si el `sueldo < 1200` se incrementa el 12%

```
sueldoNuevo = sueldo +sueldo * descuento;
```

Salida: Impresión de resultado (`sueldoNuevo`).

En la Figura 2.15 se muestra el Ejercicio 1 resuelto, utilizando la estructura selectiva simple.

```java
package capitulo2;

import java.util.Scanner;

public class SeleccionSimple {
    public static void main(String[] args) {
        float sueldo, sueldoNuevo;
        final float descuento =0.12f;
        Scanner s = new Scanner(System.in);

        System.out.print("Introduce el sueldo del trabajador ");
        sueldo = s.nextFloat();
        if(sueldo < 1200){  // Estructura selectiva simple
            sueldoNuevo = sueldo + sueldo * descuento;
            System.out.println("El nuevo sueldo del trabajador es "+sueldoNuevo);
        }
    }
}
```

Figura 2.15. Ejercicio 1 codificado, usando estructura selectiva simple.

Explicación:

Línea 9. Se declara una constante de tipo float para el descuento del 12%.

Línea 13. Se lee el sueldo del trabajador.

Línea 14. Se evalúa el sueldo del trabajador. Si (sueldo < 1200) entonces realiza las instrucciones que están entre las llaves { }, es decir las líneas 15 y 16.

En la Figura 2.16 se puede apreciar la ejecución del programa.

```
run:
Introduce el sueldo del trabajador 1100
El nuevo sueldo del trabajador es 1232.0
BUILD SUCCESSFUL (total time: 12 seconds)
```

Figura 2.16. Ejecución del programa Ejercicio 1.

Ejercicio 2. Hacer un programa en Java que reciba como entrada dos números enteros. Calcule e imprima el mayor de ellos.

Entrada: Dos números enteros (`n1, n2`).

Proceso: `Si n1 > n2 entonces n1 es el mayor, en caso contrario n2 es el mayor.`

Salida: Impresión de resultado según los valores ingresados.

En la Figura 2.17 se muestra el Ejercicio 2 resuelto, utilizando la estructura selectiva doble.

```
2   package capitulo2;
3
4   import java.util.Scanner;
5
6   public class SeleccionDoble {
7       public static void main(String[] args) {
8           int n1, n2, mayor;
9           Scanner s = new Scanner(System.in);
10
11          System.out.println("Introduzca el primer número");
12          n1 = s.nextInt();
13          System.out.println("Introduzca el segundo número");
14          n2 = s.nextInt();
15          if(n1 > n2){
16              mayor = n1;
17          }else{
18              mayor = n2;
19          }
20          System.out.println("El número mayor es "+ mayor);
21      }
22  }
```

Figura 2.17. Ejercicio 2 codificado, usando estructura selectiva doble.

Explicación:

Línea 8. Se declara una variable (`mayor`) para guardar el valor mayor de dos enteros.

Líneas 12 y 14. Lectura de dos enteros (`n1 y n2`).

Línea 15. Se evalúan los dos enteros. Si (`n1 > n2`) entonces se realiza la instrucción que está entre las llaves `{}`, es decir línea 16 (`mayor = n1`), guardando el número mayor en la variable mayor. En caso contrario (`else`) se ejecuta la línea 18 (`mayor = n2`).

Línea 20. Se imprime el resultado, según el valor que tenga la variable mayor.

En la Figura 2.18 se puede apreciar la ejecución del programa, mostrando los dos posibles resultados que se pueden tener, esto es `n1 > n2 o n2 > n1`.

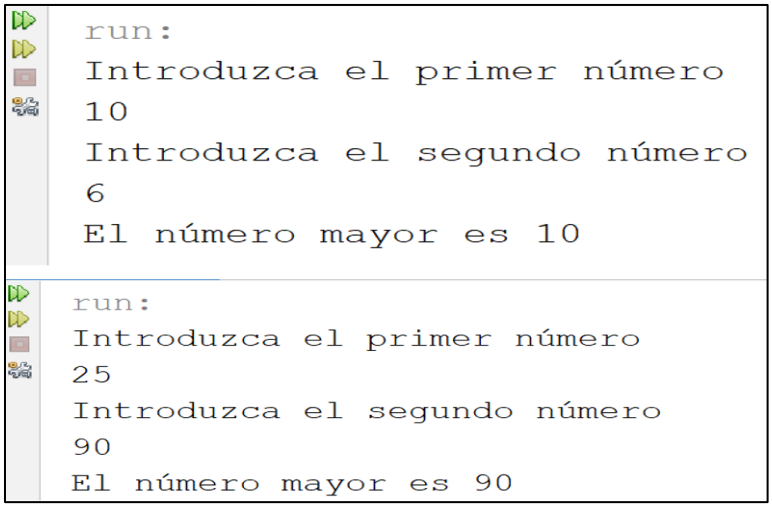


Figura 2.18. Ejecución del programa Ejercicio 2.

Ejercicio 3. Hacer un programa en Java que reciba como entrada un número entero. Calcule e imprima si el número es par o impar.

Entrada: Número entero (n).

Proceso: Se realiza operación aritmética (división con residuo)

```
resultado = n%2;
```

Si resultado == 0 entonces el número es par, si no el número es impar.

Salida: Impresión de resultado según valor ingresado.

En la Figura 2.19 se muestra el Ejercicio 3 resuelto, utilizando la estructura selectiva doble.

```java
package capitulo2;

import java.util.Scanner;

public class SeleccionDoble1 {
    public static void main(String[] args) {
        int n, resultado;
        Scanner s = new Scanner(System.in);

        System.out.print("Introduce un número: ");
        n = s.nextInt();
        resultado = n%2;
        if(resultado == 0){
            System.out.println("El número "+n+" es par");
        }else{
            System.out.println("El número "+n+" es impar");
        }
    }
}
```

Figura 2.19. Ejercicio 3 codificado, usando estructura selectiva doble.

Explicación:

Línea 8. Se declara una variable (`resultado`) para guardar el valor de la operación aritmética (n%2).

Líneas 12. Lectura del número entero(`n`).

Línea 14. Se evalúa la condición. Si (`n == 0`) entonces se realiza la instrucción que está entre las llaves `{ }`, es decir línea 15 que imprime que el número es par (`"El número "+n+" es par"`). En caso contrario (`else`) se ejecuta la línea 17 e imprime que el número es impar (`"El número "+n+" es impar"`).

En la Figura 2.20 se puede apreciar la ejecución del programa, mostrando los dos posibles resultados que se pueden tener, esto es si n es par (`resultado == 0`) o impar.

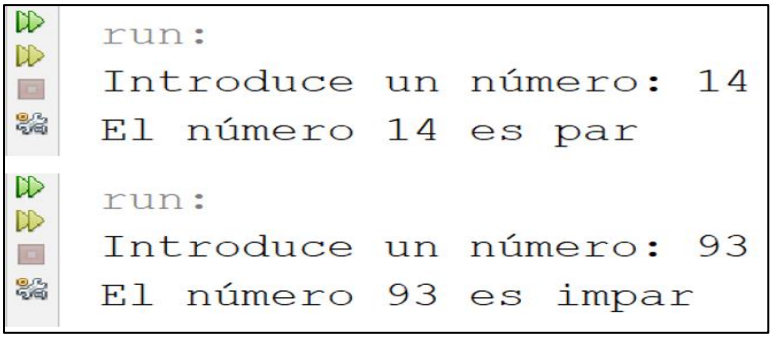


Figura 2.20. Ejecución del programa Ejercicio 3.

Ejercicio 4. Dado como datos el nivel y el sueldo de un trabajador de la empresa FACILITO, hacer un programa que incremente el sueldo del trabajador según los criterios que se muestran en la Tabla 2.1 de abajo. Al final imprima el nivel, el sueldo anterior y el nuevo sueldo del trabajador.

Tabla 2.1. Criterios.

Nivel	Aumento
1	12%
2	10%
3	8%
4	6%

Entrada: Sueldo del trabajador (`sueldo`) y nivel (`nivel`).

Proceso: Calcula el sueldo nuevo (`sueldoN`) de acuerdo con el nivel.

```
caso 1:realiza sueldoN = sueldo * 1.12
caso 2:realiza sueldoN = sueldo * 1.10
caso 3:realiza sueldoN = sueldo * 1.08
caso 4:realiza sueldoN = sueldo * 1.06
```

Salida: Impresión de sueldo nuevo (`sueldoN`) y nivel (`nivel`).

En la Figura 2.21 se muestra el Ejercicio 4 resuelto, utilizando la estructura selectiva múltiple `switch`.

```
2  package capitulo2;
3
4  import java.util.Scanner;
5
6  public class SeleccionMultipleSwitch1 {
7      public static void main(String[] args) {
8        double sueldo, sueldoN = 0;
9        int nivel;
10       Scanner s = new Scanner(System.in);
11
12         System.out.print("Introduzca el sueldo del trabajador... ");
13         sueldo = s.nextFloat();
14         System.out.print("Introduzca el nivel(1..4): ");
15         nivel = s.nextInt();
16         switch(nivel){
17             case 1:
18                 sueldoN = sueldo * 1.12;
19                 break;
20             case 2:
21                 sueldoN = sueldo * 1.10;
22                 break;
23             case 3:
24                 sueldoN = sueldo * 1.08;
25                 break;
26             case 4:
27                 sueldoN = sueldo * 1.06;
28                 break;
29         }
30         System.out.println("El nivel del trabajador es "+ nivel);
31         System.out.printf("y su nuevo sueldo es %.2f\n",sueldoN);
32     }
33 }
```

Figura 2.21. Ejercicio 4 codificado, usando estructura selectiva múltiple `switch`.

Explicación:

Líneas 8 y 9. Declaración de variables.

Líneas 13 y 15. Lectura del sueldo del trabajador (`sueldo`) y nivel (`nivel`).

Líneas de la 16 a la 29. Estructura selectiva múltiple `switch`. Se evalúa:

`caso 1: sueldoN = sueldo * 1.12`, sale de la estructura hacia la línea 30.

`caso 2: sueldoN = sueldo * 1.10`, sale de la estructura hacia la línea 30.

`caso 3: sueldoN = sueldo * 1.08`, sale de la estructura hacia la línea 30.

`caso 4: sueldoN = sueldo * 1.06`, sale de la estructura hacia la línea 30.

Líneas 30 y 31. Impresión de resultados (`nivel` y `sueldoN`).

En la Figura 2.22 se puede apreciar la ejecución del programa, mostrando diferentes resultados.

```
run:
Introduzca el sueldo del trabajador... 3800
Introduzca el nivel(1..4): 3
El nivel del trabajador es 3
y su nuevo sueldo es 4104.00
run:
Introduzca el sueldo del trabajador... 6200
Introduzca el nivel(1..4): 4
El nivel del trabajador es 4
y su nuevo sueldo es 6572.00
run:
Introduzca el sueldo del trabajador... 1100
Introduzca el nivel(1..4): 1
El nivel del trabajador es 1
y su nuevo sueldo es 1232.00
run:
Introduzca el sueldo del trabajador... 1750
Introduzca el nivel(1..4): 2
El nivel del trabajador es 2
y su nuevo sueldo es 1925.00
```

Figura 2.22. Ejecución del programa Ejercicio 4.

También podría hacerse con `if` anidados como se muestra en el fragmento del programa de la Figura 2.23.

```
16              if(nivel == 1){
17                      sueldoN = sueldo * 1.12;
18              }else if(nivel == 2){
19                      sueldoN = sueldo * 1.10;
20              }else if(nivel == 3){
21                      sueldoN = sueldo * 1.08;
22              }else
23                      sueldoN = sueldo * 1.06;
24
```

Figura 2.23. Uso de estructura selectiva anidada.

Ejercicio 5. Hacer un programa tal que dado como dato un número entero, determine e imprima si este número es positivo, negativo o cero.

Entrada: Número entero (num).

Proceso: `si (num == 0) entonces imprime "El número es cero" sino si (num > 0) entonces imprime "El número es positivo" sino imprime "El número es negativo".`

Salida: Impresión de resultado según entero ingresado.

En la Figura 2.24 se muestra el Ejercicio 5 resuelto, utilizando la estructura selectiva anidada.

```
1   package capitulo2;
2
3   import java.util.Scanner;
4
5   public class IfAnidados {
6       public static void main(String[] args) {
7           int num;
8           Scanner s = new Scanner(System.in);
9
10          System.out.print("Introduzca un número entero... ");
11          num = s.nextInt();
12          if(num == 0){
13              System.out.println("El número es cero");
14          }else if(num > 0){
15              System.out.println("El número es positivo");
16          }else{
17              System.out.println("El número es negativo");
18          }
19      }
20  }
```

Figura 2.24. Ejercicio 5 codificado, usando estructura selectiva anidada.

Explicación:

Líneas 7. Declaración de variable entera (`num`).

Líneas 11. Lectura del número (`num`).

Líneas de la 12 a la 18. Estructura selectiva anidada. Se evalúa:

```
si (num == 0) entonces se va a ejecutar la línea 13.
    sino si (num > 0) entonces se va a ejecutar la línea 15.
    sino se va a ejecutar la línea 17.
```

Líneas 13, 15 y 17. Impresión de mensajes, "`El número es cero`", "`El número es positivo`", "`El número es negativo`", respectivamente.

En la Figura 2.25 se puede apreciar la ejecución del programa, mostrando diferentes resultados.

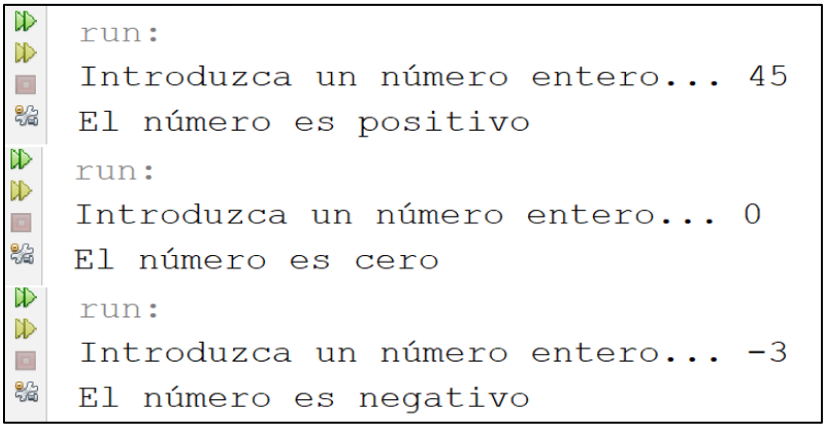


Figura 2.25. Ejecución del programa Ejercicio 5.

Ejercicio 6. En una tienda efectúan un descuento a los clientes dependiendo del monto de la compra. El descuento se hace de acuerdo con los siguientes criterios: Si el monto

Es menor que $500 no hay descuento.

Está comprendido entre $500 y $1,000 inclusive 5% de descuento.

Está comprendido entre $1,000 y $7,000 inclusive 11% de descuento.

Está comprendido entre $7,000 y $15,000 inclusive 18% de descuento.

Es mayor a $15,000 25% de descuento.

Hacer un programa en Java tal que, dado el monto de la compra de un cliente, determine lo que debe pagar.

Entrada: Monto de la compra (`compra`).

Proceso: Se evalúa la compra para determinar el descuento

si compra < $500 no hay descuento.

si compra >= $500 y <= $1000 5% de descuento.

si compra > $1000 y >= $7,000 11% de descuento.

si compra > $7,000 y <= $15,000 18% de descuento.

si compra > $15,000 25% de descuento.

Salida: Impresión de resultado según valor ingresado.

En la Figura 2.26 se muestra el Ejercicio 6 resuelto, utilizando la estructura selectiva anidada.

```
2   package capitulo2;
3
4   import java.util.Scanner;
5
6   public class IfAnidados2 {
7       public static void main(String[] args) {
8           double compra, pago;
9           Scanner s = new Scanner(System.in);
10
11          System.out.print("Monto de la compra $");
12          compra = s.nextDouble();
13          if(compra < 500){
14              pago = compra;
15          }else if(compra <= 1000){
16              pago = compra - compra * 0.05;
17          }else if(compra <= 7000){
18              pago = compra - compra * 0.11;
19          }else if(compra <= 15000){
20              pago = compra - compra * 0.18;
21          }else{
22              pago = compra - compra * 0.25;
23          }
24          System.out.printf("El cliente debe pagar $%.2f\n",pago);
25      }
26  }
```

Figura 2.26. Ejercicio 6 codificado, usando estructura selectiva anidada.

Explicación:

Líneas 8. Declaración de variables `double` (`compra y pago`).

Líneas 12. Lectura del monto de la compra (`compra`).

Líneas de la 15 a la 23. Estructura selectiva anidada. Se evalúa:

```
si (compra < 500) entonces pago = compra;
   sino si (compra <= 1000) entonces pago = compra - compra * 0.05;
   sino si (compra <= 7000) entonces pago = compra - compra * 0.11;
   sino si (compra <= 15000) entonces pago = compra - compra * 0.18;
sino pago = compra - compra * 0.25;
```

Línea 24. Impresión de resultado `"El cliente debe pagar ",pago`.

En la Figura 2.27 se puede apreciar la ejecución del programa, mostrando diferentes resultados.

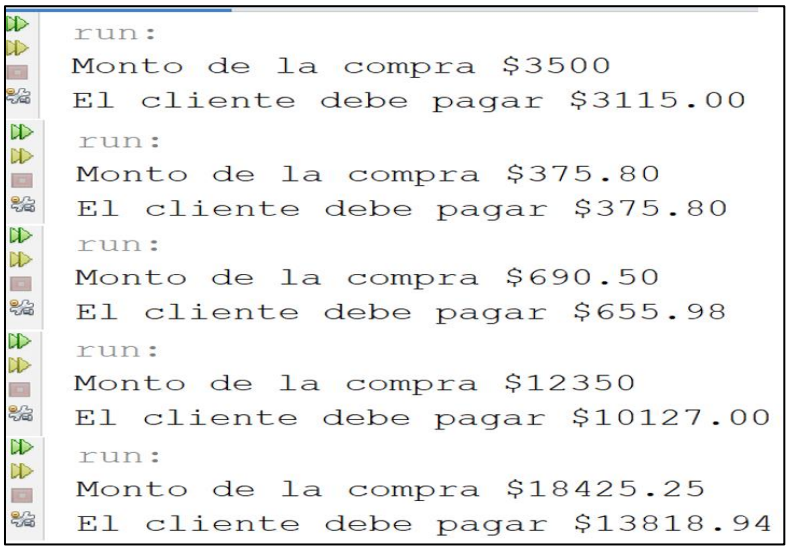


Figura 2.27. Ejecución del programa Ejercicio 6.

Ejercicio 7. El costo de las llamadas telefónicas internacionales depende de la zona geográfica en la que se encuentra el país destino y del número de minutos hablados. En la siguiente Tabla 2.2 se presenta el costo del minuto por zona. A cada zona se le ha asociado una clave.

Tabla 2.2. Costo del minuto por zona.

Clave	Zona	Precio por minuto
12	América del norte	2.0
15	América central	2.2
18	América del sur	4.5
19	Europa	3.5
23	Asia	6
25	África	6
29	Oceanía	5

Hacer un programa que le permita calcular e imprimir el costo total de una llamada.

Entrada: Leer clave y número de minutos (`clave, numMinutos`).

Proceso: Determinar el costo total mediante la estructura selectiva múltiple switch.

Salida: Impresión de resultado según los valores ingresados.

En la Figura 2.28 se muestra el Ejercicio 7 resuelto, utilizando la estructura selectiva múltiple `switch`.

Explicación:

Líneas 8 y 9. Declaración de variables.

Líneas 13 y 15. Lectura de clave (`clave`) y número de minutos (`numMin`).

Líneas de la 17 a la 38. Estructura selectiva múltiple `switch`. Se evalúa:

`caso 12 : costo = numMin * 2,` sale de la estructura hacia la línea 39.

`caso 15: costo = numMin * 2.2`, sale de la estructura hacia la línea 39.

caso `18: costo = numMin * 4.5`, sale de la estructura hacia la línea 39.

`caso 19: costo = numMin * 3.5`, sale de la estructura hacia la línea 39.

`caso 23 , 25: costo = numMin * 6`, sale de la estructura hacia la línea 39.

`caso 29: costo = numMin * 5`, sale de la estructura hacia la línea 39.

Explicación:

Líneas 8 y 9. Declaración de variables.

Líneas 13 y 15. Lectura de clave (`clave`) y número de minutos (`numMin`).

Líneas de la 17 a la 38. Estructura selectiva múltiple `switch`. Se evalúa:

`case 12 : costo = numMin * 2,` sale de la estructura hacia la línea 39.

`case 15: costo = numMin * 2.2`, sale de la estructura hacia la línea 39.

case `18: costo = numMin * 4.5`, sale de la estructura hacia la línea 39.

`case 19: costo = numMin * 3.5`, sale de la estructura hacia la línea 39.

`case 23 , 25: costo = numMin * 6`, sale de la estructura hacia la línea 39.

`case 29: costo = numMin * 5`, sale de la estructura hacia la línea 39.

Línea 39. Impresión de resultado, costo de la llamada (`costo`).

```java
package capitulo2;

import java.util.Scanner;

public class SeleccionMultipleSwitch2 {
    public static void main(String[] args) {
        int clave, numMin;
        double costo = 0;
        Scanner s = new Scanner(System.in);

        System.out.print("Introduce la clave: ");
        clave = s.nextInt();
        System.out.print("Introduce número de minutos: ");
        numMin = s.nextInt();

        switch(clave){
            case 12:
                costo = numMin * 2;
                break;
            case 15:
                costo = numMin * 2.2;
                break;
            case 18:
                costo = numMin * 4.5;
                break;
            case 19:
                costo = numMin * 3.5;
                break;
            case 23,25:
                costo = numMin * 6;
                break;
            case 29:
                costo = numMin * 5;
                break;
            default:
                System.out.println("Esa clave no existe");
        }
        System.out.println("El costo total de la llamada es $"+costo);
    }
}
```

Figura 2.28. Ejercicio 7 codificado, usando estructura selectiva múltiple switch.

En la Figura 2.29 se puede apreciar la ejecución del programa, mostrando diferentes resultados.

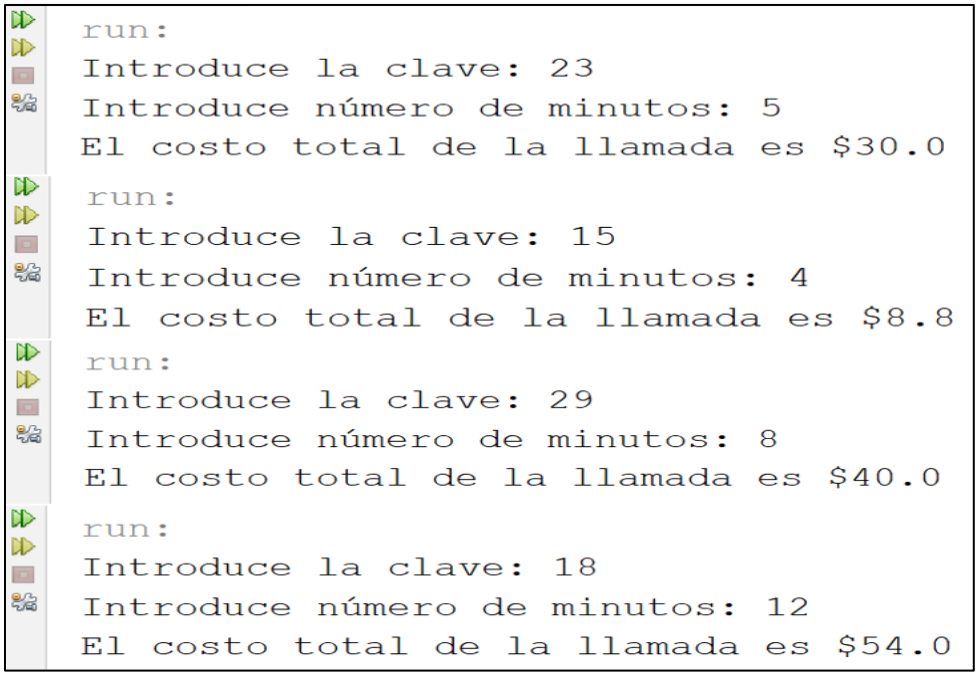


Figura 2.29. Ejecución del programa Ejercicio 7.

Ejercicio 8. Hacer un programa que reciba como entrada una hora de la siguiente forma: hora, minutos y segundos. El programa debe imprimir la hora un segundo más tarde. Ejemplo:

Si la hora ingresada es 11:45:59, debe imprimir 11:46:00 (hora actual + 1 segundo).

Entrada: Leer hora, minutos y segundos (`hora, min y seg`).

Proceso: Determinar la nueva hora mediante la estructura selectiva anidada.

Salida: Impresión de resultado según los valores ingresados.

En la Figura 2.30 se muestra el Ejercicio 8 resuelto, utilizando la estructura selectiva anidada.

```java
package capitulo2;

import java.util.Scanner;

public class IfAnidados3 {
    public static void main(String[] args) {
        int hora, min, seg;
        Scanner s = new Scanner(System.in);
        System.out.print("Introduce la hora: ");
        hora = s.nextInt();
        System.out.print("Introduce los minutos: ");
        min = s.nextInt();
        System.out.print("Introduce los segundos: ");
        seg = s.nextInt();
        seg++;
        if(seg > 59){
            seg = 0;
            min++;
            if(min > 59){
                min = 0;
                hora++;
                if(hora > 23){
                    hora = 0;
                }
            }
        }
        if((seg < 10)||(min < 10)||(hora <10)){
            System.out.printf("La hora más un segundo es: %02d:%02d:%02d\n",hora,min,seg);
        }else
        System.out.println("La hora más un segundo es: "+hora+":"+min+":"+seg);
    }
}
```

Figura 2.30. Ejercicio 8 codificado, usando estructura selectiva anidada.

Explicación:

Líneas. Declaración de variables.

Líneas 11,13 y 15. Lectura de hora(`hora`), minutos (`min`) y segundos (`seg`).

Línea 16. Iniciamos incrementando los segundos (`seg++`), ya que básicamente es lo que debe hacer el programa.

Líneas de la 17 a la 27. Estructura selectiva anidada. Se evalúa:

`si (seg > 59) entonces seg = 0` e incrementar los minutos en uno (`min++`)

`si (min > 59) entonces min = 0` e incrementar la hora en uno (`hora++`)

```
si (hora > 23) entonces hora = 0
```

Líneas de la 28 a la 31. Impresión de resultado se realiza con una estructura selectiva doble. Con la finalidad de que si la hora (`hora`), minuto (`min`) o segundo (`seg`) sean menor de 10 se anteponga un cero tal como 01, 02, 03 y así sucesivamente (líneas 28 y 29 en caso contario realiza la línea 31.

En la Figura 2.31 se puede apreciar la ejecución del programa, mostrando diferentes resultados.

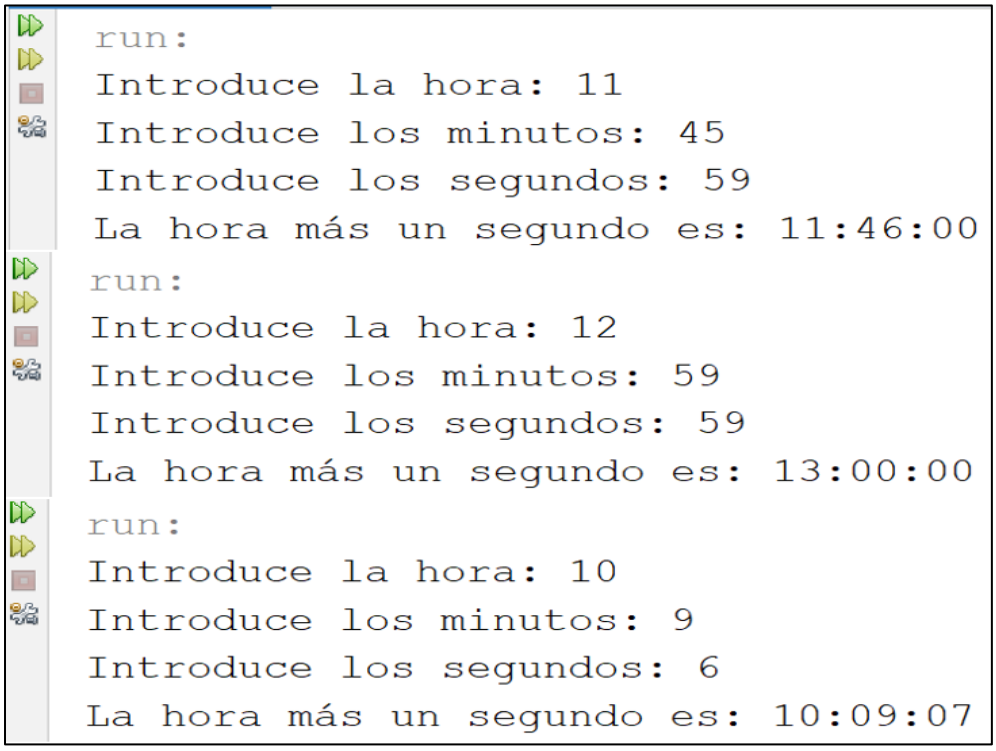


Figura 2.31. Ejecución del programa Ejercicio 8.

