



TECNOLÓGICO
NACIONAL DE MÉXICO



Tecnológico Nacional de México Instituto Tecnológico de Tijuana

Departamento de Sistemas y Computación

Elaboración de Apuntes : MANUAL DE PRÁCTICAS "Programación Orientada a Objetos"

Reporte Final

Periodo Sabático : 13 agosto 2018 al 12 agosto 2019

Dictamen No. AS-2-259/2018

MC. Alejandra Mancilla Soto

Tijuana B.C. Agosto del 2019

ÍNDICE

Objetivo	3
Justificación	3
Contenido de Prácticas para Programación Orientada a Objetos en Ing. en Sistemas Computacionales	5
Definir una clase Persona	7
Definir una clase Elevador e instanciarla	12
Resolviendo conflicto de nombres	18
Utilizando métodos que no regresan valor y métodos que regresan valor	23
Utilizando parámetros de paso de por valor	28
Heredar de la clase vehículo	34
Heredar de la clase Figura Geométrica	42
Identificar los constructores de la clase derivada	52
Definir una clase con métodos virtuales	60
Definir una clase con métodos override	68
Definir una clase abstracta	82
Utilizar una Excepción específica	88
Utilizar una Excepción específica	92
Relanzando tu propia excepción	96
Creando un archivo de texto	100
Contenido de Prácticas para Programación Orientada a Objetos en Ing. en Tecnologías de la Información y Comunicaciones	108
Definir una clase Persona	110
Define una clase con todos sus miembros	116
Define una clase definiendo sus especificadores de acceso	122
Crea un objeto inicializándolo con el constructor por defecto	128
Crea un objeto utilizando el constructor sobrecargado	133
Elabora un programa utilizando manejo de excepciones	139
Elabora un programa donde relances tu propia excepción	144
Define una clase con sus métodos	150
Define una clase donde defina un método que regrese valor	154
Crear un programa que sobrecargue un método	160
Elabora un ejercicio donde muestre la herencia entre las clases Persona y Estudiante	166
Elabora un programa donde implementes de manera implícita una interface	175
Elabora un programa donde implementes de manera explícita una interface	181
Elabora programa que capture 10 estaturas y las convierta a cm	187
Elabora un programa que capture 5 objetos tipo Estudiante y los almacene en un arreglo	191

PRESENTACIÓN

OBJETIVO

Diseñar un manual de prácticas de programación en C# que sea útil y aplicable en la materia de Programación Orientada a Objetos en las carreras de Ingeniería en Sistemas Computacionales y en Ingeniería Informática e Ingeniería en Tecnologías de la Información y Comunicaciones.

JUSTIFICACIÓN

La elaboración de este manual de prácticas tiene la finalidad de contribuir al crecimiento y fortalecimiento de una base de datos, donde podamos tener y utilizar un extenso material auxiliar para apoyar las actividades de aprendizaje de nuestros estudiantes y que ellos puedan realizar dichas actividades tanto en el salón de clases, como prácticas de laboratorio o como parte de su tarea en casa.

Este manual cubre completamente el programa de la materia de Programación Orientada a Objetos de las carreras de Ingeniería en Sistemas Computacionales y la carrera de Ingeniería en Tecnologías de la Información y Comunicaciones, la cual requiere de prácticas de los diferentes temas que se manejan en esta materia, cubriendo así las competencias que el alumno debe desarrollar en el transcurso del semestre.

El tener la oportunidad de desarrollar este trabajo me permite colaborar en la formación de los estudiantes de estas carreras ya que al contribuir en el desarrollo de material podremos favorecer las prácticas de aprendizaje que tanto docentes como alumnos puedan utilizar en clase. Considero que tengo la experiencia suficiente para elaborar este material ya he impartido esta materia durante varios años. También permitirá no solo aumentar la eficiencia en relación a los proyectos de desarrollo de largo plazo, sino planificar con mejores resultados las estrategias de codificación en los proyectos que ya se encuentran trabajando, y apoyar en los planes docentes de enseñanza de la programación y computación.

CONTENIDO

A continuación se definen cada una de las prácticas para la materia de Programación Orientada a Objetos, en el primer bloque se describen las prácticas para Ing. en Sistemas Computacionales y en el segundo bloque las de la carrera de TIC's.

CONTENIDO DE PRÁCTICAS PARA PROGRAMACIÓN ORIENTADA A OBJETOS EN ING. EN SISTEMAS COMPUTACIONALES

Nombre de la asignatura: **Programación Orientada a Objetos**
 Carrera: **Ingeniería en Sistemas Computacionales**
 Clave de la asignatura: **SCD-1020**
 (Créditos) SATCA1 **2 - 3 - 5**

Objetivo General:

Diseñar e implementar objetos de programación que permitan resolver situaciones reales y de ingeniería.

Prácticas

Núm.	Nombre de la Práctica	Objetivo Específico	Temas con los que se relacionan
1	Definir una clase Persona	Identificar los miembros que conforman una clase	Unidad 2. Clases y Objetos 2.1 Declaración de clases
2	Definir una clase Elevador e instanciarla	Definir una clase y crear un objeto de la misma y ejecutar los métodos	Unidad 2. Clases y Objetos 2.1 Declaración de clases 2.2 Instanciación de una clase
3	Resolviendo conflicto de nombres	Resolver el conflicto de nombres utilizando this	Unidad 2. Clases y Objetos 2.3 Referencia al objeto actual.
4	Utilizando métodos que no regresan valor y métodos que regresan valor	Definir e implementar la estructura de los métodos	Unidad 2. Clases y Objetos 2.4 Métodos: declaración, mensajes, paso de parámetros, retorno de valores.
5	Utilizando parámetros de paso de por valor	Utilizar el paso de parámetros por valor	Unidad 2. Clases y Objetos 2.4 Métodos: declaración, mensajes, paso de parámetros, retorno de valores.
6	Heredar de la clase vehículo	Implementar la herencia de clases	Unidad 3. Herencia 3.1 Definición: clase base, clase derivada.
7	Heredar de la clase Figura Geométrica	Implementar la herencia de clases y observar la jerarquía de clases	Unidad 3. Herencia 3.4 Referencia al objeto de la clase base.

8	Identificar los constructores de la clase derivada	Implementar la herencia de clases	Unidad 3. Herencia 3.5 Constructores y destructores en clases derivadas.
9	Definir una clase con métodos virtuales	Identificar métodos virtuales	Unidad 3. Herencia 3.6 Redefinición de métodos en clases derivadas.
10	Definir una clase con métodos override	Identificar la sustitución de métodos virtuales con métodos override	Unidad 3. Herencia 3.6 Redefinición de métodos en clases derivadas.
11	Definir una clase abstracta	Identificar las clases abstractas y sus métodos abstractos	Unidad 4. Polimorfismo 4.2 Clases abstractas: definición, métodos abstractos, implementación de clases abstractas, modelado de clases abstractas.
12	Utilizar una Excepción específica	Definir y utilizar los bloques para manejo de excepciones	Unidad 5. Excepciones 5.1 Definición. 5.2 Tipos de Excepciones.
13	Utilizar una Excepción específica	Definir y utilizar los bloques para manejo de excepciones utilizando una excepción específica	Unidad 5. Excepciones 5.1 Definición. 5.2 Tipos de Excepciones.
14	Relanzando tu propia excepción	Definir y utilizar los bloques para manejo de excepciones y crear tu propia excepción	Unidad 5. Excepciones 5.5 Creación y manejo de excepciones definidas por el usuario.
15	Creando un archivo de texto	Identificar los tipos de archivos, ejecutar una operación básica	Unidad 6. Flujos y Archivos 6.2 Clasificación: Archivos de texto y binarios. 6.3 Operaciones básicas y tipos de acceso.

Prácticas

No. de Práctica	UNO
Nombre	Definir una clase Persona
Competencia específica de la unidad 2	▪ Implementar clases y objetos cumpliendo las reglas de la programación orientada a objetos. ▪ Implementar constructores para inicializar atributos. ▪ Sobrecargar métodos para optimizar el código de una clase.
Introducción	La Programación Orientada a Objetos es un modelo de programación que utiliza clases y objetos para la resolución de problemas. En esta práctica identificaremos las entidades de interés las cuales pueden ser objetos potenciales que poseen un conjunto de atributos y métodos, útiles para resolver el problema.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente	Unidad 2. Clases y Objetos 2.1 Declaración de clases
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una persona utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase persona representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

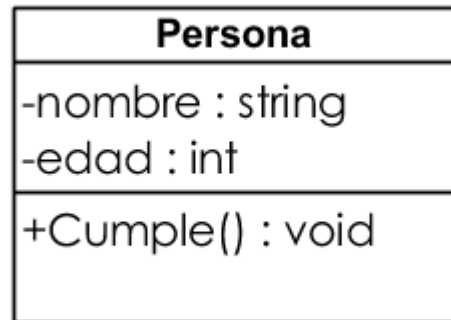
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Determina los datos necesarios de una persona que cumple años, identifica los miembros de la clase persona.

a) Datos :

- nombre
- edad
- cumpleaños

b) Diagrama de clases



c) Código C# en consola

CLASE PERSONA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Persona
{
    class Persona
    {
        // atributos
        string nombre;
        int edad;

        // propiedades
        public string Nombre
        {
```

```

        get { return nombre; }
        set { nombre = value; }
    }

    public int Edad
    {
        get { return edad; }
        set { edad = value; }
    }

    // Constructores
    public Persona() { }
    public Persona(string n, int e)
    {
        Nombre = n;
        Edad = e;
    }

    // métodos
    public void Cumple()
    {
        edad++;
    }
}

```

CLASE PRINCIPAL

```

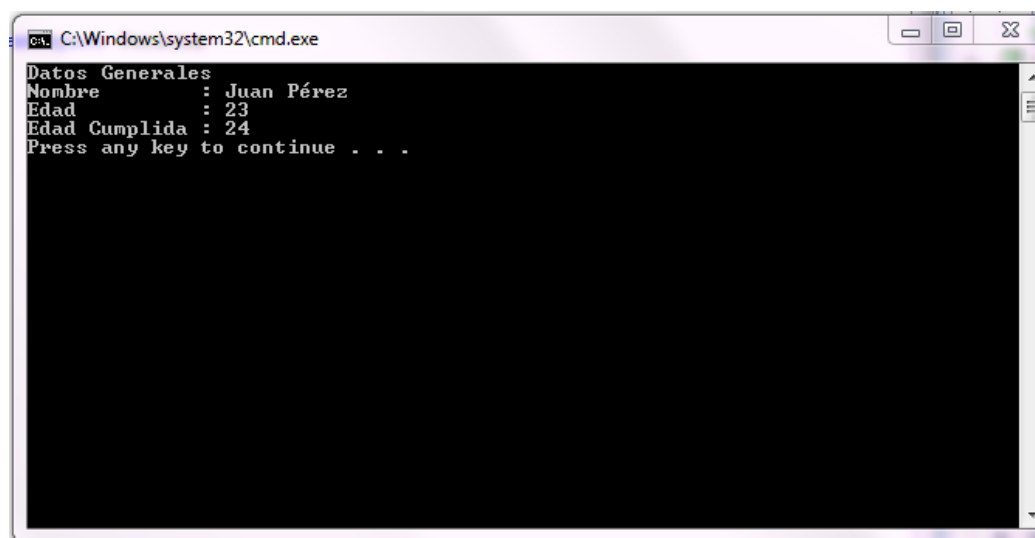
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Persona
{
    class Program
    {
        static void Main(string[] args)
        {
            // Crear el objeto
            Persona Juan = new Persona("Juan Pérez", 23);

            // mostrar los datos y ejecutar el método
            Console.WriteLine("Datos Generales");
            Console.WriteLine("Nombre           : {0} ",
Juan.Nombre);
            Console.WriteLine("Edad           : {0} ", Juan.Edad);
            Juan.Cumple();
            Console.WriteLine("Edad Cumplida : {0} ", Juan.Edad);
        }
    }
}

```

d) Corrida o Pantalla de captura



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt displays the following text: 'Datos Generales', 'Nombre : Juan Pérez', 'Edad : 23', 'Edad Cumplida : 24', and 'Press any key to continue . . .'. The rest of the window is black.

Sugerencias didácticas

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.
Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML.
Nombrar otros ejemplos de clases para identificar sus miembros.
Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno

El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.

Bibliografía

- Bell, D. and Parr, M. (2011). *C# para estudiantes*. Pearson Educación de México, S.A. de C.V.
- Ceballos Sierra, F. (2012). *Microsoft C#, Curso de Programación*. México, D.F.: Alfaomega Ra-Ma.
- Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. *La Biblia de C#*. Anaya.
- Gonzalez Seco, J. (2012). *El lenguaje de programación C#*.
- Larman, C. (2003). *UML y patrones*. Naucalpan de Juárez: Pearson
-

Educación.

Msdn.microsoft.com. (2018). *Objetos, clases y estructuras (C#)*. [online]
Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. *CSharp, Courseware Development Tools*.

No. de Práctica	DOS
Nombre	Definir una clase Elevador e instanciarla
Competencia específica de la unidad 2	Implementar clases y objetos cumpliendo las reglas de la programación orientada a objetos. Implementar constructores para inicializar atributos. Sobrecargar métodos para optimizar el código de una clase.
Introducción	En esta práctica identificaremos la entidad que define los datos necesarios para resolver el problema. Es importante identificar los miembros de la clase e identificarlos en el diagrama de clases de UML. Después se instanciará un objeto del tipo de la clase. Una clase equivale a la generalización de un tipo específico de objetos, pero cada objeto que construyamos de esa clase tendrá sus propios datos. Un objeto de determinada clase se crea en el momento en que se define una variable de dicha clase.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.1 Declaración de clases 2.2 Instanciación de una clase
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una clase Elevador utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase Elevador representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica.

5. Ejecuta el código y revisa los resultados.

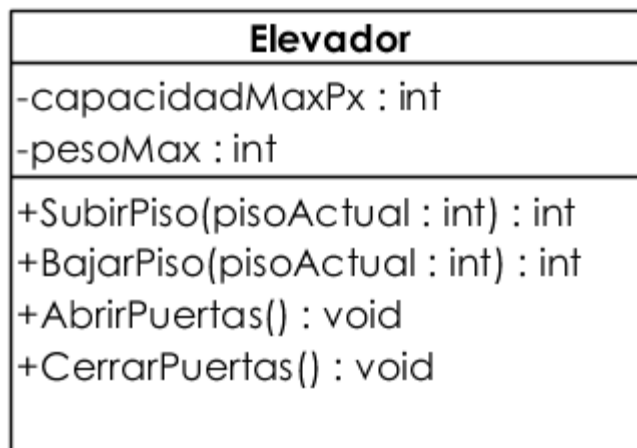
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Identifica y define los miembros de una clase Elevador, de acuerdo al funcionamiento natural de un elevador, deberá subir y bajar pisos de acuerdo en el piso que se encuentre, deberá mostrar la capacidad máxima de personas y el peso máximo en kg., además indicar cuándo se abren o cierran las puertas del elevador.

a) Datos :

- piso actual
- piso nuevo
- subir Piso
- Bajar Piso
- capacidad máxima de personas
- peso máximo
- abrir puertas
- cerrar puertas

b) Diagrama de clases



c) Código C# en consola

CLASE ELEVADOR

```
using System;  
using System.Collections.Generic;
```

```

using System.Linq;
using System.Text;

namespace Elevador
{
    class Elevador
    {
        //atributos
        int pesoMax, capacidadMaxPx;

        //propiedades
        public int PesoMax {
            get { return pesoMax; }
            set { pesoMax = value; } }

        public int CapacidadMaxPx {
            get { return capacidadMaxPx; }
            set { capacidadMaxPx = value; } }

        //constructores
        public Elevador() { }
        public Elevador(int p, int c)
        {
            pesoMax = p;
            capacidadMaxPx = c; }

        //metodos
        public int SubirPiso(int pisoActual)
        { return ++pisoActual; }

        public int BajarPiso(int pisoActual)
        { return --pisoActual; }

        public void AbrirPuertas()
        { Console.WriteLine("\a Abrir puertas \a"); }

        public void CerrarPuertas()
        { Console.WriteLine("\a Cerrar puertas \a"); }
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Elevador
{
    class Program
    {
        static void Main(string[] args)
        {
            int x = 0, pisoA=0;
            Elevador E = new Elevador();

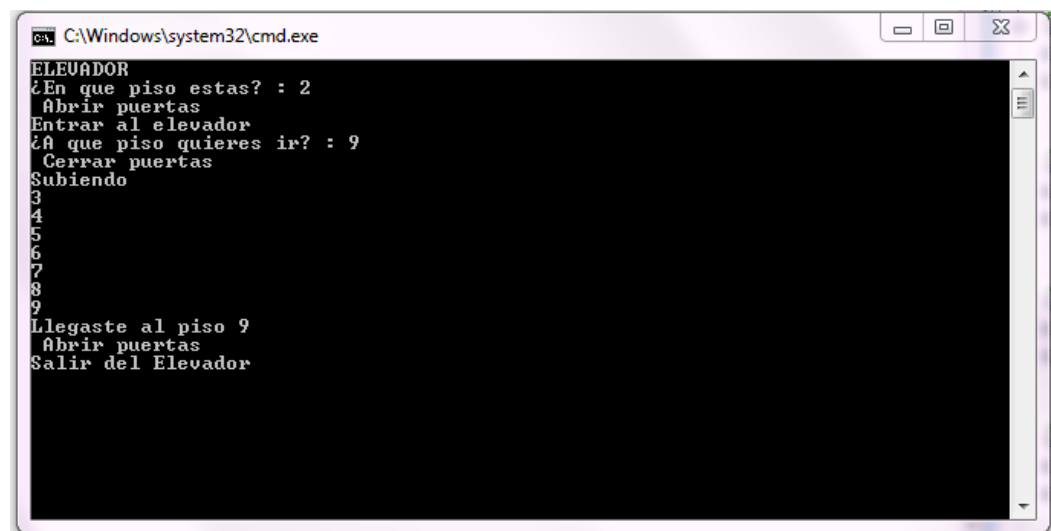
```

```

Console.WriteLine("ELEVADOR");
Console.Write("¿En que piso estas? : ");
pisoA = int.Parse(Console.ReadLine());
E.AbrirPuertas();
Console.WriteLine("Entrar al elevador");
Console.Write("¿A que piso quieres ir? : ");
int p = int.Parse(Console.ReadLine());
E.CerrarPuertas();
if (pisoA < p)
{
    Console.WriteLine("Subiendo");
    x = p - pisoA;
    for (int i = 0; i < x; i++)
    {
        pisoA = E.SubirPiso(pisoA);
        Console.WriteLine("\a" + pisoA);
    }
}
else
{
    Console.WriteLine("Bajando");
    for (int i = pisoA; i > p; i--)
    {
        pisoA=E.BajarPiso(pisoA);
        Console.WriteLine("\a" + pisoA);
    }
}
Console.WriteLine("Llegaste al piso {0}",pisoA);
E.AbrirPuertas();
Console.WriteLine("Salir del Elevador");
Console.ReadKey();
}
}
}

```

d) Corrida o Pantalla de captura



```

C:\Windows\system32\cmd.exe
ELEVADOR
¿En que piso estas? : 2
Abrir puertas
Entrar al elevador
¿A que piso quieres ir? : 9
Cerrar puertas
Subiendo
3
4
5
6
7
8
9
Llegaste al piso 9
Abrir puertas
Salir del Elevador

```

```
C:\Windows\system32\cmd.exe
ELEVADOR
¿En que piso estas? : 8
Abrir puertas
Entrar al elevador
¿A que piso quieres ir? : 1
Cerrar puertas
Bajando
7
6
5
4
3
2
1
Llegaste al piso 1
Abrir puertas
Salir del Elevador
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Instanciar un objeto del tipo de la clase. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
-------------------------------	---

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
---------------------------	--

Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i> . Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i> . México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i> . Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i> . Larman, C. (2003). <i>UML y patrones</i> . Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i> . [online]
---------------------	---

Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	TRES
Nombre	Resolviendo conflicto de nombres
Competencia específica de la unidad 2	Implementar clases y objetos cumpliendo las reglas de la programación orientada a objetos. Implementar constructores para inicializar atributos. Sobrecargar métodos para optimizar el código de una clase.
Introducción	Una clase equivale a la generalización de un tipo específico de objetos, pero cada objeto que construyamos de esa clase tendrá sus propios datos. Para inicializar un objeto se asignan valores a sus atributos pero si existe un conflicto de nombre en sus variables se deberá solucionar con la palabra reservada <i>this</i> que hace referencia al objeto actual.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.3 Referencia al objeto actual.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una clase persona utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase persona representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. Utiliza la referencia al objeto actual para resolver el conflicto de nombres, es decir, utiliza la palabra reservada <i>this</i>. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

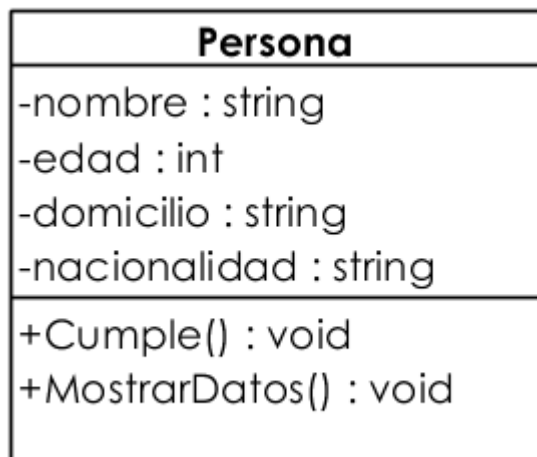
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Resuelve el conflicto de nombres a través del método constructor.