2.6 Programas propuestos.

1. En una tienda departamental hacen un descuento del 20% a todos los clientes que compren más de $3000.00. Hacer un programa que calcule e imprima cual es el descuento y cuánto debe pagar por su compra.
2. Dados dos números enteros. Hacer un programa que los imprima ordenados de forma descendente.
3. Hacer un programa tal que, dados dos números enteros, determine si un número es divisor del otro.
4. Dados cuatro números enteros a, b, c y d. Hacer un programa que calcule e imprima el resultado de las siguientes expresiones (si d = 0 imprimir mensaje apropiado):

$\frac{(a-c)^2}{d}$ y $\frac{(a-b)^3}{d}$

5. Hacer un programa. Leer 3 números enteros diferentes e imprimirlos ordenados de mayor a menor.
6. Hacer un programa. Leer 3 números enteros diferentes e imprimirlos ordenados de menor a mayor
7. Hacer un programa que permita realizar operaciones aritméticas básicas, según el operador aritmético ingresado y de acuerdo con la Tabla 2.3.

Tabla 2.3. Operadores.

Operador	Operación aritmética
+	Suma
-	Resta
*	Multiplicación
/	División

Imprima el operador ingresado y el resultado de la operación aritmética.

8. Hacer un programa que imprima cuantas cifras tiene un número entero introducido por teclado, dicho número estará comprendido entre 0 y 99999.
9. Una agencia de automóviles ofrece descuentos y bonos en sus unidades del 2024, de acuerdo con la Tabla 2.4 Si el modelo no se encuentra en la Tabla 2.4, entonces no tiene ningún tipo de descuento ni bono.

Tabla 2.4. Modelos.

Modelo	Descuento	Bono
Serie X1	5%	10,000
Serie X2	5%	10,000
New sky	8	12,000
Infinity	10	14,000
Nova Serie	10	15,000
Rapidísimo	12	18,000
Serie Z	15	20,000

Dado el modelo y el precio del vehículo, hacer un programa que calcule el nuevo precio. Al final imprimir el modelo, el precio original y el nuevo precio.

10. Hacer un programa que lee por teclado un número de mes y muestra el nombre del mes correspondiente.
11. Hacer un programa que reciba como entrada los tres lados de un triángulo y determine si el triángulo es escaleno, isósceles o equilátero.
12. Hacer un programa que solicite al usuario una fecha (día, mes y año) y muestre la fecha correspondiente al día siguiente. No considerar los años bisiestos.
13. Hacer un programa que solicite al usuario una fecha (día, mes y año) e imprima si la fecha es o no correcta. No considerar los años bisiestos.

14. Dado como datos, el sueldo, el nivel del trabajador y el número de horas extras trabajadas, hacer un programa que calcule e imprima el sueldo de un trabajador. Los empleados con categoría mayor a 3 no se les pagan las horas extras. Sólo se pueden pagar máximo 30 horas extras y se calculan mediante la Tabla 2.5.

Tabla 2.5. Calculo sueldo.

Nivel	Precio de hora extra
1	$100
2	$150
3	$200

15. Hacer un programa tal que, dado como dato una temperatura en grados Fahrenheit, determine el deporte que es apropiado para practicar a esa temperatura, teniendo en cuenta la Tabla 2.6.

Tabla 2.6. Temperaturas.

Deporte	Temperatura
Natación	>85
Tenis	70 < temp <=85
Golf	32 < temp <= 70
Esquí	10 < temp <= 32
Caminata	<= 10

16. En un hospital determinan el costo total de un paciente mediante la Tabla 2.7. Hacer un programa tal que, dado el número de días que un paciente es hospitalizado, el tipo de enfermedad y su edad (edad entre 13 y 21 años implica un costo adicional del 15%), calcule e imprima el costo total que representa un paciente.

Tabla 2.7.Costo.

Tipo de enfermedad	Costo por día
a	$550
b	$695
c	$720
d	$845

17. Hacer un programa que calcule e imprima si un año es bisiesto. Un año es bisiesto cuando es múltiplo de 4, por ejemplo 1984; sin embargo, los años que son múltiplos de 100 sólo son bisiestos cuando también son múltiplos de 400; Por ejemplo, 1800 no es bisiesto, mientras que 2024, sí lo es.
18. Una empresa tiene una vacante y necesita que el aspirante a ocupar el puesto cumpla con las siguientes condiciones:

    Su nivel de estudios sea egresado o pasante y con una experiencia mayor a 2 años, o bien que sea titulado y la experiencia mayor a 5 años.

Hacer un programa que determine e imprima si el empleado reúne las condiciones para ocupar el puesto.

19. Hacer un programa que al recibir como entrada un número entero de cuatro dígitos, determine si todos los dígitos del número son pares.
20. Hacer un programa que reciba como entrada el número de lados de un polígono e imprima si es un triángulo, cuadrilátero, pentágono o hexágono.

3. ESTRUCTURAS REPETITIVAS

A veces nos damos cuenta de que es necesario repetir varias veces un conjunto de instrucciones. Aunque esas instrucciones son las mismas, los resultados varían. Por ejemplo, si necesito calcular el promedio de calificaciones de un grupo de 40 alumnos o si deseo saber los intereses que se obtienen de una inversión en un determinado número de meses o años, o simplemente imprimir n veces un letrero. El conjunto de instrucciones que se ejecuta repetidamente recibe el nombre de ciclo.

> Para Jiménez Marín & Pérez Montes (2016)
>
> Una estructura repetitiva es un ciclo que contiene un bloque de instrucciones que se ejecutan varias veces; cada ejecución o repetición del ciclo se llama iteración. Los ciclos simplifican la escritura de programas, minimizando el código duplicado. Cualquier fragmento de programa que se necesite ejecutar varias veces seguidas es susceptible de incluirse en un ciclo. (pág. 57).

> Para Corona Nakamura & Ancona Valdez (2011)
>
> Bucle, ciclo o iteración es un segmento de un programa, cuyas instrucciones se repiten un número finito o indefinido de veces mientras se cumpla una determinada condición. En cada iteración se comprueba si la condición es verdadera, terminando el ciclo cuando es falsa. En algún momento la condición tiene que ser falsa ya que en caso contrario el bucle se hará infinito. (pág. 79).

En Java existen tres tipos de estructuras repetitivas:

- `for`
- `while`
- `do-while`

3.1 Estructura repetitiva `for`

La estructura repetitiva `for` es la estructura más utilizada, repite un conjunto de instrucciones un número determinado de veces. Es decir, se sabe de antemano cuántas veces se debe repetir un bucle, ciclo o iteración (ciclo controlado por un contador). El número de repeticiones no depende de las instrucciones dentro del ciclo. El número de repeticiones se obtiene del planteamiento del problema o de una lectura que indica que el número de iteraciones se debe realizar para N veces.

Sintaxis de la estructura repetitiva `for`:

```
for (inicialización; condición; contador)
{
      Cuerpo del ciclo
}
```

Como se puede observar el encabezado de la estructura repetitiva for consta de 3 partes: inicialización, condición y actualización; separadas de un punto y coma (`;`).

inicialización. En esta parte del encabezado se inicializa la variable que llevará el control del ciclo, indicando con que valor inicia el ciclo y se hace mediante el operador de asignación (=). Por ejemplo, `i = 1`. También puede llevar una declaración de variable en caso de que no haya sido declarada antes, tal como `int i = 1;`

condición. Es una expresión lógica que determina cuándo finalizará el ciclo o bucle. Cuando se evalúa la expresión lógica y ésta es verdadera el ciclo se repite. En el momento en que la expresión lógica es falsa se sale del ciclo.

contador. Define cómo cambiará la variable de control cada vez que se repite el bucle o ciclo, puede ser mediante incremento o decremento (`++, --`).

Las 3 partes pueden tener una o varias instrucciones, las cuales deben ir separadas por comas. La condición nos lleva al valor final. También es posible omitir cualquiera de las partes del encabezado del ciclo `for`, incluso las 3, pero los separadores de punto y coma (`;`) deben escribirse siempre.

En la estructura repetitiva `for`, primero se ejecutará la inicialización, posteriormente se evaluará la condición y, en caso de ser verdadera, se ejecutarán las instrucciones que componen el cuerpo del ciclo. Después, se realizará el contador (incremento o decremento según sea el caso) y se volverá a verificar la condición. Cuando la condición se vuelve falsa, en la siguiente evaluación se terminará el ciclo for. Si la condición nunca se vuelve falsa, la estructura repetitiva nunca terminará y el ciclo se repetirá indefinidamente hasta que se detenga en forma manual.

Por ejemplo, el siguiente fragmento de programa (Figura 3.1) calcula la suma de los primeros 5 números enteros positivos.

```
6           int i, suma = 0;
7           for (i = 1;  i <= 5; i++)   //Encabezado del ciclo for
8           {
9               suma += i;              //Cuerpo del ciclo
10          }
11          System.out.println("La suma de los primeros 5 números positivos es "+suma);
```

Figura 3.1. Suma de los primeros 5 números positivos.

La variable `i` ubicada en el encabezado del ciclo `for` (línea 7) se llama variable de control y mantiene el control del número de repeticiones. Esta variable `i`, se inicializa en 1 e inmediatamente después se evalúa la condición (`i <= 5`). Si el resultado de la condición es verdadero, se ejecuta el cuerpo del ciclo (líneas de la 8 a la 10). De otra forma, se termina la ejecución de la estructura repetitiva `for` y el control fluye a la sentencia que sigue, en este caso la línea 11 que imprime el resultado de la suma de los 5 primeros números positivos. Cada vez que se ejecuta el cuerpo del ciclo, también se ejecuta el operador de incremento (`i++`) y entonces se evalúa la condición.

En la Tabla 3.1 se muestra cómo sería la validación y verificación del ciclo `for`.

Tabla 3.1. Validación y verificación del ciclo `for`.

Número de repetición	Valor de i	Valor de suma
		0
1	1	1
2	2	3
3	3	6
4	4	10
5	5	15

En la Figura 3.2 se puede apreciar la ejecución del programa y corroborar que es correcta la validación y verificación del ciclo `for`.

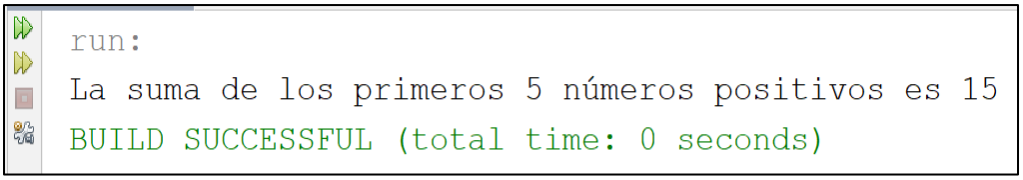


Figura 3.2. Ejecución del programa con ciclo for.

El componente inicialización también puede incluir una declaración de la variable de control tal como:

```
for (int i = 1;  i <= 5; i++)
```

en lugar de

```
int i;
for (i = 1;  i <= 5; i++)
```

La variable de control se puede inicializar en cualquier valor, aunque casi siempre es en 0 o 1.

En el ejemplo anterior, el contador incrementa la variable de control en 1, pero podemos incrementarla con otros valores diferentes de 1, incluso, con valores negativos. En la Figura 3.3 se tiene varios ejemplos de cómo se puede utilizar la estructura repetitiva `for`.

```
5           for (int i = -1; i > -10; i--)
6           {
7               System.out.println("i = "+i);//donde i toma los valores de -1, -2,-3, -4, ... -9
8           }
9           for (int j = 0; j < 100; j += 5)//donde j toma los valores de 0, 5,10, 15, ... 95
10          {
11              System.out.println("j = "+j);
12          }
13          for (int k =2; k < 40; k *= 2)//donde k toma los valores de 2, 4, 8, 16, 32
14          {
15              System.out.println("k = "+k);
16          }
17          for (int p =100; p > 0; p--)//donde p toma los valores de 100, 99,98, 97, ... 1
18          {
19              System.out.println("p = "+p);
20          }
21          for (char letra = 'A'; letra <= 'Z'; letra++) //donde letra toma los valores de
22          {                                              // A, B, C, D, E, ... Z
23              System.out.println(letra);
24          }
```

Figura 3.3. Ejemplos de uso de la estructura repetitiva `for`.

El cuerpo del ciclo (lo que está entre llaves `{ }`) puede llevar varias instrucciones, de hecho, si el ciclo solo cuenta con una instrucción, las llaves no son necesarias.

Por ejemplo, si desea calcular el promedio de 3 calificaciones de un alumno, el cuerpo del ciclo tendría varias instrucciones como se puede apreciar en la Figura 3.4.

En las líneas 8 y 9. Se presenta la declaración de las variables a utilizar donde:

`i` será la variable de control del ciclo `for`.

`cal` se utilizará para guardar la calificación cada vez que entre al ciclo.

`suma` inicializada en cero, se utilizará para hacer la suma de cada una de las calificaciones.

```
1
2   package capitulo3;
3
4   import java.util.Scanner;
5
6   public class UsoFor {
7       public static void main(String[] args) {
8           int i, cal;
9           float suma = 0, promedio;
10          Scanner s = new Scanner(System.in);
11
12          for (i = 1; i < 4; i++)
13          {
14              System.out.println("Introduce la calificación "+i);
15              cal = s.nextInt();
16              suma = suma + cal;
17          }
18          promedio = suma/3;
19          System.out.println("El promedio del alumno es "+promedio);
20      }
21  }
```

Figura 3,4. Cuerpo de ciclo for con varias instrucciones.

Línea 12. Encabezado del ciclo for, la variable i se inicializa con el valor 1, se verifica que la condición i < 4 sea verdadera y se procede a entrar al cuerpo del ciclo.

Línea 13. Inicio del cuerpo del ciclo.

Línea 14. impresión en pantalla solicitando calificación i

Línea 15. entrada de calificación en variable `cal`.

Línea 16. suma = suma + cal, para guardar la suma de las calificaciones.

Línea 17. Fin del cuerpo del ciclo. Se retorna al encabezado del ciclo (línea 12) se incrementa la variable de control `i` en uno (`i++`), y se vuelve a verificar la condición `i < 4`, si es verdadera vuelve a entrar al ciclo y así sucesivamente hasta que la condición `i < 4` sea falsa.

Línea 18. Cuando la condición es falsa sale del ciclo y realiza `promedio = suma/3`.

Línea 19. Imprime el resultado (`promedio de 3 calificaciones`).

3.2 Estructura repetitiva `while`

Esta estructura es adecuada para utilizar en un ciclo, cuando no se conoce el número de veces que éste se ha de repetir. Ese número de veces depende de una condición que se

debe inicializar antes del ciclo y debe evaluarse en el ciclo. Si la condición es verdadera el ciclo se sigue repitiendo, en caso contrario el ciclo termina.

Sintaxis de la estructura repetitiva `while`:

```
/* Forma 1 */
while (condición)
     instrucción;          //este cuerpo de ciclo solo lleva
                           //una instrucción, por lo tanto, no
                           //necesita llaves{}

/* Forma 2 */
while(condición)
{
     Cuerpo del ciclo
}
```

Esta estructura se interpreta de la siguiente forma, mientras la condición sea verdadera se ejecuta el cuerpo del ciclo, en caso contario sale del ciclo.

Por lo anterior es necesario que la condición este inicializada en verdadera antes de ser evaluada, ya que en caso contrario nunca entrará al ciclo. Además, dentro del cuerpo del ciclo debe haber un enunciado que afecte el resultado de la condición si no se volverá un ciclo infinito.

3.2.1 Contadores, acumuladores, centinelas y banderas

Contadores y acumuladores. Se refiere a las variables que van incrementando o decrementando su valor a lo largo de la ejecución del programa, normalmente son de tipo numérico. Algunos ejemplos son:

```
contador = contador + 1;
total = total + y;
r = r * 4;
```

Los contadores y acumuladores se deben inicializar antes de entrar al ciclo. Para la suma o resta se inicializa en cero antes de entrar al ciclo y para la multiplicación se inicializa en 1, porque si vale cero todo lo que se multiplique será cero.

Un **contador** es una forma de controlar un ciclo. Es una variable cuyo valor se incrementa o decremento en una cantidad constante cada vez que se produce un determinado suceso o acción en cada repetición; dicha variable controla o determina la cantidad de veces que se repite un proceso o dato. Su sintaxis es:

```
nombreContador = nombreContador + valorConstante;
```

Si es un decremento, se coloca el signo menos (-) en lugar del signo más (+).

Primero se realiza una operación de inicialización y posteriormente el incremento o decremento dentro del ciclo. La inicialización consiste en asignarle al contador un valor de inicio. se situará antes del ciclo. En la Figura 3.5 se muestran diferentes formas de utilizar un contador ya sea de incremento o decremento.

```
 9          int cuenta = 1;
10          /*Incremento*/
11          cuenta++;                   /*Forma 1*/
12          ++cuenta;                   /*Forma 2*/
13          cuenta = cuenta + 1;        /*Forma 3*/
14          cuenta += 1;                /*Forma 4*/
15          /*Decremento*/
16          cuenta--;                   /*Forma 1*/
17          --cuenta;                   /*Forma 2*/
18          cuenta = cuenta - 1;        /*Forma 3*/
19          cuenta -= 1;                /*Forma 4*/
20
```

Figura 3.5. Uso de contador (incremento y decremento).

Un **acumulador** realiza la misma función que un contador con la diferencia de que el incremento o decremento es variable en lugar de constante. Es una variable que almacena cantidades resultantes de operaciones sucesivas. Su sintaxis es:

```
nombreAcumulador = nombreAcumulador + valorVariable;
```

Si es un decremento, se coloca el signo menos (-) en lugar del signo más (+). En la Figura 3.6 se muestran diferentes formas de utilizar un acumulador ya sea de incremento o decremento.

```
10          int acumula = 0;
11          /*Incremento*/
12          acumula = acumula + valor;      /*Forma 1*/
13          acumula += valor;               /*Forma 2*/
14          /*Decremento*/
15          acumula = acumula - valor;      /*Forma 1*/
16          acumula -= valor;               /*Forma 1*/
17
```

Figura 3.6. Uso de acumulador (incremento y decremento).

Centinela. Es una variable que inicia con un valor, luego dentro del ciclo este valor cambia, haciendo falsa la condición del ciclo y por lo tanto finalizará el ciclo (el usuario puede determinar cuándo hacerlo). La repetición controlada por centinela se considera como una repetición indefinida (se desconoce el número de repeticiones).

El usuario puede introducir S o N para indicar si se desea continuar o no. El ciclo terminará cuando la respuesta del usuario sea “n” o “N”. También se podría manejar algún valor numérico como respuesta para terminar el ciclo.

Bandera. Es una variable que puede tomar solo dos valores (verdadero o falso), a lo largo de la ejecución del programa.

> Para Joyanes Aguilar & Zahonero Martínez (2011)
>
> Su valor se inicializa normalmente en falso (false) y, antes de la entrada al ciclo, se redefine cuando un suceso especifico ocurre dentro de éste. Un ciclo controlado por una bandera se ejecuta hasta que se produce el suceso anticipado y se cambia el valor del indicador (pág. 138).

A continuación, se verán ejemplos con contadores, acumuladores, centinelas y banderas.

En la Figura 3.7 se presenta un fragmento del programa que realiza la suma de los primeros 5 números positivos, utilizando la estructura repetitiva while controlado por un contador.

```
6           int i=1,suma=0;
7
8           while(i <= 5)
9           {
10              suma += i;
11              i++;
12          }
13          System.out.println("La suma de los primeros 5 números positivos es "+suma);
```

Figura 3.7. Ciclo `while` controlado por un contador.

En la línea 7 se encuentra el contador `i`, inicializado en 1, también está la variable suma inicializada en cero que hará la función de un acumulador.

En la línea 8 se encuentra el encabezado del ciclo `while`, evaluando que mientras `i <= 5` que entre al ciclo y ejecute instrucciones que se encuentran dentro de las llaves { } (líneas 10 y 11).

En la línea 10 está la instrucción `suma += i;` la cual irá sumando y acumulando los valores de 1, 2, 3, 4, 5.

El contador incrementa su valor (`i++`) y se regresa el control a la línea 8 para evaluar nuevamente la condición. Como el valor de `i` es igual a 2, la condición se cumple y vuelve a entrar al ciclo, y así sucesivamente hasta que i sea mayor de 5. Cuando i es mayor de 5 ya no entra al cuerpo del ciclo y se dirige a la línea 13 para imprimir el resultado, el cual se puede ver en la Figura 3.8.

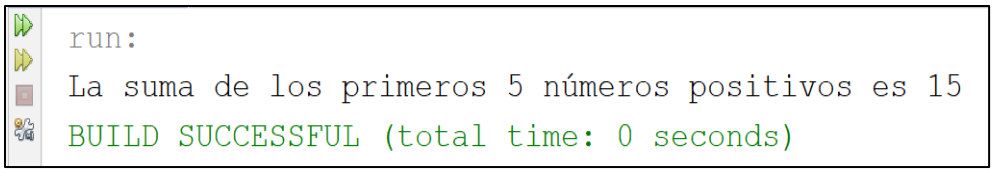


Figura 3.8. Ejecución del ciclo `while` controlado por un contador.

Ahora veremos en la Figura 3.9 un ejemplo de un ciclo `while` controlado por un acumulador. Se sumará un conjunto de números enteros (los cuales entraran por teclado) hasta que la suma sea mayor de 100. Al final imprimir el resultado de la acumulación.

```
11          int n, suma = 0;
12          while(suma <= 100)
13          {
14              System.out.print("Introduce un número entero: ");
15              n = lee.nextInt();
16              suma = suma + n;
17          }
18          System.out.println("La suma es "+suma);
```

Figura 3.9. Ciclo `while` controlado por un acumulador.

El acumulador es la variable `suma` y empieza con un valor de cero (línea 11), en la línea 12 se evalúa la condición mientras `suma <= 100`, como `suma = 0`, la condición es verdadera por lo tanto entra al ciclo, solicita un número `n` y lo acumula en `suma` (`suma = suma + n`). En este momento `suma` cambia su valor, si el valor es menor o igual a 100 volverá a entrar al ciclo en caso contrario saldrá del ciclo e imprimirá la suma (línea 18). Cuando se usa un ciclo controlado por un acumulador no se sabe exactamente cuantas veces se repetirá el ciclo, ya que depende de los valores que se introduzcan. En la Figura 3.10 podrá ver la ejecución de este programa.

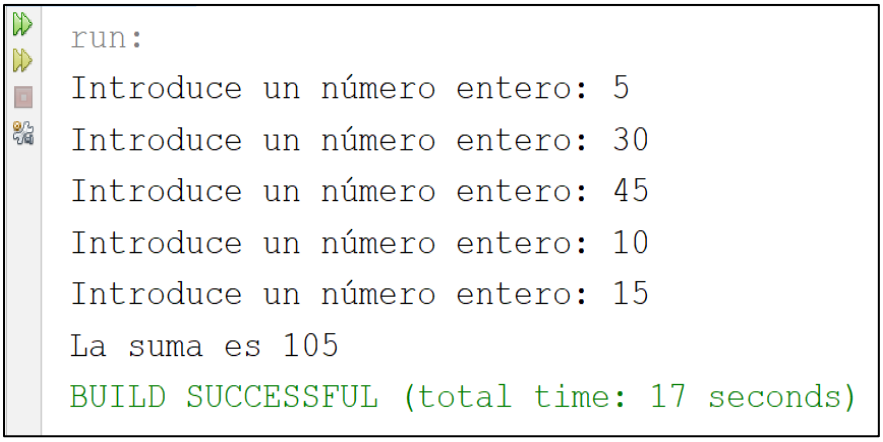


Figura 3.10. Ejecución del ciclo `while` controlado por un acumulador.

El ejemplo de Figura 3.11 muestra un ciclo while controlado por un centinela. Se calculará el producto (multiplicación) entre un conjunto de números reales, hasta que el número introducido sea cero. Al final imprimir el resultado de la multiplicación.

```
10          float n, mult = 1;
11          System.out.print("Introduce un número real: ");
12          n = lee.nextFloat();
13          while(n != 0)
14          {
15              mult = mult * n;
16              System.out.print("Introduce un número real: ");
17              n = lee.nextFloat();
18          }
19          System.out.printf("El producto es %.2f\n",mult);
```

Figura 3.11. Ciclo `while` controlado por un centinela.

En este caso el centinela es el cero, mientras el número introducido sea diferente de cero `(n != 0)` entrará al ciclo, de lo contario no entra al ciclo. Por lo tanto, si el primer número introducido es cero (línea 12) jamás entrará al ciclo y la variable `mult` será igual a 1, ya que ese es el valor con el que se inicializó (línea 10). La variable `mult` está inicializada con el valor 1, para que cuando realice la primera operación de multiplicación el resultado sea el número introducido, Sí inicializa con cero entonces multiplicaría por cero y el resultado siempre sería cero. En la Tabla 3.2 puede observar la ejecución manual o prueba de escritorio del programa para un mejor entendimiento.

Tabla 3.2. Prueba de escritorio del ciclo while controlado por un centinela.

Iteración	mult	n
	1	3.5
1	3.5	4.8
2	16.80	6.7
3	112.56	10.45
4	1176.25	0

En la Figura 3.12 se puede apreciar la ejecución del programa y corroborar que es correcta la validación y verificación del ciclo `while` controlado por un centinela.

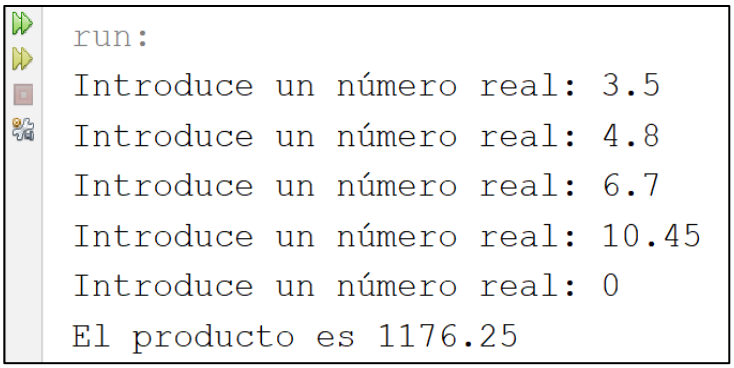


Figura 3.12. Ejecución del ciclo `while` controlado por un centinela.

El ejemplo de Figura 3.13 muestra un ciclo `while` controlado por una bandera. Se desea leer un dato numérico x con valor mayor que cero para calcular la función `f(x)=x*log(x)`.

```
11          double f,x = 0;
12          boolean xpositivo;
13          xpositivo = false;
14          while(!xpositivo){
15              System.out.println("\n Valor de x: ");
16              x = ent.nextDouble();
17              xpositivo = (x > 0.0);
18          }
19          f = x*Math.log(x);
20          System.out.println("f("+x+")="+f);
```

Figura 3.13. Ciclo `while` controlado por una bandera.

La bandera `xpositivo` se utiliza para representar que el dato leído sea mayor que cero, por lo cual se inicializa con un valor de `false` antes de que se ejecute el ciclo y de que se lea el dato de entrada (x). El ciclo debe continuar mientras el número capturado sea negativo o cero. En el momento en que se introduzca un número mayor a cero, `xpositivo` cambia su valor a `true`. Por lo anterior, la condición del ciclo debe ser `!xpositivo`, ya que ésta es `true` cuando `xpositivo` es `false`. En la Figura 3.14 podrá ver la ejecución de este programa.

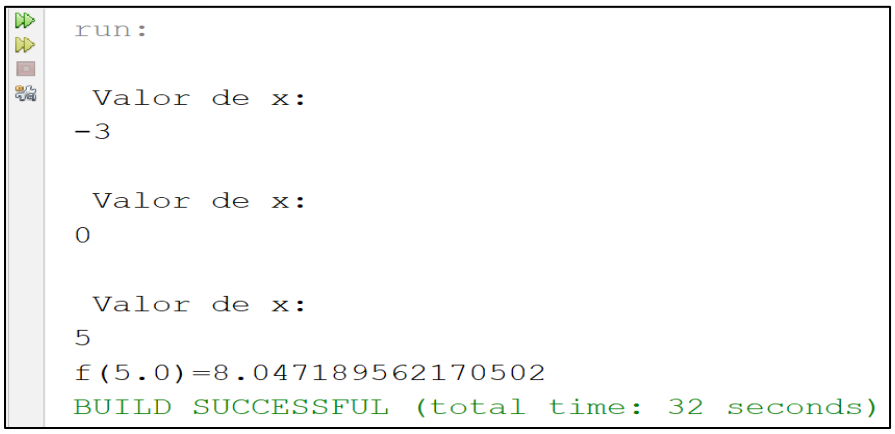


Figura 3.14. Ejecución del ciclo `while` controlado por una bandera.

3.3 Estructura repetitiva `do-while`

La estructura repetitiva `do-while` se utiliza para especificar un ciclo que se ejecuta por lo menos una vez. Su sintaxis es la siguiente:

```
/* Forma 1 */
do instrucción while (condición);

/* Forma 2 */
do{
      Cuerpo del ciclo
}while(condición);
```

Esta estructura se interpreta de la siguiente forma: Hacer (`do`) el cuerpo del ciclo (lo que está entre llaves `{}`), mientras la condición sea verdadera (`while (condición)`). Cuando la condición es falsa, sale del ciclo.

Al igual que la estructura repetitiva `while`, la estructura `do-while` puede ser controlada por un contador, un acumulador, un centinela o bandera.

La estructura `do-while` es similar a la estructura `while`, la diferencia radica en el cuerpo del ciclo, que en el caso de `do-while` siempre se ejecuta por lo menos una vez.

Se presentarán los mismos ejemplos realizados con la estructura while, con la finalidad de que aprecie la similitud de las estructuras.

En la Figura 3.15. se muestra un ejemplo (fragmento de programa) del uso de la estructura repetitiva `do-while`, con el ejemplo utilizado en el tema anterior (Figura 3.7. Ciclo `while` controlado por un contador).

```
6           int n = 1;
7           int suma = 0;
8           do{
9               suma += n;
10              n++;
11          }while(n <= 5);
12          System.out.println("La suma de los primeros 5 números positivos es "+suma);
```

Figura 3.15. Estructura repetitiva `do-while` controlada por un contador.

Como la evaluación de la condición `n <= 5` (línea 11) se encuentra después del cuerpo del ciclo (líneas 9 y 10), primero entra al ciclo realiza lo que está entre las llaves `{}` y después evalúa nuevamente la condición, si esta es verdadera regresa al cuerpo del ciclo, ejecuta las instrucciones por segunda ocasión, y así sucesivamente hasta que `n` sea mayor que 5 (condición = falsa). En la Figura 3.16 se puede visualizar la ejecución del programa con el mismo resultado presentado en la Figura 3.8 del tema anterior.

```
run:
La suma de los primeros 5 números positivos es 15
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 3.16. Ejecución del ciclo `do-while` controlado por un contador.

En la Figura 3.17 se muestra un fragmento del programa, utilizando la estructura `do-while` controlada por un acumulador.

```
10          int n, suma = 0;
11          do{
12              System.out.print("Introduce un número entero: ");
13              n = lee.nextInt();
14              suma = suma + n;
15          }while(suma <= 100);
16          System.out.println("La suma es "+suma);
```

Figura 3.17. Estructura repetitiva do-while controlada por un acumulador.

En la Figura 3.18 se muestra la ejecución de la estructura do-while controlada por un acumulador, con los mismos valores utilizados en la Figura 3.10.

```
run:
Introduce un número entero: 5
Introduce un número entero: 30
Introduce un número entero: 45
Introduce un número entero: 10
Introduce un número entero: 15
La suma es 105
BUILD SUCCESSFUL (total time: 13 seconds)
```

Figura 3.18. Ejecución del ciclo do-while controlado por un acumulador.

A continuación, se muestra el ejemplo de la estructura do-while controlada por un centinela y su ejecución (Figura 3.19 y 3.20) respectivamente.

```
12          do{
13
14              System.out.print("Introduce un número real: ");
15              n = lee.nextFloat();
16              if(n != 0){
17                  mult = mult * n;
18              }
19          }while(n != 0);
20          System.out.printf("El producto es %.2f\n",mult);
21      }
```

Figura 3.19. Estructura do-while controlada por un centinela.

En este código se hizo una modificación, en la línea 16 se agregó una condición simple, con la finalidad de que, si se captura un cero, no se realice la multiplicación de la línea 17 (`mult = mult * n`). Si no existiera la condición y se captura un cero, se realiza la multiplicación y el resultado sería cero.

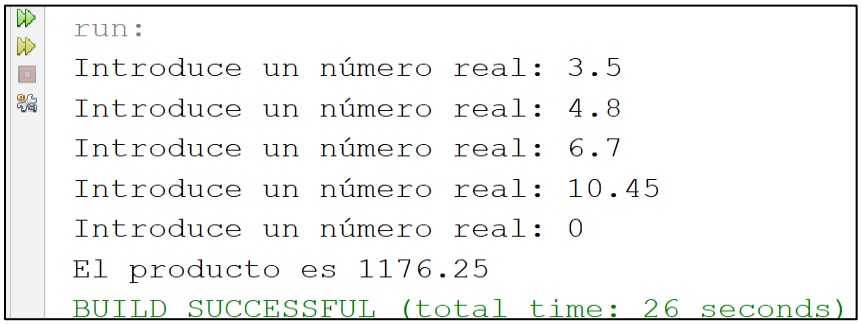


Figura 3.20. Ejecución del ciclo `do-while` controlado por un centinela.

En la Figura 3.21. se muestra fragmento de programa con la estructura `do-while` controlada por una bandera. El código es muy similar a la estructura repetitiva `while` de la Figura 3.13.

```
13              do{
14                  System.out.println("\n Valor de x: ");
15                  x = ent.nextDouble();
16                  xpositivo = (x > 0.0);
17              }while(!xpositivo);
18              f = x*Math.log(x);
19              System.out.println("f("+x+")="+f);
```

Figura 3.21. Estructura `do-while` controlada por una bandera.

En este caso no hay necesidad de inicializar la variable `xpositivo`, ya que en este tipo de estructura primero entra al ciclo y después evalúa la condición.

Por último, en la Figura 3.22 podrá ver la ejecución del programa con los mismos valores presentados en la Figura 3.14.

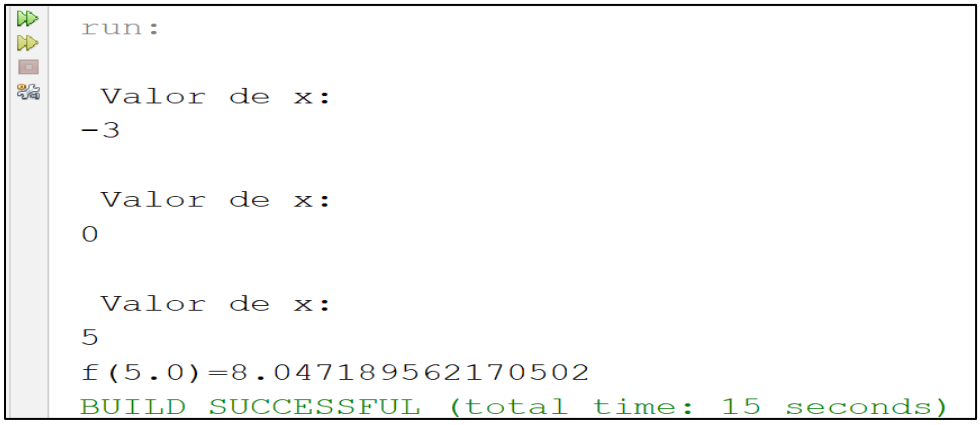


Figura 3.22. Ejecución del ciclo `do-while` controlado por una bandera.

Finalmente, dependiendo del análisis del problema se utilizará la estructura repetitiva optima, en la Tabla 3.3 se muestran las características de cada una de las estructuras.

Tabla 3.3. Características de las estructuras repetitivas.

Estructura repetitiva	Características
for	Se utiliza cuando el número de repeticiones se conoce por anticipado y puede controlarlo un contador; también es adecuado para ciclos que implican control no contable del ciclo con simples etapas de inicialización y actualización; la evaluación de la condición precede a la ejecución del cuerpo del ciclo.
while	Su uso más frecuente es cuando se desconoce el número de repeticiones. La evaluación de la condición precede a cada repetición del ciclo; el cuerpo del ciclo puede no ser ejecutado. Se debe utilizar cuando se desea saltar el ciclo si la condición es falsa
do-while	Es apropiado cuando se necesita que el ciclo se ejecute al menos una vez.

3.4 Estructuras repetitivas anidadas

De igual forma que se pueden anidar estructuras selectivas, es posible anidar ciclos, los cuales tienen un ciclo externo y uno o más internos; cada vez que se repite el externo, los internos también lo hacen; los componentes de control se vuelven a evaluar y se ejecutan todas las repeticiones requeridas.

Cada vez que se ejecuta un ciclo externo, se ejecuta en su totalidad el ciclo interno. Para mayor claridad vea el fragmento de programa de la Figura 3.23.

```
8       for (int i = 1; i <= 3; i++)
9       {
10          System.out.printf("Externo %4d\n",i);
11          for (int j = 1; j <= i; j++)
12          {
13              System.out.printf("%10s","Interno");
14              System.out.printf("%10d\n",j);
15          }
16      }
```

Figura 3.23. Ciclos for anidados.

Se dice que son ciclos anidados porque dentro de un ciclo existe otro ciclo, en este caso el ciclo for representado por la variable de control `j` (línea 11, ciclo interno) está dentro del ciclo for representado por la variable de control `i` (línea 8, ciclo externo). Cada vez que se ejecute el ciclo externo (3 veces para este caso), el ciclo interno se ejecutará en su totalidad, esto es:

Cuando sea `i = 1` (ciclo externo), el ciclo interno `j` de ejecutará una vez.

Cuando sea `i = 2` (ciclo externo), el ciclo interno `j` de ejecutará dos veces.

Cuando sea `i = 3` (ciclo externo), el ciclo interno `j` de ejecutará tres veces.

Lo anterior lo puede verificar en la ejecución del programa que se encuentra en la Figura 3.24.

```
run:
                i         j
Externo     1
    Interno             1
Externo     2
    Interno             1
    Interno             2
Externo     3
    Interno             1
    Interno             2
    Interno             3
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 3.24. Ejecución de ciclos `for` anidados.

Es posible anidar cualquier tipo de estructura repetitiva, como se verá en el siguiente fragmento de programa de la Figura 3.25. Elevar un número x a una potencia y, además preguntarle al usuario si desea realizar cálculo de otra potencia.

```
12          while(resp == 'S' || resp == 's')
13          {
14              res = 1;
15              System.out.print("Número que desea elevar a una potencia ");
16              x = s.nextInt();
17              System.out.print("¿A que potencia elevará el número? ");
18              y = s.nextInt();
19              for (cont = 1;  cont <= y; cont++)
20              {
21                  res = res * x;
22              }
23              System.out.println("La potencia de "+x+ " elevado a "+y+" es "+res );
24              System.out.println("Desea realizar otra potencia S/N ");
25              resp = s.next().charAt(0);
26          }
```

Figura 3.25. Estructuras repetitivas `while` y `for` anidadas.

En este ejemplo tenemos una estructura repetitiva `for`, dentro de una estructura repetitiva `while`, por lo tanto, están anidados. Esto quiere decir que cada vez, que se ejecute el ciclo externo `while` (dependiendo de que `resp == 'S' || resp == 's'`), se ejecutará el ciclo interno `for` en su totalidad, el número de veces dependerá del valor que se le de a la variable y (línea 18).

Cada vez que la `resp == 'S' o resp == 's'` (línea 12) entrará al ciclo while (línea 13 a la 26), se capturará un número x para elevar a una potencia y (líneas 16 y 18). Ejecutará el ciclo `for` (`res = res * x`) desde `cont = 1` hasta `cont <= y`. En la Figura 3.26 podrá observar la ejecución del programa con diferentes valores.

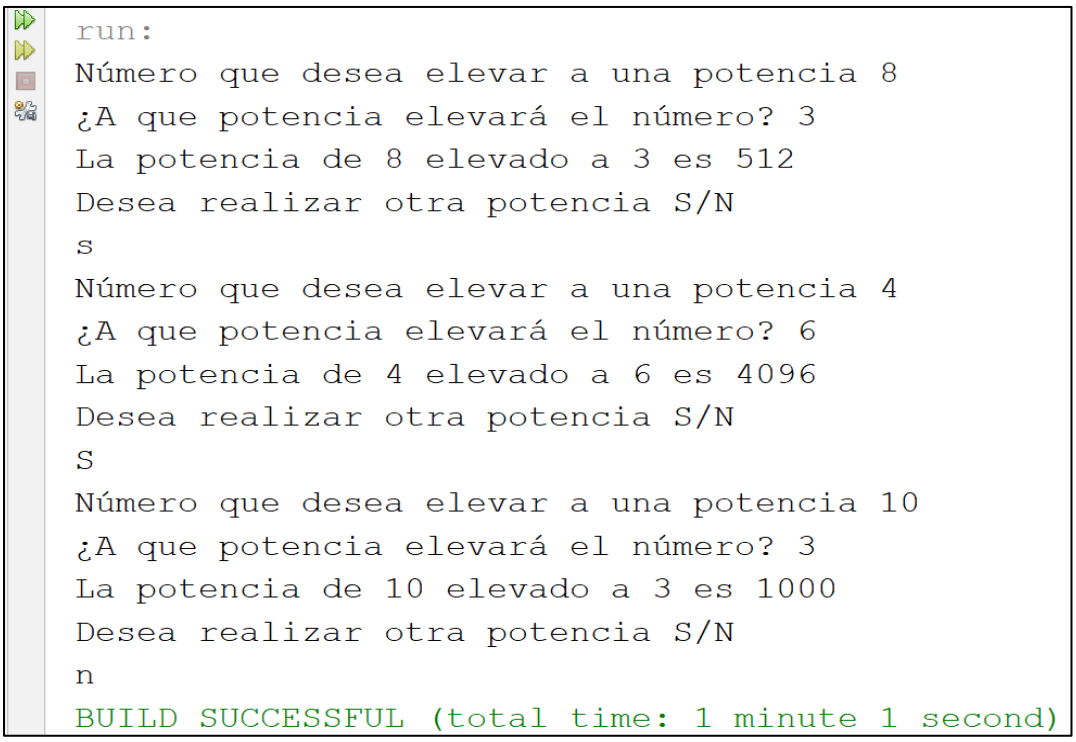


Figura 3.26. Ejecución de estructuras repetitivas `while` y `for` anidadas.

3.5 Programas resueltos

Ejercicio 1. Hacer un programa en Java tal que, dado como datos n números enteros, obtenga el total de ceros que se introduzcan por teclado.

Entrada: n . Cantidad de números a leer.

`num`. Variable para leer los números de entrada.

Proceso:

```
Para (i = 1; i <= n; i++)
     Solicitar número (num)
     Si (num == 0)
          cuentaCeros++;
```

Salida: Impresión de resultado (`cuentaCeros`).

En la Figura 3.27 se muestra el Ejercicio 1 resuelto, utilizando la estructura repetitiva `for`.

```
2   package capitulo3;
3
4   import java.util.Scanner;
5
6   public class CicloFor {
7       public static void main(String[] args) {
8           Scanner s = new Scanner(System.in);
9           int i, num,n, cuentaCeros = 0;
10
11          System.out.print("¿Cuántos números va a introducir? ");
12          n = s.nextInt();
13          for (i = 1; i <= n; i++)
14          {
15              System.out.println("Introduce el número "+i+": ");
16              num =s.nextInt();
17              if(num == 0)
18                  cuentaCeros++;
19          }
20          System.out.println("Total de ceros que ingresó son: "+cuentaCeros);
21      }
22  }
```

Figura 3.27. Ejercicio 1 codificado, usando estructura repetitiva `for`.

Explicación:

Línea 9. Se declaran las variables:

`i`. Variable de control del ciclo `for`.

`num`. Variable para leer los números.

`n`. Variable para saber cuántos números se ingresarán.

`cuentaCeros`. Variable para contar los ceros que se ingresen.

Línea 12. Se lee n (números a introducir), con este valor se puede saber cuántas veces se repetirá el ciclo `for`.

Líneas 13 a la 19. Ciclo `for`, la Tabla 3.4 muestra la ejecución manual o prueba de escritorio del ciclo, con los valores de n = 5, num = 3, 0, 10, 0 y 87.

Tabla 3.4. Ejecución manual del ciclo `for` del Ejercicio 1.

iteración	n	i	num	num == 0	cuentaCero
					0
1	5	1	3	false	
2		2	0	true	1
3		3	10	false	
4		4	0	true	2
5		5	87	false	

Línea 20. Una vez que sale del ciclo imprime el resultado (`cuentaCero = 2`).

En la Figura 3.28 se puede apreciar la ejecución del programa con los valores anteriores.

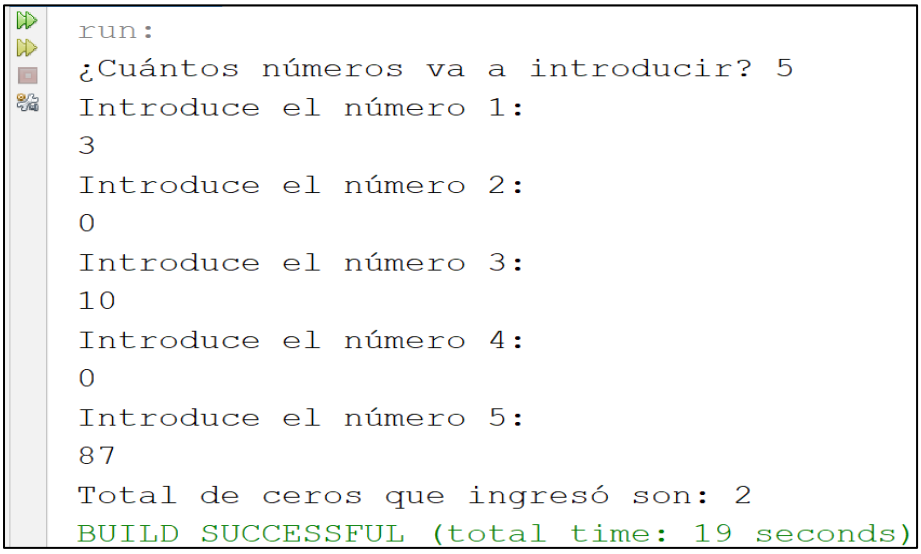


Figura 3.28. Ejecución del programa Ejercicio 1.

Ejercicio 2. Hacer un programa en Java tal que, dado un número `num`, calcule e imprima el factorial de dicho número.

Entrada: `num` . Variable a la que se le calculará el factorial.

Proceso:
```
para (int i = num; i >= 1; i--)
      fact = fact * i;
```

Salida: Impresión de resultado (`fact`).

En la Figura 3.29 se muestra el Ejercicio 2 resuelto, utilizando la estructura repetitiva `for`.

```java
package capitulo3;

import java.util.Scanner;

public class CicloFor2 {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int fact= 1, num;
        System.out.println("Programa que calcula el factorial de un número entero ");
        System.out.println("Introduce un número ");
        num = s.nextInt();
        for (int i = num; i >= 1; i--)
        {
            fact = fact * i;
        }
        System.out.println("El factorial de "+num+" es "+fact);
    }
}
```

Figura 3.29. Ejercicio 2 codificado usando estructura repetitiva `for`.

Explicación:

Línea 8. Se declaran las variables:

`num`. Variable para leer un número entero.

`fact`. Variable para calcular el factorial, se inicializa en 1.

Línea 11. Entrada del número que se calculará el factorial.

Líneas 12 a la 15. Ciclo for, en el encabezado se declara la variable de control del ciclo (`i`) la Tabla 3.5 muestra la ejecución manual o prueba de escritorio del ciclo, con el valor de `num = 5`.

Tabla 3.5. Ejecución manual del ciclo `for`

iteración	num	i	fact
	5		1
1		5	5
2		4	20
3		3	60
4		2	120
5		1	120

Línea 16. Una vez que sale del ciclo imprime el resultado (`fact = 120`).

En la Figura 3.30 se puede apreciar la ejecución del programa con el valor anterior.

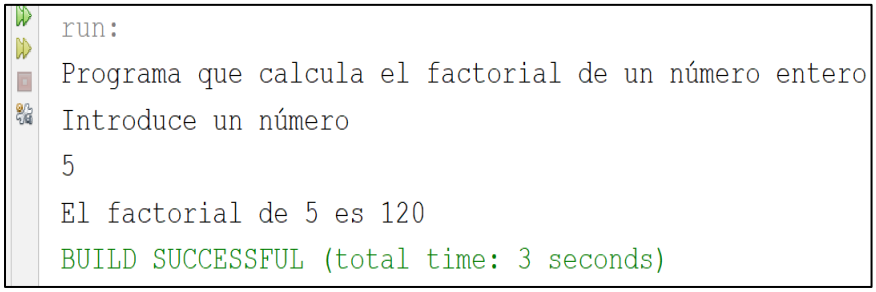


Figura 3.30. Ejecución del programa Ejercicio 2.

Ejercicio 3. Hacer un programa en Java tal que, dado un número `n`, calcule e imprima si es o no un número primo. Los números primos son aquellos que sólo se pueden dividir entre 1 y ellos mismos, es decir solo tiene 2 divisores.

Entrada: `n` . Variable de entrada para leer un número entero.

Proceso:
```
mientras (bandera == verdadera y contador < n)
    Si(n % contador == 0)
       bandera = false;
    sino
       contador++;
```

Salida: Impresión de resultado, si la bandera es `true` entonces el número es primo, si la bandera es `false`, el número no es primo.

En la Figura 3.31 se muestra el Ejercicio 3 resuelto, utilizando la estructura repetitiva `while`.

Explicación:

Líneas 8 y 9. Se declaran las variables:

`n`. Variable para leer un número entero.

`contador`. Variable inicializada en 2, llevará el conteo de divisores

`bandera`. Variable inicializada en `true`

Línea 11. Entrada del número `n`, el cual se determinará si es o no un número primo.

```java
package capitulo3;

import java.util.Scanner;

public class NumeroPrimo {
    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);
        int n, contador = 2;
        boolean bandera = true;
        System.out.print("Introduce un número ");
        n = s.nextInt();
        while(bandera && contador < n){
            if(n % contador == 0) {
                bandera = false;
            }
            else{
                contador++;
            }
        }
        if(bandera)
            System.out.println("El número "+n+" es primo");
        else
            System.out.println("El número "+n+" no es primo");
    }
}
```

Figura 3.31. Ejercicio 3 codificado, usando estructura repetitiva `while`.

Líneas 12 a la 19. El encabezado del ciclo `while` tiene dos condiciones (`bandera && contador < n`) que deben ser verdaderas para que entre al ciclo y realice las divisiones necesarias, si el número tiene más de dos divisores (es decir `n % contador == 0`) entonces el número no es primo. La Tabla 3.6 muestra la ejecución manual o prueba de escritorio del ciclo, con el valor de `n = 5`.

Tabla 3.6. Prueba de escritorio del ciclo `while`.

Iteración	bandera && contador < n	n % contador == 0	contador	bandera	n
			2	True	5
1	true	false	3		
2	true	false	4		
4	true	false	5		
5	false				

En esta prueba de escritorio se puede observar que la variable bandera nunca cambió su valor, lo que significa que el número 5 sólo tiene dos divisores (el 1 y el mismo).

Líneas 20 a la 23. La condicional dice que si bandera es verdadera imprimirá El número 5 es primo.

En la Figura 3.32 se muestra la ejecución del programa con dos ejemplos: un número primo y un número que no es primo.

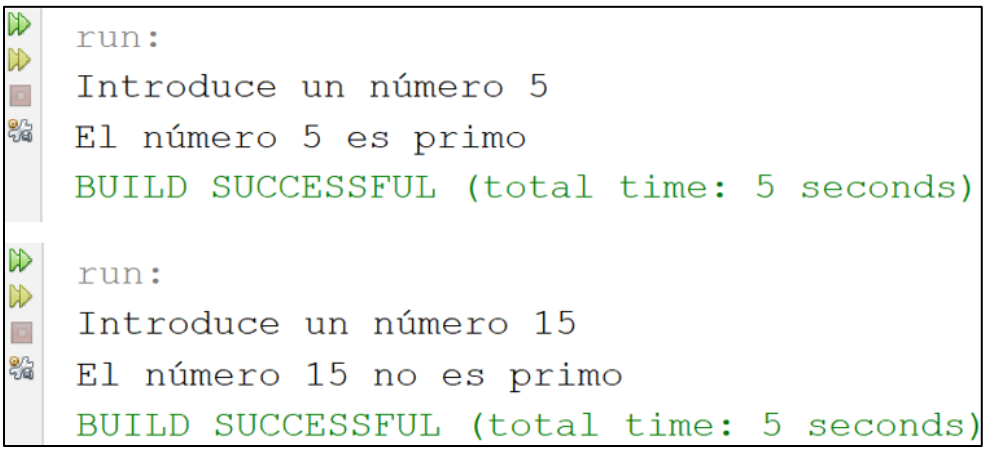


Figura 3.32. Ejecución del programa Ejercicio 3.

Ejercicio 4. Hacer un programa en Java que obtenga la suma e imprima los términos de la siguiente serie:

2, 5, 7, 10, 12, 15, 17, …,80.

Entrada: No habrá datos de entrada desde el teclado, los términos de la serie se generarán dentro del ciclo.

Proceso:

```
bandera = true; i = 2;
mientras (i <= 60)
    sumaSerie = sumaSerie + i;
    imprimir i;
    Si(bandera == true)
        i += 3;
        bandera = false;
    sino
         i += 2;
         bandera = true;
```

Salida: Impresión de la serie y suma de la serie.

En la Figura 3.33 se muestra el Ejercicio 4 resuelto, utilizando la estructura repetitiva `while`.

```java
package capitulo3;

public class Serie {
    public static void main(String[] args) {
        boolean bandera = true;
        int sumaSerie = 0, i = 2;
         System.out.println("Serie");
        while(i <= 80)
        {
            sumaSerie = sumaSerie + i;
            System.out.printf("%-3d",i);
            if(bandera)
            {
                i += 3;
                bandera = false;
            }
            else{
                i += 2;
                bandera = true;
            }
        }
         System.out.println("\nLa suma de la serie es "+sumaSerie);
    }
}
```

Figura 3.33. Ejercicio 4 codificado, usando estructura repetitiva `while`.

Explicación:

Líneas 5. Se declara la variable `bandera` de tipo `boolean`, se utilizará para saber cuándo incrementar en 2 o 3 la serie y se inicializa en `true` con la finalidad de que entre al ciclo desde la primera vez, es la variable de control del ciclo `while`.

Línea 6. Se declara la variable `sumaSerie` y se inicializa en cero, llevará la suma de los términos que se generen, En la variable `i` se generaran los términos y se inicializa en 2, debido a que la serie empieza con dicho número.

Líneas 8 a la 21. Ciclo `while`, se explica en la Tabla 3.7 con la ejecución manual o prueba de escritorio del ciclo, con 8 iteraciones.

Tabla 3.7. Prueba de escritorio del ciclo `while`.