a) Datos :

- nombre
- edad
- domicilio
- nacionalidad
- cumple años
- mostrar datos

b) Diagrama de clases



c) Código C# en consola

CLASE PERSONA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Persona
{
```

```
class Persona
{
    // atributos
    string nombre, domicilio, nacionalidad;
    int edad;

    // propiedades
    public string Nombre
    {
        get { return nombre; }
        set { nombre = value; }
    }

    public string Domicilio
    {
        get { return domicilio; }
        set { domicilio = value; }
    }

    public string Nacionalidad
    {
        get { return nacionalidad; }
        set { nacionalidad = value; }
    }

    public int Edad
    {
        get { return edad; }
        set { edad = value; }
    }

    // Constructores
    public Persona() { }
    public Persona(string nombre, string domicilio, string
nacionalidad, int edad)
    {
        this.nombre = nombre;
        this.domicilio = domicilio;
        this.nacionalidad=nacionalidad;
        this.edad = edad;
    }

    // métodos
    public void Cumple()
    {
        edad++;
    }

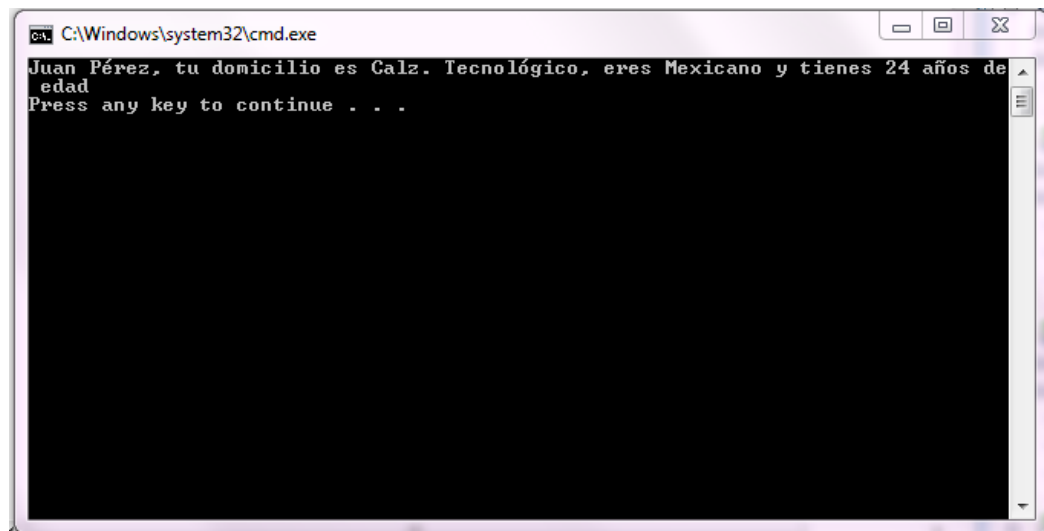
    public void MostrarDatos()
    {
        Console.WriteLine("{0}, tu domicilio es {1}, eres
{2} y tienes {3} años de edad",nombre,domicilio,
nacionalidad,edad);
    }
}
}
```

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Constructor
{
    class Program
    {
        static void Main(string[] args)
        {
            Persona Juan = new Persona("Juan Pérez", "Calz.
Tecnológico", "Mexicano", 23);
            Juan.Cumple();
            Juan.MostrarDatos();
        }
    }
}
```

d) Corrida o Pantalla de captura



Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML.
-------------------------------	---

Utiliza la referencia al objeto actual para resolver el conflicto de nombres, es decir, utiliza la palabra reservada *this*.
Nombrar otros ejemplos de clases para identificar sus miembros.
Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
---------------------------	--

Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i>. [online] Available at: https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx [Accessed 6 Nov. 2018]. Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.
---------------------	---

No. de Práctica	CUATRO
Nombre	Utilizando métodos que no regresan valor y métodos que regresan valor
Competencia específica de la unidad 2	Implementar clases y objetos cumpliendo las reglas de la programación orientada a objetos. Implementar constructores para inicializar atributos. Sobrecargar métodos para optimizar el código de una clase.
Introducción	La biblioteca de C# proporciona métodos predefinidos, pero también podemos definir nuestros propios métodos e invocarlos de la misma forma que los predefinidos. Podemos definir métodos que pasan parámetros y métodos que retornan o no valores. En esta práctica se utilizarán métodos que retornan valor y métodos que no retornan valor.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.4 Métodos: declaración, mensajes, paso de parámetros, retorno de valores.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para resolver el problema utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

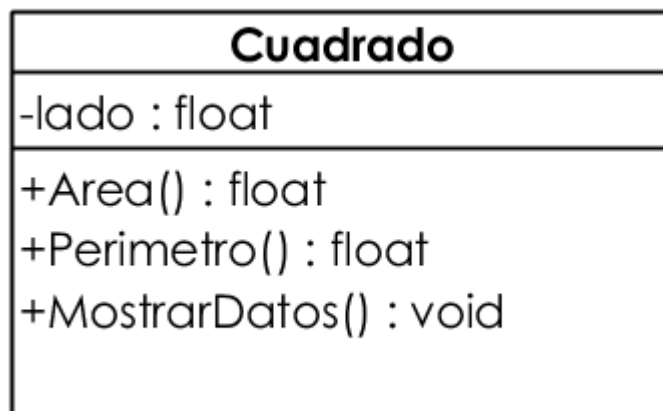
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Utilizando métodos que no regresan valor y métodos que regresan valor para resolver el problema de calcular el área y perímetro de la figura geométrica como el Cuadrado, agrega los atributos, propiedades, constructores y métodos. Crea una objeto de la figura con todos sus datos y ejecuta sus métodos.

a) Datos :

- lado
- área
- perímetro

b) Diagrama de clases



c) Código C# en consola

CLASE CUADRADO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Metodos
{
    class Cuadrado
    {
        float lado;

        public float Lado
        {
            get { return lado; }
            set { lado = value; }
        }
    }
}
```

```

        public Cuadrado() { }

        public Cuadrado(float la)
        {
            lado = la;
        }

        // métodos que regresan valor
        public float Area()
        {
            return lado * lado;
        }

        public float Perimetro()
        {
            return lado * 4;
        }

        // métodos que no regresan valor
        public void MostrarDatos()
        {
            Console.WriteLine("Area y Perimetro del Cuadrado");
            Console.WriteLine("Lado      = {0:N2} cm", lado);
            Console.WriteLine("Area      = {0:N2} cm2", Area());
            Console.WriteLine("Perimetro = {0:N2} cm",
Perimetro());
        }
    }
}

```

CLASE PRINCIPAL

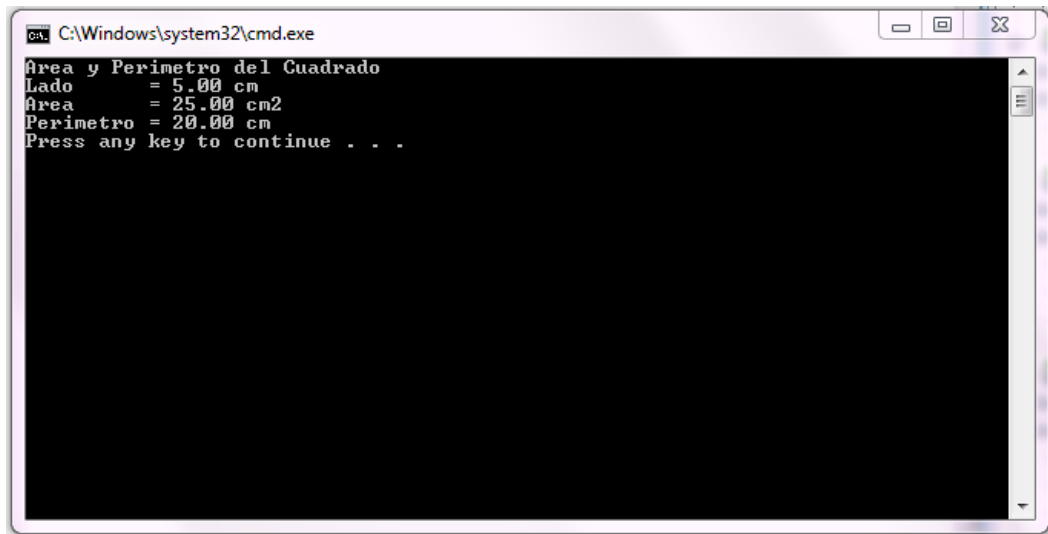
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Metodos
{
    class Program
    {
        static void Main(string[] args)
        {
            Cuadrado C = new Cuadrado(5);
            C.MostrarDatos();
        }
    }
}

```

d) Corrida o Pantalla de captura



Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Utilizar métodos que no regresan valor y métodos que regresan valor, utilizando el mecanismo de paso de parámetros por valor. Realiza otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i>. [online]

Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	CINCO
Nombre	Utilizando parámetros de paso de por valor
Competencia específica de la unidad 2	Implementar clases y objetos cumpliendo las reglas de la programación orientada a objetos. Implementar constructores para inicializar atributos. Sobrecargar métodos para optimizar el código de una clase.
Introducción	La biblioteca de C# proporciona métodos predefinidos, pero también podemos definir nuestros propios métodos e invocarlos de la misma forma que los predefinidos. Podemos definir métodos que pasan parámetros y métodos que retornan o no valores. En la definición de métodos también podemos utilizar el mecanismo de paso de parámetros, en este caso el paso de parámetros por valor. Cuando un método en C# invoca a otro método y le pasa un argumento de un tipo primitivo, pasa una copia de ese argumento. En esta práctica se utilizarán métodos que retornan valor y métodos que no retornan valor, además de paso de parámetros por valor.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.4 Métodos: declaración, mensajes, paso de parámetros, retorno de valores
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan un objeto tipo Empleado utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase persona representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de

clases.

4. Compila el programa y corrige errores de sintaxis y lógica.

5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Utilizando métodos que no regresan valor y métodos que regresan valor para resolver el problema de calcular el sueldo de un empleado, agrega los atributos, propiedades, constructores y métodos para calcular el sueldo semanal y mostrar todos sus datos. Utiliza paso de parámetros por valor. Crea una objeto de la clase con todos sus datos y ejecuta sus métodos.

a) Datos :

- nombre
- no empleado
- sueldo
- departamento
- días trabajados
- Calcular el sueldo semanal

b) Diagrama de clases

Empleado
-nombre : string -noEmp : int -sueldo : float -departamento : string
+CalcularSueldoSemanal(dias : int) : float +MostrarDatos(dias : int) : void

c) Código C# en consola

CLASE EMPLEADO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Empleado
{
    class Empleado
    {
        string nombre, departamento;
        int noEmp;
        float sueldo;

        // propiedades
        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
        public string Depto
        {
            get { return departamento; }
            set { departamento = value; }
        }

        public int NoEmp
        {
            get { return noEmp; }
            set { noEmp = value; }
        }

        public float Sueldo
        {
            get { return sueldo; }
            set { sueldo = value; }
        }

        public Empleado() {}
        public Empleado(string n, string d, int e, float s)
        {
            nombre = n;
            departamento = d;
            noEmp = e;
            sueldo = s;
        }

        public float CalcularSueldoSemanal(int dias)
```

```

        {
            return sueldo * dias;
        }

        public void MostrarDatos(int dias)
        {
            Console.WriteLine("Datos del Empleado");
            Console.WriteLine("No. Emp : {0} Empleado : {1} ",
noEmp, nombre);
            Console.WriteLine("Departamento      : {0} ", departamento);
            Console.WriteLine("Sueldo Diario      : {0:c2}", sueldo);
            Console.WriteLine("Días trabajados : {0}", dias);
            Console.WriteLine("Sueldo Semanal :
0:c2}", CalcularSueldoSemanal(dias));
        }
    }
}

```

CLASE PRINCIPAL

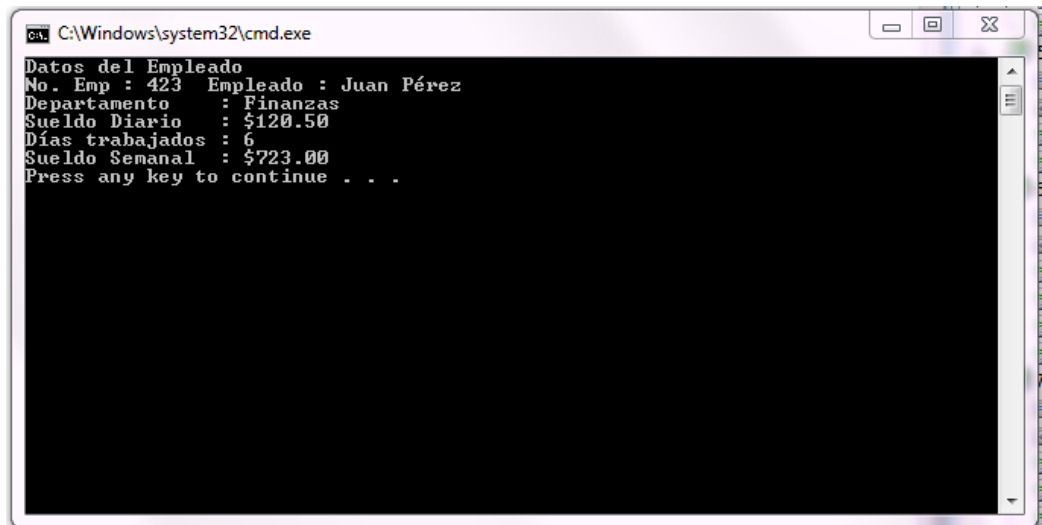
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Empleado
{
    class Program
    {
        static void Main(string[] args)
        {
            Empleado Juan = new Empleado("Juan Pérez", "Finanzas",
423, 120.5f);
            Juan.MostrarDatos(6);
        }
    }
}

```

d) Corrida o Pantalla de captura



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The command prompt displays the following text: 'Datos del Empleado', 'No. Emp : 423 Empleado : Juan Pérez', 'Departamento : Finanzas', 'Sueldo Diario : \$120.50', 'Días trabajados : 6', 'Sueldo Semanal : \$723.00', and 'Press any key to continue . . .'. The background is black, and the text is white.

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Utilizar métodos que no regresan valor y métodos que regresan valor, utilizando el mecanismo de paso de parámetros por valor. Realiza otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación.

Msdn.microsoft.com. (2018). *Objetos, clases y estructuras (C#)*. [online]
Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	SEIS
Nombre	Heredar de la clase vehículo
Competencia específica de la unidad 3	Implementar la herencia en clases derivadas para reutilizar los miembros de una clase base.
Introducción	El mecanismo de herencia de POO permite que una clase nueva herede todas las características de otra clase ya existente, garantizando la reutilización de código. Con la herencia todas las clases están clasificadas en una jerarquía, donde cada clase puede tener una o más subclases. El término heredar significa que las subclases disponen de todos los métodos y propiedades de su superclase. Este mecanismo proporciona una forma rápida y cómoda de extender la funcionalidad de una clase. En C# solo se puede heredar de una clase base, a esto se le llama herencia simple.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Herencia 3.1 Definición: clase base, clase derivada.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan las clases utilizando el mecanismo de abstracción y herencia. 2. Clasifica los datos y define la clase base y la clase derivada representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a las clases mostradas en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica.

5. Ejecuta el código y revisa los resultados.

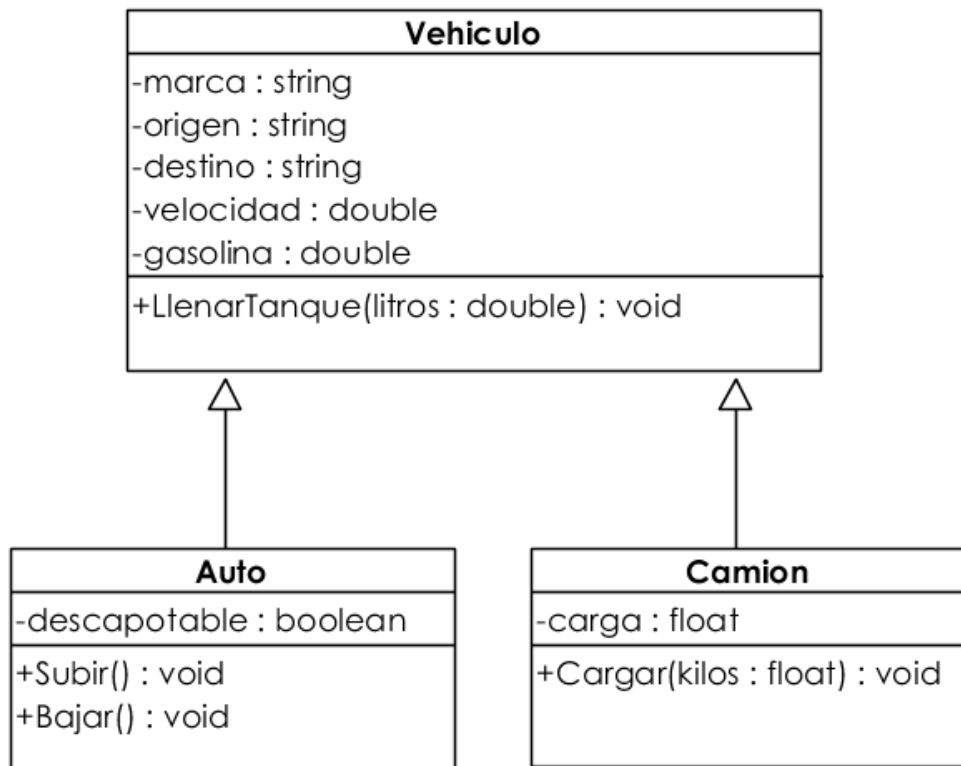
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Hereda de la Clase Vehículo a sus clases derivadas Auto y Camión, agrega los atributos, propiedades, constructores y métodos correspondientes a cada clase.

a) Datos :

- marca
- origen
- destino
- velocidad
- gasolina
- llenar tanque
- auto techo descapotable
- subir techo
- bajar techo
- carga en kg
- llevar carga

b) Diagrama de clases



c) Código C# en consola

CLASE BASE VEHICULO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Vehiculo
    {
        double velocidad, gasolina;
        string marca, origen, destino;

        public double Velocidad
        {
            get { return velocidad; }
            set { velocidad = value; }
        }
    }
}
```

```

        public double Gasolina
        {
            get { return gasolina; }
            set { gasolina = value; }
        }
        public string Marca
        {
            get { return marca; }
            set { marca = value; }
        }
        public string Origen
        {
            get { return origen; }
            set { origen = value; }
        }
        public string Destino
        {
            get { return destino; }
            set { destino = value; }
        }

        public Vehiculo() { }
        public Vehiculo(double vel, double gas, string mar, string
ori, string des)
        {
            velocidad = vel;
            gasolina = gas;
            marca = mar;
            origen = ori;
            destino = des;

        }
        public void Llenartanque(double litros)
        {
            Gasolina = Gasolina + litros;
        }
    }
}

```

CLASE DERIVADA AUTO

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Auto:Vehiculo
    {
        bool descapotable;

        public bool Descapotable
        {
            get { return descapotable; }

```

```

        set { descapotable = value; }
    }
    public Auto() { }
    public Auto(double vel, double gas, string mar, string ori,
string des, bool desca): base(vel, gas, mar, ori, des)
    {
        descapotable = desca;
    }
    public void Subir()
    {
        Console.WriteLine("\n Subiendo Capo");
    }
    public void Bajar()
    {
        Console.WriteLine("\n Bajando Capo");
    }
}
}

```

CLASE DERIVADA CAMION

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ConsoleApplication1
{
    class Camion:Vehiculo
    {
        float carga;

        public float Carga
        {
            get { return carga; }
            set { carga = value; }
        }
        public Camion(): base() { }
        public Camion(double vel, double gas, string mar, string
ori, string des, float car): base(vel, gas, mar, ori, des)
        {
            carga = car;
        }
        public void Cargar(float kilos)
        {
            Carga = Carga + kilos;
        }
    }
}

```

CLASE PRINCIPAL

```

using System;

```

```

using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Vehiculo
{
    class Program
    {
        static void Main(string[] args)
        {
            // Creamos un objeto de tipo Auto que heredará de
            Vehículo
            Auto miCoche = new Auto(90, 50, "Honda", "Tijuana",
            "Mexicali", true);

            Console.WriteLine("\n\nDatos del Automóvil ");
            Console.WriteLine("La marca del auto es : {0}",
            miCoche.Marca);
            Console.WriteLine("Lleva una velocidad de {0} Km/h",
            miCoche.Velocidad);
            miCoche.Llenartanque(10);
            Console.WriteLine("El tanque tiene {0} litros",
            miCoche.Gasolina);
            Console.WriteLine("El lugar de partida es : {0}",
            miCoche.Origen);
            Console.WriteLine("El lugar de destino es : {0}",
            miCoche.Destino);

            if (miCoche.Descapotable == true)
            {
                Console.WriteLine("El Automóvil es descapotable");
                // métodos definidos en la clase Auto
                miCoche.Subir();
                miCoche.Bajar();
            }

            // Creamos un objeto de tipo Camion que heredará de
            Vehículo
            Camion miCamion= new
            Camion(100,25,"Reno","Tecate","Ensenada",146.23f);