Iteración	i	bandera	sumaSerie	Impresión i
	2	true	0	
1	5	false	2	2
2	7	true	7	5
3	10	false	14	7
4	12	true	24	10
5	15	false	36	12
6	17	true	51	15
7	20	false	68	17
8	22	true	88	20

Cada vez que entra al ciclo imprime un término de la serie, el cual está representado por la variable `i` (línea 11). Al final, cuando sale del ciclo imprime la suma de la serie (línea 22).

En la Figura 3.34 puede verificar la ejecución del programa con los datos de la prueba de escritorio.

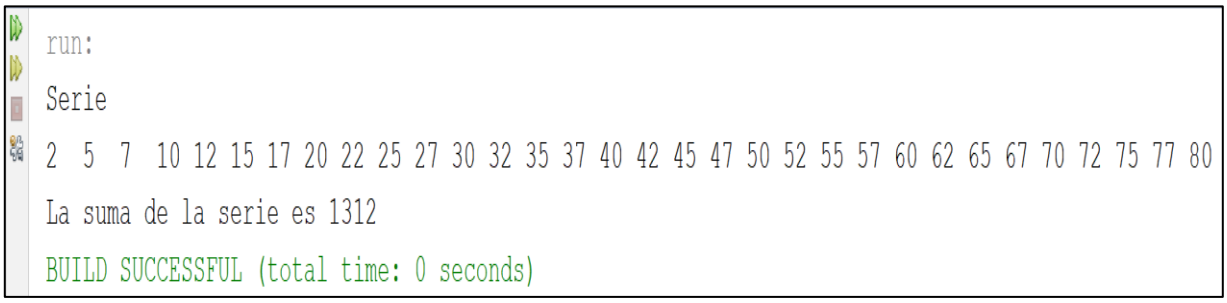


Figura 3.34. Ejecución del programa Ejercicio 4.

Ejercicio 5. Hacer un programa en Java que imprima los números pares que se encuentran en el rango entre 1 y 50 e imprima la suma de ellos.

Entrada: No habrá datos de entrada desde el teclado, los números pares se generarán dentro del ciclo.

Proceso:

```
par = 2; suma = 2;
hacer
   imprimir par;
   suma += par;
   par += 2;
mientras(par <= 50);
```

Salida: Impresión de los números pares y suma de los números pares.

En la Figura 3.35 se muestra el Ejercicio 5 resuelto, utilizando la estructura repetitiva do-while.

```
1   package capitulo3;
2
3   public class SumaPares {
4       public static void main(String[] args) {
5           int par = 2, suma =0;
6           System.out.println("Suma de número pares en el rango de 1 a 50\n");
7           do{
8               System.out.printf("%-3d",par);
9               suma += par;
10              par += 2;
11          }while(par <= 50);
12          System.out.println("\nLa suma de los números pares es: "+suma);
13      }
14  }
```

Figura 3.35. Ejercicio 5 codificado, usando estructura repetitiva do-while.

Explicación:

Líneas 5. Se declaran e inicializan las variables, `par = 2` y `suma = 0`. La variable par se usará para generar los números pares y además será la variable de control del ciclo do-while, y suma llevará el registro de la sumatoria.

Líneas 7 a la 11. Ciclo do-while, se explica en la Tabla 3.8 con la ejecución manual o prueba de escritorio del ciclo, con 13 iteraciones.

Tabla 3.8. Prueba de escritorio del ciclo do-while.

Iteración	Impresión par	suma	Par
		0	2
1	2	2	4
2	4	6	6
3	6	12	8
4	8	20	10
5	10	30	12
6	12	42	14
7	14	56	16
8	16	72	18
9	18	90	20
10	20	110	22
11	22	132	24
12	24	156	26
13	26	182	28

Cuando par es mayor a 50, se sale del ciclo do-while e imprime la suma de los números pares (línea 12).

En la Figura 3.36 puede verificar la ejecución del programa con los datos de la prueba de escritorio.

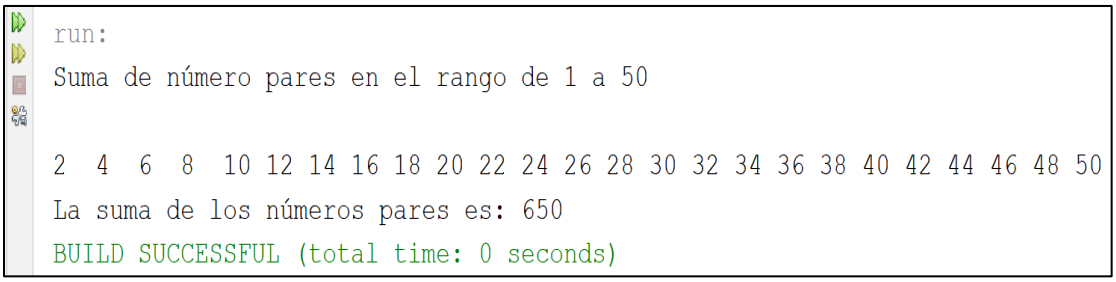


Figura 3.36. Ejecución del programa Ejercicio 5.

Ejercicio 6. Estas en el super haciendo las compras de la semana, de acuerdo con la cantidad de artículos que lleves y su precio, calcula e imprime la cantidad total a pagar.

Entrada: Variables `cantidad, precio, resp;`

Proceso:

```
total = 0;
hacer
     imprime "Ingresa cantidad del artículo ";
     Leer cantidad;
     Imprime "Precio del artículo ";
     Leer precio;
     total = total + (cantidad * precio);
     ¿Deseas ingresar otro artículo? S/N: ;
     Leer resp;
 mientras(resp != 'N' y resp != 'n');
```

Salida: Impresión del total a pagar (total).

En la Figura 3.37 se muestra el Ejercicio 6 resuelto, utilizando la estructura repetitiva do-while.

```
1
2   package capitulo3;
3
4   import java.util.Scanner;
5
6   public class Compra {
7       public static void main(String[] args) {
8           Scanner s = new Scanner(System.in);
9           double precio, total=0;
10          int cantidad;
11          char resp;
12
13          do{
14              System.out.print("Ingresa cantidad del artículo ");
15              cantidad = s.nextInt();
16              System.out.print("Precio del artículo ");
17              precio = s.nextDouble();
18              total = total + (cantidad * precio);
19              System.out.print("¿Deseas ingresar otro artículo? S/N: ");
20              resp = s.next().charAt(0);
21          }while(resp != 'N' && resp != 'n');
22          System.out.println("Total a pagar por su compra $"+total);
23      }
24  }
```

Figura 3.37. Ejercicio 6 codificado, usando estructura repetitiva do-while.

Explicación:

Líneas 9. Se declara e inicializa la variable, `total = 0`, la cual acumulara la cantidad a pagar. La variable `precio` almacenará los precios de los artículos y la variable cantidad guardará la cantidad de artículos que compre.

Línea 11 Se declaró una variable `resp` de tipo char (carácter) será la variable de control del ciclo `do-while`. Cuando el usuario introduzca el carácter 'N' o 'n', el ciclo se detendrá y dará paso a la siguiente instrucción (línea 22).

Líneas 13 a la 21. Ciclo `do-while`, se explica en la Tabla 3.9 con la ejecución manual o prueba de escritorio del ciclo, con 3 iteraciones.

Tabla 3.9. Prueba de escritorio del ciclo do-while.

Iteración	cantidad	precio	total	Resp
		0	0	
1	2	150.30	300.60	S
2	5	37.50	488.10	S
3	8	146.80	1662.50	N

Cuando `resp` es igual a 'N' o 'n', el ciclo se termina e imprime el total a pagar (línea 22).

En la Figura 3.38 puede verificar la ejecución del programa con los datos de la prueba de escritorio.

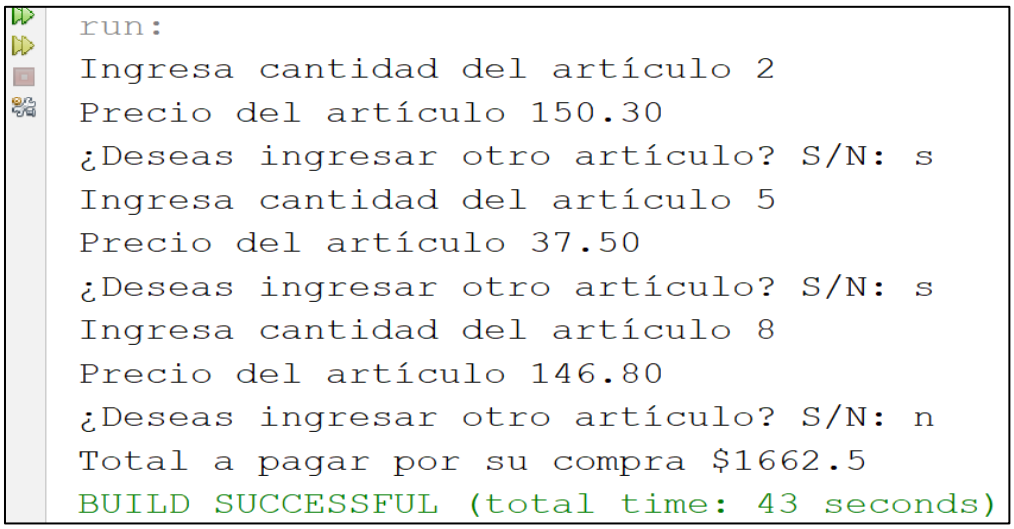


Figura 3.38. Ejecución del programa Ejercicio 6.

Ejercicio 7. Dado un número entero, hacer un programa en Java que imprima un triángulo isósceles con asteriscos, como lo muestra la Figura 3.39. Si número de filas = 5.

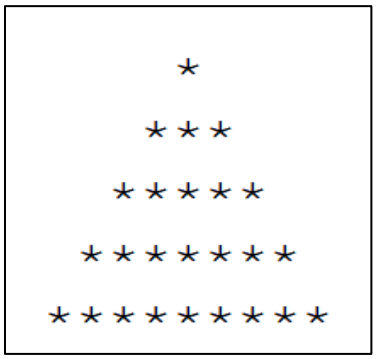


Figura 3.39. Triángulo isósceles.

Entrada: Variable `int numfila`

Constantes de tipo `char espacio = ' '`, `asterisco = '*'`

Proceso: leer `numFila`

```
para (fila = 1; fila <= numFila; fila++){
 para (blancos = numFila-fila; blancos > 0; blancos--)
```

```
            imprimir espacio;
        para (cuenta_A = 1; cuenta_A < 2*fila; cuenta_A++)
            imprimir asterisco;
        imprimir un salto de línea;
    }//fin del ciclo fila
```

Salida: Impresión del triángulo isósceles mediante los ciclos `for`.

En la Figura 3.40 se muestra el Ejercicio 7 resuelto, utilizando las estructuras anidadas `for`

```
1  package capitulo3;
2
3  import java.util.Scanner;
4
5  public class TrianguloIsosceles {
6      public static void main(String[] args) {
7          Scanner s = new Scanner(System.in);
8          final char espacio = ' ';
9          final char asterisco = '*';
10         int numFila;
11         System.out.println("Imprime un triángulo isosceles");
12         System.out.print("Número de filas que tendrá el triángulo: ");
13         numFila = s.nextInt();
14         for (int fila = 1; fila <= numFila; fila++){
15             for (int blancos = numFila-fila; blancos > 0; blancos--){
16                 System.out.print(espacio);
17             }
18             for (int cuenta_A = 1; cuenta_A < 2 * fila; cuenta_A++){
19                 System.out.print(asterisco);
20             }
21             System.out.println();
22         }
23     }
24 }
```

Figura 3.40. Ejercicio 7 codificado, usando estructuras anidadas `for`.

Explicación:

Líneas 8. Se declara e inicializa la constante, `char espacio = ' '` se utilizará para imprimir espacios en blanco, antes de imprimir los asteriscos.

Línea 9. Se declara e inicializa la constante, `char asterisco = '*'` se utilizará para imprimir filas de asteriscos.

Línea 13. Se solicita la entrada de número de filas que tendrá el triángulo isósceles (`numFila`).

Líneas 14 a la 22. Ciclos anidados, se explica en la Tabla 3.10 con la ejecución manual o prueba de escritorio de los ciclos, con `numFila = 5`, en negrita están representadas todas las iteraciones.

Tabla 3.10. Prueba de escritorio de ciclos anidados.

fila	fila < numFila	blancos	blancos > 0	cuenta_A	cuenta_A < 2*fila	Imprime
1	True	**4**	true			blanco
		3	true			blanco
		2	true			blanco
		1	true			blanco
		0	false			
				1	true	asterisco
				2	false	salto de línea
2	True	**3**	true			blanco
		2	true			blanco
		1	true			blanco
		0	false			
				1	true	asterisco
				2	true	asterisco
				3	true	asterisco
				4	false	salto de línea
3	true	**2**	true			blanco
		1	true			blanco
		0	false			
				1	true	asterisco
				2	true	asterisco
				3	true	asterisco
				4	true	asterisco
				5	true	asterisco
				6	false	salto de línea
4	true	**1**	true			blanco
		0	false			
				1	true	asterisco
				2	true	asterisco
				3	true	asterisco
				4	true	asterisco
				5	true	asterisco
				6	true	asterisco
				7	true	asterisco
				8	false	salto de línea
5	true	**0**	false			
				1	true	asterisco
				2	true	asterisco
				3	true	asterisco
				4	true	asterisco
				5	true	asterisco
				6	true	asterisco
				7	true	asterisco
				8	true	asterisco
				9	true	asterisco
				10	false	Salto de línea
6	false					

En la Figura 3.41 puede verificar la ejecución del programa con los datos de la prueba de escritorio.

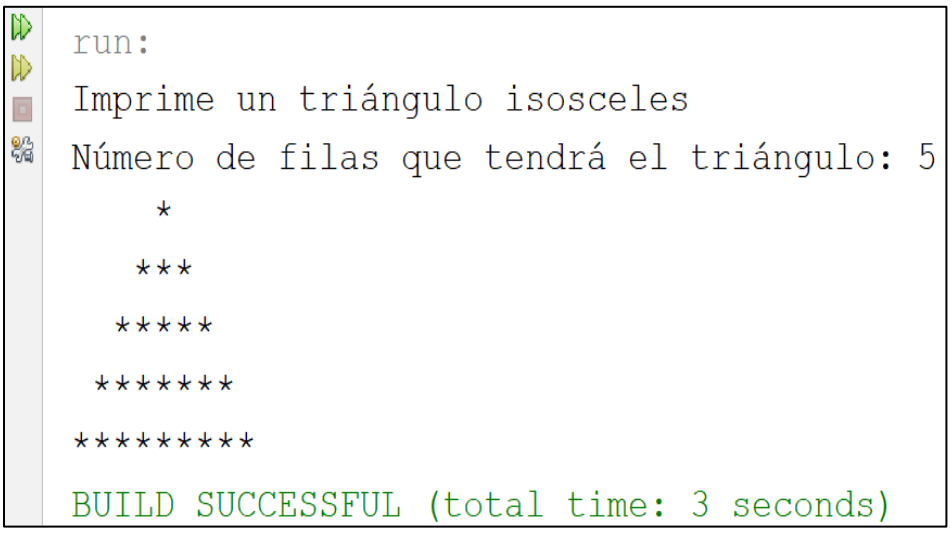


Figura 3.41. Ejecución del programa Ejercicio 7.

Ejercicio 8. Hacer un programa en Java que imprima todos los números primos que se encuentren entre dos números ingresados por el teclado. Ejemplo: Los números primos entre 5 y 15 son 5, 7, 11, 13.

Entrada: Variables enteras `ini, fin`

Proceso:

```
para (i = ini; i <= fin; i++) {
    c = 2;
    bandera = verdadera;
    mientras (bandera y c < i) {
        Si (i % c == 0)
          bandera = falsa;
        sino
          c++;
    }//fin del ciclo mientras
    Si (bandera = verdadera)
       imprime i;
}//fin del ciclo para
```

Salida: Impresión de todos los números primos que estén entre `ini` y `fin`.

En la Figura 3.42 se muestra el Ejercicio 8 resuelto, utilizando las estructuras anidadas `for` y `while`.

```
1   package capitulo3;
2
3   import java.util.Scanner;
4
5   public class Primos {
6       public static void main(String[] args) {
7           Scanner s = new Scanner(System.in);
8           int ini, fin, c; boolean bandera;
9           System.out.print("Introduce el primer valor de entrada ");
10          ini = s.nextInt();
11          System.out.print("Introduce el segundo valor de entrada ");
12          fin = s.nextInt();
13          System.out.println("\nLos números primos entre "+ini+" y "+fin+" son: ");
14          for (int i = ini; i <= fin; i++) {
15              c = 2; bandera = true;
16              while (bandera && c < i) {
17                  if (i % c == 0)
18                      bandera = false;
19                  else
20                      c++;
21              }
22              if (bandera)
23                  System.out.println(i);
24          }
25      }
26  }
```

Figura 3.42. Ejercicio 8 codificado, usando estructuras anidadas `for` y `while`.

Explicación:

En este programa se tienen dos ciclos anidados, el ciclo `for` (líneas 14 a la 24) que es el ciclo externo y el ciclo `while` (líneas 16 a la 21) que es el ciclo interno.

Líneas 10 y 12. Se introduce por teclado el valor con el que va a iniciar el ciclo `for` (variable `ini`) y el valor con el que terminará (variable `fin`). Recordando que cuando hay ciclos anidados, se empieza con el ciclo externo y después con el interno.

En la Tabla 3.11 se realizará la prueba de escritorio. Considere que `ini = 5` y `fin = 15`. Cada vez que entra al ciclo `while`, las variables se inicializan (`bandera = true y c = 2`). Recuerde que para que un número sea primo solo debe tener 2 divisores (el 1 y el mismo).

Tabla 3.11. Prueba de escritorio de ciclos anidados.

i	c	bandera	while (bandera && c<i)	if (i%c == 0)	c++	if(bandera) Imprime i	i++	i<=fin
5	2	True	true	falso	3			
	3		true	falso	4			
	4		true	falso	5			
	5		false			5	6	true
6	2	True	true	true				
		False					7	true
7	2	True	true	false	3			
	3		true	false	4			
	4		true	false	5			
	5		true	false	6			
	6		true	false	7			
	7		false			7	8	true
8	2	True	true	true				
		False					9	true
9	2	True	true	false	3			
	3		true	true				
		False					10	true
10	2	True	true	true				
		False					11	true
11	2	True	true	false	3			
	3		true	false	4			
	4		true	false	5			
	5		true	false	6			
	6		true	false	7			
	7		true	false	8			
	8		true	false	9			
	9		true	false	10			
	10		true	false	11			
	11		false			11	12	true
12	2	True	true	true				
		False					13	true
13	2	True	true	false	3			
	3		true	false	4			
	4		true	false	5			
	5		true	false	6			
	6		true	false	7			
	7		true	false	8			
	8		true	false	9			
	9		true	false	10			
	10		true	false	11			
	11		true	false	12			

	12		true	false	13			
	13		false			13	14	true
14	2	True	true	true				
		False					15	true
15	2	True	false				16	false

En la Figura 3.43 puede verificar la ejecución del programa con los datos de la prueba de escritorio.

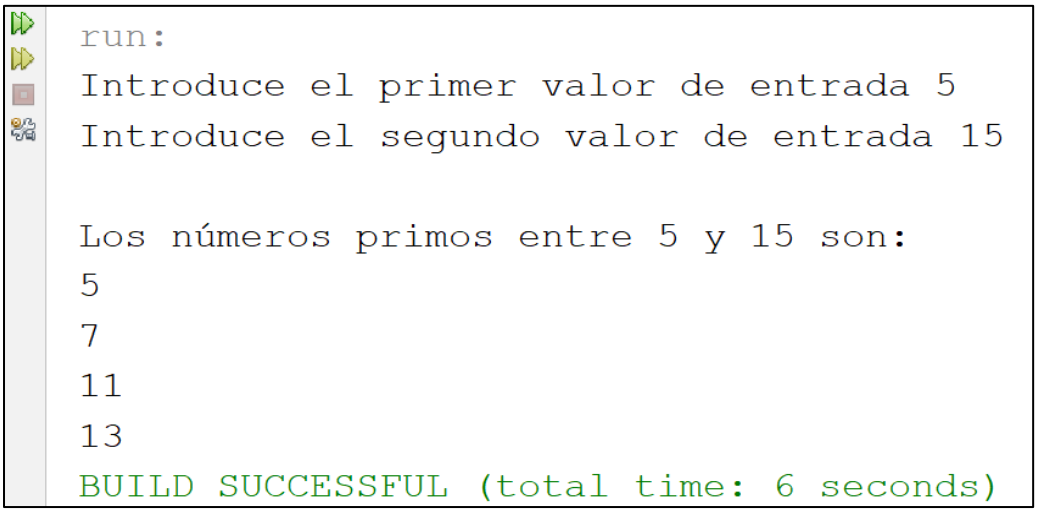


Figura 3.43. Ejecución del programa Ejercicio 8.

3.6 Programas propuestos

1. Hacer un programa en Java que calcule la suma de todos los números entre 1 y un número leído (n) por el usuario e imprima el resultado.
2. Hacer un programa en Java que imprima las letras del abecedario en mayúsculas.
3. Hacer un programa en Java que imprima el abecedario en orden descendente (de la z a la a)
4. Hacer un programa en Java que imprima en pantalla la tabla de multiplicar de cualquier número entero.
5. Dado el número de horas trabajadas y el precio por hora, calcular e imprimir el sueldo de 5 empleados.
6. Hacer un programa en Java que calcule el promedio de calificaciones de un grupo de alumnos.
7. Hacer un programa en Java para obtener la tabla de multiplicar de un número entero k, comenzando desde 1 hasta el 10.
8. Hacer un programa en Java que lea un número entero n y calcule el resultado de la siguiente serie:

$$1+\frac{1}{2}+\frac{1}{3}+\frac{1}{4}+\cdots+\frac{1}{n}$$

9. Para integrar los equipos de basquetbol se requiere de las siguientes características: Para el equipo varonil que su altura sea mayor a 1.75 y para el femenil que su altura sea mayor que 1.65. Imprimir de los interesados cuántos fueron seleccionados para el equipo varonil y cuántos para el equipo femenil.
10. Hacer un programa en Java que al recibir como dato un número entero n, obtenga el resultado de la siguiente serie:

$$1^1 - 2^2 + 3^3 - \cdots \pm n^n$$

11. Dado el sueldo de n trabajadores, considere un aumento del 12% a cada uno de ellos, si su sueldo es inferior a $800.00. Imprima el sueldo con el aumento incorporado (si corresponde).
12. Calcule el aumento de sueldos de n empleados, bajo los siguientes criterios:
    - Si el sueldo es menor de $10,000.00 aumento de 10%
    - Si el sueldo es entre $10,000.00 y $25,000.00 aumento de 7%
    - Si el sueldo es mayor de $25,000.00 aumento de 5%

    Imprimir el nuevo sueldo del empleado y el monto total de la nómina considerando el aumento.

13. Una persona invierte $1000.00 en una cuenta de ahorro que produce el 3% de interés. Suponiendo que todo el interés se deposita en la cuenta, calcule e imprima el monto de dinero en la cuenta al final de cada año, durante 5 años. Use la siguiente fórmula para determinar los montos:

    $a = p\,(1 + r)^n$

    Donde: p es el monto que se invirtió originalmente.

    r es la tasa de interés anual.

    n es el número de años.

    a es la cantidad depositada al final de cada año.

14. Hacer un programa en Java que reciba como entrada 24 números reales que representan las temperaturas del exterior en un periodo de 24 horas. Encuentre la temperatura media, así como la más alta y la más baja del día.
15. Realizar las tablas de multiplicar del 1 al 5 hasta el múltiplo de 15.
16. Hacer un programa que reciba como entrada 10 números enteros e imprima la suma de los que sean primos.
17. Hacer un programa el Java que calcule e imprima el factorial de un número entero, el programa se detendrá cuando se ingrese un número menor de 1.
18. Hacer un programa en Java que genere e imprima los primeros 30 números de la serie Fibonacci.

19. Hacer un programa que al recibir como entrada un número positivo, imprima en pantalla una figura como la que se muestra a continuación (ejemplo para n = 5):

```
1
1 2
1 2 3
1 2 3 4
1 2 3 4 5
1 2 3 4
1 2 3
1 2
1
```

20. Hacer un programa que al recibir como entrada un número positivo, imprima en pantalla una figura como la que se muestra a continuación (ejemplo para n = 5):

```
*
* * *
* * * * *
* * * * * *
```

21. Introducir el número de alumnos de un grupo y contabilizar por separado en qué sector del Estado de México viven; para tal efecto se desplegará un menú con las siguientes opciones:
    ¿Dónde vives?
    a. Toluca
    b. Metepec
    c. Lerma
    d. Zinacantepec
    e. Otro
    f. Salir

    Imprimir el total de alumnos por municipio.

22. Hacer un programa en Java que reciba como entrada una secuencia de n números y determinar: El mayor de los que son múltiplos de 5 y el menor de los que son múltiplos de 3.
23. Hacer un programa en Java que reciba como entrada una secuencia de n números y determinar: La suma de los pares y el producto de los que son múltiplos de 7.
24. Hacer un programa que reciba como entrada n números, e imprimir cuántos son positivos, cuántos negativos y cuántos fueron ceros.
25. Hacer un programa en Java que reciba como entrada n números enteros, calcule e imprima la suma, promedio, producto, el mayor y el menor.
26. Hacer un programa en Java que calcule e imprima el promedio de los números múltiplos de 9 que hay en el rango de 45 a 194.
27. Calcular los cuadrados, los cubos y las raíces cuadradas de los números del 1 al 9, utilice tabuladores para imprimir la siguiente tabla:

Tabla 3.12.Cuadrado, cubo y raíz cuadrada.

Número	Cuadrado	Cubo	Raíz cuadrada
1	1	1	1
2	4	8	1.414
3	9	27	1.732
.	.	.	.
.	.	.	.
.	.	.	.
9	81	729	3

28. Hacer un programa en Java que reciba como entrada un número entero positivo e imprimir si es o no un número perfecto. Un número es perfecto cuando es igual a la suma de sus divisores excepto él mismo. Por ejemplo, el número 28 es perfecto porque 28 = 1+2+4+7+14
29. Hacer un programa en Java que calcule e imprima los números del 1 al 100 excepto los múltiplos de 7.
30. Hacer un programa en Java que imprima el abecedario alternando mayúscula y minúscula. Por ejemplo: Aa, Bb, Cc, Dd, Ee, Ff y así sucesivamente hasta Zz.
31. Hacer un programa en Java que imprima el abecedario en forma inversa (de la z a la a) y luego vaya eliminando de una letra en una, empezando por la letra z hasta que quede la letra a. Ejemplo:
    zywvtsrqponmlkjihgfedcba
    ywvtsrqponmlkjihgfedcba
    wvtsrqponmlkjihgfedcba
    vtsrqponmlkjihgfedcba
    tsrqponmlkjihgfedcba
    . . .
    edcba
    dcba
    cba
    ba
    a

4. ARREGLOS

En los temas anteriores se ha trabajado con tipos de datos simples, que representan valores como un número entero, real o un carácter. En muchas ocasiones se necesita procesar una colección de datos que están relacionados entre sí, por ejemplo, una lista de calificaciones, o de temperaturas tomadas a lo largo de un día o de un mes, etcétera. Este tipo de procesamiento utilizando datos simples puede ser engorroso, por ello muchos de los lenguajes de programación incluyen características de estructura de datos. Estas estructuras de datos se llaman arreglos.

Los arreglos pueden ser unidimensionales (representados por una fila de tamaño n), bidimensionales (representados por filas y columnas) o multidimensionales (formados por planos, filas y columnas).

4.1 Definición de arreglo

“Un arreglo es una secuencia de posiciones de la memoria central a las que se puede acceder directamente, que contiene datos del mismo tipo y pueden ser seleccionados individualmente mediante el uso de subíndices” (Joyanes Aguilar, 2020, pág. 255).

“Un arreglo es un grupo de variables (llamadas **elementos** o **componentes**) que contienen valores del mismo tipo. Los arreglos son objetos, por lo que se consideran como tipos de referencia” (Deitel & Deitel, 2016, pág. 245).