            Console.WriteLine("\n\nDatos del Camion ");
            Console.WriteLine("La marca del Camion es : {0}",
            miCamion.Marca);
            Console.WriteLine("Lleva una velocidad de {0} Km/h",
            miCamion.Velocidad);

            // Carga se define en la clase Camion
            miCamion.Cargar(60);
            Console.WriteLine("El tanque tiene {0} litros",
            miCamion.Gasolina);
            Console.WriteLine("El lugar de partida es : {0}",
            miCamion.Origen);
            Console.WriteLine("El lugar de destino es : {0}",
            miCamion.Destino);

```

```

        Console.WriteLine("LLeva una carga de {0} Kg.",
miCamion.Carga);

    }

}

```

d) Corrida o Pantalla de captura

```

C:\Windows\system32\cmd.exe

Datos del Automóvil
La marca del auto es : Honda
Lleva una velocidad de 90 Km/h
El tanque tiene 60 litros
El lugar de partida es : Tijuana
El lugar de destino es : Mexicali
El Automóvil es descapotable

Subiendo Capo
Bajando Capo

Datos del Camion
La marca del Camion es : Reno
Lleva una velocidad de 100 Km/h
El tanque tiene 25 litros
El lugar de partida es : Tecate
El lugar de destino es : Ensenada
LLeva una carga de 206.23 Kg.
Press any key to continue . . .

```

Sugerencias didácticas

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.
 Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML.
 Identificar la clase base y las clases derivadas con sus miembros heredados.
 Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno

El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.

Bibliografía

Bell, D. and Parr, M. (2011). *C# para estudiantes*. Pearson Educación de México, S.A. de C.V.

Ceballos Sierra, F. (2012). *Microsoft C#*. México, D.F.: Alfaomega.

Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). *La Biblia de C#*. Anaya.

Gonzalez Seco, J. (2012). *El lenguaje de programación C#*.

Larman, C. (2003). *UML y patrones*. Naucalpan de Juárez: Pearson Educación.

Msdn.microsoft.com. (2018). *Objetos, clases y estructuras (C#)*. [online]
Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	SIETE
Nombre	Heredar de la clase Figura Geométrica
Competencia específica de la unidad 3	Implementar la herencia en clases derivadas para reutilizar los miembros de una clase base.
Introducción	Una clase puede redefinir cualquier método heredado de una clase padre siempre que sea necesario que su comportamiento en la subclase sea diferente. Es posible referirse a un miembro de la clase base que se necesita redefinir en la clase hija utilizando la palabra reservada base y después el miembro que se redefine. El modificador base hace referencia a los métodos y atributos pero en este caso perteneciente a la clase padre del objeto que se está invocando un método o propiedad también se utiliza para indicar cómo deseamos inicializar los campos heredados de la clase padre.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Herencia 3.4 Referencia al objeto de la clase base.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan las clases utilizando el mecanismo de abstracción y herencia. 2. Clasifica los datos y define la clase base y la clase derivada representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a las clases mostradas en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica.

5. Ejecuta el código y revisa los resultados.

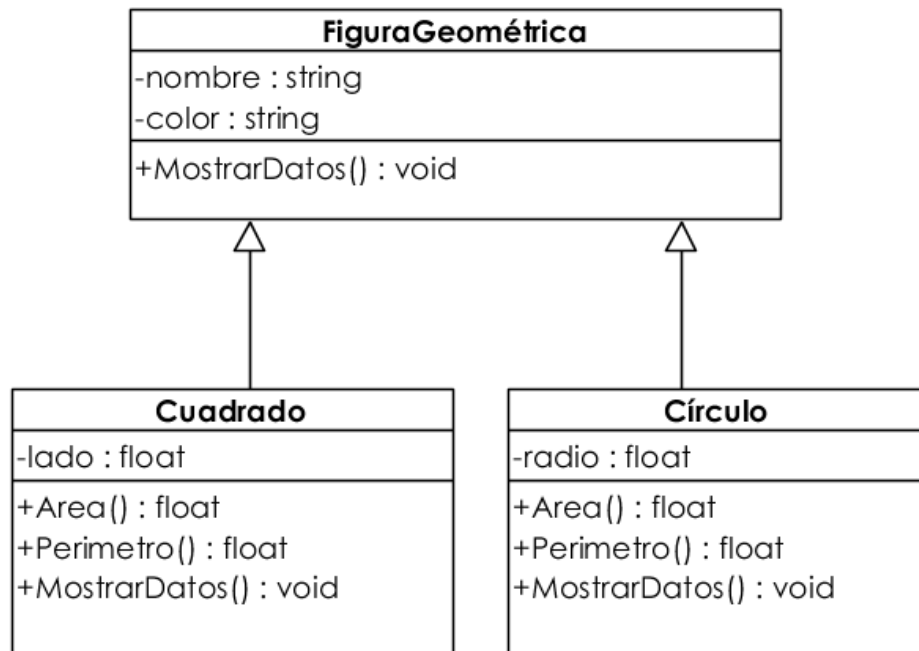
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Hereda de la clase base Figura Geométrica a las clases derivadas Cuadrado y Circulo, agrega los atributos, propiedades, constructores y métodos para calcular el área y perímetro correspondientes a cada clase.

a) Datos :

- nombre de la figura
- color
- lado del cuadrado
- área del cuadrado
- perímetro del cuadrado
- radio del círculo
- área del círculo
- perímetro del círculo

b) Diagrama de clases



c) Código C# en consola

CLASE BASE FIGURAGEOMETRICA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FiguraGeometrica
{
    class FiguraGeometrica
    {
        // atributos
        string nombre, color;

        // propiedades
        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
        public string Color
        {
            get { return color; }
            set { color = value; }
        }

        // Método Constructor por defecto
        public FiguraGeometrica() { }

        // Método Construntor sobrecargado
        public FiguraGeometrica(string n, string c)
        {
            nombre = n;
            color = c;
        }

        public void MostrarDatos()
        {
            Console.WriteLine("La Figura Geometrica es : {0}",
nombre);
            Console.WriteLine("El color de la figura es : {0}",
color);
        }
    }
}
```

CLASE DERIVADA CUADRADO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FiguraGeometrica
{
    class Cuadrado : FiguraGeometrica
    {
        // atributos
        float lado;

        // Propiedades
        public float Lado
        {
            get { return lado; }
            set { lado = value; }
        }

        // Método Constructor por defecto de la clase derivada
        public Cuadrado(): base() { }

        // Método Constructor sibrecargado de la clase derivada
        public Cuadrado(float la, string n, string c):base(n,c)
        {
            lado = la;
        }

        // métodos que regresan valor
        public float Area()
        {
            return lado * lado;
        }

        public float Perimetro()
        {
            return lado * 4;
        }

        // métodos que no regresan valor
        public void MostrarDatos()
        {
            base.MostrarDatos();
            Console.WriteLine("Area y Perimetro del
{0}", Nombre);
            Console.WriteLine("Lado      = {0:N2} cm", lado);
            Console.WriteLine("Area      = {0:N2} cm2", Area());
            Console.WriteLine("Perimetro = {0:N2} cm",
Perimetro());
        }
    }
}
```

CLASE DERIVADA CIRCULO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FiguraGeometrica
{
    class Circulo : FiguraGeometrica
    {
        // Atributos
        float radio;

        // Propiedades
        public float Radio
        {
            get { return radio; }
            set { radio = value; }
        }

        // Método Constructor por defecto
        public Circulo():base() { }

        // Método constructor sobrecargado
        public Circulo(float r, string n, string c):base(n, c)
        {
            r = radio;
        }

        //Métodos
        public float Area()
        {
            return (float) (Math.PI * Math.Pow((double)radio,2.0));
        }

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

        // métodos que no regresan valor
        public void MostrarDatos()
        {
            base.MostrarDatos();
            Console.WriteLine("Area y Perimetro del
{0}",Nombre);
            Console.WriteLine("Radio      = {0:N2} cm", radio);
            Console.WriteLine("Area      = {0:N2} cm2", Area());
            Console.WriteLine("Perimetro = {0:N2} cm",
Perimetro());
        }
    }
}
```

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace FiguraGeometrica
{
    class Program
    {
        static void Main(string[] args)
        {
            int opc = 0;

            do
            {
                Console.Clear();
                Console.WriteLine("Menú de Figuras
Geométricas");
                Console.WriteLine("\n 1.Cuadrado\n 2.Círculo\n
3.Salir");
                Console.Write("Seleccione opción : ");
                opc= int.Parse(Console.ReadLine());

                switch (opc)
                {

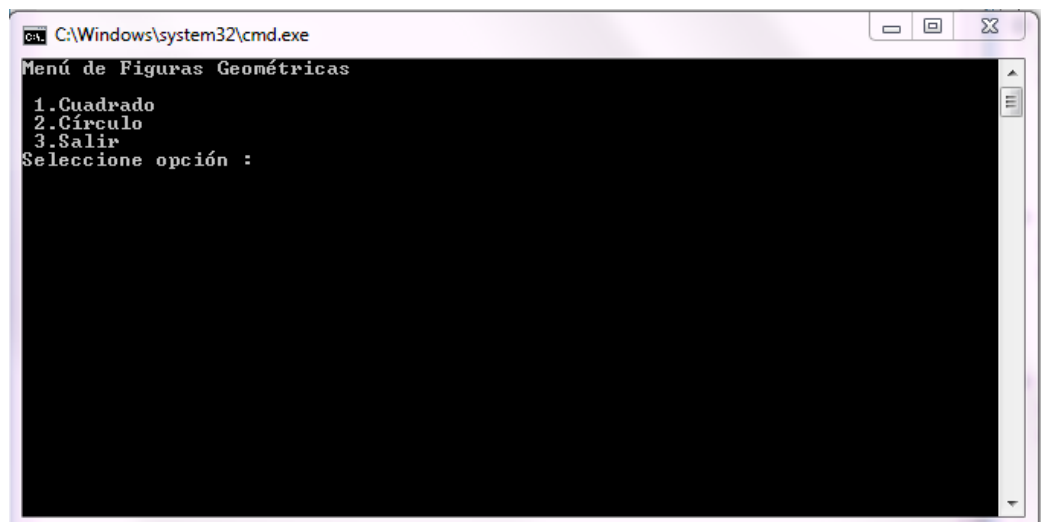
                    case 1:
                        Console.Clear();
                        Cuadrado Cu = new Cuadrado();
                        Console.WriteLine("Datos de la Figura");
                        Console.Write("Figura : ");
                        Cu.Nombre = Console.ReadLine();
                        Console.Write("Color : ");
                        Cu.Color = Console.ReadLine();
                        Console.Write("Lado : ");
                        Cu.Lado =
float.Parse(Console.ReadLine());
                        Cu.MostrarDatos();
                        Console.ReadKey();
                        break;

                    case 2:
                        Console.Clear();
                        Circulo Ci = new Circulo();
                        Console.WriteLine("Datos de la Figura");
                        Console.Write("Figura : ");
                        Ci.Nombre = Console.ReadLine();
                        Console.Write("Color : ");
                        Ci.Color = Console.ReadLine();
                        Console.Write("Radio : ");
                        Ci.Radio =
float.Parse(Console.ReadLine());
```

```
        Ci.MostrarDatos();  
        Console.ReadKey();  
        break;  
  
        case 3:  
            Console.Clear();  
            Console.WriteLine("\nSalir del Menu de  
Figuras ");  
            break;  
        }  
    } while (opc != 3);  
}  
}
```

d) Corrida o Pantalla de captura

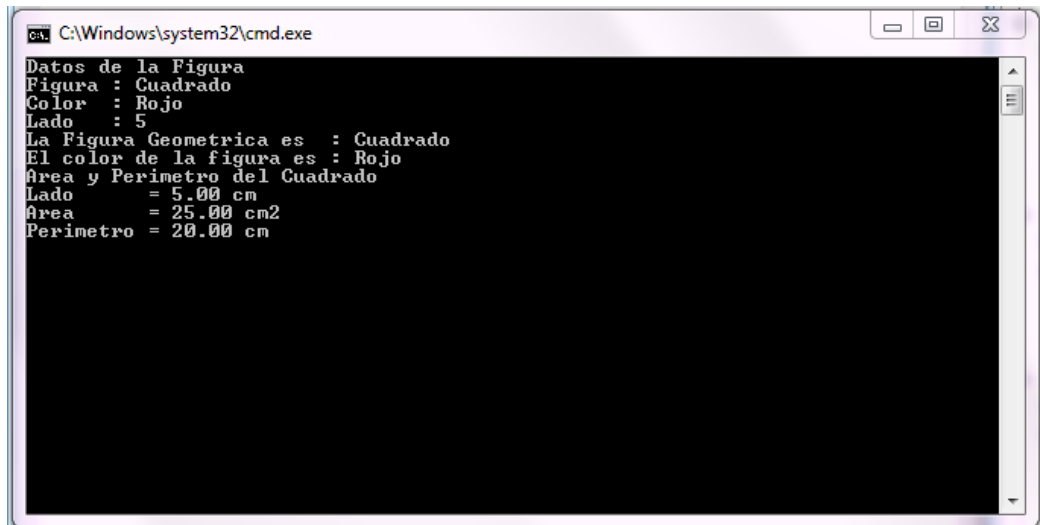
Menú :



The screenshot shows a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text:

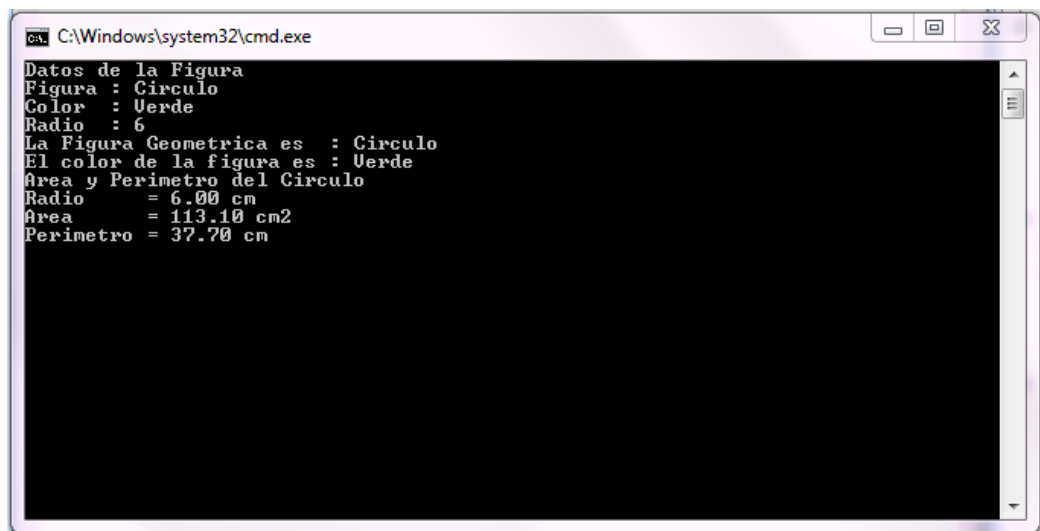
```
Menú de Figuras Geométricas  
1.Cuadrado  
2.Círculo  
3.Salir  
Seleccione opción :
```

opción 1 :



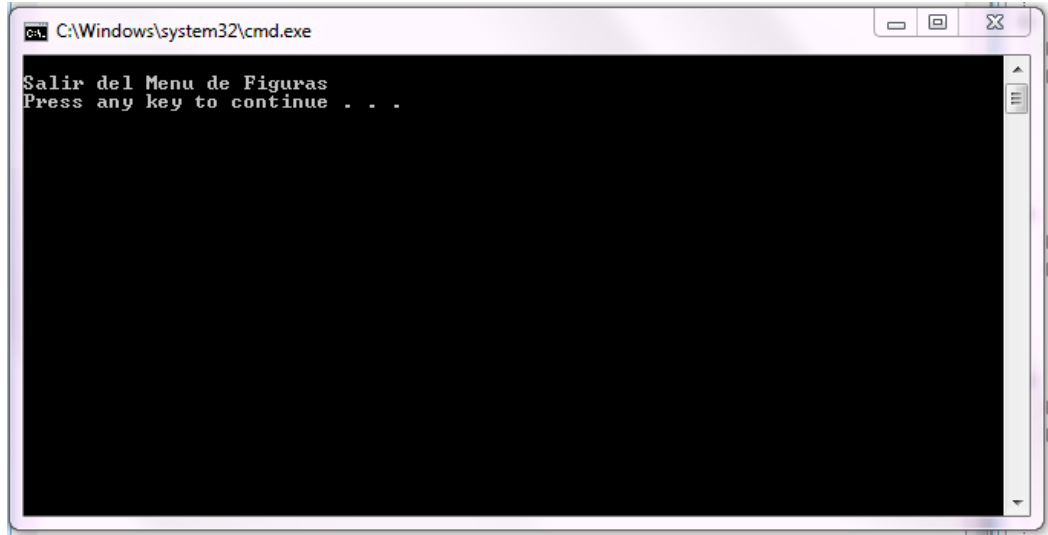
```
C:\Windows\system32\cmd.exe
Datos de la Figura
Figura : Cuadrado
Color : Rojo
Lado : 5
La Figura Geometrica es : Cuadrado
El color de la figura es : Rojo
Area y Perimetro del Cuadrado
Lado      = 5.00 cm
Area      = 25.00 cm2
Perimetro = 20.00 cm
```

opción 2 :



```
C:\Windows\system32\cmd.exe
Datos de la Figura
Figura : Circulo
Color : Verde
Radio : 6
La Figura Geometrica es : Circulo
El color de la figura es : Verde
Area y Perimetro del Circulo
Radio     = 6.00 cm
Area      = 113.10 cm2
Perimetro = 37.70 cm
```

opción 3 :



Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Identificar la clase base y las clases derivadas con sus miembros heredados. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i> . Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i> . México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i> . Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i> . Larman, C. (2003). <i>UML y patrones</i> . Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i> . [online] Available at: https://msdn.microsoft.com/es-

es/library/ms173109(v=vs.80).aspx [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	OCHO
Nombre	Identificar los constructores de la clase derivada
Competencia específica de la unidad 3	Implementar la herencia en clases derivadas para reutilizar los miembros de una clase base.
Introducción	Si en la definición del constructor de alguna clase que derive de otra no incluimos inicializador base el compilador considera que este es <i>base()</i> , por ello hay que estar seguro de que si que no incluye base en la definición del constructor, el tipo padre del tipo al que pertenezca disponga de constructor sin parámetros.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Herencia 3.5 Constructores y destructores en clases derivadas
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan las clases utilizando el mecanismo de abstracción y herencia. 2. Clasifica los datos y define la clase base y la clase derivada representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a las clases mostradas en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

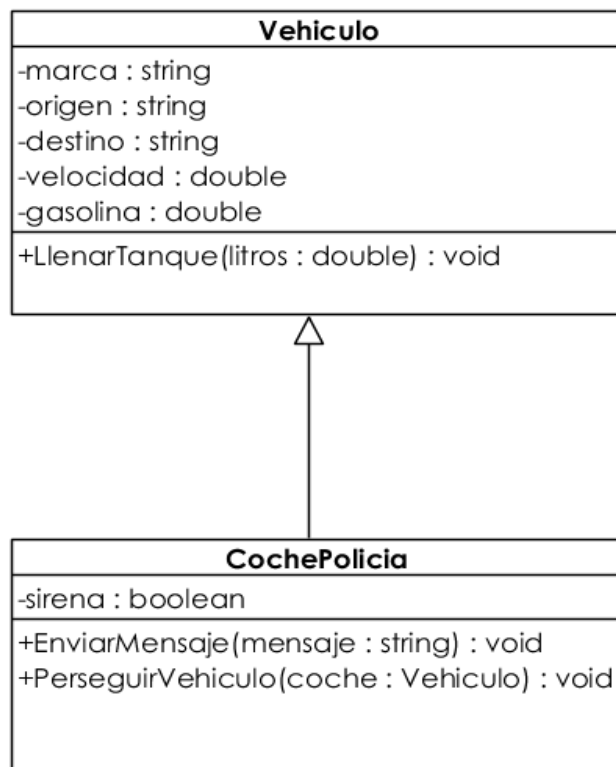
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Hereda de la clase vehículo a la clase derivada CochePolicia que ejecute un método para enviar mensaje y le envíe como parámetro el mensaje y otro método para perseguir el vehículo y envíe como parámetro el objeto vehículo que está persiguiendo.

a) Datos :

- marca
- origen
- destino
- velocidad
- gasolina
- llenar tanque
- litros de gasolina
- sirena del coche de policia
- perseguir vehiculo
- enviar mensaje

b) Diagrama de clases



c) Código C# en consola

CLASE BASE VEHICULO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HerenciaConstructores
{
    class Vehiculo
    {
        // Atributos
        string marca, origen, destino;
        double velocidad, gasolina;

        // propiedades
        public double Gasolina
        {
            get { return gasolina; }
            set { gasolina = value; }
        }
        public double Velocidad
        {
            get { return velocidad; }
            set
            {
                if (value < 0)
                    velocidad = 0;
                else
                    velocidad = value;
            }
        }
        public string Marca
        {
            get { return marca; }
            set { marca = value; }
        }
        public string Origen
        {
            get { return origen; }
            set { origen = value; }
        }
        public string Destino
        {
            get { return destino; }
            set { destino = value; }
        }

        // Método constructor por defecto
        public Vehiculo() { }
    }
}
```

```

        // Método constructor sobrecargado
        public Vehiculo(string m, string o, string d, double
v, double g)
        {
            marca = m;
            origen = o;
            destino = d;
            velocidad = v;
            gasolina = g;
        }

        // otros metodos
        public void Imprimir()
        {
            Console.WriteLine("\n\nEl Vehiculo es de la marca:
{0}", marca);
            Console.WriteLine("Tiene en el tanque de gasolina
{0} litros de gasolina", gasolina);
            Console.WriteLine("El Vehiculo se dirige de {0} a
{1} a {2} Km/h",origen, destino, velocidad);
        }

        public void LlenarTanque(double litros)
        {
            gasolina = gasolina + litros;
        }
    }
}