“Un arreglo se define como estructuras compuestas por un conjunto de celdas donde se puede almacenar datos de un mismo tipo. En cada celda se puede acceder por su posición, mediante un índice” (Gómez Jiménez & Moreno Nuñez, 2019, pág. 41).

Por lo anterior, un arreglo es un conjunto de elementos del mismo tipo, los cuales se pueden acceder mediante un índice. Los arreglos deben ser finitos, homogéneos y ordenados.

- Finito. Se debe determinar el tamaño del arreglo.
- Homogéneo. Todos los elementos del arreglo son del mismo tipo de datos.
- Ordenado. Se puede determinar cuál es el primer elemento, el segundo, el tercero y el n-ésimo elemento.

4.2 Arreglo unidimensional

Los arreglos unidimensionales también llamados vectores o listas, se representan por una sola fila, un índice representará la posición de la celda en la fila para el primer elemento, el segundo, tercero y n-ésimo elemento. En la Figura 4.1. se muestra un arreglo unidimensional que contiene datos de tipo cadena (nombres). Debajo de la estructura se encuentra su respectivo índice. El arreglo en cuestión tiene 6 elementos. En Java los índices siempre tienen un índice inferior de cero y como índice superior el tamaño del arreglo menos 1.

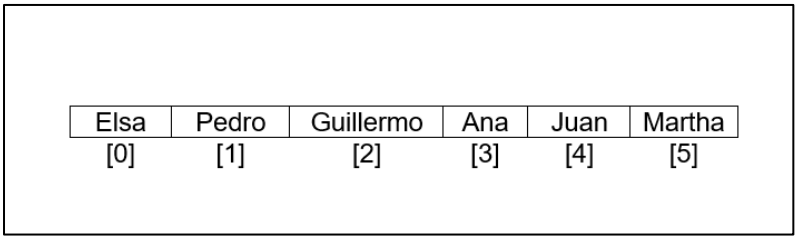


Figura 4.1. Estructura de un arreglo unidimensional.

Como puede ver, cada valor tiene una posición dentro del conjunto de elementos del arreglo. Si desea acceder algún elemento del arreglo debe escribir el nombre del arreglo, seguido de los corchetes ([ ]) y dentro de ellos el valor numérico del elemento que desea acceder. Por ejemplo, si el arreglo se llama nombre y quiere acceder al elemento Ana, sería nombre[3]. Como puede ver este arreglo tiene 6 elementos y las posiciones (índices) van desde cero hasta 5 (tamaño del arreglo – 1).

La declaración de un arreglo es similar a otros tipos de datos, la diferencia radica en los corchetes. Ejemplos:

```
int [] vector;
float arreglo[];
```

Los corchetes se pueden colocar antes o después de nombre del arreglo (identificador). Su sintaxis es:

```
tipo_dato [] nombreArreglo;
tipo_dato nombreArreglo [];
```

La primera forma indica que todos los identificadores son arreglos del mismo tipo, la segunda forma, sólo es arreglo el identificador al que le siguen los corchetes.

Para Java un arreglo es una referencia a un objeto, por lo tanto, para que cree o instancie un arreglo, se usa el operador new, como lo puede ver en la línea 8 de la Figura 4.2.

```
7        float [] calificaciones;              //Declaración del arreglo
8        calificaciones = new float [40];      //Aquí se instancia el arreglo
9
```

Figura 4.2. Declaración de un arreglo y su instancia.

Se puede escribir en una sola sentencia:

```
float [ ] calificaciones = new float [40];
```

Entonces la sintaxis para declarar y definir un arreglo es:

```
tipo_dato nombreArreglo [ ] = new tipo_dato [numeroDeElementos];
```

o bien,

```
tipo_dato nombreArreglo [ ];
nombreArreglo = new tipo_dato [numeroDeElementos];
```

Se puede referenciar a los elementos del arreglo utilizando fórmulas para el subíndice; el cual puede ser una constante, una variable o una expresión siempre que se evalúe un entero. Ver el fragmento de programa de la Figura 4.3.

```
7           double []estatura,promedio;   //Se declaran 2 arreglos
8           estatura = new double[45];    //Se crea el arreglo de 45 elementos
9           promedio = new double[20];    //Se crea el arreglo de 20 elementos
10          final int NUM = 40;
11          int i = 2, j= 3;
12          estatura[15]= 1.67;           //Accede al elemento estatura[15]
13          estatura[i + j]= 1.56;        //Accede al elemento estatura[5]
14          promedio[NUM-32]= 1.20;       //Accede al elemento promedio[13]
15          promedio[i * j]= 1.30;        //Accede al elemento promedio[6]
16          estatura[NUM]= 1.40;          //Accede al elemento estatura[40]
```

Figura 4.3. Ejemplos para acceder a los elementos de un arreglo.

Se puede crear un arreglo y a la vez inicializar los elementos de éste, con valores constantes y así determinar su tamaño; estos valores se separan por comas y se encierran entre llaves, los ejemplos los puede ver en la Figura 4.4:

```
6   byte edad[] = {18, 26, 35,12, 10, 42, 38};      //Arreglo con 7 elementos
7   char letra[] = {'D','i','a','n','a'};           //Arreglo con 5 elementos
8   double temp[] = {45.2, 25.3, 32.8};             //Arreglo con 3 elementos
```

Figura 4.4. Creación e inicialización de un arreglo.

Pueden asignarse valores a un arreglo utilizando las estructuras repetitivas `for`, `while` o `do-while`, de hecho, es lo más usual. A continuación, en la Figura 4.5 se presenta un programa en el cual se declara y crea un arreglo llamado vector con 6 elementos `int`, los cuales están inicializados con el valor de cero, qué es el valor predeterminado para las variables `int`. Con ayuda del ciclo for se muestra el índice, así como los valores de cada elemento (cero). También muestra la ejecución del programa.

```
1   package capitulo4;
2
3   public class Arreglo1 {
4       public static void main(String[] args) {
5           //Declaración de la variable vector y la inicializa con un objeto vector
6           int vector[]= new int[10];  //crea el objeto vector
7
8           System.out.printf("%s%8s\n","Indice","Valor");
9           for (int sub = 0; sub < 10; sub++) {
10              System.out.printf("%3d%9d\n",sub,vector[sub]);
11          }
12      }
13  }
```

```
Output - Arreglo1 (run)
run:
Indice   Valor
  0        0
  1        0
  2        0
  3        0
  4        0
  5        0
  6        0
  7        0
  8        0
  9        0
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 4.5. Inicialización de elementos de un arreglo con valores predeterminados y ejecución.

En la línea 6 se declara vector y luego se inicializa la variable con una referencia a un objeto arreglo que contiene 10 elementos `int`.

El ciclo `for` (líneas 9 a la 11) imprime el índice representado por `sub` (variable de control del ciclo `for`), así como el valor de cada elemento del arreglo representado por `vector[i]`. La variable `sub` inicia con cero para poder acceder al primer elemento del arreglo, mientras la condición (`sub <10`) del ciclo `for` se cumpla, la ejecución del ciclo continuará e imprimirá los índices (`sub`) así como los elementos del arreglo (`vector[sub]`).

Ahora vamos a imprimir un arreglo inicializado con elementos desde su creación. Usaremos gran parte del código anterior (ver Figura 4.6). El arreglo `vector` esta inicializado con 7 elementos que se encuentran entre llaves `{}` (`35, 7, 43, 90, 12, 50, 74`). En este caso la longitud del arreglo se determina con base en el número de elementos que se encuentran entre las llaves `{}`. El elemento vector[0] se inicializa con 35, vector[1] con 7 y así sucesivamente. También se incorpora la expresión `vector.length` (línea 8) la cual determina la longitud del arreglo.

```java
package capitulo4;

public class Arreglo2 {
    public static void main(String[] args) {
        int vector[]= {35,7,43,90,12,50,74};

        System.out.printf("%s%8s\n","Indice","Valor");
        for (int i = 0; i < vector.length; i++) {
            System.out.printf("%3d%9d\n",i,vector[i]);
        }
    }
}
```

```
Output - Arreglo2 (run)
run:
Indice   Valor
  0       35
  1        7
  2       43
  3       90
  4       12
  5       50
  6       74
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 4.6. Programa y ejecución de Arreglo2.

4.2.1 Operaciones con arreglos

Las operaciones más comunes para realizar en un arreglo son:

- Asignación
- Lectura/escritura
- Recorrido (acceso secuencial)
- Actualización (añadir, insertar, borrar)
- Ordenación
- Búsqueda.

A continuación, se analizará cada una de estas operaciones.

Asignación. La asignación de valores se realiza mediante la instrucción de asignación como se muestra en la figura 4.7.

```
12          System.out.println("Operaciones con arreglos unidimensionales");
13          System.out.println("ASIGNACIÓN");
14          System.out.println("ASIGNACIÓN DE VALORES MEDIANTE LA INSTRUCCIÓN DE ASIGNACIÓN ");
15          int arreglo[]= new int[6];
16          arreglo[2]=100;         //Asignación
17          arreglo[4]=90;          //Asignación
```

Figura 4.7. Asignación de valores en un arreglo.

Se declara un arreglo (línea 15) y se asignan los valores (líneas 16 y 17). También es posible asignar todos los valores a un arreglo, con ayuda de las estructuras repetitivas (`for`, `while` o `do-while`). Por ejemplo, en la Figura 4.8 se presenta un fragmento de programa, si se desea dar el mismo valor a todos los elementos de un arreglo, mediante el ciclo `for`. En este caso todos los elementos tendrán un valor de 15.

```
20              for (int i = 0; i < arreglo.length; i++)
21              {
22                  arreglo[i]= 15;
23              }
```

Figura 4.8. Asignación mediante ciclo `for`.

Lectura/escritura. También llamadas operaciones de entrada/salida, normalmente se realizan con estructuras repetitivas como puede observarse en la Figura 4.9. En la línea 26 se da la entrada de un dato al arreglo en la posición 5, en la línea 27 se muestra por pantalla (salida) el elemento 5 del arreglo.

```
24          System.out.println("LECTURA Y ESCRITURA");
25          System.out.print("Introduce el elemento 5 del arreglo ");
26          arreglo[5]= s.nextInt();    //Lectura
27          System.out.println("arreglo[5] = "+arreglo[5]); //Escritura
```

Figura 4.9. Lectura/escritura en un arreglo.

Recorrido (acceso secuencial). Es posible acceder a los elementos de un arreglo para introducir datos o bien para ver su contenido. A esta acción se le conoce como recorrido del arreglo. El recorrido de un arreglo se realiza utilizando las estructuras repetitivas, en donde las variables de control se utilizan como índices del arreglo. El incremento del contador del ciclo hace que se procesen todos los elementos del arreglo. En la Figura 4.10 (fragmento de programa) puede ver el recorrido de un arreglo mediante el ingreso de datos en él (Líneas 13 a la 16) y la impresión del arreglo (líneas 18 a la 20).

```
11          System.out.println("RECORRIDO DEL ARREGLO (ACCESO SECUENCIAL)");
12          System.out.println("LLENADO DE ARREGLO");
13          for (int i = 0; i < arreglo.length; i++) {
14              System.out.print("Ingrese arreglo[" + i + "]: ");
15              arreglo[i] = s.nextInt();
16          }
17          System.out.println("IMPRESIÓN DEL ARREGLO");
18          for (int i = 0; i < arreglo.length; i++) {
19              System.out.println("arreglo[" + i + "] = " + arreglo[i]);
20          }
```

Figura 4.10. Recorrido (acceso secuencial) de un arreglo.

Actualización (insertar, borrar, modificar). La actualización de un arreglo puede constar a su vez de 3 operaciones importantes: insertar, borrar y modificar un arreglo.

Añadir. Añadir datos en un arreglo, quiere decir introducir elementos al final de un arreglo; la única condición para realizarlo es verificar que haya espacio en el arreglo, esto es, que el arreglo no contenga todos los elementos con que fue definido al principio.

Insertar. Insertar un elemento consiste en introducir un elemento en el interior del arreglo y los demás elementos deberán desplazarse hacia abajo para insertar el nuevo elemento. Por ejemplo, se tiene un arreglo vector de 10 elementos que contiene 5 valores enteros ordenados ascendentemente y se desea insertar el valor 10 (ver Figura 4.11). Como el valor 10 está entre los elementos 8 y 14 (posiciones 1 y 2 del arreglo), se deberán desplazar hacia abajo los elementos 14, 25 y 30. El elemento 14 pasará a la posición 3, el 25 a la posición 4 y el 30 a la 5.

```
Impresión del arreglo vector
vector[0] = 2
vector[1] = 8
vector[2] = 14
vector[3] = 25
vector[4] = 30
vector[5] = 0
vector[6] = 0
vector[7] = 0
vector[8] = 0
vector[9] = 0
```

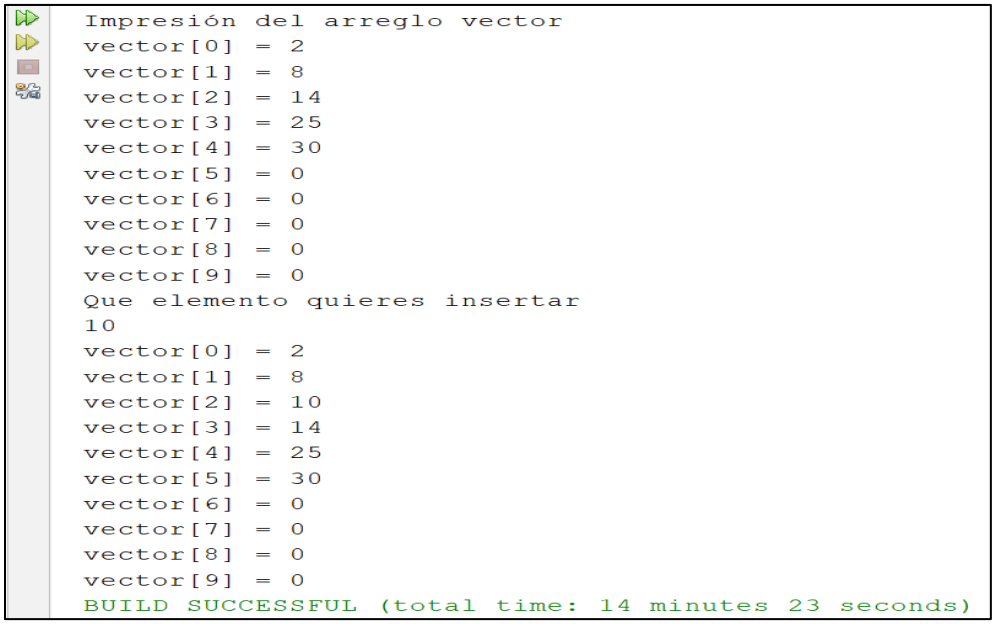
Figura 4.11. Impresión de arreglo vector.

En la Figura 4.12 se muestra el fragmento de programa que realiza el desplazamiento de los elementos, suponiendo que tiene suficiente espacio en el arreglo (como en la Figura 4.11). Primero se debe calcular la posición que ocupará el nuevo elemento (líneas 33 a la 43) y después el desplazamiento de los elementos (líneas 45 a la 50) y por último insertar el nuevo elemento (línea 51).

```
31          //System.out.println("Calcular posición del elemento a insertar");
32          System.out.println("Que elemento quieres insertar");
33          ele = s.nextInt();
34          int c = 0,pos=0; boolean band = true;
35          do
36          {
37              if (ele < vector[c]){
38                  pos = c;
39                  band = false;
40              }else{
41                 c++;
42              }
43          }while(band);
44          //Desplazamiento de elementos
45          int tam = vector.length-2;
46          while(tam >= pos)
47          {
48              vector[tam+1]= vector[tam];
49              tam--;
50          }
51          vector[pos]=ele;    //Insertar el nuevo elemento
```

Figura 4.12. Insertar un elemento en un arreglo.

En la Figura 4.13 puede ver la ejecución del programa, donde se solicita el elemento que desea agregar, luego se muestra el arreglo con el elemento nuevo y el desplazamiento de los elementos 14, 25 y 30



```
Impresión del arreglo vector
vector[0] = 2
vector[1] = 8
vector[2] = 14
vector[3] = 25
vector[4] = 30
vector[5] = 0
vector[6] = 0
vector[7] = 0
vector[8] = 0
vector[9] = 0
Que elemento quieres insertar
10
vector[0] = 2
vector[1] = 8
vector[2] = 10
vector[3] = 14
vector[4] = 25
vector[5] = 30
vector[6] = 0
vector[7] = 0
vector[8] = 0
vector[9] = 0
BUILD SUCCESSFUL (total time: 14 minutes 23 seconds)
```

Figura 4.13. Ejecución del programa que inserta elementos en un arreglo.

Borrar. Eliminar un elemento al final de un arreglo no presenta ningún problema. Sin embargo, eliminar un elemento del interior del arreglo, provoca un desplazamiento hacia arriba de los elementos inferiores a él para reorganizar el arreglo. Cabe aclarar que el borrado es un borrado lógico ya que, al crear un arreglo, en automático se inicializa en cero.

Para hacer este programa es necesario primero saber la posición en que se encuentra el elemento que se desea borrar, para este ejemplo se ha declarado un arreglo e inicializado con valores (`int numero[]= {3,4,8,32,23,65,10};`). En la Figura 4.14 se encuentra un fragmento de programa que sirve para conocer la posición del elemento que se desea borrar del arreglo.

```
17          //System.out.println("Calcular posición del elemento a borrar");
18          System.out.print("¿Que elemento quieres borrar? ");
19          ele = s.nextInt();
20          do
21          {
22              if (ele == numero[c]){
23                  pos = c;
24                  band = false;
25              }else{
26                 c++;
27              }
28          }while(band);
```

Figura 4.14. Encontrar posición de elemento a borrar.

Una vez que se encuentre el elemento en el arreglo (que se cumpla la condición de la línea 22, `ele == numero[c]`) entonces a la variable `pos` se le asignará el índice del elemento y sabremos que elemento borrar. El desplazamiento de los elementos hacia arriba es muy sencillo, se utiliza un ciclo for, el cual inicia a partir de la posición del elemento a borrar, y se presenta en la Figura 4.15.

```
30          for (i = pos; i < numero.length-1; i++) {
31              numero[i]= numero[i+1];
32          }
33          numero[i]= 0;
```

Figura 4.15. Borrado de un elemento y desplazamiento hacia arriba.

En la Figura 4.16 podrá ver el funcionamiento del borrado de un elemento, así como el desplazamiento de los elementos.

```
run:
Impresión del arreglo numero
vector[0] = 3
vector[1] = 4
vector[2] = 8
vector[3] = 32
vector[4] = 23
vector[5] = 65
vector[6] = 10
¿Que elemento quieres borrar? 32
Impresión del arreglo numero
vector[0] = 3
vector[1] = 4
vector[2] = 8
vector[3] = 23
vector[4] = 65
vector[5] = 10
vector[6] = 0
BUILD SUCCESSFUL (total time: 11 seconds)
```

Figura 4.16. Borrado de un elemento del arreglo.

Ordenación. Para ordenar un arreglo ya sea de forma ascendente o descendente, utilizaré el método burbuja, es el más fácil de comprender cuando se inicia en la programación.

Este método realiza varias “pasada”, es decir, hace un recorrido completo a través del arreglo a ordenar. Se comparan los datos que se encuentran en las casillas adyacentes (por ejemplo, el `elemento[0]` y el `elemento[1]`) y se intercambian de orden si es necesario. Así se continúa hasta recorrer todo el arreglo.

Una “pasada” no garantiza que los datos queden ordenados, por lo que es necesario realizar varias “pasadas”. El número de pasadas es tamaño del arreglo -1, en la última pasada se detecta que no se hicieron cambios, lo cual indica que todos los datos están en el orden deseado. En la Figura 4.17 podrá ver el código de ordenamiento por burbuja.

```
14          //System.out.println("Ordenamiento por Método burbuja");
15
16          for (int i = 1; i < vector.length-1; i++) {
17              for (int j = 0; j <= vector.length-2; j++) {
18                  if(vector[j] > vector[j+1])
19                  {
20                      temp = vector[j];
21                      vector[j]= vector[j+1];
22                      vector[j+1]= temp;
23                  }
24              }
25          }
```

Figura 4.17. Método de burbuja para ordenar un arreglo.

Como puede ver en la Figura 4.17 se tienen ciclo anidados `for`, el primero (línea 16) lleva el conteo del número de pasadas que debe realizar para asegurar el ordenamiento deseado, el segundo `for` (línea 17) se utiliza para las comparaciones entre los elementos (el primero con el segundo, el segundo con el tercero, el tercero con el cuarto, y así sucesivamente hasta llegar al elemento `n - 1`). Cada vez que se cumple la condición (`vector[j] > vector[j+1]`) se realiza el intercambio de los elementos para ser ordenados. A continuación, en la Figura 4.18 se presenta la ejecución del método de burbuja.

```
run:
Impresión de arreglo desordenado
arreglo1[0] = 3
arreglo1[1] = 41
arreglo1[2] = 8
arreglo1[3] = 32
arreglo1[4] = 4
arreglo1[5] = 65
arreglo1[6] = 10

Impresión de arreglo ordenado de forma ascendente
arreglo1[0] = 3
arreglo1[1] = 4
arreglo1[2] = 8
arreglo1[3] = 10
arreglo1[4] = 32
arreglo1[5] = 41
arreglo1[6] = 65
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 4.18. Ejecución del método burbuja.

4.3 Arreglo bidimensional

Los arreglos bidimensionales (dos dimensiones) con frecuencia representan una tabla de valores, para acceder a estos valores se requiere de dos índices que representan las filas y columnas de la tabla o arreglo bidimensional, también conocido como matriz. Por convención, el primer índice identifica la fila del elemento y el segundo la columna. La figura 4.19 representa un arreglo bidimensional, una tabla o una matriz de 20 elementos (4X5) con 4 filas y 5 columnas. Los índices de las filas y columnas se inicializan en cero.

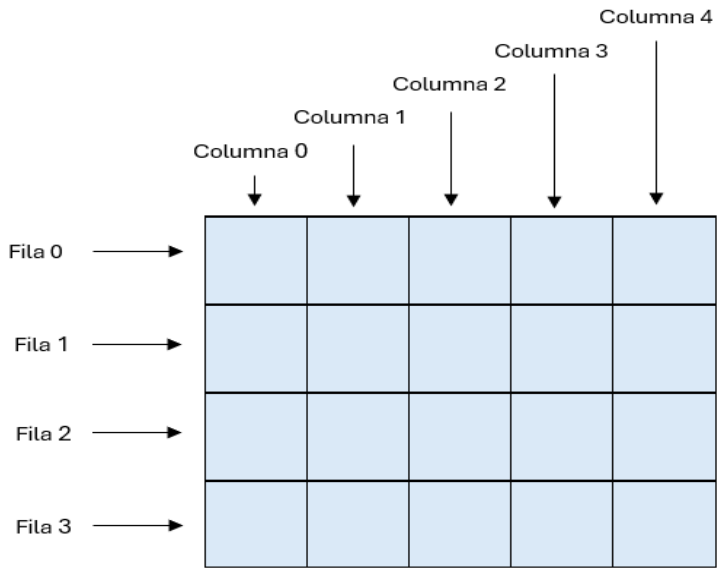


Figura 4.19. Representación de un arreglo bidimensional.

Al igual que un arreglo unidimensional o vector (de 20 elementos), cada uno de ellos tiene el mismo nombre. Supongamos que el arreglo bidimensional se llama `matriz` y se quiere acceder a un elemento en particular, escribiríamos `matriz[2][3]` para acceder al elemento del arreglo matriz que se ubica en la fila 2, columna 3.

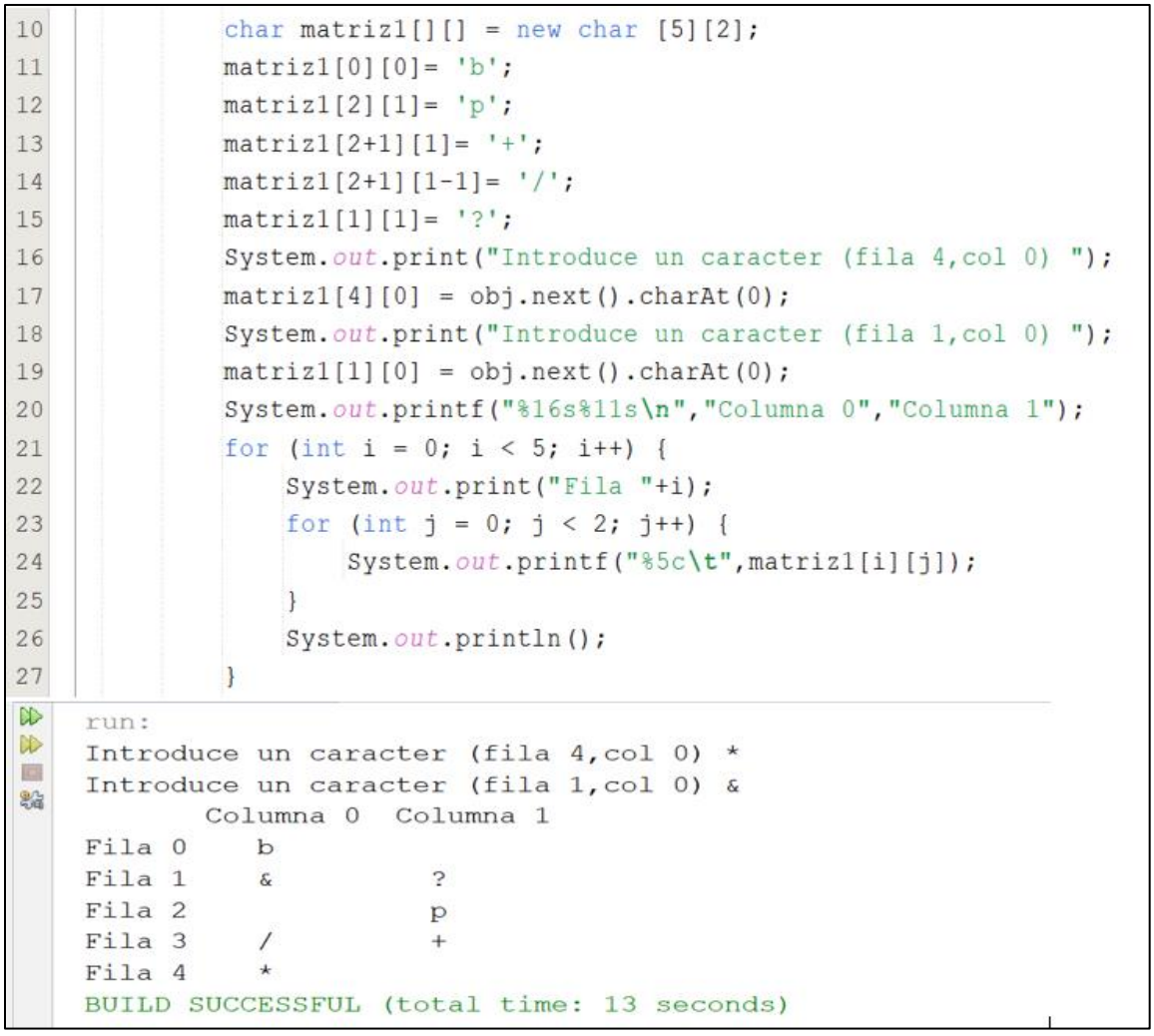
La declaración de un arreglo bidimensional o matriz es muy similar que la del unidimensional, la única diferencia son los índices, el vector tiene un índice y la matriz tiene 2, lo puede observar en la Figura 4.20.

```
9           //Declaración en una sola sentencia
10          char matriz1[][] = new char [5][2];
11
12          //Declaración en dos sentencias
13          float matriz2[][];
14          matriz2 = new float [4][5];
15
16          //Declaración e inicialización de la matriz
17          int matriz3[][]= {{3,0,1},{2,4,6},{10,7,9}};
```

Figura 4.20. Formas de declarar un arreglo bidimensional.

Una vez declarada la matriz, se puede dar valores a sus elementos a través de asignaciones, o por medio de un ciclo y la lectura correspondiente de los valores.

En la Figura 4.21 se ha declarado un arreglo bidimensional de tipo `char` (`matriz1`) de 5 filas y 2 columnas, se hicieron varios tipos de asignaciones (líneas 11 a la 15), entradas por teclado (líneas 17 y 19) y por medio de ciclos `for` anidados se imprimió el arreglo. Al final está la ejecución del programa.