```

CLASE DERIVADA COCHEPOLICIA

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HerenciaConstructores
{
    class CochePolicia : Vehiculo
    {
        bool sirena = false;

        public bool Sirena
        {
            get { return sirena; }
            set { sirena = value; }
        }

        // Método constructor por defecto
        public CochePolicia() : base () {}

        // Método constructor sobrecargado
        public CochePolicia(string m, string o, string d,
double v, double g, bool s)

```

```

        : base(m, o, d, v, g)
    {
        sirena = s;
    }

    public void EnviarMensaje(string mensaje)
    {
        Console.WriteLine("Se ha enviado el mensaje
siguiente a la central: {0}", mensaje);
    }

    public void PerseguirVehículo(Vehiculo coche)
    {
        // cambia el valor del atributo destino
        Destino = coche.Destino;
        Console.WriteLine("nuevo destino : {0}", Destino);
    }
}
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace HerenciaConstructores
{
    public class Principal
    {
        public static void Main()
        {
            // crear objeto usando el constructor por defecto
            Vehiculo miCoche = new Vehiculo();

            Console.WriteLine("Datos del Vehiculo");
            Console.Write("La marca del vehículo : ");
            miCoche.Marca = Console.ReadLine();
            Console.Write("La velocidad es de : ");
            miCoche.Velocidad =
double.Parse(Console.ReadLine());
            Console.Write("Con cuantos litros quieres cargar el
tanque : ");
            double litros=double.Parse(Console.ReadLine());
            miCoche.LlenarTanque(litros);
            Console.Write("El lugar de partida es : ");
            miCoche.Origen = Console.ReadLine();
            Console.Write("El lugar de destino es : ");
            miCoche.Destino = Console.ReadLine();
            Console.Clear();
            Console.WriteLine("Datos del Vehiculo");
            miCoche.Imprimir();
        }
    }
}

```

```

        // Creamos un objeto de tipo CochePolicía usando el
        constructor sobrecargad
        CochePolicia miCochePolicia = new
        CochePolicia("Ford","Tijuana", "Mexicali",100,20,true);

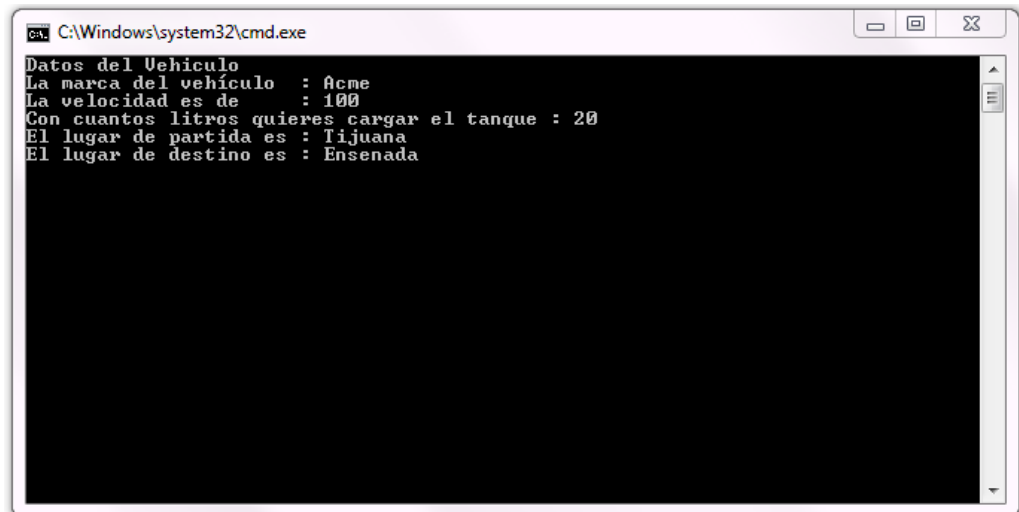
        Console.WriteLine("\n\nDatos del CochePolicia");

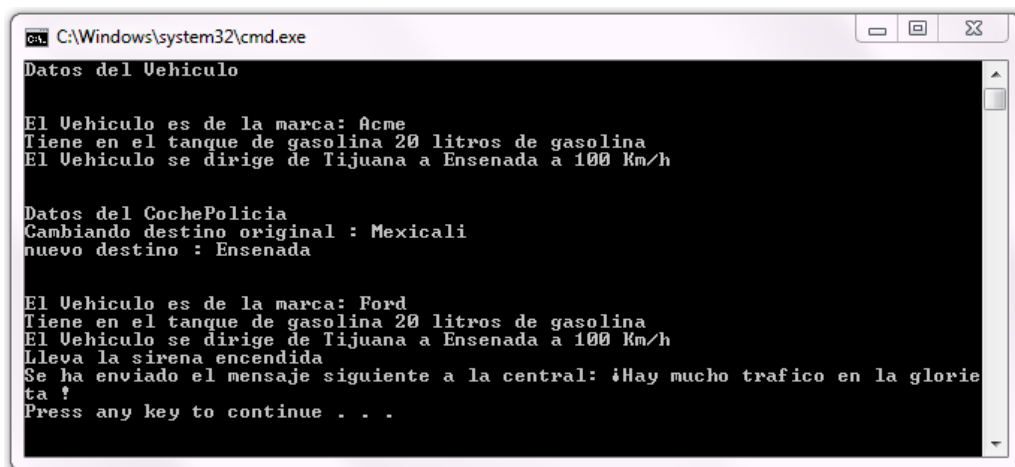
        Console.WriteLine("Cambiando destino original :
{0}", miCochePolicia.Destino);
        // PerseguirVehículo se define en la clase
        CochePolicía enviamos el objeto vehiculo como parametro
        miCochePolicia.PerseguirVehículo(miCoche);

        // Imprimir se hereda de la clase Vehículo
        miCochePolicia.Imprimir();
        if (miCochePolicia.Sirena == true)
            Console.WriteLine("Lleva la sirena encendida
");
        else
            Console.WriteLine("Lleva la sirena apagada ");
        // EnviarMensaje se define en la clase CochePolicía
        miCochePolicia.EnviaMensaje(";Hay mucho trafico en
la glorieta !");
    }
}
}

```

d) Corrida o Pantalla de captura





```
C:\Windows\system32\cmd.exe
Datos del Vehiculo

El Vehiculo es de la marca: Acme
Tiene en el tanque de gasolina 20 litros de gasolina
El Vehiculo se dirige de Tijuana a Ensenada a 100 Km/h

Datos del CochePolicia
Cambiano destino original : Mexicali
nuevo destino : Ensenada

El Vehiculo es de la marca: Ford
Tiene en el tanque de gasolina 20 litros de gasolina
El Vehiculo se dirige de Tijuana a Ensenada a 100 Km/h
Lleva la sirena encendida
Se ha enviado el mensaje siguiente a la central: ¡Hay mucho trafico en la glorie
ta !
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Identificar los miembros heredados, definir el constructor por defecto y los constructores sobrecargados. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
-------------------------------	--

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
---------------------------	--

Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i>. [online] Available at: https://msdn.microsoft.com/es-
---------------------	--

es/library/ms173109(v=vs.80).aspx [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

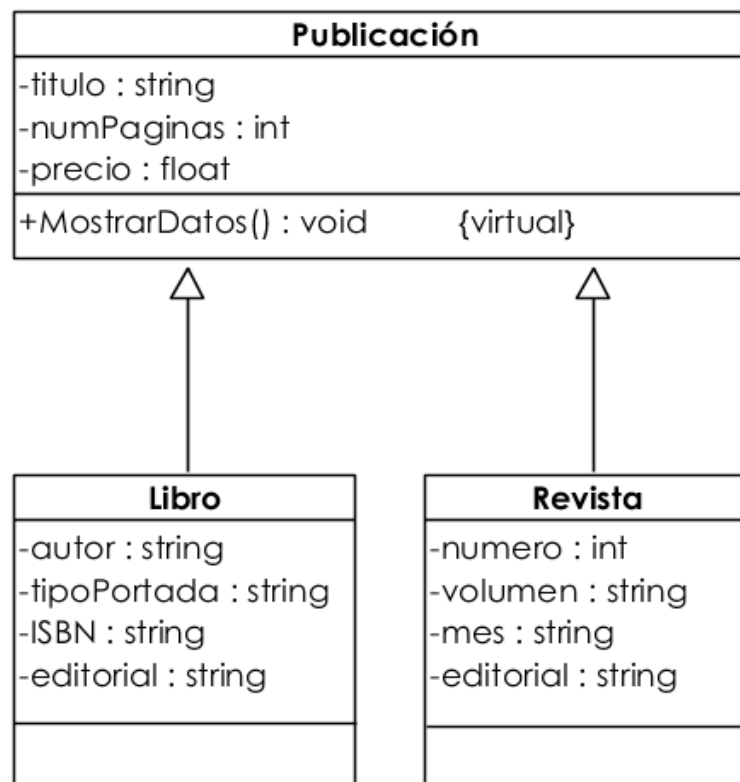
No. de Práctica	NUEVE
Nombre	Definir una clase con métodos virtuales
Competencia específica de la unidad 3	Implementar la herencia en clases derivadas para reutilizar los miembros de una clase base.
Introducción	Un método virtual especifica una implementación de un método que puede ser sustituida por polimorfismo en una clase derivada. Del mismo modo, un método virtual no especifica la única implementación de un método. Los métodos virtuales se declaran con la palabra reservada virtual y deben contener cuerpo, no son privados ni estáticos.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Herencia 3.6 Redefinición de métodos en clases derivadas.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una persona utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase persona representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Define una clase **Publicación** con los siguientes atributos: **titulo**, **numPaginas** y **precio**, define un método virtual para mostrar los datos. Hereda de esta clase a una clase **Libro** con los siguientes atributos : **autor**, **tipoPortada**, **ISBN**, **editorial**, y otra clase **Revista** con los siguientes atributos: **numero**, **volumen**, **mes**, **editorial**. Redefine el método heredado.

a) Datos :

- titulo
- num de paginas
- precio
- autor
- tipo de portada
- ISBN
- editorial del libro
- numero de revista
- volumen
- mes de la revista
- editorial de la revista
- mostrar los datos

b) Diagrama de clases



c) Código C# en consola

CLASE BASE PUBLICACION

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Publicacion
{
    class Publicacion
    {
        // Atributos para Publicación:
        string titulo;
        int numPaginas;
        float precio;

        // propiedades

        public string Titulo
        {
            get { return titulo; }
            set { titulo = value; }
        }
        public int NumPag
        {
            get { return numPaginas; }
            set { numPaginas = value; }
        }
        public float Precio
        {
            get { return precio; }
            set { precio = value; }
        }

        // Método Constructor por defecto
        public Publicacion() { }

        // Método Constructor por sobrecargado
        public Publicacion(string t, int n, float p)
        {
            titulo = t;
            numPaginas = n;
            precio = p;
        }

        // Método
        public virtual void MostrarDatos()
        {
            Console.WriteLine(" Titulo           : {0}", titulo);
            Console.WriteLine(" Num. Páginas    : {0}",
```

```

numPaginas);
        Console.WriteLine(" Precio           : {0:c2}",
precio);
    }
}
}

```

CLASE DERIVADA LIBRO

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Publicacion
{
    class Libro : Publicacion
    {
        // Atributos
        string tipoPortada, ISBN, autor, editorial;

        // propiedades
        public string Portada
        {
            get { return tipoPortada; }
            set { tipoPortada = value; }
        }
        public string Isbn
        {
            get { return ISBN; }
            set { ISBN = value; }
        }
        public string Autor
        {
            get { return autor; }
            set { autor = value; }
        }
        public string Editorial
        {
            get { return editorial; }
            set { editorial = value; }
        }

        // Método Constructor por defecto
        public Libro():base() { }

        // Método Constructor por sobrecargado
        public Libro(string t, int n, float p, string tp, string i,
string a, string e) : base (t,n,p)
        {
            tipoPortada=t;
            ISBN=i;
            Autor=a;
            Editorial=e;

```

```

    }

    // redefinicion de Método virtual
    public override void MostrarDatos()
    {
        Console.WriteLine(" \n Publicación de Libro ");
        base.MostrarDatos();
        Console.WriteLine(" Tipo de Portada : {0}",
tipoPortada);
        Console.WriteLine(" ISBN           : {0}", ISBN);
        Console.WriteLine(" Autor         : {0}", autor);
        Console.WriteLine(" Editorial      : {0}",
editorial);
    }
}
}

```

CLASE DERIVADA REVISTA

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Publicacion
{
    class Revista:Publicacion
    {
        // Atributos para Revista :
        int numero;
        string volumen, mes, editorial;

        // propiedades
        public int Num
        {
            get { return numero; }
            set { numero = value; }
        }
        public string Volumen
        {
            get { return volumen; }
            set { volumen = value; }
        }
        public string Mes
        {
            get { return mes; }
            set { mes = value; }
        }
        public string Editorial
        {
            get { return editorial; }

```

```

        set { editorial = value; }
    }

    // Método Constructor por defecto
    public Revista():base() { }

    // Método Constructor por sobrecargado
    public Revista(string t, int n, float p,int nu, string
vol, string m, string e) : base (t,n,p)
    {
        numero=nu;
        volumen=vol;
        mes=m;
        Editorial=e;
    }

    // redefinicion de Método virtual
    public override void MostrarDatos()
    {
        Console.WriteLine(" \nPublicación de Revista ");
        base.MostrarDatos();
        Console.WriteLine(" Número           : {0}", numero);
        Console.WriteLine(" Volumen           : {0}",
volumen);
        Console.WriteLine(" Mes             : {0}", mes);
        Console.WriteLine(" Editorial        : {0}",
editorial);
    }
}
}

```

CLASE PRINCIPAL

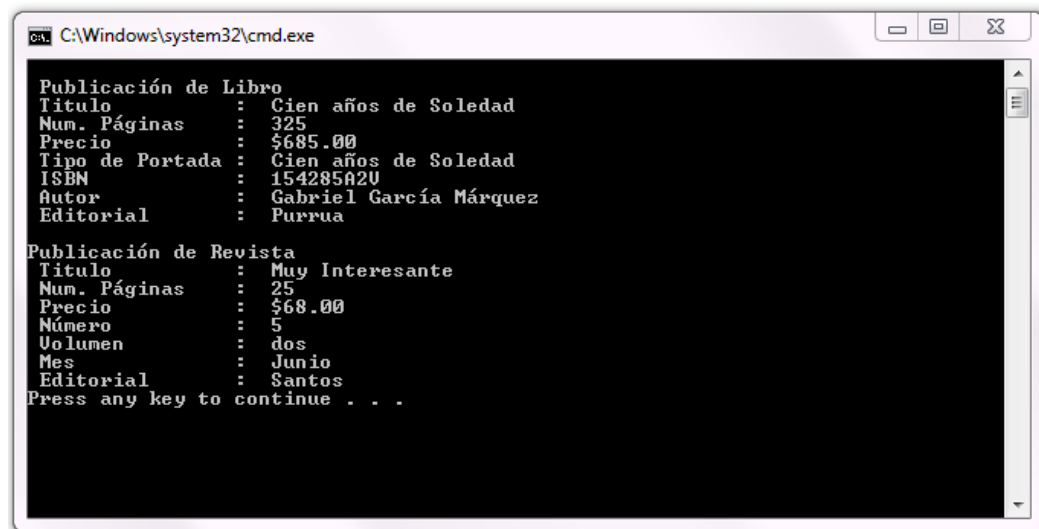
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Publicacion
{
    class Program
    {
        static void Main(string[] args)
        {
            Libro L = new Libro("Cien años de
Soledad",325,685,"Dura","154285A2V","Gabriel García
 Márquez","Purrua");
            L.MostrarDatos();
            Revista R = new Revista("Muy
Interesante",25,68,5,"dos","Junio","Santos");
            R.MostrarDatos();
        }
    }
}

```

d) Corrida o Pantalla de captura



```
C:\Windows\system32\cmd.exe

Publicación de Libro
Titulo      : Cien años de Soledad
Num. Páginas : 325
Precio      : $685.00
Tipo de Portada : Cien años de Soledad
ISBN       : 154285A2U
Autor      : Gabriel García Márquez
Editorial   : Purrua

Publicación de Revista
Titulo      : Muy Interesante
Num. Páginas : 25
Precio      : $68.00
Número     : 5
Volumen    : dos
Mes        : Junio
Editorial   : Santos
Press any key to continue . . .
```

Sugerencias didácticas

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.
Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML.
Identificar los miembros heredados, definir el constructor por defecto y los constructores sobrecargados.
Redefine el método virtual heredado.
Nombrar otros ejemplos de clases para identificar sus miembros.
Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno

El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.

Bibliografía

Bell, D. and Parr, M. (2011). *C# para estudiantes*. Pearson Educación de México, S.A. de C.V.

Ceballos Sierra, F. (2012). *Microsoft C#*. México, D.F.: Alfaomega.

Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). *La Biblia de C#*. Anaya.

Gonzalez Seco, J. (2012). *El lenguaje de programación C#*.

Larman, C. (2003). *UML y patrones*. Naucalpan de Juárez: Pearson Educación.

Msdn.microsoft.com. (2018). *Objetos, clases y estructuras (C#)*. [online]

Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	DIEZ
Nombre	Definir una clase con métodos override
Competencia específica de la unidad 3	Implementar la herencia en clases derivadas para reutilizar los miembros de una clase base.
Introducción	Un método override especifica otra implementación de un método virtual. Los métodos virtuales definidos en una clase base pueden ser sustituidos por polimorfismo en una clase derivada. Se usa la palabra reservada override y sustituyen a los métodos virtuales.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Herencia 3.6 Redefinición de métodos en clases derivadas.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una persona utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase persona representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición :

La clase **Animalito** es una clase con cuatro atributos miembros:

- a) Un *String* indicando la **especie** (león, águila, abeja),
- b) Un *String* indicando el **nombre** del animal concreto (Simba, Maya, RinTinTin).
- c) Un *double* indicando el **peso** el kg.
- d) Un *int* indicando el número de **jaula** que se asigna al animal.

Además, la clase **Animalito** declara un método **virtual** **QueClaseDeAnimalitoEres()** que habrá que definir en las clases derivadas.

La clase **Mamifero** no añade nuevos atributos miembro, aunque deberá implementar el método **QueClaseDeAnimalitoEres()**.

La clase **Ave** tiene dos nuevos atributos :

- a) Un *String* **colorPlumaje** indicando el color predominante y
- b) Un *double* indicando la **alturaMaximaVuelo**.

La clase **Insecto** tiene un nuevo atributo miembro de tipo *boolean* llamada **vuela** que indica si el insecto vuela o no.

Para realizar este ejercicio se pide lo siguiente:

I. **Crear las cuatro clases indicadas**, con los correspondientes constructores y sobrecarga de constructores, Como ayuda, se indica que el orden de los argumentos en el constructor parametrizado de la clase base es:

```
public Animalito(string especie, string nombre, double peso, int jaula){...}
```

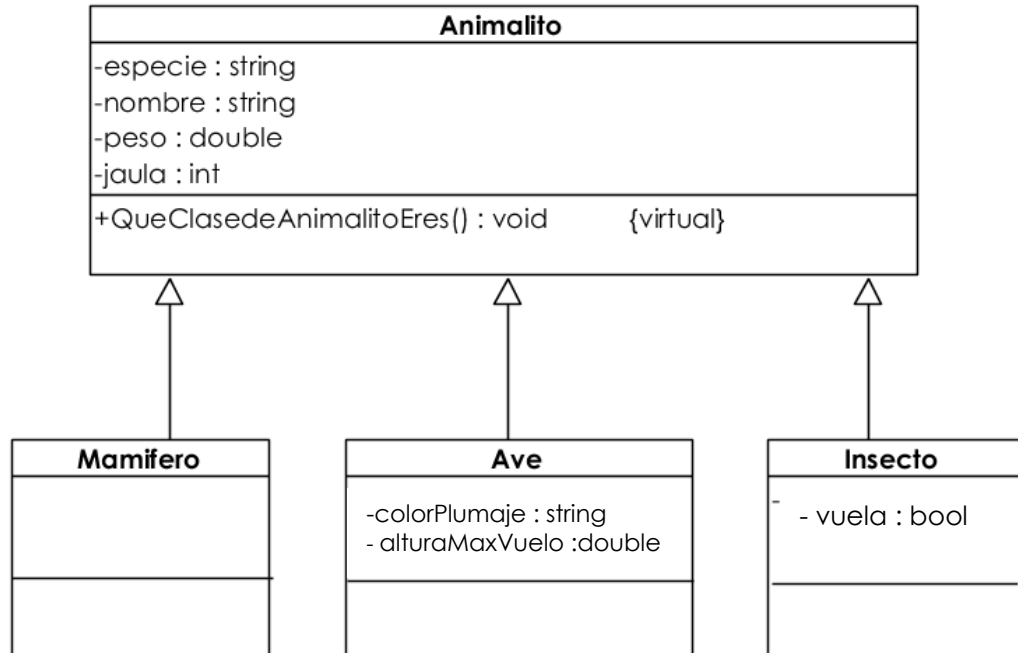
II. **Definir** los métodos llamados **QueClaseDeAnimalitoEres()** en cada una de las clases derivadas de **Animalito**. Este método no tiene valor de retorno (es *void*) ni argumentos.

Debe ser capaz de mostrar por la pantalla la información correspondiente al animalito de que se trate, utilizando para ello la información almacenada en las variables miembro.

a) Datos :

- especie
 - nombre
 - peso
 - numero de jaula
 - color del plumaje
 - altura máxima de vuelo
 - vuela el insecto
-

b) Diagrama de clases



c) Código C# en consola

CLASE BASE ANIMALITO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Zoologico
{
    abstract class Animalito
    {
        //Propiedades
        string especie, nombre;
        double peso;
        int jaula;

        //Propiedades
        public string Especie
        {
            get { return especie; }
            set { especie = value; }
        }
    }
}
```

```

    }
    public string Nombre
    {
        get { return nombre; }
        set { nombre = value; }
    }
    public double Peso
    {
        get { return peso; }
        set { peso = value; }
    }
    public int Jaula
    {
        get { return jaula; }
        set { jaula = value; }
    }

    //Constructores
    public Animalito() { }
    public Animalito(string e, string n, double p, int j)
    {
        Especie = e;
        Nombre = n;
        Peso = p;
        Jaula = j;
    }

    //Metodos
    public virtual void QueClasedeAnimalitoEres()
    {
        Console.WriteLine("Soy {0}", Nombre);
        Console.WriteLine("De la especie {0}", Especie);
        Console.WriteLine("Peso {0} Kg ", Peso);
        Console.WriteLine("Estoy en la jaula # {0}", Jaula);
    }
}
}

```

CLASE DERIVADA MAMIFERO

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Zoologico
{
    class Mamifero:Animalito
    {
        //Constructores
        public Mamifero() : base() { }
        public Mamifero(string e, string n, double p, int
j):base(e,n,p,j){}

```

```

        //Metodos
        public override void QueClasedeAnimalitoEres()
        {
            Console.WriteLine("Soy un Mamifero");
            base.QueClasedeAnimalitoEres();
        }
    }
}

```

CLASE DERIVADA AVE

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Zoologico
{
    class Ave:Animalito
    {
        //Atributos
        string colorPlumaje;
        double alturaMaxVuelo;

        //Propiedades
        public string ColorPlumaje
        {
            get { return colorPlumaje; }
            set { colorPlumaje = value; }
        }
        public double AlturaMaximaVuelo
        {
            get { return alturaMaxVuelo; }
            set { alturaMaxVuelo = value; }
        }

        //Constructores
        public Ave() : base() { }
        public Ave(string cp,double am,string e, string n,
double p, int j):base(e,n,p,j)
        {
            ColorPlumaje = cp;
            AlturaMaximaVuelo = am;
        }

        //Metodos
        public override void QueClasedeAnimalitoEres()
        {
            Console.WriteLine("Soy una Ave");
            base.QueClasedeAnimalitoEres();
            Console.WriteLine("Plumas de color {0} ",
ColorPlumaje);
            Console.WriteLine("Altura Maxima de vuelo : {0}

```

```
        km/hr ", AlturaMaximaVuelo);
    }
}
}
```

CLASE DERIVADA INSECTO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Zoologico
{
    class Insecto:Animalito
    {
        //Atributos
        bool vuela;

        //Propiedades
        public bool Vuela
        {
            get { return vuela; }
            set { vuela = value; }
        }

        //Constructores
        public Insecto() : base() { }
        public Insecto(bool v,string e, string n, double p, int
j):base(e,n,p,j)
        {
            Vuela = v;
        }

        //Metodos
        public override void QueClasedeAnimalitoEres()
        {
            Console.WriteLine("Soy un Insecto");
            base.QueClasedeAnimalitoEres();
            if (Vuela == true)
                Console.WriteLine("Si Vuela");
            else
                Console.WriteLine("No Vuela");
        }
    }
}
```

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Zoologico
{
    class Program
    {
        static void Main(string[] args)
        {
            int op = 0;
            do{
                Console.Clear();
                Console.WriteLine("\n\t\t\tMenú\n1-Mamífero\n2-Ave\n3-Insecto\n4-Salir\n\n\tOpcion: ");
                op = int.Parse(Console.ReadLine());
                switch (op)
                {
                    case 1:
                        Console.Clear();
                        Mamifero mamifero = new Mamifero();
                        Console.WriteLine("\t\t Datos del Mamífero ");
                        Console.WriteLine("Ingrese nombre del mamífero : ");
                        mamifero.Nombre = Console.ReadLine();
                        Console.WriteLine("Ingrese Especie : ");
                        mamifero.Especie = Console.ReadLine();
                        Console.WriteLine("Ingrese peso en kg : ");
                        mamifero.Peso = double.Parse(Console.ReadLine());
                        Console.WriteLine("Ingrese numero de jaula : ");
                        mamifero.Jaula = int.Parse(Console.ReadLine());
                        Console.Clear();
                        mamifero.QueClasedeAnimalitoEres();
                        break;

                    case 2:
                        Console.Clear();
                        Ave ave = new Ave();
                        Console.WriteLine("\t\t Datos del Ave ");
                        Console.WriteLine("Ingrese nombre del ave : ");
                        ave.Nombre = Console.ReadLine();
                        Console.WriteLine("Ingrese Especie : ");
                        ave.Especie = Console.ReadLine();
                        Console.WriteLine("Ingrese peso en kg
```

```

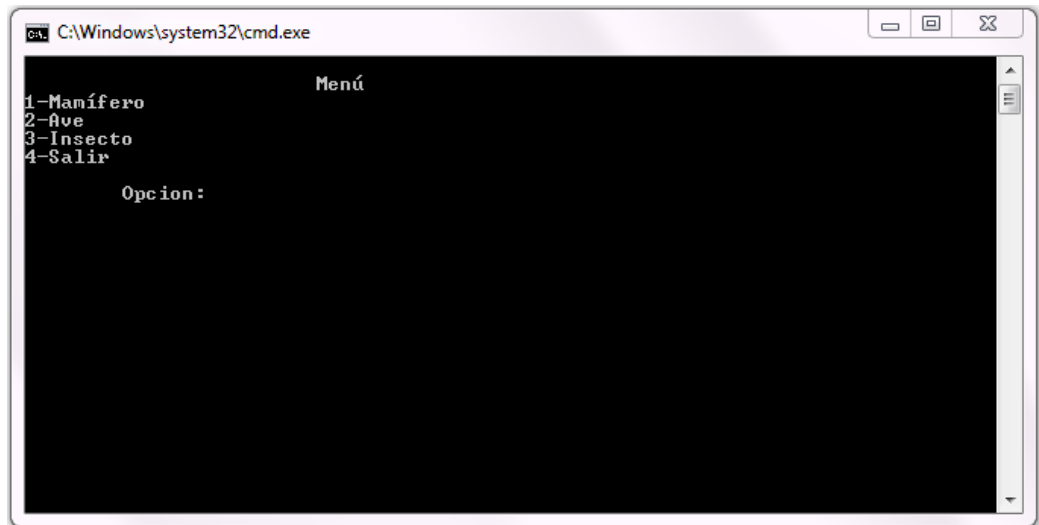
: ");
                                ave.Peso =
double.Parse(Console.ReadLine());
                                Console.Write("Ingrese numero de jaula
: ");
                                ave.Jaula =
int.Parse(Console.ReadLine());
                                Console.Write("Ingrese Color de plumas
: ");
                                ave.ColorPlumaje = Console.ReadLine();
                                Console.Write("Ingrese Altura Maxima de
vuelo : ");
                                ave.AlturaMaximaVuelo =
double.Parse(Console.ReadLine());
                                Console.Clear();
                                ave.QueClasedeAnimalitoEres();
                                break;
case 3:
    Console.Clear();
    Insecto Insecto = new Insecto();
    Console.WriteLine("\t\t Datos del
Insecto ");
    Console.Write("Ingrese nombre del
Insecto : ");
    Insecto.Nombre = Console.ReadLine();
    Console.Write("Ingrese Especie
: ");
    Insecto.Especie = Console.ReadLine();
    Console.Write("Ingrese peso en kg
: ");
    Insecto.Peso =
double.Parse(Console.ReadLine());
    Console.Write("Ingrese numero de jaula
: ");
    Insecto.Jaula =
int.Parse(Console.ReadLine());
    Console.Write("El insecto vuela(si/no)
: ");
    if (Console.ReadLine() == "si")
        Insecto.Vuela = true;
    else
        Insecto.Vuela = false;
    Console.Clear();
    Insecto.QueClasedeAnimalitoEres();
    break;
case 4 :
    Console.WriteLine("\n\t Salir del
Menú");
    break;
default :
    Console.WriteLine("\n\t Opción
inválida");
    break;
} Console.ReadKey();
} while (op<4);
}

```

```
}  
}
```

d) Corrida o Pantalla de captura

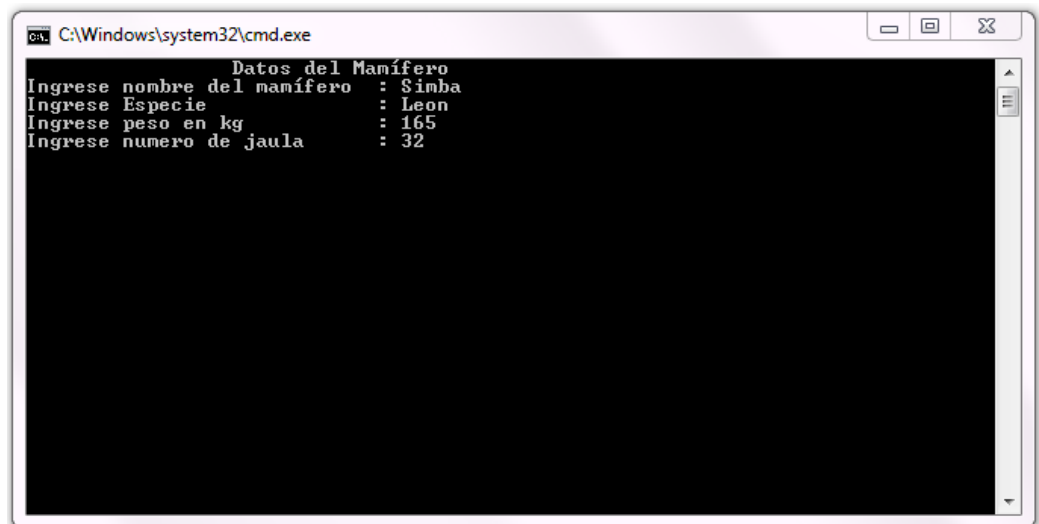
Menú



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. At the top right, the word "Menú" is displayed. Below it, a list of options is shown: "1-Mamífero", "2-Ave", "3-Insecto", and "4-Salir". Further down, the text "Opcion:" is displayed, indicating where the user should enter their choice.

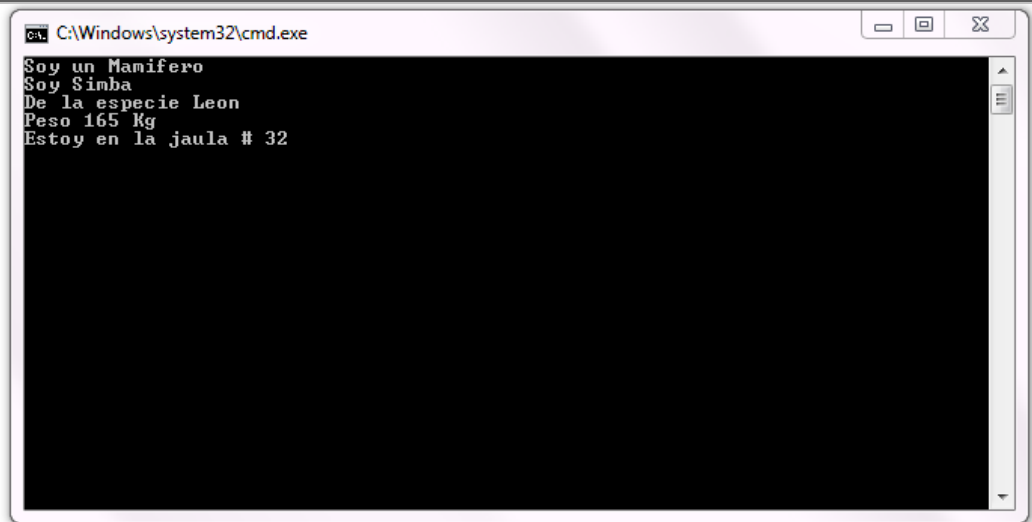
```
C:\Windows\system32\cmd.exe  
  
Menú  
1-Mamífero  
2-Ave  
3-Insecto  
4-Salir  
  
Opcion:
```

opción 1:



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background with white text. At the top right, the text "Datos del Mamífero" is displayed. Below it, four lines of data are shown, each with a label and a value separated by a colon: "Ingrese nombre del mamífero : Simba", "Ingrese Especie : Leon", "Ingrese peso en kg : 165", and "Ingrese numero de jaula : 32".

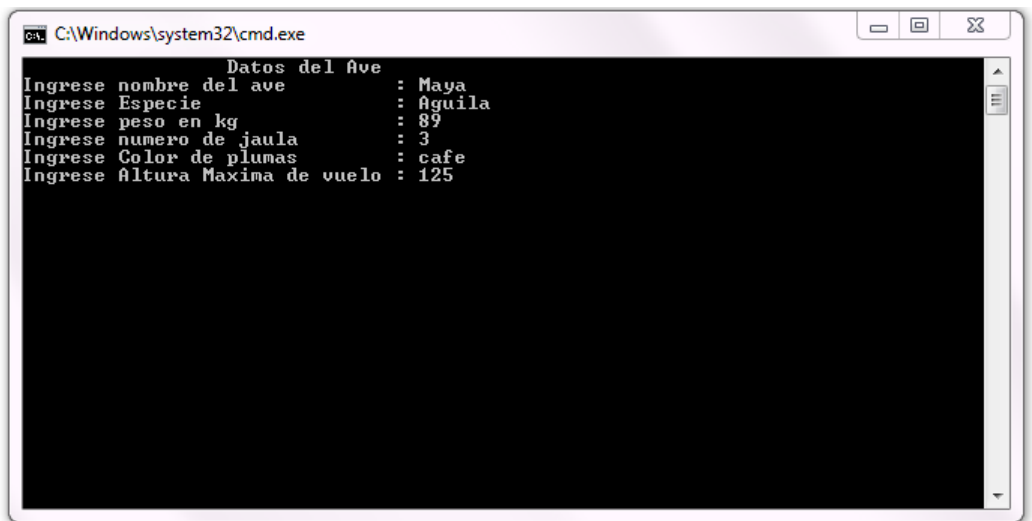
```
C:\Windows\system32\cmd.exe  
  
Datos del Mamífero  
Ingrese nombre del mamífero : Simba  
Ingrese Especie : Leon  
Ingrese peso en kg : 165  
Ingrese numero de jaula : 32
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background and white text. The text displayed is:

```
Soy un Mamifero  
Soy Simba  
De la especie Leon  
Peso 165 Kg  
Estoy en la jaula # 32
```

opción 2:



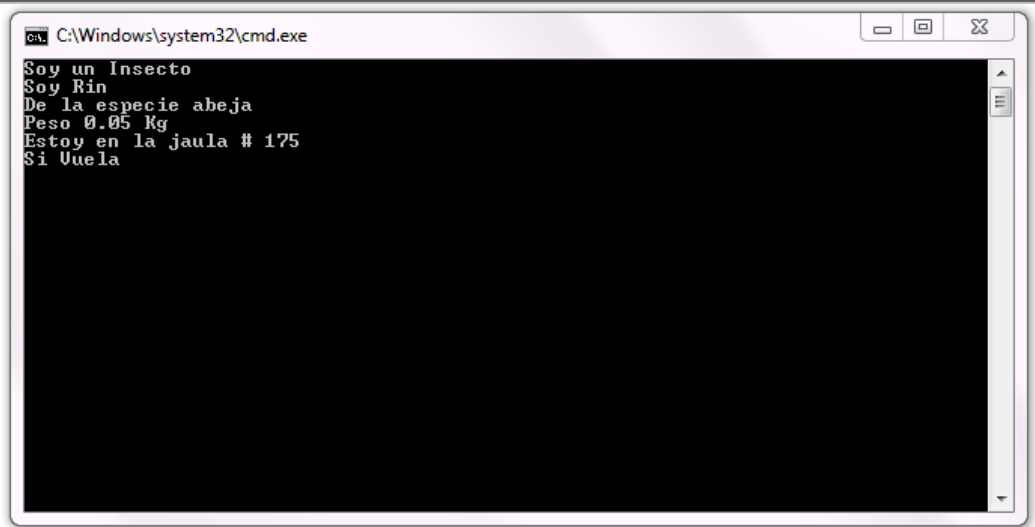
A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window has a black background and white text. The text displayed is:

```
                Datos del Ave  
Ingrese nombre del ave : Maya  
Ingrese Especie       : Aguila  
Ingrese peso en kg    : 89  
Ingrese numero de jaula : 3  
Ingrese Color de plumas : cafe  
Ingrese Altura Maxima de vuelo : 125
```

```
C:\Windows\system32\cmd.exe
Soy una Ave
Soy Maya
De la especie Aguila
Peso 89 Kg
Estoy en la jaula # 3
Plumas de color cafe
Altura Maxima de vuelo : 125 km/hr
```

opción 3 :

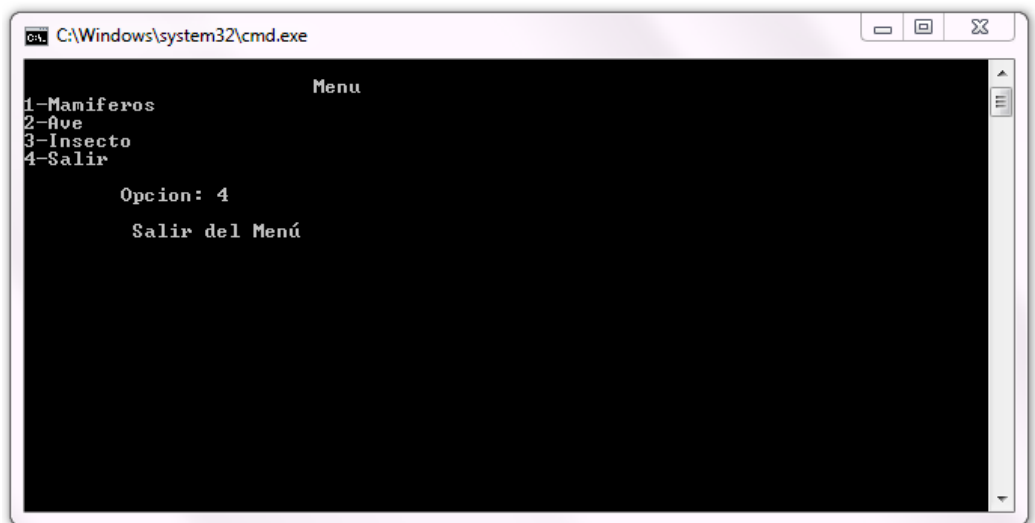
```
C:\Windows\system32\cmd.exe
Datos del Insecto
Ingrese nombre del Insecto : Rin
Ingrese Especie : abeja
Ingrese peso en kg : 0.05
Ingrese numero de jaula : 175
El insecto vuela(si/no) : si
```



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
Soy un Insecto  
Soy Rin  
De la especie abeja  
Peso 0.05 Kg  
Estoy en la jaula # 175  
Si Uuela
```

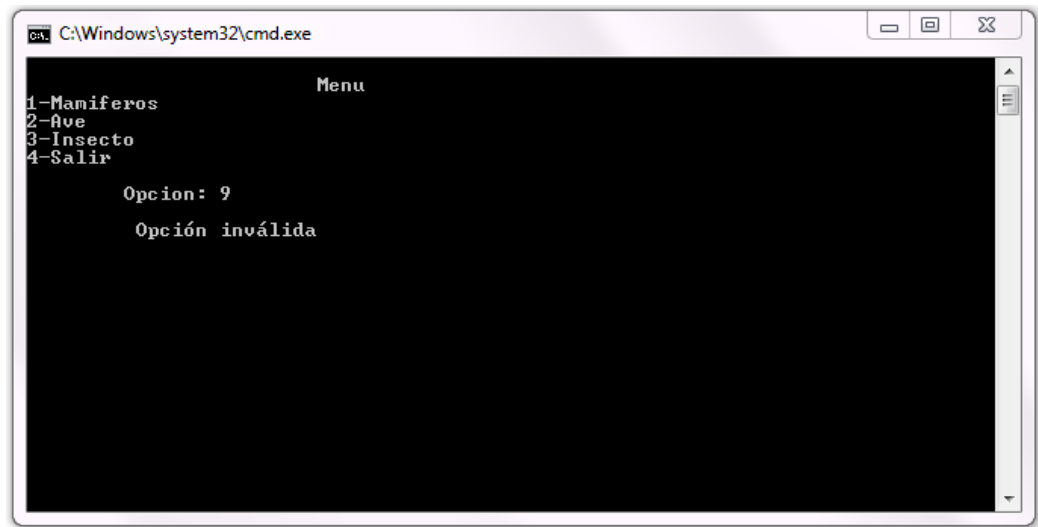
opción 4:



A screenshot of a Windows command prompt window. The title bar shows the path 'C:\Windows\system32\cmd.exe'. The window contains the following text:

```
Menu  
1-Maniferos  
2-Ave  
3-Insecto  
4-Salir  
  
Opcion: 4  
Salir del Menú
```

opción inválida:



```
C:\Windows\system32\cmd.exe

Menu
1-Maniferos
2-Ave
3-Insecto
4-Salir

Opcion: 9
Opción inválida
```

**Sugerencias
didácticas**

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.

Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML.

Identificar los miembros heredados, definir el constructor por defecto y los constructores sobrecargados.

Redefine el método virtual heredado.

Utiliza la sustitución de métodos override.

Nombrar otros ejemplos de clases para identificar sus miembros.

Realizar otros ejercicios para practicar a partir del diagrama de clases.

**Reporte del
Alumno**

El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.

Bibliografía

Bell, D. and Parr, M. (2011). *C# para estudiantes*. Pearson Educación de México, S.A. de C.V.

Ceballos Sierra, F. (2012). *Microsoft C#*. México, D.F.: Alfaomega.

Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). *La Biblia de C#*. Anaya.

Gonzalez Seco, J. (2012). *El lenguaje de programación C#*.

Larman, C. (2003). *UML y patrones*. Naucalpan de Juárez: Pearson Educación.

Msdn.microsoft.com. (2018). *Objetos, clases y estructuras (C#)*. [online]
Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	ONCE
Nombre	Definir una clase abstracta
Competencia específica de la unidad 4	Implementar interfaces y clases polimórficas
Introducción	Se necesita un mecanismo que haga imposible la creación de instancias de estas clases base para ello se debe marcar la clase base como abstracta utilizando la palabra reservada <i>abstract</i>. La clase base existe únicamente para introducir el nombre del método en la jerarquía. Las clases abstractas se emplean para proporcionar implementaciones parciales de clases que se completan con clases derivadas concretas. En esta práctica se definirá una clase abstracta, veremos cómo heredar de ella y que miembros puede contener.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 4. Polimorfismo 4.2 Clases abstractas: definición, métodos abstractos, implementación de clases abstractas, modelado de clases abstractas.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una persona utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase persona representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

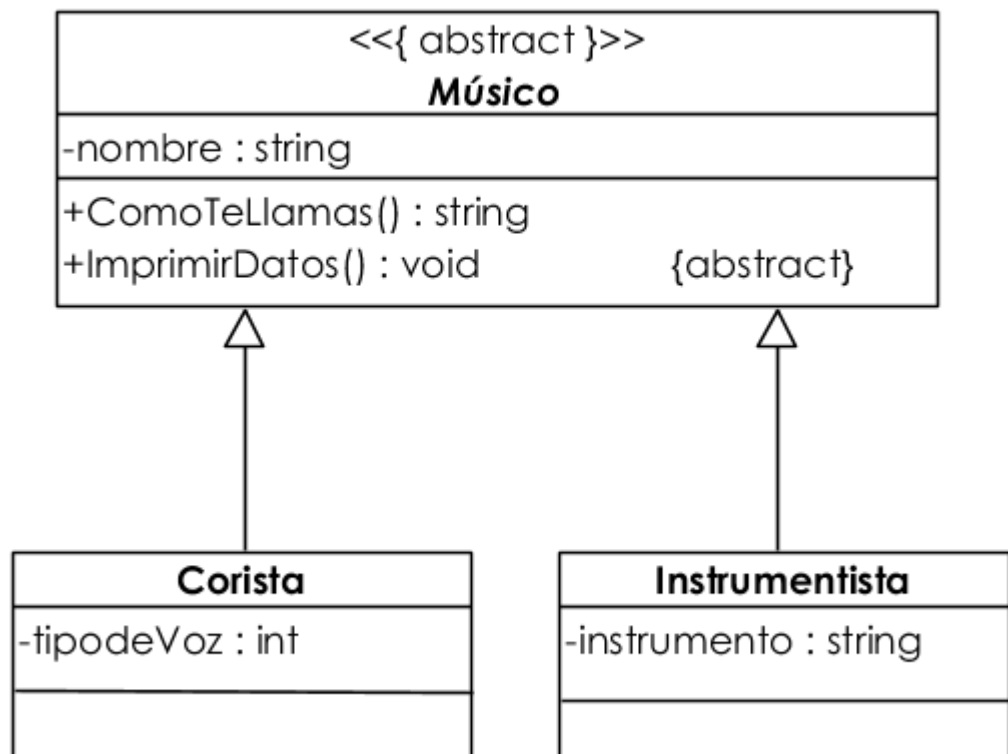
I. Definición : Define una clase abstracta llamada **Músico**, con el atributo nombre, un método que llamado **ComoTeLlamas** que regrese el nombre del músico y un método abstracto llamado **ImprimirDatos** que me permita mostrar todos los datos de cada músico.

Hereda de esta clase a una clase llamada **Corista** que serán los músicos del coro en el cual identificaran su tipo de voz de acuerdo a los siguientes códigos de **TipodeVoz** : 0.**soprano**; 1.**contralto**; 2.**tenor**; 3.**bajo**. Otra clase **Instrumentista** que tendrá a los músicos indicando que instrumento tocan. Crea todos los objetos que necesites para mostrar los tipos de músicos.

a) Datos :

- nombre
- tipo de voz
- instrumento
- ComoTeLlamas
- ImprimirDatos

b) Diagrama de clases



c) Código C# en consola

CLASE BASE MUSICO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Abstracta
{
    public abstract class Musico
    {
        string nombre;
        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
        public Musico() { }
        public Musico(string name) { nombre = name; }
        public string ComoTeLLamas()
        { return nombre; }
        public abstract void imprimirDatos();
    }
}
```

CLASE DERIVADA CORISTA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Abstracta
{
    public class Corista : Musico
    {
        int tipodeVoz;
        public int TipodeVoz
        {
            get { return tipodeVoz; }
            set { tipodeVoz = value; }
        }
        public Corista(): base()
        { }
        public Corista(string nom, int tipo): base(nom)
        { tipodeVoz = tipo; }
        public override void imprimirDatos()
        {
            string voz = "";

```

```

        Console.WriteLine("{0} Es miembro del Coro
", ComoTeLLamas());
        if (TipodeVoz == 0)
        { voz = "soprano"; }
        else if (TipodeVoz == 1)
        { voz = "contralto"; }
        else if (TipodeVoz == 2)
        { voz = "tenor"; }
        else if (TipodeVoz == 3)
        { voz = "bajo"; }
        Console.WriteLine("y canta en la cuerda de {0}", voz);
    }

}
}

```

CLASE DERIVADA INSTRUMENTISTA

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Abstracta
{
    public class Instrumentista: Musico
    {
        string instrumento;
        public string Instrumento
        {
            get { return instrumento; }
            set { instrumento = value; }
        }
        public Instrumentista(): base()
        { }
        public Instrumentista(string nom, string inst): base(nom)
        { instrumento=inst; }
        public override void imprimirDatos()
        {
            Console.WriteLine("{0} Es miembro de la Orquesta y
toca {1}", ComoTeLLamas(), Instrumento);
        }
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Abstracta

```

```

{
    class Program
    {

        public static void Main()
        {

            Corista a = new Corista("Juan Perez",3);
            Corista b = new Corista("Rocio Miro", 1);
            Corista c = new Corista("Alma Ros", 2);

            Instrumentista i = new Instrumentista("Mar
Cisneros","Flautin");
            Instrumentista j = new Instrumentista("Robert Mark",
"Trombon");
            Instrumentista k = new Instrumentista("Marco Miller",
"Guitarra");

            Console.WriteLine("\tLos Coristas son ");
            a.imprimirDatos();
            b.imprimirDatos();
            c.imprimirDatos();

            Console.WriteLine("\n\n\tInstrumentistas ");
            i.imprimirDatos();
            j.imprimirDatos();
            k.imprimirDatos();

        }
    }
}

```

d) Corrida o Pantalla de captura

```

C:\Windows\system32\cmd.exe
Los Coristas son
Juan Perez Es miembro del Coro y canta en la cuerda de bajo
Rocio Miro Es miembro del Coro y canta en la cuerda de contralto
Alma Ros Es miembro del Coro y canta en la cuerda de tenor

Instrumentistas
Mar Cisneros Es miembro de la Orquesta y toca Flautin
Robert Mark Es miembro de la Orquesta y toca Trombon
Marco Miller Es miembro de la Orquesta y toca Guitarra
Press any key to continue . . .

```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Identificar los miembros heredados, definir el constructor por defecto y los constructores sobrecargados. Redefine el método abstracto heredado. Utiliza la sustitución de métodos override. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i>. [online] Available at: https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx [Accessed 6 Nov. 2018]. Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	DOCE
Nombre	Utilizar una Excepción específica
Competencia específica de la unidad 5	Identificar, manejar, gestionar y crear las condiciones de error que interrumpan el flujo normal de ejecución de un programa.
Introducción	Las características de control de excepciones del lenguaje C# proporcionan una manera de afrontar cualquier situación inesperada o excepcional que se presente mientras se ejecuta un programa. En esta práctica se manejarán los tipos de cláusulas catch que se pueden utilizar como son: la cláusula catch específica, cláusula catch específica con identificador y cláusula catch general.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 5. Excepciones 5.1 Definición. 5.2 Tipos de Excepciones.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para resolver el problema que determina si un número es par o impar 2. Escribe el código utilizando los tipos de excepciones necesarias. 3. Compila el programa y corrige errores de sintaxis y lógica. 4 Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es : I. Definición : Captura un número entero y determina si es par o impar. Utiliza las excepciones necesarias para atrapar posibles errores y muestra el mensaje del error que corresponda.

-
- a) Datos :
- número 1
 - número 2

c) Código C# en consola

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Excepciones
{
    class Program
    {
        static void Main(string[] args)
        {
            bool ban = true;
            do
            {
                try
                {
                    int n1;
                    Console.Write("Ingrese un Numero : ");
                    n1 = int.Parse(Console.ReadLine());

                    if ((n1 % 2) == 0)
                    {
                        Console.WriteLine("Su Numero es Par");
                        ban = true;
                    }
                    else
                    {
                        Console.WriteLine("Su Numero es Impar");
                        ban = true;
                    }
                }

                // tipos de clausulas catch
                // clausula catch especifica con identificador
                catch (FormatException id)
                {
                    Console.WriteLine(id.Message);
                    ban = false;
                }

                // clausula catch especifica
                catch (OverflowException id)
                {

```

```

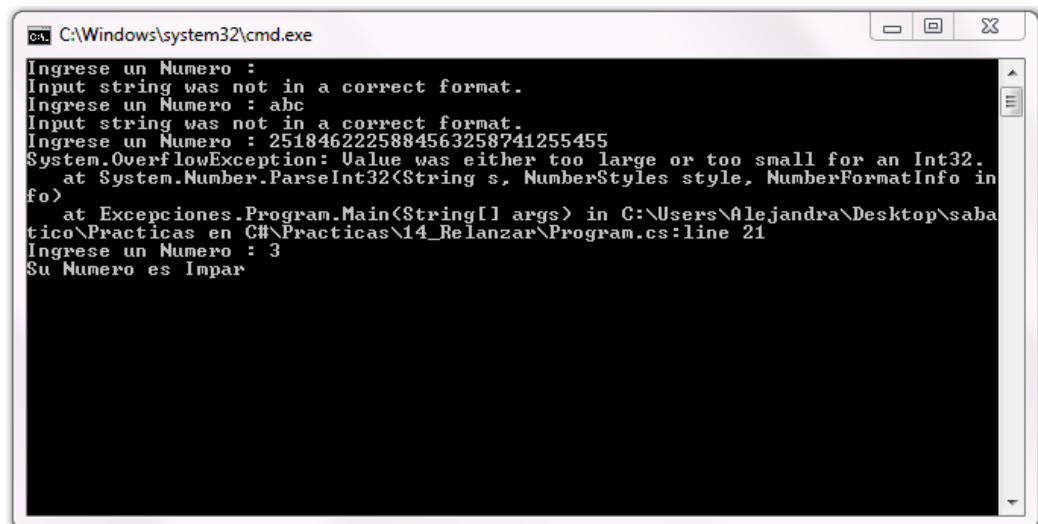
        Console.WriteLine(id);
        ban = false;
    }

    // clausula catch general
    catch
    {
        ban = false;
    }

    } while (ban == false); Console.ReadLine();
    }
}

```

d) Corrida o Pantalla de captura



Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Utiliza los tipos de excepciones mostrando el mensaje de error correspondiente para identificar la excepción que se ejecuto. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i>. [online] Available at: https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx [Accessed 6 Nov. 2018]. Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	TRECE
Nombre	Utilizar una Excepción específica
Competencia específica de la unidad 5	Identificar, manejar, gestionar y crear las condiciones de error que interrumpen el flujo normal de ejecución de un programa.
Introducción	En el lenguaje C#, los errores del programa se difunden en tiempo de ejecución a través del programa mediante un mecanismo denominado excepciones. Las excepciones se producen cuando el código encuentra un error y se detectan mediante el código que puede corregir el error.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 5. Excepciones 5.1 Definición. 5.2 Tipos de Excepciones.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para resolver el problema. 2. Escribe el código utilizando los tipos de excepciones necesarias. 3. Compila el programa y corrige errores de sintaxis y lógica. 4 Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es : I. Definición : Crea una tabla con 5 elementos numéricos, lee toda la tabla y después pregunta que elemento deseas mostrar. Utiliza las excepciones necesarias para atrapar posibles errores y muestra el mensaje del error que corresponda.

c) Código C# en consola

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ExcepcionEspecifica
{
    class Program
    {
        static void Main(string[] args)
        {
            int [] tabla=new int [5];

            try
            {
                for (int i = 0; i < tabla.Length; i++)
                {
                    Console.Write("Numero : ");
                    tabla[i] = int.Parse(Console.ReadLine());
                }
                Console.WriteLine("Toda la tabla");
                for (int i = 0; i < tabla.Length; i++)
                {
                    Console.WriteLine("Numero : " + tabla[i]);
                }

                Console.Write("Que elemento deseas ver de la
tabla : ");
                int p = int.Parse(Console.ReadLine());
                Console.WriteLine("El elemento es el :
"+tabla[p]);
            }

            catch (IndexOutOfRangeException e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

d) Corrida o Pantalla de captura



```
C:\Windows\system32\cmd.exe
Numero : 6
Numero : 5
Numero : 9
Numero : 2
Numero : 4
Toda la tabla
Numero : 6
Numero : 5
Numero : 9
Numero : 2
Numero : 4
Que elemento deseas ver de la tabla : 8
Index was outside the bounds of the array.
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Utiliza los tipos de excepciones mostrando el mensaje de error correspondiente para identificar la excepción que se ejecuto. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i> . Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i> . México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i> . Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i> . Larman, C. (2003). <i>UML y patrones</i> . Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i> . [online] Available at: https://msdn.microsoft.com/es-

es/library/ms173109(v=vs.80).aspx [Accessed 6 Nov. 2018].

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	CATORCE
Nombre	Relanzando tu propia excepción
Competencia específica de la unidad 5	Identificar, manejar, gestionar y crear las condiciones de error que interrumpen el flujo normal de ejecución de un programa.
Introducción	Las clases derivadas de <i>Exception</i> representan estas excepciones. Esta clase identifica el tipo de excepción y contiene las propiedades que albergan detalles acerca de ésta. Producir una excepción implica crear una instancia de una clase derivada de la excepción, configurar opcionalmente las propiedades de la excepción y, a continuación, iniciar el objeto con la palabra clave <i>throw</i> .
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 5. Excepciones 5.5 Creación y manejo de excepciones definidas por el usuario.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para resolver el problema creando tu propia excepción. 2. Escribe el código utilizando los tipos de excepciones necesarias. 3. Compila el programa y corrige errores de sintaxis y lógica. 4 Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es : I. Definición : Captura el saldo de tu tarjeta de credito, captura la cantidad a retirar, determina si es el saldo suficiente si no relanza la

excepción correspondiente con el mensaje de “Saldo Insuficiente”.

c) Código C# en consola

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace ExcepcionRelanzar
{
    class Program
    {
        static void Main(string[] args)
        {
            int saldo=0,cant=0;

            // agregar
            try
            {
                Console.Write("Introducir saldo : ");
                saldo = int.Parse(Console.ReadLine());

                Console.Write("Introducir cant. retirar : ");
                cant = int.Parse(Console.ReadLine());

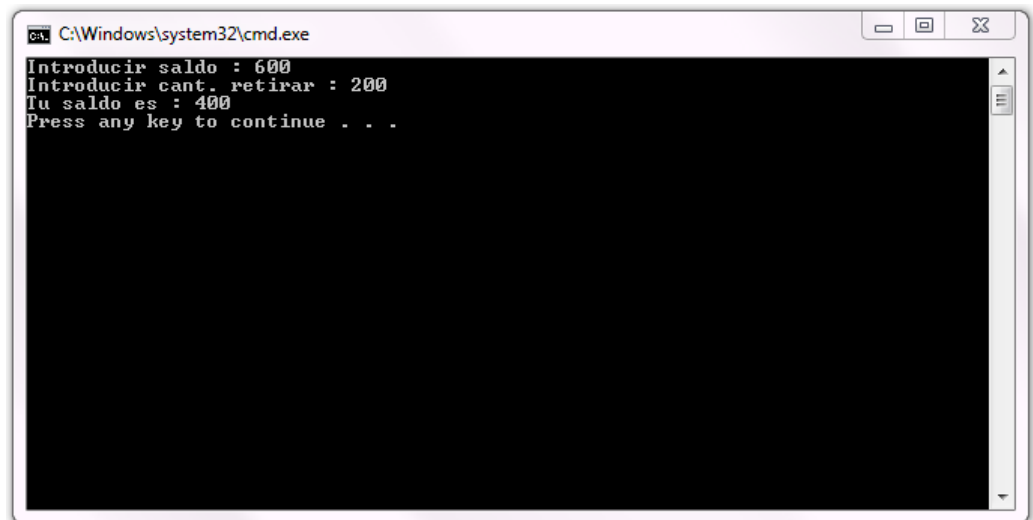
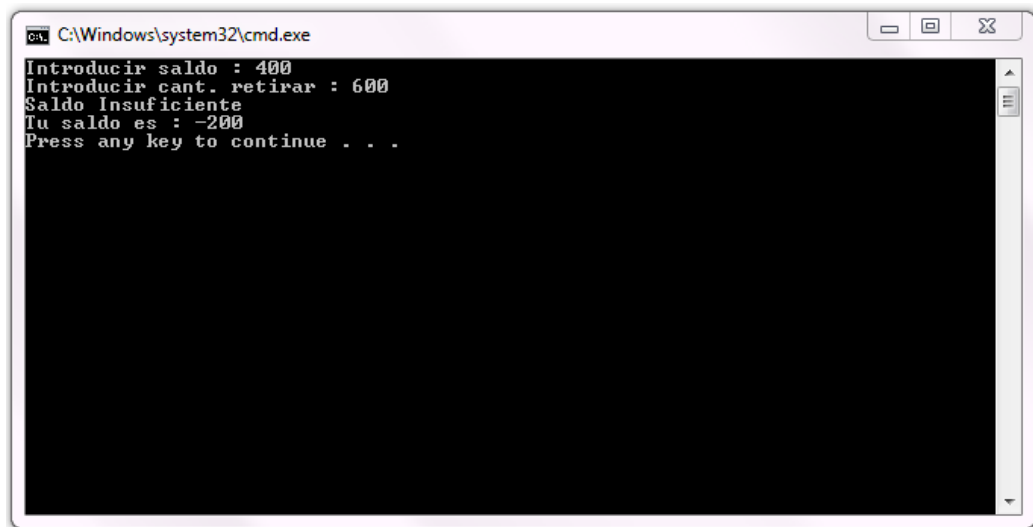
                if (cant > saldo)
                    throw new Exception("Saldo Insuficiente");
            }
            // atrapar excepciones
            catch (OverflowException)
            {
                Console.WriteLine("\n Error de desbordamiento\n");
            }
            catch (FormatException)
            {
                Console.WriteLine("Vuelve a Introducir el saldo " );
                saldo = int.Parse(Console.ReadLine());
            }

            catch (Exception e)
            {
                Console.WriteLine(e.Message);
            }

            // finalizar
            finally
            {
                if (saldo>0 && cant>0)
```