```
10          char matriz1[][] = new char [5][2];
11          matriz1[0][0]= 'b';
12          matriz1[2][1]= 'p';
13          matriz1[2+1][1]= '+';
14          matriz1[2+1][1-1]= '/';
15          matriz1[1][1]= '?';
16          System.out.print("Introduce un caracter (fila 4,col 0) ");
17          matriz1[4][0] = obj.next().charAt(0);
18          System.out.print("Introduce un caracter (fila 1,col 0) ");
19          matriz1[1][0] = obj.next().charAt(0);
20          System.out.printf("%16s%11s\n","Columna 0","Columna 1");
21          for (int i = 0; i < 5; i++) {
22              System.out.print("Fila "+i);
23              for (int j = 0; j < 2; j++) {
24                  System.out.printf("%5c\t",matriz1[i][j]);
25              }
26              System.out.println();
27          }
```

```
run:
Introduce un caracter (fila 4,col 0) *
Introduce un caracter (fila 1,col 0) &
       Columna 0  Columna 1
Fila 0     b
Fila 1     &          ?
Fila 2                p
Fila 3     /          +
Fila 4     *
BUILD SUCCESSFUL (total time: 13 seconds)
```

Figura 4.21 Asignación, lectura e impresión de una matriz.

El primer ciclo for (línea 21) se utiliza para las filas, y el segundo for (línea 23) para las columnas.

Se puede acceder a los elementos de una matriz por medio de las filas o las columnas según el planteamiento del problema.

En el siguiente fragmento de programa se declaró una matriz de 3X3 (`tabla`) e inicializó con valores reales ({3.5,4.8,1.3},{2.6,4.0,6.2},{10.4,7.7,9.0}), se sumaron los elementos de cada columna y se imprimió tanto la matriz como los resultados de cada columna, ver la Figura 4.22 (código y ejecución del programa).

```
15          //Suma de elementos por columna
16          for (int i = 0; i < 3; i++) {
17              suma = 0;
18              for (int j = 0; j < 3; j++) {
19                  suma = suma + tabla[j][i];
20              }
21              System.out.println("Suma de la columna "+i+ " es "+suma);
22          }
```

```
run:
Tabla de 3*3
3.5     4.8     1.3
2.6     4.0     6.2
10.4    7.7     9.0

Suma de la columna 0 es 16.5
Suma de la columna 1 es 16.5
Suma de la columna 2 es 16.5
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 4.22. Suma de elementos por columnas.

4.4 Arreglo multidimensional (tridimensional)

Es posible crear arreglos de tantas dimensiones como las aplicaciones requieren: 3, 4 o más, aunque raramente los datos del mundo real requieren más de 3 dimensiones. Los arreglos tridimensionales (tres dimensiones) son similares a los bidimensionales, sólo hay que agregar un índice más para referenciar a un elemento. La Figura 4.23 muestra el arreglo tridimensional `tri` que contiene 27 elementos de tipo entero. Observe que el arreglo tiene 3 filas, 3 columnas y 3 planos de profundidad, por lo tanto, para acceder a cada uno de los elementos se necesitan 3 índices. Si queremos acceder al elemento de la segunda fila, primera columna segundo plano de profundidad, escribiremos `tri[1][0][1]`. El valor de `tri[2][0][1]` es 62, el de `tri[0][1][2]` es 10. La suma de `tri[1][1][1]` y `tri[1][1][2]` es 50, el producto de `tri[0][0][0]` y `tri[2][1][0]` es 23.

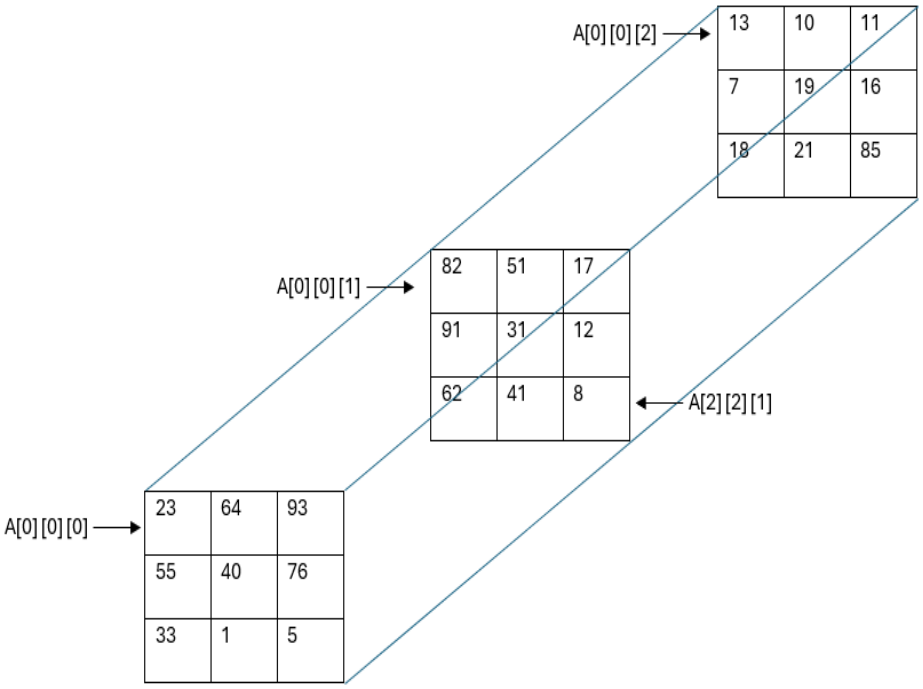


Figura 4.23.Índices y elementos de un arreglo tridimensional.

Para una mejor comprensión del uso de los arreglos tridimensionales se realizará un ejemplo. En el Instituto Tecnológico de Toluca se guarda la información sobre el número de alumnos que han ingresado en 4 de las diferentes carreras, por semestre, en los últimos 3 años. Hacer un programa en Java que calcule e imprima lo siguiente:

a) Año en que ingreso el mayor número de alumnos al Instituto.
b) Carrera que obtuvo el mayor número de alumnos el último año.
c) ¿En qué año la carrera de Sistemas obtuvo el mayor número de alumnos?

Las carreras tienen un valor numérico y son las siguientes:

1. Industrial
2. Química
3. Sistemas
4. Tic´s

Para este caso se ha declarado una matriz tridimensional de 4X2X3, ya que son 4 carreras, hay ingreso de alumnos semestralmente (2 semestres al año) y la información es de los últimos 3 años. También es necesario utilizar 3 ciclos for anidados para el acceso a los elementos del arreglo.

En la Figura 4.24 se muestra la declaración de 3 constantes (línea 8) que se utilizarán para los índices del arreglo tridimensional (`carr` para la carrera, `sem` para el semestre y a para el año). El ciclo `for (int k = 0; k < a; k++)` llevará el conteo de los años (3), el

ciclo `for (int i = 0; i < carr; i++)` controlará las carreras (4), y el for `(int j = 0; j < sem; j++)` hará lo propio para el semestre.

```
8           final int carr = 4, sem = 2, a = 3;
9           int mayor, an, suma;
10          int tec[][][] = new int[carr][sem][a];
11          Scanner s = new Scanner(System.in);
12
13          //Almacenar información en el arreglo
14          System.out.println("Número de alumnos que ingresaron");
15          for (int k = 0; k < a; k++) {
16              for (int i = 0; i < carr; i++) {
17                  for (int j = 0; j < sem; j++) {
18                      System.out.printf("Año: %d\tCarrera: %d\tSemestre: %d: ",k+1,i+1,j+1);
19                      tec[i][j][k] = s.nextInt();
20                  }
21              }
22          }
```

Figura 4.24. Código para almacenar información en arreglo tridimensional.

La ejecución de esta parte del programa se muestra en la Figura 4.25.

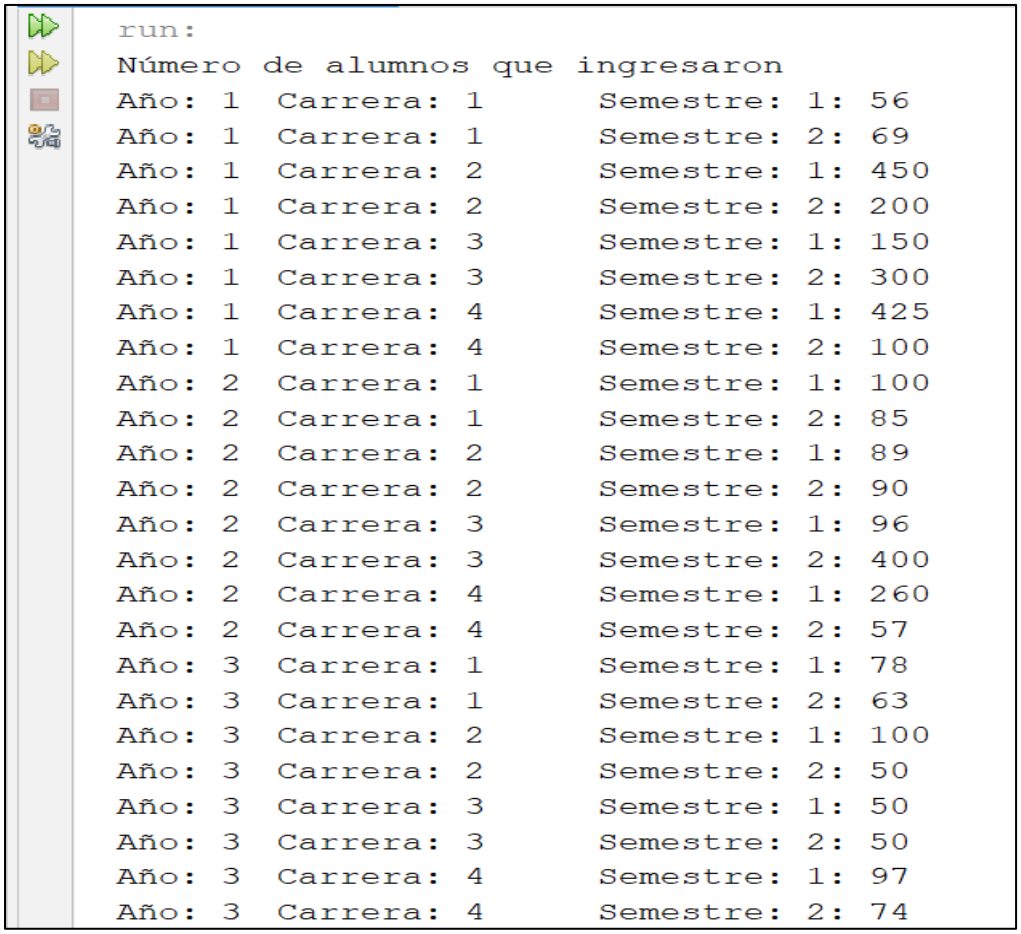


Figura 4.25. Ejecución del programa (parte 1 de 4).

Una vez almacenados los datos en el arreglo, se procede a resolver el inciso a) Año en que ingresó el mayor número de alumnos al Instituto. Observe en la Figura 4.26 que el ciclo externo (línea 25) corresponde a los años, por cada año se suma el ingreso de los dos semestres de las 4 carreras, después se verifica que año tuvo mayor ingreso de alumnos (línea 31) y se guarda el año en la variable an. Por último, se imprimen los resultados (año y total de alumnos inscritos.

```
23          //Año en que ingresó el mayor número de alumnos al Instituto.
24          mayor = 0; an = -1;
25          for (int k = 0; k < a; k++) {
26              suma = 0;
27              for (int i = 0; i < carr; i++)
28                  for (int j = 0; j < sem; j++) {
29                      suma += tec[i][j][k];
30                  }
31              if(suma > mayor){
32                  mayor = suma;
33                  an = k;
34              }
35          }
36          System.out.println("Año en que ingresó el mayor número de alumnos: "+(an+1));
37          System.out.println("Total de alumnos que ingresaron: "+mayor);
```

Figura 4.26. Año en que ingresó el mayor número de alumnos al Instituto.

Con los datos ingresados anteriormente (Figura 4.25), el resultado quedaría como se muestra en la Figura 4.27.

```
Año en que ingresó el mayor número de alumnos: 1
Total de alumnos que ingresaron: 1750
```

Figura 4.27. Ejecución del programa (parte 2 de 4).

El inciso b) Carrera que obtuvo el mayor número de alumnos el último año. En la Figura 4.28 puede ver el código para este inciso, cómo puede darse cuenta no se utiliza un ciclo para los planos de la profundidad, ya que es un dato (a). En otras palabras, como ya sabemos que la información que se necesita es del último año y el año es 3, no es necesario hacer un ciclo.

```
39          //Carrera con mayor número de alumnos en el último año
40          mayor = 0; an = -1;
41          for (int i = 0; i < carr; i++) {
42              suma = 0;
43              for (int j = 0; j < sem; j++)
44                  suma += tec[i][j][a-1];
45              if(suma > mayor){
46                  mayor = suma;
47                  an = i;
48              }
49          }
50          System.out.println("Carrera con mayor número de alumnos "+(an+1));
51          System.out.println("Total alumnos que ingresaron en último año(3): "+mayor);
```

Figura 4.28. Carrera que obtuvo el mayor número de alumnos el último año.

La impresión de resultados quedaría como se muestra en la Figura 4.29.

```
Carrera con mayor número de alumnos 4
Total alumnos que ingresaron en último año(3): 171
```

Figura 4.29. Ejecución del programa (parte 3 de 4).

Por último, nos piden ¿En qué año la carrera de Sistemas obtuvo el mayor número de alumnos? En la Figura 4.30 puede ver el código, no se utiliza el ciclo de la carrera (`carr`) porque ya sabemos que a la carrera de sistemas le corresponde el número 3, así que sólo usaremos el ciclo del año y el del semestre.

```
52          //Año en que la carrera de Sistemas obtuvo mayor número de alumnos
53          mayor = 0; an = -1;
54          for (int k = 0; k < a; k++)
55          {
56              suma = 0;
57              for (int j = 0; j < sem; j++)
58                  suma += tec[carr-2][j][k];
59              if(suma > mayor){
60                  mayor = suma;
61                  an = k;
62              }
63          }
64          System.out.println("Carrera Sistemas 3");
65          System.out.println("Año en que la carrera de Sistemas obtuvo el mayor número de alumnos "+(an+1));
66          System.out.println("Total alumnos de Sistemas: "+mayor);
```

Figura 4.30. Año en que la carrera de Sistemas obtuvo el mayor ingreso de alumnos.

En la Figura 4.31. se muestran los resultados al ejecutar el programa.

```
Año en que la carrera de Sistemas obtuvo el mayor número de alumnos 2
Total alumnos de Sistemas: 496
```

Figura 4.31. Ejecución del programa (parte 4 de 4).

4.5 Programas resueltos

Ejercicio 1. Inicialice un arreglo unidimensional con 10 elementos reales. Calcule e imprima la suma de los elementos del arreglo.

Entrada: No hay entrada por teclado ya que el arreglo unidimensional esta inicializado.

Proceso:

```
para (i = 0; i < 10; i++)
    suma += vector[i];
```

Salida: Impresión de resultado, se imprime el arreglo para mayor claridad y el resultado de la suma (`suma`).

En la Figura 4.32 se muestra el Ejercicio 1 resuelto, utilizando un ciclo for para acceder a los elementos del arreglo y realizar la suma.

```
1   package capitulo4;
2
3   public class Ejercicio1 {
4       public static void main(String[] args) {
5           double vector[]= {5.6, 8.9, 3.9,6.5, 4.98, 2.04, 7.8, 9.7,10.3, 2.5};
6           double suma =0;
7           System.out.println("Impresión de los elementos del arreglo");
8           for (int i = 0; i < vector.length; i++) {
9               System.out.printf("vector[%d]= %.2f\n",i,vector[i]);
10          }
11          System.out.println();
12          for (int i = 0; i < vector.length; i++)
13          {
14              suma += vector[i];
15          }
16          System.out.println("La suma del arreglo es "+suma);
17      }
18  }
```

Figura 4.32. Ejercicio 1 codificado, usando arreglo unidimensional.

Explicación:

Línea 5. Se declara e inicializa el arreglo unidimensional (`vector[]`) con 10 elementos reales (`double`).

Línea 6. Se declara e inicializa en cero la variable `suma`, la cual acumulará todos los valores del arreglo.

Líneas 8 a la 10. Se imprime el arreglo mediante un primer ciclo `for` para visualizar los valores.

Líneas 12 a la 15. Se utiliza un segundo ciclo `for` para acceder a cada uno de los elementos del arreglo, recuerde que la variable de control (`i`) del ciclo for se utiliza como índice del arreglo. Este ciclo se repetirá 10 veces (del 0 al 9) porque son 10 elementos en el arreglo y se sumaran a la variable `suma`.

Línea 16. Imprime el resultado de la suma de los 10 elementos que están almacenados en el arreglo.

En la Figura 4.33 podrá ver la ejecución del Ejercicio 1.

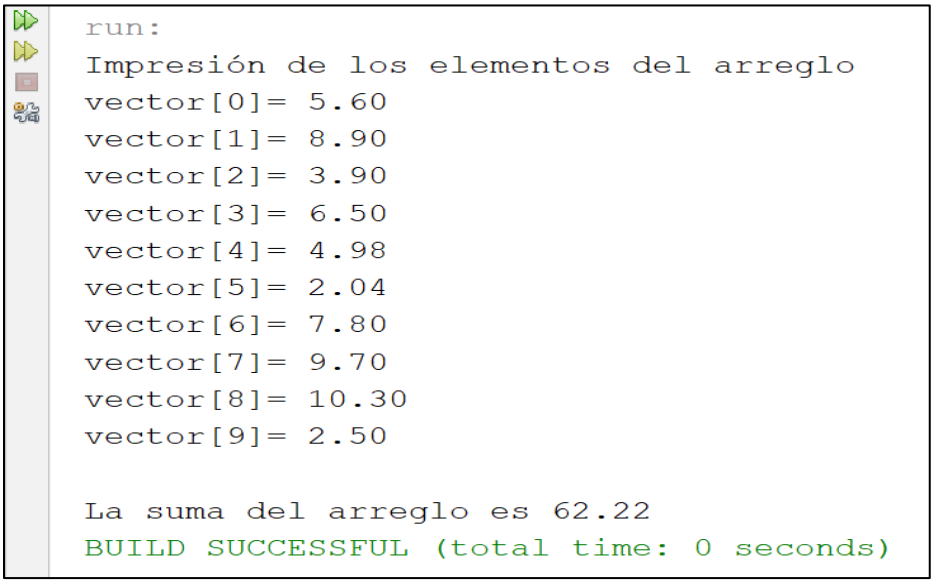


Figura 4.33. Ejecución del programa Ejercicio 1.

Ejercicio 2. Dados dos vectores `a` y `b` enteros de 5 elementos cada uno, multiplicar el primer elemento del arreglo `a`, por el primer elemento del arreglo `b`, el resultado almacenarlo en la primera posición de un tercer arreglo `c`, luego el segundo elemento del arreglo `a` multiplicarlo por el segundo elemento del arreglo `b` y guardar el resultado en la segunda posición del arreglo `c` y así sucesivamente. Imprimir los tres arreglos en formato de columnas.

Entrada: arreglos unidimensionales `a[]` y `b[]`.

Proceso:

```
para (i = 0; i < 5; i++)
    Leer a[i];
    Leer b[i];
    c[i]=a[i]*b[i];
```

Salida: Imprimir los tres arreglos (`a[]`, `b[]` y `c[]`) en formato de columna.

En la Figura 4.34 se muestra el código del Ejercicio 2 resuelto, utilizando un ciclo `for` para acceder a los elementos del arreglo y realizar la multiplicación de los elementos de los arreglos.

```
1   package capitulo4;
2
3   import java.util.Scanner;
4
5   public class Ejercicio2 {
6       public static void main(String[] args) {
7           int []a,b,c;
8           a = new int[5];
9           b = new int[5];
10          c = new int[5];
11          Scanner s = new Scanner(System.in);
12
13          for (int i = 0; i < 5; i++) {
14              System.out.println("Captura de datos arreglo a");
15              System.out.print("Introduce el elemento "+i+": ");
16              a[i]=s.nextInt();
17              System.out.println("Captura de datos arreglo b");
18              System.out.print("Introduce el elemento "+i+": ");
19              b[i]=s.nextInt();
20              c[i]=a[i]*b[i];     //Multiplicación
21          }
22          System.out.println("Impresión de arreglos");
23          System.out.printf("%5s%5s%5s\n","a","b","c");
24          for (int i = 0; i < 5; i++) {
25              System.out.printf("%5d%5d%5d\n",a[i],b[i],c[i]);
26          }
27      }
28  }
```

Figura 4.34. Ejercicio 2 codificado, usando arreglos unidimensionales.

Explicación:

Línea 7. Se declara tres arreglos de tipo entero (`a[], b[]` y `c[]`).

Líneas 8 a la 10. Se crean los tres arreglos, recordar que para Java los arreglos son objetos.

Líneas 13 a la 21. Se utiliza un ciclo `for`, para capturar los datos del arreglo `a` y del arreglo `b` (líneas 16 y 19). En la línea 20 se realiza la multiplicación y se asigna al arreglo `c`.

Línea 23. Se imprimen los títulos de los arreglos mediante el método `printf().`

Líneas 24 a la 26. Por medio del ciclo for se imprimen los tres arreglos, se utiliza el método `printf()` para imprimir con formato de columnas

En la Figura 4.35 podrá ver la ejecución del Ejercicio 2.

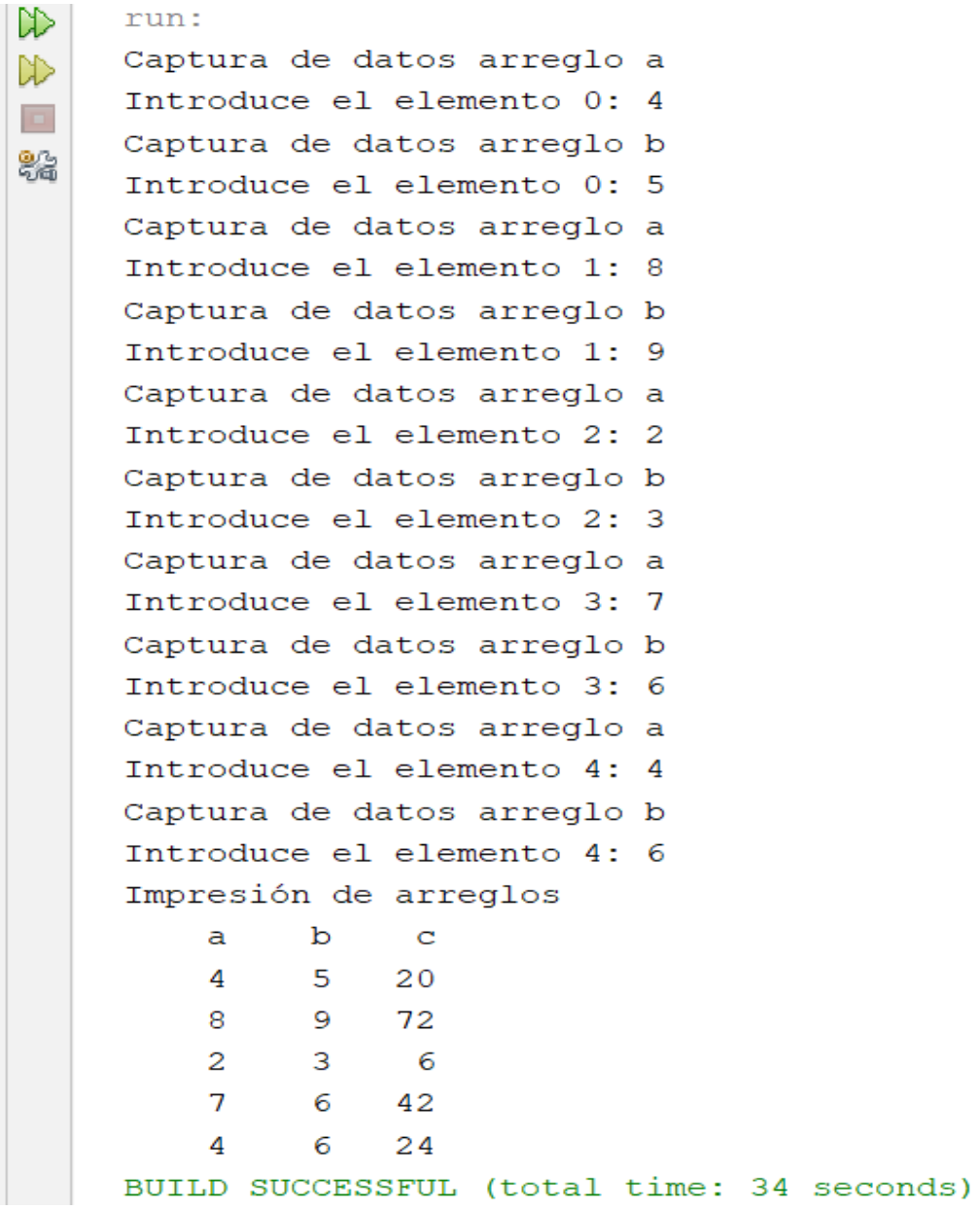


Figura 4.35. Ejecución del programa Ejercicio 2.

Ejercicio 3 . Almacene en una lista 15 números enteros e imprimir

a) cuántos son cero.
b) cuántos son negativos
c) cuántos positivos.
d) la suma de los negativos y la suma de los positivos.

Entrada: arreglo unidimensional `lista[];`

Proceso:

```
para (int i = 0; i < 15; i++)
   leer lista[i];
   si (lista[i]== 0)
      cero++;
   sino si(lista[i]>0){
      pos++;
      sumaPos += lista[i];
   sino
      neg++;
      sumaNeg += lista[i];
```

Salida: Imprimir los resultados: Total de ceros (`cero`), Total de positivos (`pos`), Total negativos (`neg`), suma de negativos(`sumaNeg`) y suma positivos (`sumaPos`).

En la Figura 4.36 se muestra el código del Ejercicio 3 resuelto, utilizando un ciclo `for` para acceder a los elementos del arreglo `lista[]` y un `if` anidado para verificar si el número introducido es cero, negativo o positivo.

```java
package capitulo4;

import java.util.Scanner;

public class Ejercicio3 {
    public static void main(String[] args) {
        int lista[]=new int[15];
        int pos = 0,neg = 0,cero = 0,sumaPos = 0,sumaNeg = 0;
        Scanner s = new Scanner(System.in);

        for (int i = 0; i < lista.length; i++) {
            System.out.print("lista["+i+"]= ");
            lista[i]= s.nextInt();
            if(lista[i]== 0){
                cero++;
            }else if(lista[i]>0){
                pos++;
                sumaPos += lista[i];
            }else{
                neg++;
                sumaNeg += lista[i];
            }
        }
        System.out.println("Total de números ceros: "+cero);
        System.out.println("Total de números negativos: "+neg);
        System.out.println("Total de números positivos: "+pos);
        System.out.println("Suma de números negativos: "+sumaNeg);
        System.out.println("Suma de números positivos: "+sumaPos);
    }
}
```

Figura 4.36. Ejercicio 3 codificado, usando arreglos unidimensionales.

Explicación:

Línea 7. Se declara y se crea el arreglo de tipo entero (`lista[]`).

Líneas 8. Se declaran e inicializan en cero las variables, `pos, neg y cero` contarán los números positivos, negativos y cero respectivamente, `sumaPos` sumará los números positivos y `sumaNeg` sumará los números negativos.

Líneas 11 a la 23. Se utiliza un ciclo `for`, para capturar los datos del arreglo `lista[].` Una vez capturado el primer elemento `lista[0]` (líneas 13), se verifica si es igual a cero (línea 14), si la condición es verdadera se incrementa el contador de ceros(`cero++`) sino evalúa la condición, si el elemento es mayor a cero (línea 16), si la condición es verdadera, se incrementa el contador de números positivos (`pos++`) y se suma el elemento a `sumaPos` (línea 18), sino se incrementa el contador de los números negativos (`neg++`) y se suma el elemento a `sumaNeg` (línea 21), y así sucesivamente hasta llegar al elemento 14 del arreglo.

Líneas 24 a 28. Se imprimen los resultados solicitados.

En la Figura 4.37 podrá ver la ejecución del Ejercicio 3 y comprobar los resultados.

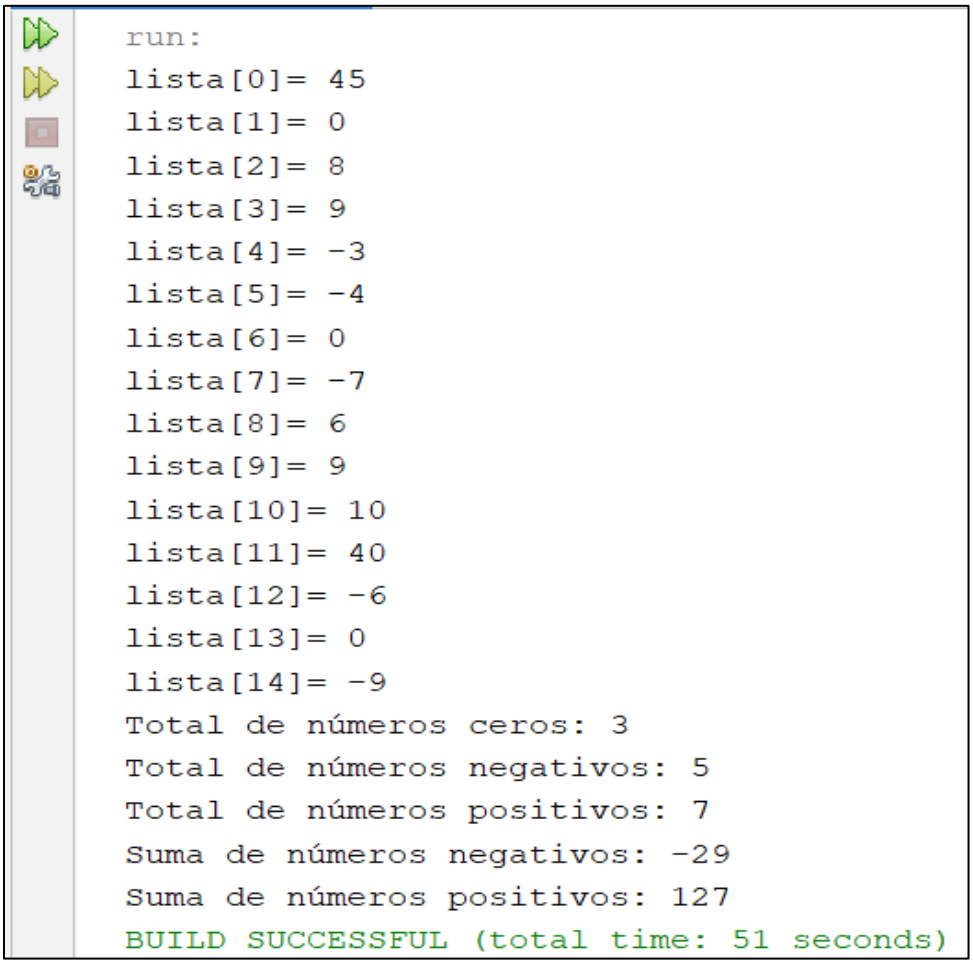


Figura 4.37 Ejecución del programa Ejercicio 3.

Ejercicio 4. Inicializar con números enteros una matriz de 4X4, imprimir los elementos de la diagonal principal y la diagonal secundaria de la matriz.

Entrada: No hay entrada por teclado ya que el arreglo bidimensional esta inicializado.

Proceso: Diagonal principal

```
para (i = 0; i < 4; i++)
    imprimir matriz[i][i];
```

Diagonal secundaria

```
j =3;
para (int i = 0; i < 4; i++)
  imprimir matriz[i][j];
  j--;
```

Salida: Imprimir los resultados: mat[][], los elementos de la diagonal principal (`mat[0][0], mat[1][1], mat[2][2]` y `mat[3][3]`), los elementos de la diagonal secundaria (`mat[0][3], mat[1][2], mat[2][1]` y `mat[3][0]`)

Puede darse cuenta de que todos los elementos de la diagonal principal tienen el índice de filas y el índice de columnas iguales, por lo tanto, con un solo ciclo `for` se puede imprimir dichos elementos. Para el caso de la diagonal secundaria, mientras que el índice de las filas se va incrementando, el índice de las columnas se va decrementando, también se utiliza sólo un ciclo `for`. En la Figura 4.38 se muestra el código del Ejercicio 4 resuelto.

```
1  package capitulo4;
2
3  public class Ejercicio4 {
4      public static void main(String[] args) {
5          int matriz[][]= {{20,35,45,10},{8,16,12,90},{4,55,62,30},{25,30,47,60}};
6
7          System.out.println("Impresión de matriz");
8          for (int i = 0; i < 4; i++) {
9              for (int j = 0; j < 4; j++) {
10                 System.out.printf("%d\t",matriz[i][j]);
11             }
12             System.out.println();
13         }
14         System.out.println("Impresión elementos de diagonal principal");
15         for (int i = 0; i < 4; i++) {
16             System.out.println(matriz[i][i]);
17         }
18         System.out.println("Impresión elementos de diagonal secundaria");
19         int j =3;
20         for (int i = 0; i < 4; i++) {
21             System.out.println(matriz[i][j]);
22             j--;
23         }
24     }
25 }
```

Figura 4.38. Ejercicio 4 codificado, usando arreglo bidimensional.

Explicación:

Línea 5. Se declara y se inicializa el arreglo bidimensional de 4X4 de tipo entero (`mat[][]`).

Líneas 15 a la 17. Diagonal principal, se utiliza un ciclo `for`, ya que los índices de las filas y columnas son iguales, la variable de control `i`, del ciclo representará ambos índices. Cada vez que el ciclo se repite imprime un elemento de la diagonal principal (línea 16).

Líneas 11 a la 23. Diagonal secundaria, Se utiliza un ciclo `for` para el control de los índices de las filas (`i`) y para las columnas se utiliza una variable entera (j) inicializada en 3, con la finalidad de que cada vez que el ciclo se repita, la variable `j` se decremente hasta llegar a cero. Cada vez que el ciclo se repite imprime un elemento de la diagonal secundaria (línea 21).

En la Figura 4.39 podrá ver la ejecución del Ejercicio 4 y comprobar los resultados.

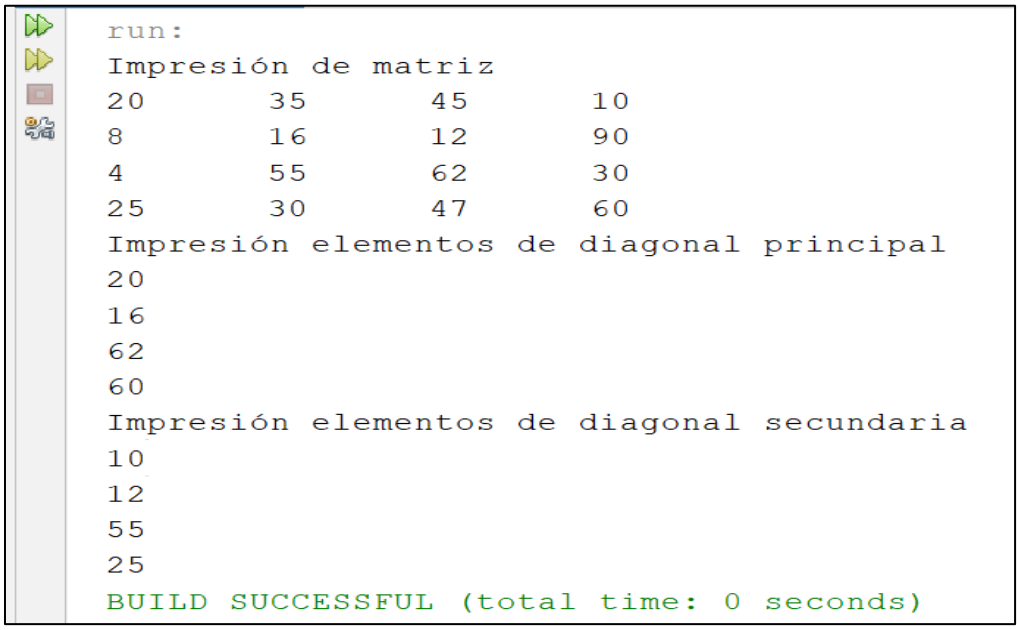


Figura 4.39 Ejecución del programa Ejercicio 4.

Ejercicio 5. Dada una matriz cuadrada `mat`, hacer un programa que permita determinar si dicha matriz es simétrica. Se considera que una matriz es simétrica sí `mat[i][j] = mat[j][i]` y esto se cumple para todos los elementos `i,j` de la matriz.

Entrada: Tamaño del arreglo `tam` y Arreglo bidimensional `mat[][]` de tipo entero.

Proceso:

```
bandera = verdadera;  i = 0;
mientras((i < tam) y (bandera = verdadera))
    j = 0;
    mientras((j<=(i-1)) y (bandera = verdadera))
```

```
            si(mat[i][j] = mat[j][i])
               j++;
            sino
               bandera = false;
      i++;
```

Salida: Imprimir los resultados:

```
si(bandera = verdadera)
   imprimir "Es simétrica";
sino
    imprimir "No es simétrica";
```

Una vez que se introduzca el tamaño (`tam`) de la matriz se puede llenar del arreglo bidimensional por medio de un un ciclo `for`. Sin embargo, para las comparaciones (`mat[i][j] == mat[j][i]`) es mejor usar un ciclo `while`, para que en el momento que `mat[i][j] ≠ mat[j][i]`, bandera se vuelve falsa y ya no se repite el ciclo. En la Figura 4.40. se muestra el código del Ejercicio 5 resuelto.

Explicación:

Línea 11. Se solicita el tamaño del arreglo bidimensional.

Línea 12. Se declara y crea el arreglo bidimensional `mat[][]` de tamaño `tam`.

Líneas 13 a la 18. Se capturan los datos en el arreglo mediante ciclos anidados `for`.

Líneas 20 a la 29. Mediante ciclos anidados while se verifica si `mat[i][j] == mat[j][i]`, se utiliza este ciclo ya que no es necesario comparar con los elementos que tiene el índice igual tal como `mat[0][0], mat[1][1], mat[2][2]`, etcétera.

En la Tabla 4.1 se realizará la prueba de escritorio. Considere que `tam = 3` y que la matriz si es simétrica

Tabla4.1. Prueba de escritorio Ejercicio 5.

bandera	i	while((i<tam) && (bandera))	j	while((j<=(i-1)) &&(bandera))	if (mat[i][j] == mat[j][i])	If (bandera)	imprime
true	0	true	0	false			
	1	true	0	true	true		
			1	false			
	2	true	0	true	true		
			1	true	true		
			2	false			
	3	false				true	Es simétrica

```java
package capitulo4;

import java.util.Scanner;

public class Ejercicio5 {
    public static void main(String[] args) {
        int tam,i,j;
        boolean bandera;
        Scanner s = new Scanner(System.in);
        System.out.print("Ingrese el tamaño de la matriz cuadrada: ");
        tam = s.nextInt();
        int mat[][]= new int[tam][tam];
        for (i = 0; i < tam; i++) {
            for (j = 0; j < tam; j++) {
                System.out.print("Ingrese el elemento mat["+i+"]["+j+"]= ");
                mat[i][j]= s.nextInt();
            }
        }
        bandera = true;  i = 0;
        while((i < tam)&&(bandera)){
            j = 0;
            while((j<=(i-1))&&(bandera)){
                if(mat[i][j] == mat[j][i])
                    j++;
                else
                    bandera = false;
            }
            i++;
        }
        if(bandera)
            System.out.println("Es simétrica");
        else
            System.out.println("No es simétrica");
    }
}
```

Figura 4.40. Ejercicio 5 codificado, usando arreglo bidimensional.

En la Figura 4.41 puede observar la ejecución del programa Ejercicio 5, cuando la matriz es simétrica, recordar que es simétrica si `mat[i][j] = mat[j][i]` para todos los elementos de la matriz

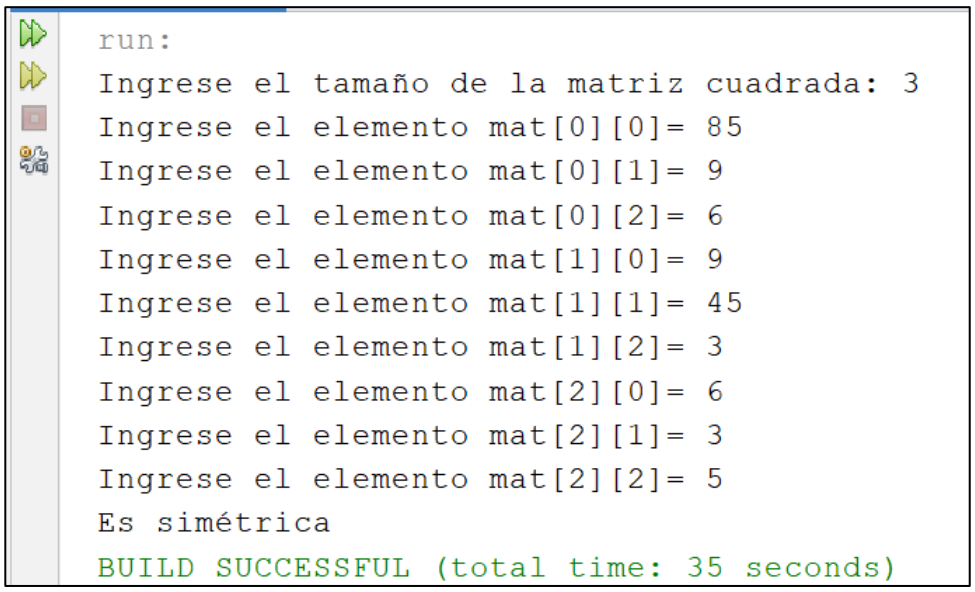


Figura 4.41. Ejecución del Ejercicio 5 para matriz simétrica.

En la Figura 4.42 verá la ejecución del mismo programa, pero ahora indicando que no es simétrica. El elemento `mat[0][2] ≠ mat[2][0]`.

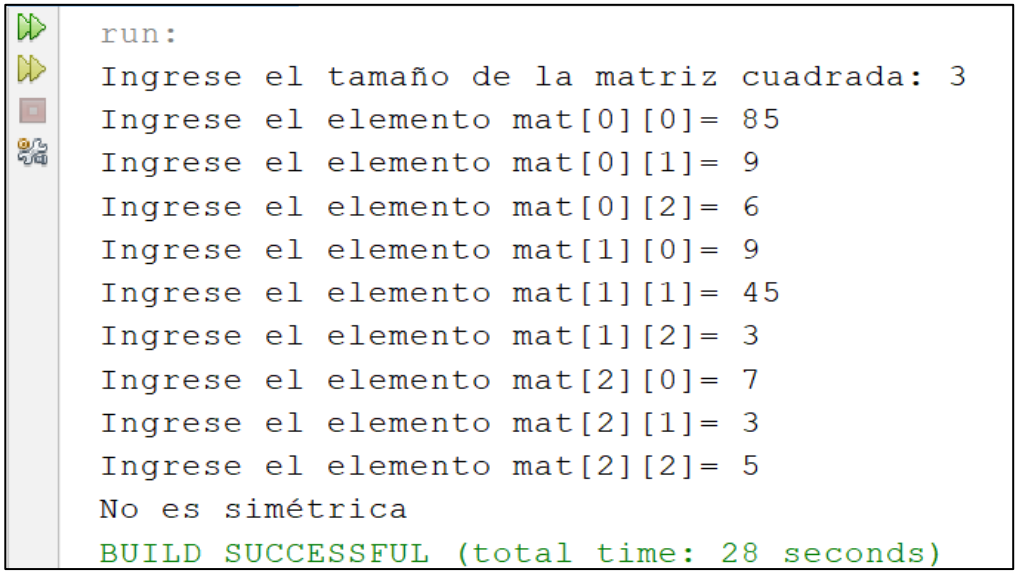


Figura 4.42. Ejecución del Ejercicio 5 para matriz no simétrica.

Ejercicio 6. Se tienen los costos de producción de 3 departamentos (dulces, bebidas y conservas), correspondientes a los 12 meses del año anterior. Hacer un programa que proporcione la siguiente información:

a) ¿En qué mes se registró el mayor costo de producción de dulces?
b) Promedio anual de los costos de producción de bebidas.
c) ¿En qué mes se registró el mayor costo de producción en bebidas, y en qué mes el menor costo?
d) ¿Cuál fue el departamento que tuvo el menor costo de producción en diciembre?

Entrada: Arreglo bidimensional `prod[12][3]` de tipo real.

Proceso: Para el inciso a)

```
mayorDulce = prod[0][0];
para (i = 1; i < 12; i++)
   si(mayorDulce < prod[i][0])
      mayorDulce = prod[i][0];
      mes = i;
```

Para el inciso b)

```
para (i = 0; i < 12; i++)
     suma += prod[i][1];
```

Para el inciso c)

```
mesMayor = 0; mesMenor = 0;
mayorBeb = prod[0][1]; menorBeb = prod[0][1];
para (i = 1; i < 12; i++)
    si(mayorBeb < prod[i][1])
       mayorBeb = prod[i][1];
       mesMayor = i;
 sino si(menorBeb > prod[i][1]){
     menorBeb = prod[i][1];
     mesMenor = i;
```

Para el inciso d)

```
menor = prod[11][0];
para (j = 1; j < 3; j++)
    si(menor > prod[11][j])
```

```
menor = prod[11][j];
depto = j;
```

Salida: Imprimir los resultados:

a) Mes en que se registró mayor producción de dulces (`mes`).
b) Promedio anual de los costos de producción de bebidas (`suma/12`).
c) Mes con mayor costo de producción en bebidas(`mayorBeb`) y mes con el menor costo (`menorBeb`).
d) Departamento que tuvo el menor costo de producción en diciembre (`depto`).

Debido al tamaño del programa se presentará por partes. En la Figura 4.43 se muestra parte del código del Ejercicio 6, la declaración y creación del arreglo (`float prod[][] = new float[12][3]`), las variables que se utilizan (`mes, mesMayor, mesMenor, depto, mayorDulce, suma, mayorBeb, menorBeb,menor`) y la entrada de datos al arreglo. Como se solicitan los costos de producción de los 12 meses del año anterior de tres departamentos (dulces, bebidas y conservas), se crea un arreglo bidimensional de 12X3 (línea 8). Se capturan los datos en las líneas 12 a la 18 mediante ciclos `for` anidados, donde `i` representará los meses del año y, `j` representará los departamentos.

```
1  package capitulo4;
2
3  import java.util.Scanner;
4
5  public class Ejercicio6 {
6      public static void main(String[] args) {
7          Scanner s= new Scanner(System.in);
8          float prod[][] = new float[12][3];
9          int mes = 0,mesMayor, mesMenor,depto = 0;
10         float mayorDulce,suma, mayorBeb,menorBeb,menor;
11
12         for (int i = 0; i < 12; i++) {
13             System.out.println("Mes "+(i+1));
14             for (int j = 0; j < 3; j++) {
15                 System.out.print("Ingrese el costo de producción del departamento "+(j+1)+": ");
16                 prod[i][j]= s.nextFloat();
17             }
18         }
```

Figura 4.43. Código del Ejercicio 6 (parte 1 de 4).

En la Figura 4.44 se muestra el código para resolver dos de los incisos del problema.

Explicación:

a) ¿En qué mes se registró el mayor costo de producción de dulces?
En la línea 30 se inicializa la variable `mayorDulce = prod[0][0]`, que es el primer costo de producción del departamento de dulces del primer mes. Como puede observar sólo se utiliza un ciclo `for`, ya que el índice de las columnas representa el departamento de dulces (índice 0) y este valor no cambiará en las iteraciones del ciclo. Cuando inicie el ciclo `for` (línea 31) se evaluará mayorDulce < prod[i][0], si la evaluación es verdadera, se realizará mayorDulce = prod[i][0], guardando el nuevo valor y se asignará a la variable `mes` el valor de `i`, recordar que `i` representa los meses, en caso contrario se incrementará la variable de control `i` del ciclo `for` y se repetirá mientras `i<12`.

```
29          //a)¿En qué mes se registró el mayor costo de producción de dulces? índice 0
30          mayorDulce = prod[0][0];
31          for (int i = 1; i < 12; i++) {
32              if(mayorDulce < prod[i][0]){
33                  mayorDulce = prod[i][0];
34                  mes = i;
35              }
36          }
37          System.out.println("En el mes "+(mes+1)+" se registró el mayor costo de producción de dulces");
38          System.out.printf("La producción fue de %.2f\n",mayorDulce);
39          System.out.println();
40
41          //b)Promedio anual de los costos de producción de bebidas. índice 1
42          suma = 0;
43          for (int i = 0; i < 12; i++) {
44                  suma += prod[i][1];
45          }
46          System.out.printf("El promedio anual de los costos de producción de bebidas es %.2f\n",(suma/12));
```

En la Figura 4.44 Código del Ejercicio 6 (parte 2 de 4).

b) Promedio anual de los costos de producción de bebidas. Para este inciso se inicializa suma = 0 (línea 42), la cual acumulará todos los costos de producción del departamento de bebidas, por tal motivo sólo se utiliza un ciclo `for` ya que los departamentos están representados por el índice de la columna, y para este caso el departamento de bebidas le corresponde el índice 1.

En la Figura 4.45 se presenta el código para resolver el siguiente inciso:

c)¿En qué mes se registró el mayor costo de producción en bebidas, y en qué mes el menor costo?

Al igual que en el anterior inciso sólo se utiliza un ciclo `for` debido a que los departamentos están representados por el índice de la columna. Se inicializan las variables `mayorBeb` y `menorBeb` con el primer costo de producción del departamento de bebidas (`prod[0][1]`), luego entra al ciclo for (línea 53) evalúa la condición `mayorBeb < prod[i][1]` (línea 54) para determinar si es el mayor costo de producción y asignar a `mesMayor` el valor de i que corresponde al mes de mayor producción, en caso contrario evalúa la condición `menorBeb > prod[i][1]` (línea 57) para determinar si es el menor costo de producción y asignar a `mesMenor` el valor de i que corresponde al mes de menor producción.

```
49          //c)¿En qué mes se registró el mayor costo de producción en bebidas, y en qué mes el menor costo?
50          mesMayor = 0; mesMenor = 0;
51          mayorBeb = prod[0][1];
52          menorBeb = prod[0][1];
53          for (int i = 1; i < 12; i++) {
54              if(mayorBeb < prod[i][1]){
55                 mayorBeb = prod[i][1];
56                 mesMayor = i;
57              }else if(menorBeb > prod[i][1]){
58                  menorBeb = prod[i][1];
59                  mesMenor = i;
60              }
61          }
62          System.out.println("En el mes "+(mesMayor+1)+" se registro el mayor costo de producción de bebidas");
63          System.out.printf("Total de producción: %.2f\n",mayorBeb);
64          System.out.println("En el mes "+(mesMenor+1)+" se registro el menor costo de producción de bebidas");
65          System.out.printf("Total de producción: %.2f\n",menorBeb);
```

Figura 4.45. Código del Ejercicio 6 (parte 3 de 4).

d)¿Cuál fue el departamento que tuvo el menor costo de producción en diciembre? Se inicializa la variable `menor = prod[11][0]`, se sabe que el índice de las filas corresponde a los meses del año y que al mes de diciembre le corresponde el índice 11, por tal motivo no es necesario un ciclo para las filas, sólo para las columnas por qué se tiene que verificar qué departamento tuvo el menor costo de producción. El ciclo `for` empezará con `j = 1` que corresponde al segundo departamento (bebidas) debido a que el costo de producción del primer departamento (dulces) ya se asignó a la variable `menor` (línea 69), se evalua la condición `menor > prod[11][j]` (comparando los datos del departamento de dulces y bebidas), si se cumple la condición entonces `menor = prod[11][j]` y `depto = j`, teniendo de esta forma el departamento que tuvo menor producción, en caso contrario `j` se incrementa en uno y vuelve a evaluar la condición por última vez.

```
68          //d)¿Cuál fue el departamento que tuvo el menor costo de producción en diciembre?
69          menor = prod[11][0];
70          for (int j = 1; j < 3; j++) {
71              if(menor > prod[11][j]){
72                 menor = prod[11][j];
73                 depto = j;
74              }
75          }
76          System.out.println("El departamento "+(depto+1)+" tuvo el menor costo de producción en diciembre");
77          System.out.printf("Total de producción: %.2f\n",menor);
```

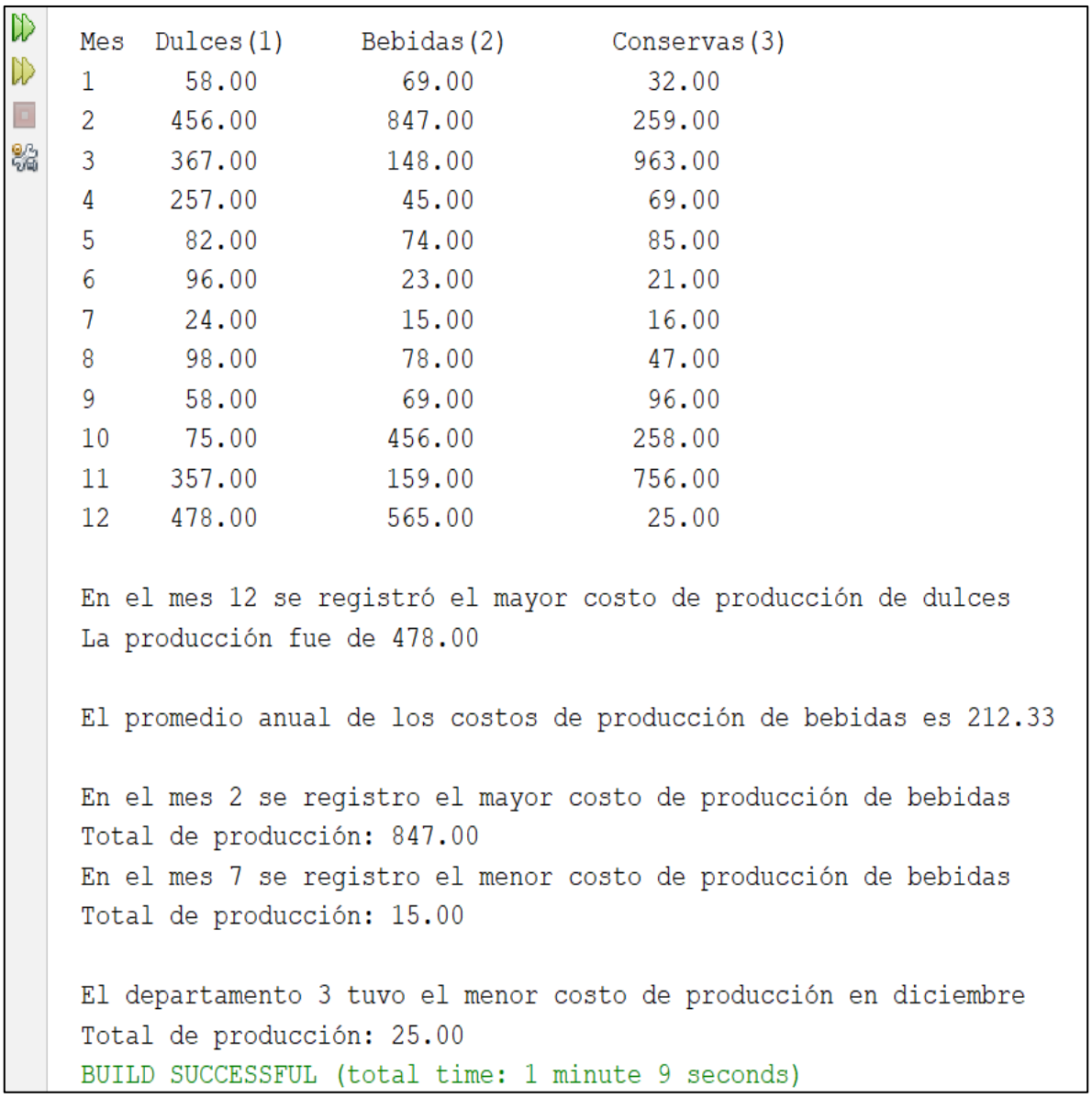
Figura 4.46. Código del Ejercicio 6 (parte 4 de 4).