```
        { saldo = saldo - cant; }  
        else{Console.WriteLine("Hubo un error y NO  
pudiste capturar cant. a retirar ");}  
        }Console.WriteLine("Tu saldo es : {0} ", saldo);  
    }  
}  
}
```

d) Corrida o Pantalla de captura



Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Utiliza los tipos de excepciones mostrando el mensaje de error
-------------------------------	--

correspondiente para identificar la excepción que se ejecuto.
Utiliza la instrucción correspondiente para crear tu propia excepción y relanzarla por el método solicitante.
Nombrar otros ejemplos de clases para identificar sus miembros.
Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
---------------------------	--

Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Msdn.microsoft.com. (2018). <i>Objetos, clases y estructuras (C#)</i>. [online] Available at: https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx [Accessed 6 Nov. 2018]. Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.
---------------------	---

No. de Práctica	QUINCE
Nombre	Creando un archivo de texto
Competencia específica de la unidad 6	Implementar aplicaciones orientadas a objetos que creen y manipulen archivos para guardar y recuperar información.
Introducción	La necesidad de manipular información permite que se implementen aplicaciones para guardar y recuperar la información utilizando archivos, que en este caso almacenamos en formato de texto. En esta práctica se utilizarán las operaciones básicas para manipular un archivo de texto.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 6. Flujos y Archivos 6.2 Clasificación: Archivos de texto y binarios. 6.3 Operaciones básicas y tipos de acceso.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para manipular archivos de texto. 2. Identifica las clases para flujo de datos. 3. Escribe el código para ejecutar las operaciones básicas para manipular un archivo. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es : I. Definición : Elabora un programa utilizando archivos de texto, utiliza un menú para definir cada una de las operaciones básicas para archivos de texto como crear, agregar, leer y eliminar.

c) Código C# en consola

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.IO;

namespace ArchivoTexto
{
    class Program
    {
        static void Main(string[] args)
        {
            int opcMenu = 0;

            do
            {
                try
                {
                    Console.Write("\n\t\tMenú de Archivos de
Texto\n1.-Crear\n2.-Agregar\n3.-Leer\n4.-Eliminar\n5.-
Salir\n\n\tOpción: ");
                    opcMenu = int.Parse(Console.ReadLine());

                    switch (opcMenu)
                    {
                        case 1:
                            crear();
                            break;

                        case 2:
                            agregar();
                            break;

                        case 3:
                            leer();
                            break;

                        case 4:
                            eliminar();
                            break;

                        case 5:
                            Console.WriteLine("\n\n Salir del
Menú");
                            break;

                        default:
                            Console.WriteLine("\n\n opción
inválida");
                    }
                }
            }
        }
    }
}
```

```

        break;

    }
    Console.ReadKey();
    Console.Clear();
}

catch (FileNotFoundException e)
{
    Console.WriteLine("\n\t\tNo se encontró el
archivo: {0} ", e.FileName);
    Console.ReadKey();
    Console.Clear();
}

} while (opcMenu != 5);
}

public static void crear()
{
    FileStream objCrear = new
FileStream("ArchivoEjemplo.txt", FileMode.OpenOrCreate,
FileAccess.Write);
    StreamWriter objWriter = new StreamWriter(objCrear);
    objWriter.Write("Se ejecuto la opción 1 : crear
archivo, contenido inicial \n");
    Console.WriteLine("\n Se ha creado el archivo...");
    objWriter.Close();
    objCrear.Close();
}

public static void agregar()
{
    FileStream objAgregar = new
FileStream("ArchivoEjemplo.txt", FileMode.Append,
FileAccess.Write);
    StreamWriter objWriter = new
StreamWriter(objAgregar);
    Console.WriteLine("Escribe el contenido que desea
agregar al archivo");
    string texto = Console.ReadLine();
    objWriter.Write(texto);
    Console.WriteLine("\nSe ejecuto la opción 2: Agregar
texto al archivo");
    objWriter.Close();
    objAgregar.Close();
}

public static void leer()
{
    FileStream objLeer = new
FileStream("ArchivoEjemplo.txt", FileMode.Open,
FileAccess.Read);
    StreamReader objReader = new StreamReader(objLeer);
    Console.WriteLine("Contenido del archivo : \n");

```

```

        Console.WriteLine(objReader.ReadToEnd());
        objReader.Close();
        objLeer.Close();
    }

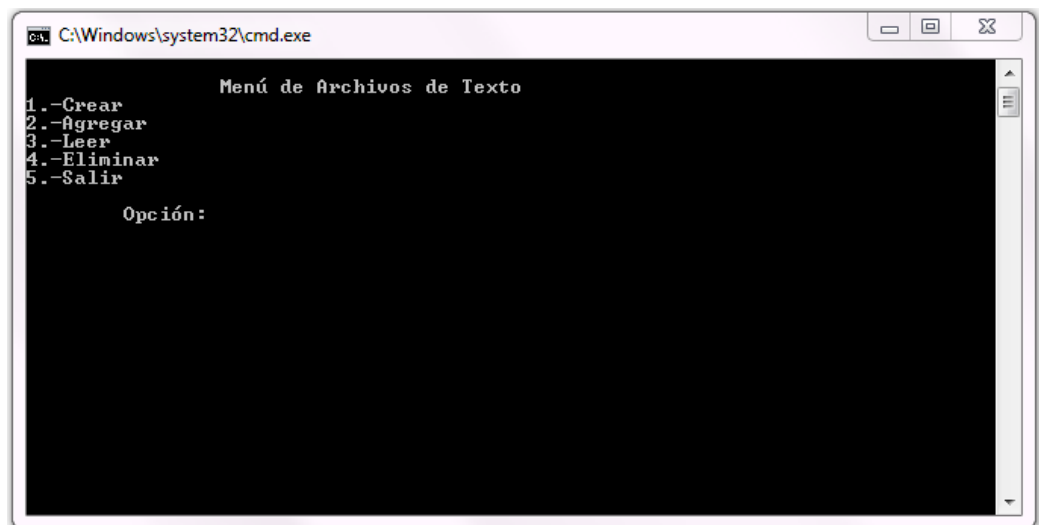
    public static void eliminar()
    {
        leer();
        Console.Write("\n\t¿Desea eliminar este archivo?
(S/N) : ");
        string charSN = Console.ReadLine();
        if (charSN.ToLower() == "s")
        {
            File.Delete("ArchivoEjemplo.txt");
            Console.WriteLine("\nArchivo Eliminado.");
        }

        else
        {
            Console.WriteLine("\nArchivo no Eliminado.");
        }
    }
}
}

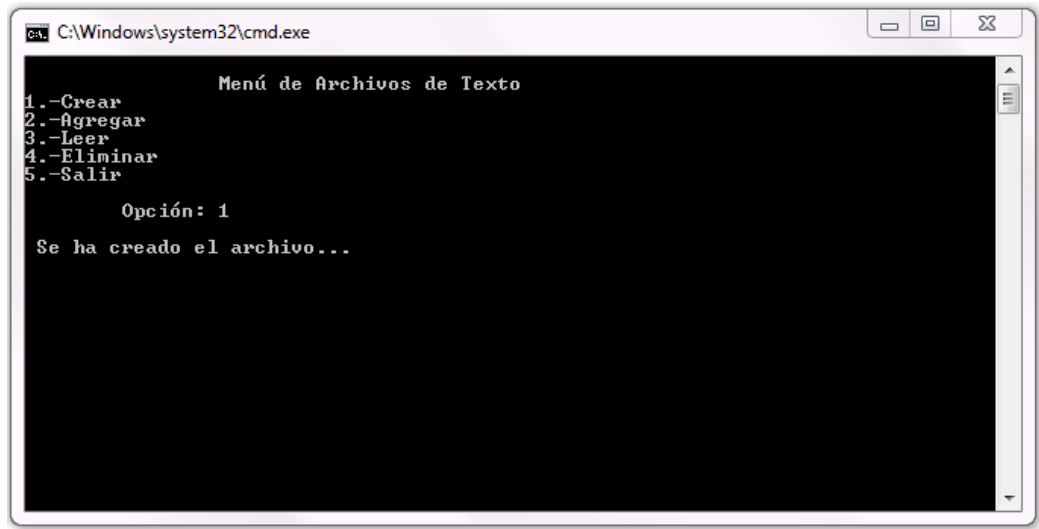
```

d) Corrida o Pantalla de captura

Menú



opción 1 :

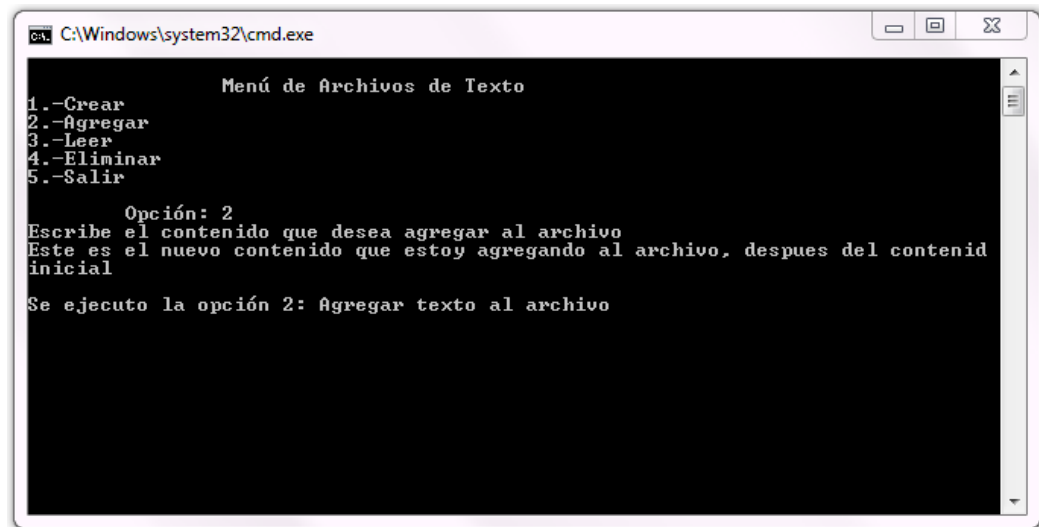


```
C:\Windows\system32\cmd.exe

Menú de Archivos de Texto
1.-Crear
2.-Agregar
3.-Leer
4.-Eliminar
5.-Salir

Opción: 1
Se ha creado el archivo...
```

opción 2 :

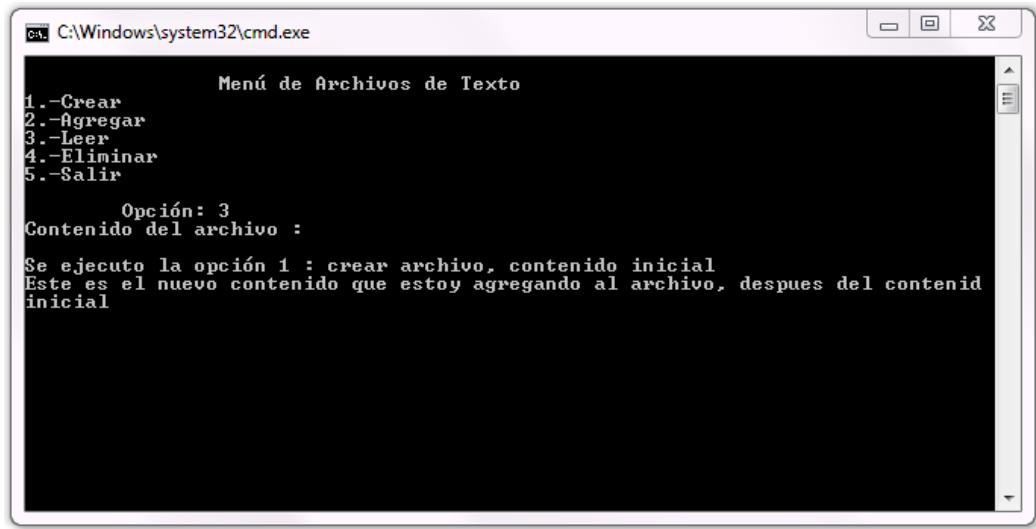


```
C:\Windows\system32\cmd.exe

Menú de Archivos de Texto
1.-Crear
2.-Agregar
3.-Leer
4.-Eliminar
5.-Salir

Opción: 2
Escribe el contenido que desea agregar al archivo
Este es el nuevo contenido que estoy agregando al archivo, despues del contenid
inicial
Se ejecuto la opción 2: Agregar texto al archivo
```

opción 3 :



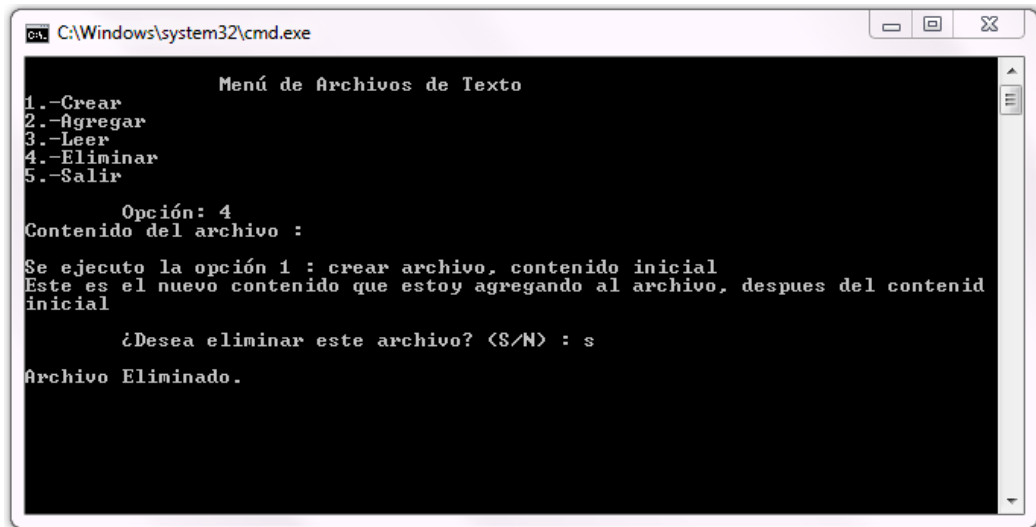
```
C:\Windows\system32\cmd.exe

Menú de Archivos de Texto
1.-Crear
2.-Agregar
3.-Leer
4.-Eliminar
5.-Salir

Opción: 3
Contenido del archivo :

Se ejecuto la opción 1 : crear archivo, contenido inicial
Este es el nuevo contenido que estoy agregando al archivo, despues del contenid
inicial
```

opción 4 :



```
C:\Windows\system32\cmd.exe

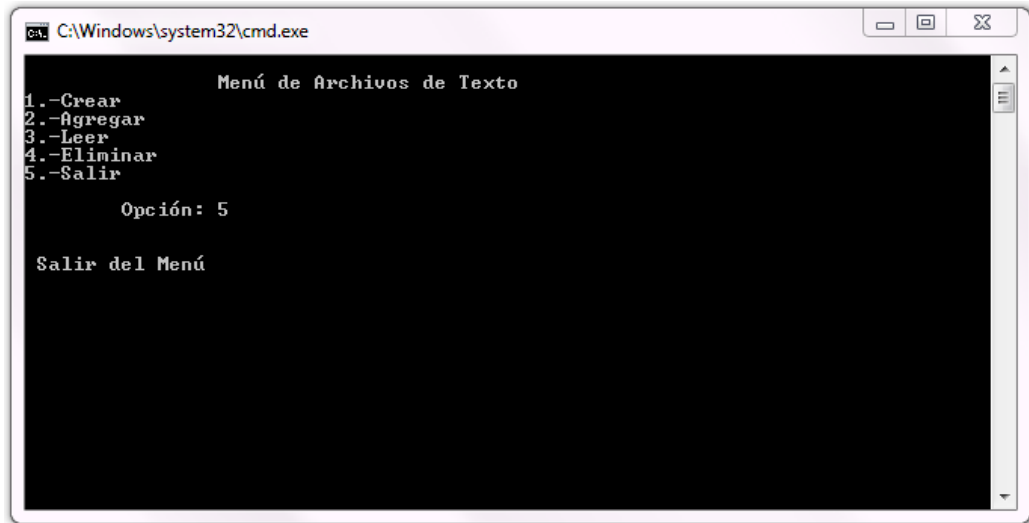
Menú de Archivos de Texto
1.-Crear
2.-Agregar
3.-Leer
4.-Eliminar
5.-Salir

Opción: 4
Contenido del archivo :

Se ejecuto la opción 1 : crear archivo, contenido inicial
Este es el nuevo contenido que estoy agregando al archivo, despues del contenid
inicial

¿Desea eliminar este archivo? (S/N) : s
Archivo Eliminado.
```

opción 5 :



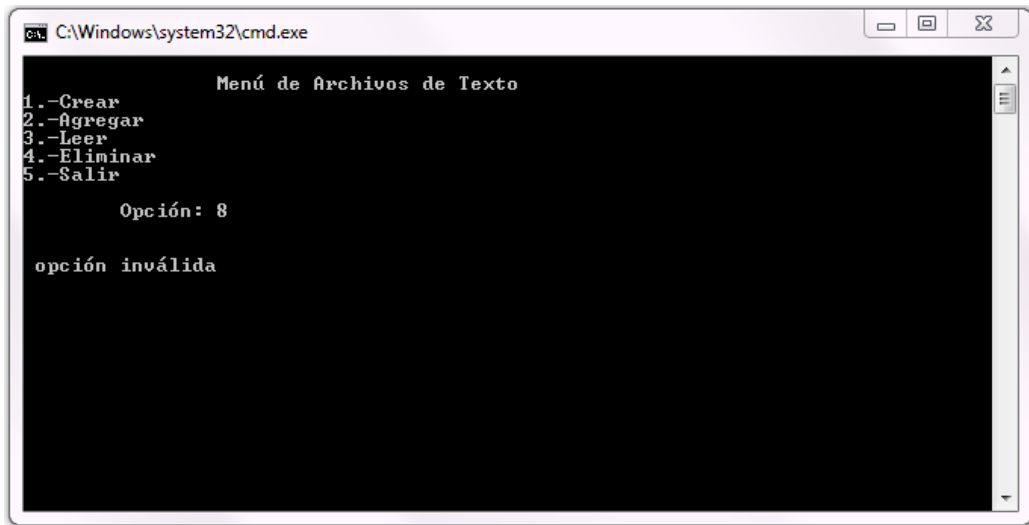
```
C:\Windows\system32\cmd.exe

Menú de Archivos de Texto
1.-Crear
2.-Agregar
3.-Leer
4.-Eliminar
5.-Salir

Opción: 5

Salir del Menú
```

opción inválida :



```
C:\Windows\system32\cmd.exe

Menú de Archivos de Texto
1.-Crear
2.-Agregar
3.-Leer
4.-Eliminar
5.-Salir

Opción: 8

opción inválida
```

**Sugerencias
didácticas**

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.
Utilizar la librería para archivos *System.IO*
Nombrar otros ejemplos de clases para identificar sus miembros.
Realizar otros ejercicios para practicar a partir del diagrama de clases.

**Reporte del
Alumno**

El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.

Bibliografía

- Bell, D. and Parr, M. (2011). *C# para estudiantes*. Pearson Educación de México, S.A. de C.V.
- Ceballos Sierra, F. (2012). *Microsoft C#*. México, D.F.: Alfaomega.
- Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). *La Biblia de C#*. Anaya.
- Gonzalez Seco, J. (2012). *El lenguaje de programación C#*.
- Larman, C. (2003). *UML y patrones*. Naucalpan de Juárez: Pearson Educación.
- Msdn.microsoft.com. (2018). *Objetos, clases y estructuras (C#)*. [online] Available at: [https://msdn.microsoft.com/es-es/library/ms173109\(v=vs.80\).aspx](https://msdn.microsoft.com/es-es/library/ms173109(v=vs.80).aspx) [Accessed 6 Nov. 2018].
- Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.
- Walker, L. (2018). *CSharp, Courseware Development Tools*.
-

CONTENIDO DE PRÁCTICAS PARA PROGRAMACIÓN ORIENTADA A OBJETOS EN ING. EN TECNOLOGÍAS DE LA INFORMACIÓN Y COMUNICACIONES

Nombre de la asignatura: **Programación Orientada a Objetos**
Carrera: **Ingeniería Informática e Ingeniería en Tecnologías de la Información y Comunicaciones**
Clave de la asignatura: **AEB-1054**
(Créditos) SATCA1 **1 - 4 - 5**

Objetivo General:

Aplica el paradigma orientado a objetos para el desarrollo de aplicaciones que solucionen problemas del entorno.