Se ha agregado al código una impresión del arreglo bidimensional en forma de tabla para mayor comprensión, el cual puede ver en la Figura 4.47.

```
20          System.out.printf("%s%11s%15s%19s\n","Mes","Dulces(1)","Bebidas(2)","Conservas(3)");
21          for (int i = 0; i < 12; i++) {
22              System.out.printf("%-2d",(i+1));
23              for (int j = 0; j < 3; j++) {
24                  System.out.printf("%10.2f\t",prod[i][j]);
25              }
26              System.out.println();
27          }
```

Figura 4.47. Código de arreglo bidimensional con formato de tabla.

Por último, en la Figura 4.48 se muestra la ejecución del Ejercicio 6.



```
Mes   Dulces(1)      Bebidas(2)        Conservas(3)
1       58.00          69.00            32.00
2      456.00         847.00           259.00
3      367.00         148.00           963.00
4      257.00          45.00            69.00
5       82.00          74.00            85.00
6       96.00          23.00            21.00
7       24.00          15.00            16.00
8       98.00          78.00            47.00
9       58.00          69.00            96.00
10      75.00         456.00           258.00
11     357.00         159.00           756.00
12     478.00         565.00            25.00

En el mes 12 se registró el mayor costo de producción de dulces
La producción fue de 478.00

El promedio anual de los costos de producción de bebidas es 212.33

En el mes 2 se registro el mayor costo de producción de bebidas
Total de producción: 847.00
En el mes 7 se registro el menor costo de producción de bebidas
Total de producción: 15.00

El departamento 3 tuvo el menor costo de producción en diciembre
Total de producción: 25.00
BUILD SUCCESSFUL (total time: 1 minute 9 seconds)
```

Figura 4.48. Ejecución del Ejercicio 6 con arreglo bidimensional.

Ejercicio 7. En el arreglo tridimensional `pro`, se guardan las ventas mensuales de los últimos 3 años de 3 productos de una empresa farmacéutica: paracetamol, diclofenaco e ibuprofeno. Hacer un programa que permita proporcionar la siguiente información:

a) Las ventas totales de la empresa en el segundo año.
b) El producto que tuvo las mayores ventas en el último año, incluyendo también el importe de las ventas.
c) Producto, mes y año con la mayor venta, incluyendo el importe de las ventas.

Entrada: Arreglo tridimensional `prod[12][3][3]` de tipo real.

Proceso: Para el inciso a)

```
anio1 = 2;
para (i = 0; i < mes; i++)
   para (int j = 0; j < art; j++)
      suma += prod[i][j][anio1 – 1];
```

Para el inciso b)

```
para (i = 0; i < mes; i++)
  para (j = 0; j < art; j++)
     switch (j + 1)
         caso 1:
             suma1 += prod[i][j][anio - 1];
         caso 2:
             suma2 += prod[i][j][anio - 1];
         caso 3:
             suma3 += prod[i][j][anio - 1];
si ((suma1 > suma2) y (suma1 > suma3))
  imprimir "Producto con mayor número de ventas en el
           último año: Paracetamol";
  imprimir "Ventas: ", suma1;
sino si (suma2 > suma3)
  imprimir "Producto con mayor número de ventas en el
           último año: Diclofenaco");
  imprimir "Ventas: ", suma2);
sino
  imprimir "producto con mayor número de ventas en el
           último año: Ibuprofeno";
  imprimir "Ventas: ", suma3;
```

Para el inciso c)

```
ventas = -1;
para (k = 0; k < anio; k++)
  para (i = 0; i < mes; i++)
    para (j = 0; j < art; j++)
```

```
si (prod[i][j][k] > ventas)
    ventas = prod[i][j][k];
    art1 = j;
    mes1 = i;
    anio1 = k; mesMayor = 0; mesMenor = 0;
```

Salida: Imprimir los resultados:

a) Imprimir Ventas totales de la empresa en el segundo año (`suma`).
b) Imprimir El producto (paracetamol, diclofenaco o ibuprofeno) que tuvo las mayores ventas en el último año (3), incluyendo también el importe de las ventas (`suma1, suma2 o suma 3`)
c) Imprimir el producto (paracetamol, diclofenaco e ibuprofeno), mes (`mes1`) y año (`anio1`) con la mayor venta, incluyendo el importe de las ventas (`ventas`).

Debido al tamaño del programa se presentará por partes. En la Figura 4.49 se muestra parte del código del Ejercicio 7, la declaración y creación del arreglo (`double prod[][][] = new double[12][3][3]`), las constantes (`mes = 12; art = 3` y `anio = 3`), las variables que se utilizan (`anio1, art1, mes1, suma, suma1, suma2, suma3, ventas`) y la entrada de datos al arreglo. Se capturan los datos en la línea 24 mediante ciclos `for` anidados, recuerde que los arreglos tridimensionales tienen 3 índices `prod[i][j][k]`, el primero representa las filas el segundo representa las columnas y el tercero la profundidad o plano. A sí mismo `k` representa los años (3), `i` representa los meses (12) y `j` representa los productos (3).

```java
package capitulo4;

import java.util.Scanner;

public class Ejercicio7 {

    public static void main(String[] args) {
        Scanner s = new Scanner(System.in);

        final int mes = 12;
        final int art = 3;
        final int anio = 3;

        double prod[][][] = new double[mes][art][anio];
        int anio1, art1, mes1;
        double suma, suma1, suma2, suma3, ventas;

        //Lectura de datos
        for (int k = 0; k < anio; k++) {
            for (int i = 0; i < mes; i++) {
                for (int j = 0; j < art; j++) {
                    System.out.printf("Año: %d\tMes: %d\tDepartamento: %d: ", k + 1, i + 1, j + 1);
                    prod[i][j][k] = s.nextFloat();
                }
            }
        }
```

Figura 4.49. . Código del Ejercicio 7 (parte 1 de 4).

Explicación:

Una vez capturados los datos en el arreglo se imprime en formato de tabla para una mejor comprensión, puede ver el código en la Figura 4.50 (líneas 30 a la 39).

Líneas 42 a la 49. Se resuelve el inciso a) Las ventas totales de la empresa en el segundo año. Como la variable k lleva el control de los años y el planteamiento del problema solicita las ventas totales del segundo año, no es necesario hacer 3 ciclos `for`, con 2 es suficiente, por eso se inicializa la variable `anio1 = 2` (líneas 43)

Línea 46. Se acumulas todas las ventas almacenadas en el arreglo (`suma += prod[i][j][anio1 - 1]`), para después imprimir la variable (`suma`).

```
27  //Impresión de la matriz
28  System.out.println();
29  System.out.printf("%s%6s%15s%16s%16s\n","Año", "Mes", "Paracetamol(1)", "Diclofenaco(2)", "Ibuprofeno(3)");
30  for (int k = 0; k < anio; k++) {
31      System.out.printf("%-2d\n", (k + 1));
32      for (int i = 0; i < mes; i++) {
33          System.out.printf("%8d", (i + 1));
34          for (int j = 0; j < art; j++) {
35              System.out.printf("%10.2f\t", prod[i][j][k]);
36          }
37          System.out.println();
38      }
39  }
40  System.out.println();
41  //Ventas totales de la empresa en el segundo año
42  suma = 0;
43  anio1 = 2;
44  for (int i = 0; i < mes; i++) {
45      for (int j = 0; j < art; j++) {
46          suma += prod[i][j][anio1 - 1];
47      }
48  }
49  System.out.printf("Ventas totales de la empresa en el segundo año: %.2f\n", suma);
```

Figura 4.50. Código del Ejercicio 7 (parte 2 de 4).

El código que resuelve el producto que tuvo las mayores ventas en el último año, incluyendo también el importe de las ventas se encuentra en la Figura 4.51.

Líneas 52 a la 54. Se inicializan las variables con cero (suma1, suma2, suma3), estas variables acumularán las ventas por producto, del último año.

Líneas 55 a la 68. Nuevamente se utilizan solo dos ciclos `for` anidados, debido a que ya conocemos el valor del último año (`anio = 3`).

Líneas 57 a la 67. Se utiliza la estructura de selección múltiple `switch`, la cual nos ayuda a determinar mediante el selector (`j + 1`) si el caso es 1 entonces realiza la sumatoria (`suma1 += prod[i][j][anio - 1]`) de las ventas del producto 1 (paracetamol), si el caso es 2 entonces realiza la sumatoria (`suma2 += prod[i][j][anio - 1]`) de las ventas del producto 2 (diclofenaco) por último si el caso es 3 entonces realiza la sumatoria (`suma3 += prod[i][j][anio - 1]`) de las ventas del producto 3 (ibuprofeno). Lo anterior se repetirá para `i < mes` y `j < art`.

Líneas 69 a la 78. Se evalúan las tres variables (`suma1, suma2, suma3`) para determinar cuál es la venta mayor, se imprime la venta mayor al igual que el producto (Paracetamol, Diclofenaco o Ibuprofeno).

```
51      //Producto que obtuvo las mayores ventas en el último año y el monto.
52      suma1 = 0;
53      suma2 = 0;
54      suma3 = 0;
55      for (int i = 0; i < mes; i++)
56          for (int j = 0; j < art; j++) {
57              switch (j + 1) {
58                  case 1:
59                      suma1 += prod[i][j][anio - 1];
60                      break;
61                  case 2:
62                      suma2 += prod[i][j][anio - 1];
63                      break;
64                  case 3:
65                      suma3 += prod[i][j][anio - 1];
66                      break;
67              }
68          }
69      if ((suma1 > suma2) && (suma1 > suma3)) {
70          System.out.println("Producto con mayor número de ventas en el último año: Paracetamol");
71          System.out.printf("Ventas: %.2f\n", suma1);
72      } else if (suma2 > suma3) {
73          System.out.println("Producto con mayor número de ventas en el último año: Diclofenaco");
74          System.out.printf("Ventas: %.2f\n", suma2);
75      } else {
76          System.out.println("producto con mayor número de ventas en el último año: Ibuprofeno");
77          System.out.printf("Ventas: %.2f\n", suma3);
78      }
```

Figura 4.51. Código del Ejercicio 7 (parte 3 de 4).

Por último, en la Figura 4.52 se encuentra el código para resolver el último inciso. Solicita el producto, mes y año con la mayor venta, incluyendo el importe de las ventas.

Líneas 82 a la 84. Se inicializan las variables (`anio1 = 0, art1 = 0, mes1 = 0` y `ventas = -1`)

Líneas 86 a la 97. Se tienen 3 ciclos `for` anidados, dentro del ciclo interno `for (int j = 0; j < art; j++)` existe una condición `prod[i][j][k] > ventas` (`ventas = -1`), si la condición es verdadera, entonces se realizan las operaciones que están en las líneas 90 a la 93 (esto es, a `ventas` se le asigna el primer elemento de la matriz

tridimensional (`prod[0][0][0]`), a la variable `art1` se le asigna el valor de `j` que es igual a cero, a la variable `mes1` se le asigna el valor de i que es igual a cero y a la variable `anio1` el valor de k que es igual a cero). Lo anterior es con j = 0, faltaría cuando `j = 1` y cuando `j = 2`, una vez que termina el ciclo de la variable de control `j`. Este arreglo tiene 108 elementos almacenados (3X3X12 = 108).

```
81  //Producto, mes y año con mayor venta
82  anio1 = 0;
83  art1 = 0;
84  mes1 = 0;
85  ventas = -1;
86  for (int k = 0; k < anio; k++) {
87      for (int i = 0; i < mes; i++) {
88          for (int j = 0; j < art; j++) {
89              if (prod[i][j][k] > ventas) {
90                  ventas = prod[i][j][k];
91                  art1 = j;
92                  mes1 = i;
93                  anio1 = k;
94              }
95          }
96      }
97  }
98  switch (art1) {
99      case 0:
100         articulo = "Paracetamol";
101         break;
102     case 1:
103         articulo = "Diclofenaco";
104         break;
105     default:
106         articulo = "Ibuprofeno";
107         break;
108 }
109 System.out.println("Producto con la mayor venta");
110 System.out.printf("Producto: %s\tMes: %d\tAño: %d\n", articulo, mes1 + 1, anio1 + 1);
111 System.out.printf("Ventas: %.2f\n", ventas);
```

Figura 4.52. Código del Ejercicio 7 (parte 4 de 4).

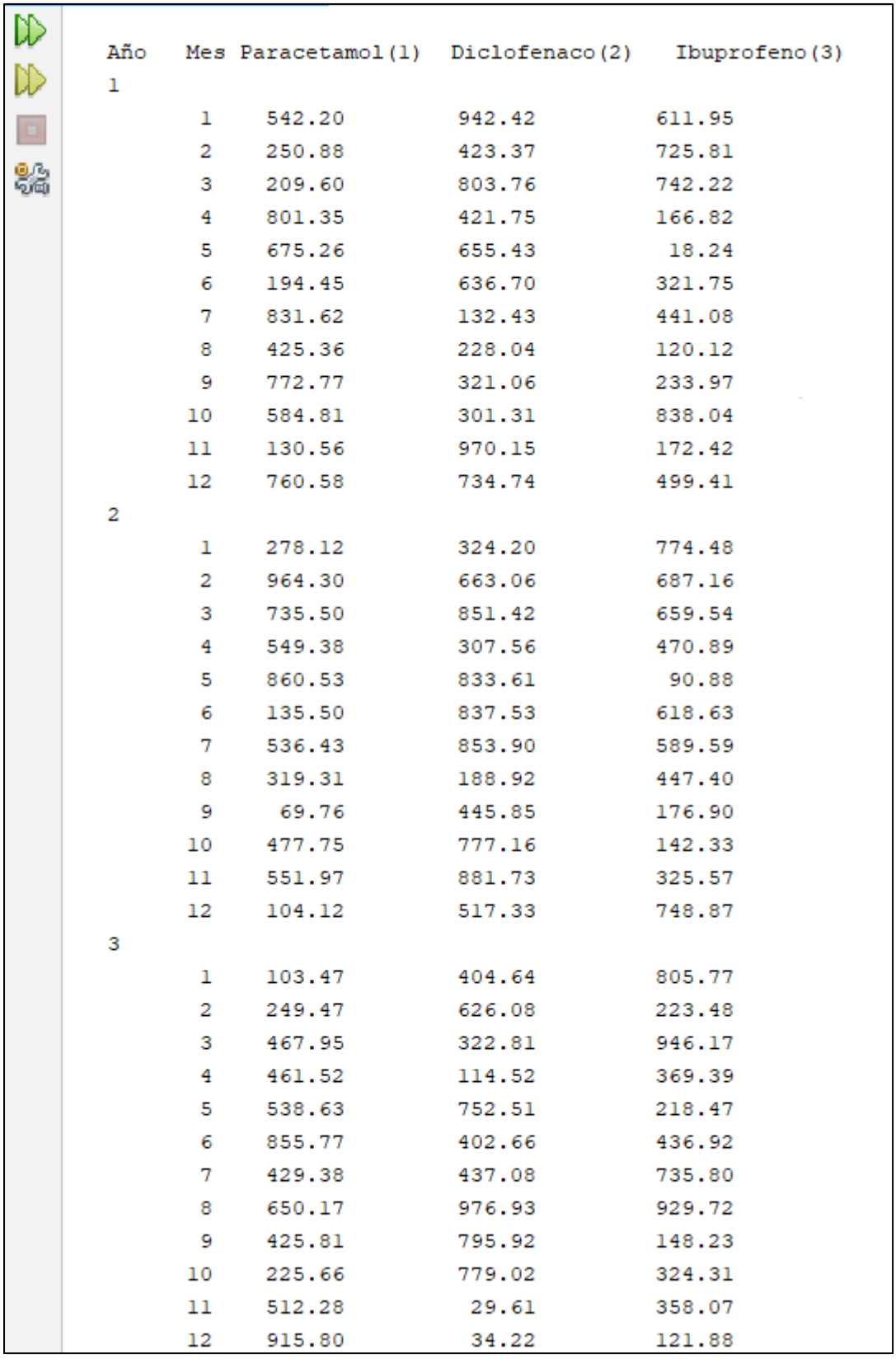
En la Tabla 4.2 se presenta una prueba de escritorio con algunos valores.

Tabla 4.2. Prueba de escritorio del Ejercicio 7.

k	i	j	prod[i][j][k] > ventas	Líneas 82 a la 84
0	0	0	true	Realizan asignaciones
		1	true	Realizan asignaciones
		2	false	
	1	0	true	Realizan asignaciones
		1	false	
		2	false	
	2	0	false	
		1	false	
		2	true	Realizan asignaciones
1	0	0	true	Realizan asignaciones
		1	false	
		2	false	
	1	0	false	
		1	false	
		2	false	
	2	0	true	Realizan asignaciones
		1	false	
		2	false	
2	0	0	true	Realizan asignaciones
		1	false	
		2	false	
	1	0	false	
		1	false	
		2	false	
	2	0	false	

Después de los ciclos anidados se imprime el resultado cómo se muestra en las líneas 110 y 111.

En las Figuras 4.53 y 4.54 puede ver la ejecución del programa.



Año	Mes	Paracetamol(1)	Diclofenaco(2)	Ibuprofeno(3)
1				
	1	542.20	942.42	611.95
	2	250.88	423.37	725.81
	3	209.60	803.76	742.22
	4	801.35	421.75	166.82
	5	675.26	655.43	18.24
	6	194.45	636.70	321.75
	7	831.62	132.43	441.08
	8	425.36	228.04	120.12
	9	772.77	321.06	233.97
	10	584.81	301.31	838.04
	11	130.56	970.15	172.42
	12	760.58	734.74	499.41
2				
	1	278.12	324.20	774.48
	2	964.30	663.06	687.16
	3	735.50	851.42	659.54
	4	549.38	307.56	470.89
	5	860.53	833.61	90.88
	6	135.50	837.53	618.63
	7	536.43	853.90	589.59
	8	319.31	188.92	447.40
	9	69.76	445.85	176.90
	10	477.75	777.16	142.33
	11	551.97	881.73	325.57
	12	104.12	517.33	748.87
3				
	1	103.47	404.64	805.77
	2	249.47	626.08	223.48
	3	467.95	322.81	946.17
	4	461.52	114.52	369.39
	5	538.63	752.51	218.47
	6	855.77	402.66	436.92
	7	429.38	437.08	735.80
	8	650.17	976.93	929.72
	9	425.81	795.92	148.23
	10	225.66	779.02	324.31
	11	512.28	29.61	358.07
	12	915.80	34.22	121.88

Figura 4.53. Ejecución del Ejercicio 7 con arreglo tridimensional (parte 1 de 2).

```
Ventas totales de la empresa en el segundo año: 18797.16

Producto con mayor número de ventas en el último año: Paracetamol
Ventas: 5835.91

Producto con la mayor venta
Producto: Diclofenaco    Mes: 8  Año: 3
Ventas: 976.93
BUILD SUCCESSFUL (total time: 0 seconds)
```

Figura 4.54. Ejecución del Ejercicio 7 con arreglo tridimensional (parte 2 de 2).

4.6 Programas propuestos.

1. Calcular el promedio de las calificaciones de las materias del semestre anterior, utilizando un arreglo unidimensional para almacenar todas las calificaciones y el promedio guárdelo en la siguiente posición después de la última calificación.
2. Hacer un programa que almacene en un arreglo unidimensional los primeros 30 números de la serie fibonacci e imprima el arreglo correspondiente.
3. Almacene en un arreglo unidimensional a, un conjunto de n elementos de tipo entero (máximo 15), almacene en el arreglo b unidimensional los elementos del arreglo a de forma invertida.
4. Guardar en un vector 10 números y cambie algún número por otro del mismo arreglo, realice el cambio y muestre el arreglo modificado.
5. Almacene en un vector de tamaño 10, números reales. Calcular el promedio e indicar cuántos elementos del vector son mayores que el promedio, y genere otro arreglo con los menores o iguales al promedio.
6. Hacer un programa que al recibir como datos dos arreglos unidimensionales de tipo entero, desordenados, de `n` y `m` elementos respectivamente, genere un nuevo arreglo unidimensional ordenado en forma descendente de `n + m` elementos de tipo entero, mezclando los 2 primeros arreglos. Imprimir los 3 arreglos.
7. Hacer un programa que al recibir como datos dos arreglos unidimensionales de tipo entero, el primero ordenado en forma ascendente y el segundo en forma descendente, de `n` y `m` elementos respectivamente, genere un nuevo arreglo unidimensional ordenado en forma ascendente de `n + m` elementos de tipo entero, mezclando los dos primeros arreglos. Imprimir los 3 arreglos.
8. Almacenar en un vector `n` números enteros y determiné ¿Cuántas veces se repite cada uno de ellos? Por ejemplo, si n=6 y los elementos del vector son `4, 7, 7, 7, 4, 6` se imprimirá:
   4 = 2
   6 = 1
   7 = 3

9. Almacenar en un vector la temperatura de cada día de una determinada semana y realizar lo siguiente:
    a) Temperatura promedio
    b) Encontrar la menor temperatura y el número de día en que ocurrió.
    c) Encontrar la mayor temperatura y el número de día en que ocurrió.
10. Guardar en vector, n elementos máximo 30 imprimir la suma de:
    a) los números pares.
    b) Los números impares.
    c) Los elementos del arreglo.
11. Crear un vector de tamaño 20 con números aleatorios entre 0 y 150 y genere otras listas con los siguientes criterios:
    a) Si los números están comprendidos entre 0 y 50 se guardarán en la lista1.
    b) Si los números están comprendidos entre 51 y 100 se guardarán en la lista2.
    c) Sí los números son mayores a 101 se guardarán en la lista 3.

    Al final imprimir las cuatro listas.

12. Hacer un programa en Java que inserte y elimine elementos en un arreglo unidimensional de tipo entero qué se encuentra desordenado. Considere que no se pueden insertar elementos repetidos.
13. En un vector de tipo real se tienen almacenadas las toneladas mensuales de trigo cosechadas durante el año anterior. Hacer un programa que calcule e imprima lo siguiente:
    a) Promedio anual de toneladas cosechadas.
    b) ¿Cuántos meses tuvieron una cosecha superior al promedio anual?
    c) ¿En qué mes se produjo el mayor número de toneladas y cuántas fueron?
14. Hacer un programa que lea las dimensiones de una matriz, la visualice y encuentre su elemento mayor y menor, así como sus posiciones respectivas.
15. En la clase de programación asiste un grupo de n alumnos y el maestro aplica tres exámenes durante el semestre. Hacer un programa que resuelva lo siguiente:
    a) promedio de calificaciones de cada alumno.
    b) Promedio del grupo en cada examen.
    c) Examen que obtuvo el mayor promedio de calificación.
16. Hacer un programa que intercambie las n columnas de un arreglo bidimensional. Los elementos de la primera columna se intercambian con los elementos de la última, los de la segunda con los de la penúltima, y así sucesivamente.
17. Hacer un programa que al recibir como dato una matriz de n x m, calcular su traspuesta. La traspuesta de una matriz es el resultado de cambiar filas por columnas y columnas por filas.
18. Hacer un programa que coloque un 1 en las diagonales principales de una matriz cuadrada. El resto se debe completar con ceros.
19. Hacer un programa que, al recibir los montos de ventas mensuales de 5 departamentos de una fábrica, proporcione la siguiente información:
    a) las ventas mensuales de la fábrica, incluido el monto anual.
    b) El departamento que tuvo mayor venta en el mes de Julio, incluyendo el monto de la venta.

c) El mes en el que se obtuvieron las mayores y menores ventas del departamento 3.

20. Hacer un programa que el recibir como dato un arreglo bidimensional de tipo entero, recorra este arreglo en forma de espiral. Observe la Figura 4.55.

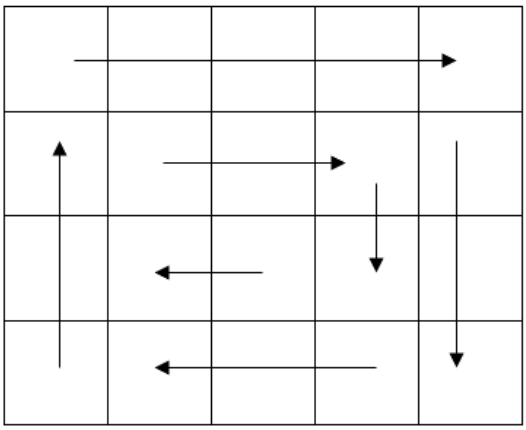


Figura 4.55. Recorrido en forma de espiral.

21. Dada una matriz cuadrada de `m x m`. Hacer un programa que encuentre la suma de todos los elementos que no pertenecen a la diagonal principal.
22. Una empresa tiene `n` sucursales, y a almacenado a lo largo de `m` años las ventas de su negocio en un arreglo bidimensional. Hacer un programa que proporcione la siguiente información:
    a) Sucursal que más ha vendido en los m años.
    b) Promedio de ventas por año.
    c) Año con mayor promedio de ventas.

5. Bibliografía

Cairó Battistutti, O. (2005). *Metodología de la programación: Algoritmos, diagramas de flujo y programas.* México: Alfaomega.

Cairó Battistutti, O. (2006). *Fundamentos de programación: Piensa en C.* México: Pearson.

Cairó Battistutti, O. (2022). *Aprende a programar en Java: De cero al infinito.* México: Alfaomega.

Corona Nakamura, M. A., & Ancona Valdez, M. d. (2011). *Diseño de algoritmos y su codificación en lenguaje C.* México: McGraw-Hill.

Deitel, P., & Deitel, H. (2016). *Cómo programar en Java* (Decima ed.). México: Pearson.

Gómez Jiménez, E., & Moreno Nuñez, J. (2019). *Fundamentos de programación JAVA con NetBeans 8.2.* México: Alfaomega.

Jiménez Marín, A., & Pérez Montes, F. M. (2016). *Aprende a programar con Java: Un enfoque práctico partiendo de cero.* Madrid, España: Paraninfo.

Joyanes Aguilar, L. (2020). *Fundamentos de programación: Algoritmos, estructura de datos y objetos* (Quinta ed.). México: McGraw-Hill.

Joyanes Aguilar, L., & Zahonero Martínez, I. (2011). *Programación en Java 6.* México: Mc Graw Hill.

Joyanes Aguilar, L., & Zahonero Martínez, I. (2014). *PROGRAMACIÓN en C, C++, Java y UML.* México: McGraw-Hill.

Oracle. (2024). *La forma mas inteligente y rápida de programar*. Obtenido de https://www.oracle.com/mx/tools/technologies/netbeans-ide.html

Sánchez Sánchez, M. (10 de agosto de 2015). Programación Orientada a Objetos. Metepec, México, México.

Sznajdleder, P. A. (2016). *Java a fondo. Curso de programación.* Buenos Aires, Argentina: Alfaomega.

POLÍTICAS ACADÉMICAS GENERALES DEL AÑO SABÁTICO DEL TECNOLÓGICO NACIONAL DE MÉXICO

CARTA DE RECONOCIMIENTO DEL AUTOR DE LOS DERECHOS A FAVOR DEL TECNM

Ciudad de México, 20/09/2024

TECNOLÓGICO NACIONAL DE MÉXICO.

PRESENTE

Bajo protesta de decir verdad, Maria del Socorro Moreno Figueroa, personal docente adscrito al Instituto Tecnológico de Toluca del Tecnológico Nacional de México, manifiesto que en cumplimiento de mis actividades relacionadas con el Año Sabático elaboré la obra titulada "Programando en java".

Con base en lo anterior, y con fundamento en los artículos 83 de la Ley Federal del Derecho de Autor y 46 de su Reglamento, reconozco que el Tecnológico Nacional de México es titular de los derechos patrimoniales sobre la misma y le corresponden las facultades relativas a la divulgación, integridad de la obra y de colección, conservando el derecho a figurar como autor.

Asimismo, respondo por la autoría y originalidad de la citada obra; y relevo de toda responsabilidad al Tecnológico Nacional de México de cualquier demanda o reclamación que llegara a formular alguna persona física o moral que considere que con esta obra es afectado en alguno de los derechos protegidos por la Ley en cita, asumiendo todas las consecuencias legales y económicas.

ATENTAMENTE

AUTOR