Prácticas

Núm.	Nombre de la Práctica	Objetivo Específico	Temas con los que se relacionan
1	Define una clase Persona	Identificar los componentes de una clase	Unidad 2. Clases y Objetos 2.2 Declaración de clases 2.3 Miembros de una clase
2	Define una clase con todos sus miembros	Definir una clase y sus miembros	Unidad 2. Clases y Objetos 2.2 Declaración de clases 2.3 Miembros de una clase
3	Define una clase definiendo sus especificadores de acceso	Identificar los especificadores de acceso a nivel de clase y sus miembros	Unidad 2. Clases y Objetos 2.5 Especificadores de acceso
4	Crea un objeto inicializándolo con el constructor por defecto	Definir una clase y crear un objeto de la misma y ejecutar los métodos	Unidad 2. Clases y Objetos 2.6 Creación de objetos
5	Crea un objeto utilizando el constructor sobrecargado	Definir una clase y crear un objeto de la misma y ejecutar los métodos	Unidad 2. Clases y Objetos 2.6 Creación de objetos

6	Elabora un programa utilizando manejo de excepciones	Realizar un ejercicio utilizando las instrucciones para manejo de excepciones como try, catch y finally.	Unidad 2. Clases y Objetos 2.9 Manejo de excepciones.
7	Elabora un programa donde relances tu propia excepción	Realizar un ejercicio utilizando las instrucción para crear su propia excepción	Unidad 2. Clases y Objetos 2.9 Manejo de excepciones.
8	Define una clase con sus métodos	Definir la estructura de un método	Unidad 3. Métodos 3.1 Definición de un método. 3.2 Estructura de un método.
9	Define una clase donde defina un método que regrese valor	Definir la estructura de un método que regresa valor	Unidad 3. Métodos 3.1 Definición de un método. 3.2 Estructura de un método. 3.3 Valor de retorno.
10	Crear un programa que sobrecargue un método	Definir y observar la sobrecarga de métodos	Unidad 3. Métodos 3.8 Sobrecarga de métodos.
11	Elabora un ejercicio donde muestre la herencia entre las clases Persona y Estudiante	Aplicar la reutilización de código para el diseño de clases nuevas a partir de clases ya probadas	Unidad 4. Herencia y Polimorfismo 4.2 Definición de una clase base. 4.3 Definición de una clase derivada.
12	Elabora un programa donde implementes de manera implícita una interface	Implementar polimorfismo con el uso de interfaces implícitas	Unidad 4. Herencia y Polimorfismo 4.5 Interfaces. 4.5.1 Definición 4.5.2 Implementación
13	Elabora un programa donde implementes de manera explícita una interface	Implementar polimorfismo con el uso de interfaces explícitas	Unidad 4. Herencia y Polimorfismo 4.5 Interfaces. 4.5.1 Definición 4.5.2 Implementación
14	Elabora programa que capture 10 estaturas y las convierta a cm	Realizar un ejercicio de inicialización de un arreglo y la determinación del número de elementos del arreglo.	Unidad 5. Arreglos 5.1 Unidimensional.
15	Elabora un programa que capture 5 objetos tipo Estudiante y los almacene en un arreglo	Realizar un ejercicio de arreglos para almacenar objetos	Unidad 5. Arreglos 5.1 Unidimensional.

Prácticas

No. de Práctica	UNO
Nombre	Definir una clase Persona
Competencia específica de la unidad 2	Comprende y aplica la estructura de clases para la creación de objetos y utiliza clases predefinidas para facilitar el desarrollo de aplicaciones.
Introducción	La Programación Orientada a Objetos es un modelo de programación que utiliza clases y objetos para la resolución de problemas. En esta práctica identificaremos las entidades de las cuales se pueden crear objetos potenciales que poseen un conjunto de atributos y métodos, útiles para resolver el problema.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente	Unidad 2. Clases y Objetos 2.2 Declaración de clases 2.3 Miembros de una clase
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una persona utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase persona representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Determina los datos necesarios de una persona que puede cambiar su nacionalidad y determinar su estatura en centímetros, identifica los miembros de la clase persona.

a) Datos :

- nombre
- edad
- domicilio
- nacionalidad
- estatura
- Cambiar Nacionalidad
- Convertir estatura a cm

b) Diagrama de Clases

Persona
- nombre : string - edad: int - domicilio: string - nacionalidad : string - estatura : float
+ CambiarNacionalidad(): void + ConvertirEstaturaCM(): float

c) Código C# en consola

CLASE PERSONA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Persona
{
    class Persona
    {
```

```
// atributos
string nombre, domicilio, nacionalidad;
int edad;
float estatura;

// propiedades
public string Nombre
{
    get { return nombre; }
    set { nombre = value; }
}

public string Domicilio
{
    get { return domicilio; }
    set { domicilio = value; }
}

public string Nacionalidad
{
    get { return nacionalidad; }
    set { nacionalidad = value; }
}

public int Edad
{
    get { return edad; }
    set { edad = value; }
}

public float Estatura
{
    get { return estatura; }
    set { estatura = value; }
}

// Constructores
public Persona() { }
public Persona(string n, string d, string nac, int e,
float es)
{
    nombre = n;
    domicilio=d;
    nacionalidad=nac;
    estatura=es;
    edad = e;
}

// métodos
public void CambiarNacionalidad()
{
    Console.WriteLine("Nueva Nacionalidad : ");
    nacionalidad = Console.ReadLine();
}
```

```

        public float ConvertirEstaturaCM()
        {
            return estatura * 100;
        }
    }
}

```

CLASE PRINCIPAL

```

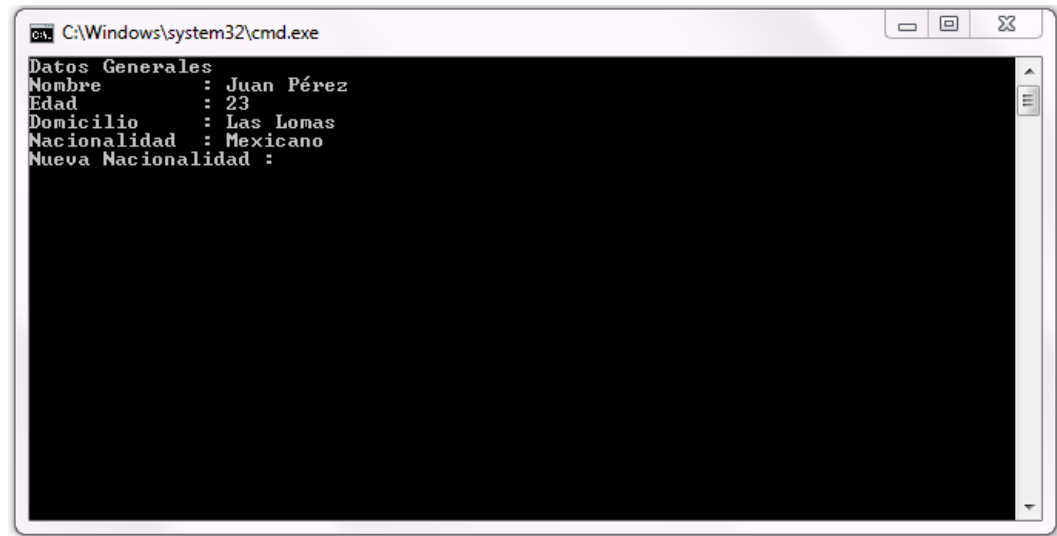
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Persona
{
    class Program
    {
        static void Main(string[] args)
        {
            Persona Juan = new Persona("Juan Pérez", "Las
Lomas", "Mexicano", 23, 1.74f);

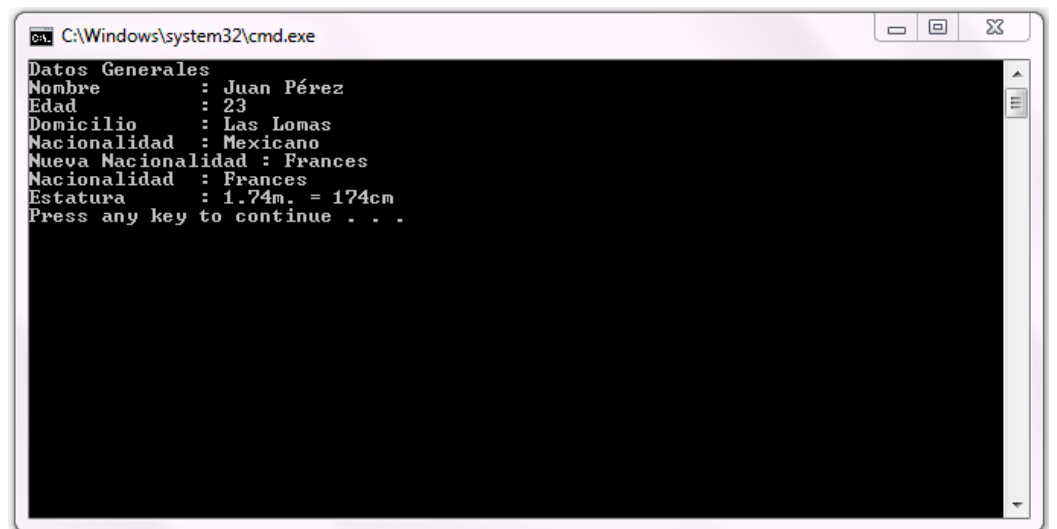
            Console.WriteLine("Datos Generales");
            Console.WriteLine("Nombre          : {0} ",
Juan.Nombre);
            Console.WriteLine("Edad          : {0} ", Juan.Edad);
            Console.WriteLine("Domicilio       : {0} ",
Juan.Domicilio);
            Console.WriteLine("Nacionalidad   : {0} ",
Juan.Nacionalidad);
            Juan.CambiarNacionalidad();
            Console.WriteLine("Nacionalidad   : {0} ",
Juan.Nacionalidad);
            Console.WriteLine("Estatura       : {0}m. = {1}cm ",
Juan.Estatura, Juan.ConvertirEstaturaCM());
        }
    }
}

```

d) Corrida o Pantalla de captura



```
C:\Windows\system32\cmd.exe
Datos Generales
Nombre : Juan Pérez
Edad : 23
Domicilio : Las Lomas
Nacionalidad : Mexicano
Nueva Nacionalidad :
```



```
C:\Windows\system32\cmd.exe
Datos Generales
Nombre : Juan Pérez
Edad : 23
Domicilio : Las Lomas
Nacionalidad : Mexicano
Nueva Nacionalidad : Frances
Nacionalidad : Frances
Estatura : 1.74m. = 174cm
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
-------------------------------	---

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
---------------------------	--

-
- Bibliografía**
- Bell, D. and Parr, M. (2011). *C# para estudiantes*. Pearson Educación de México, S.A. de C.V.
- Ceballos Sierra, F. (2012). *Microsoft C#, Curso de Programación*. México, D.F.: Alfaomega Ra-Ma.
- Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. *La Biblia de C#*. Anaya.
- Gonzalez Seco, J. (2012). *El lenguaje de programación C#*.
- Larman, C. (2003). *UML y patrones*. Naucalpan de Juárez: Pearson Educación.
- Docs.microsoft.com. (2019). *Clases y Objetos (C#)*. [online] Available at: <https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/classes-and-objects>.
- Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.
- Walker, L. *CSharp, Courseware Development Tools*.
-

No. de Práctica	DOS
Nombre	Define una clase con todos sus miembros
Competencia específica de la unidad 2	Comprende y aplica la estructura de clases para la creación de objetos y utiliza clases predefinidas para facilitar el desarrollo de aplicaciones.
Introducción	En esta práctica identificaremos la entidad que define los datos necesarios para resolver el problema. Es importante identificar los miembros de la clase e identificarlos en el diagrama de clases de UML. Después se instanciará un objeto del tipo de la clase. Una clase equivale a la generalización de un tipo específico de objetos, pero cada objeto que construyamos de esa clase tendrá sus propios datos. Un objeto de determinada clase se crea en el momento en que se define una variable de dicha clase.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.2 Declaración de clases 2.3 Miembros de una clase
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan una clase Empleado utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase Empleado representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

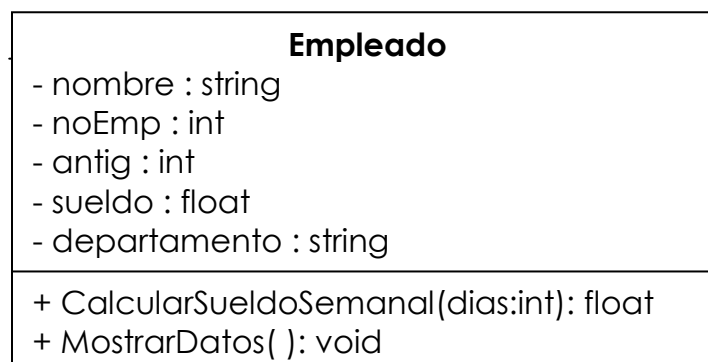
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Identifica los datos que definen a un Empleado, agrúpalos en una clase de ese tipo

a) Datos :

- nombre
- número de empleado
- antigüedad en trabajo
- sueldo diario
- departamento
- Calcular el Sueldo Semanal

b) Diagrama de clases



c) Código C# en consola

CLASE EMPLEADO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Empleado
{
    class Empleado
    {
        string nombre, departamento;
```

```

int noEmp, antig;
float sueldo;

// propiedades
public string Nombre
{
    get { return nombre; }
    set { nombre = value; }
}
public string Depto
{
    get { return departamento; }
    set { departamento = value; }
}

public int NoEmp
{
    get { return noEmp; }
    set { noEmp = value; }
}

public int Antig
{
    get { return antig; }
    set { antig = value; }
}
public float Sueldo
{
    get { return sueldo; }
    set { sueldo = value; }
}

// Métodos constructores
public Empleado() { }
public Empleado(string n, string d, int e, int a, float
s)
{
    nombre = n;
    departamento = d;
    noEmp = e;
    antig = a;
    sueldo = s;
}

// Métodos
public float CalcularSueldoSemanal(int dias)
{
    return sueldo * dias;
}

public void MostrarDatos()
{
    Console.WriteLine("Datos del Empleado");
    Console.WriteLine("No. Emp : {0} Empleado : {1} ",
noEmp, nombre);
    Console.WriteLine("Departamento      : {0} ",

```

```

departamento);
        Console.WriteLine("Años de Antigüedad : {0}", antig);
        Console.WriteLine("Sueldo Diario      : {0:c2}",
sueldo);
    }

}
}

```

CLASE PRINCIPAL

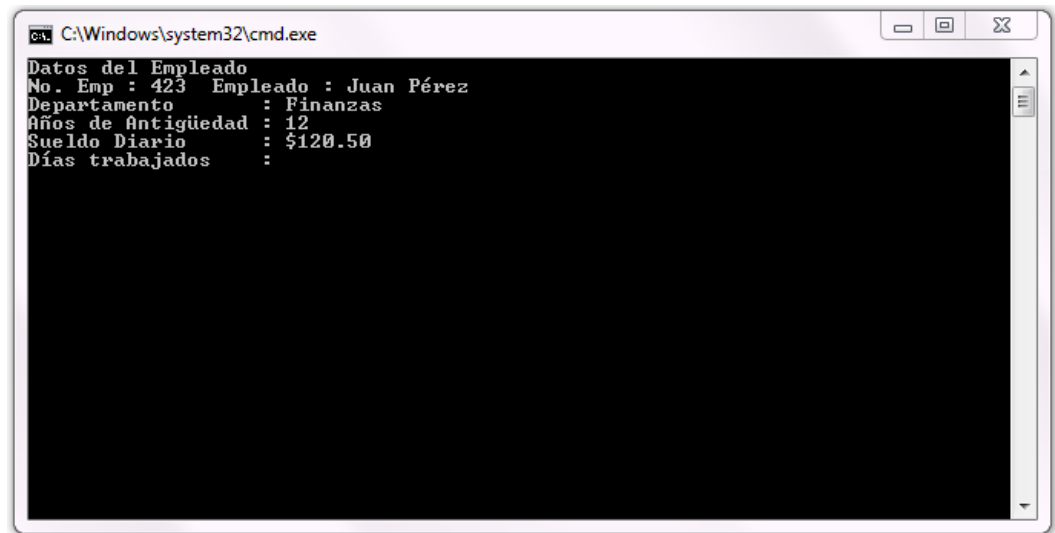
```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Empleado
{
    class Program
    {
        static void Main(string[] args)
        {
            Empleado Juan = new Empleado("Juan Pérez",
"Finanzas", 423, 12, 120.5f);
            Juan.MostrarDatos();
            Console.Write("Días trabajados : ");
            int dias=int.Parse(Console.ReadLine());
            Console.WriteLine("Sueldo Semanal      : {0:c2}",
Juan.CalcularSueldoSemanal(dias));
        }
    }
}

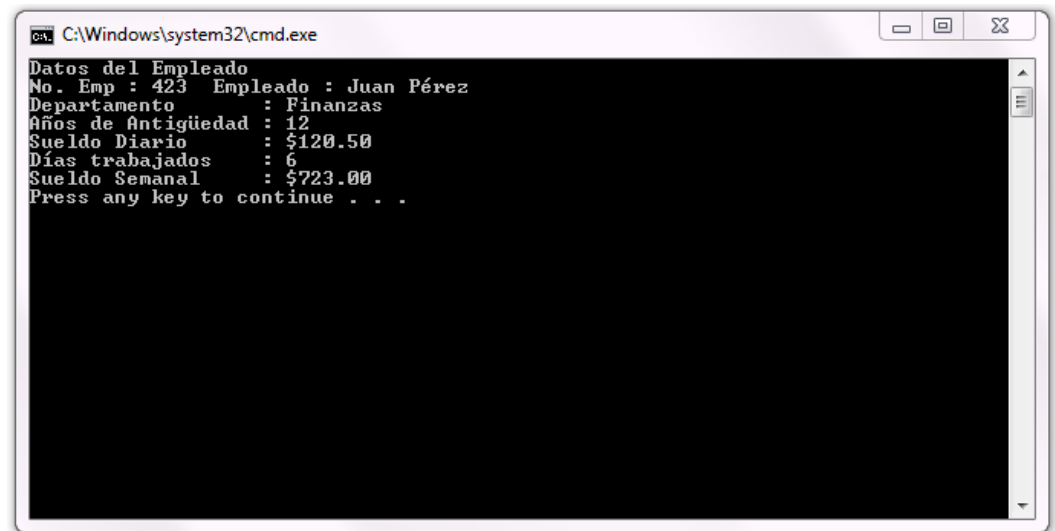
```

d) Corrida o Pantalla de captura



```
C:\Windows\system32\cmd.exe

Datos del Empleado
No. Emp : 423 Empleado : Juan Pérez
Departamento : Finanzas
Años de Antigüedad : 12
Sueldo Diario : $120.50
Días trabajados :
```



```
C:\Windows\system32\cmd.exe

Datos del Empleado
No. Emp : 423 Empleado : Juan Pérez
Departamento : Finanzas
Años de Antigüedad : 12
Sueldo Diario : $120.50
Días trabajados : 6
Sueldo Semanal : $723.00
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Instanciar un objeto del tipo de la clase. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
-------------------------------	--

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y Objetos (C#)</i>. [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/classes-and-objects. Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

TRES

No. de
Práctica

Nombre

Define una clase definiendo sus especificadores de acceso

Competencia específica de la unidad 2 Comprende y aplica la estructura de clases para la creación de objetos y utiliza clases predefinidas para facilitar el desarrollo de aplicaciones.

Introducción Todos los atributos y métodos de una clase son siempre accesibles para el código de la propia clase. Para controlar el acceso desde otras clases y para controlar la herencia por las subclases, los miembros de las clases tienen cuatro modificadores:

- * **public** : los miembros declarados public son accesibles en cualquier lugar en que sea accesible la clase y son heredados por las subclases.
- * **private** : los miembros declarados private son accesibles sólo en la propia clase.
- * **protected** : los miembros declarados protected son accesibles para las subclases y heredados por ellas y son accesibles por el código del mismo paquete.
- * **paquete**: los miembros declarados sin modificador de acceso son accesibles sólo por el código y heredados sólo por las subclases del mismo paquete.

Los modificadores de visibilidad se indican en los diagramas de clase son los símbolos o iconos que se describen a continuación :

Para atributos :

- **public** (+, ): Indica que el atributo será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
- **private** (-, ): Indica que el atributo sólo será accesible desde dentro de la clase (sólo sus métodos lo pueden acceder).

Para Métodos :

- **public** (+, ): Indica que el método será visible tanto dentro como fuera de la clase, es decir, es accesible desde todos lados.
- **private** (-, ): Indica que el método sólo será accesible desde dentro de la clase (sólo otros métodos de la clase lo pueden acceder).

En resumen podemos decir que para mantener el control de visibilidad :

- Los métodos son **públicos**, accesibles desde el exterior
- Los datos son **privados**, accesibles sólo desde el interior

Especificar la correlación con los temas y subtemas del Programa de estudio vigente.

Unidad 2. Clases y Objetos
2.5 Especificadores de acceso

Material y equipo necesario

- Cuaderno y Lápiz
- Computadora personal ya sea de escritorio o laptop
- Lenguaje de Programación instalado de C# versión 3.0 o posterior
- Editor de código Visual Studio 2010 o posterior
- Apuntes de la clase

Metodología

1. Analiza el problema e identifica los elementos necesarios que definan una clase Alumno utilizando el mecanismo de abstracción.
2. Clasifica los datos y define la clase Alumno representándola a través del diagrama de clases en UML.
3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. Especifica el modificador de acceso que se presenta en el diagrama de clases.
4. Compila el programa y corrige errores de sintaxis y lógica.
5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Determina los especificadores de acceso adecuados a cada miembro de la clase Alumno

a) Datos :

- nombre
 - matricula
 - semestre
 - carrera
 - Mostrar los Datos del alumno
-

b) Diagrama de clases



c) Código C# en consola

CLASE ALUMNO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Alumno
{
    class Alumno
    {
        // atributos con el modificador de acceso privado
        private string nombre, carrera;
        private long matricula;
        private int semestre;

        // propiedades con el modificador de acceso público
        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
        public string Carrera
        {
            get { return carrera; }
            set { carrera = value; }
        }
        public long Matricula
        {
            get { return matricula; }
            set { matricula = value; }
        }
        public int Semestre
        {
            get { return semestre; }
        }
    }
}
```

```

        set { semestre = value; }
    }

    // Constructor por defeto
    public Alumno() { }

    // Constructor sobrecargado
    public Alumno(string n, string c, long m, int s)
    {
        nombre = n;
        carrera = c;
        matricula = m;
        semestre = s;
    }

    // método con el modificador de acceso público
    public void MostrarDatos()
    {
        Console.WriteLine("\n\tDatos del Alumno");
        Console.WriteLine("Matricula : {0} {1}", matricula,
nombre);
        Console.WriteLine("Carrera : {0} - Semestre {1}",
carrera, semestre);
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

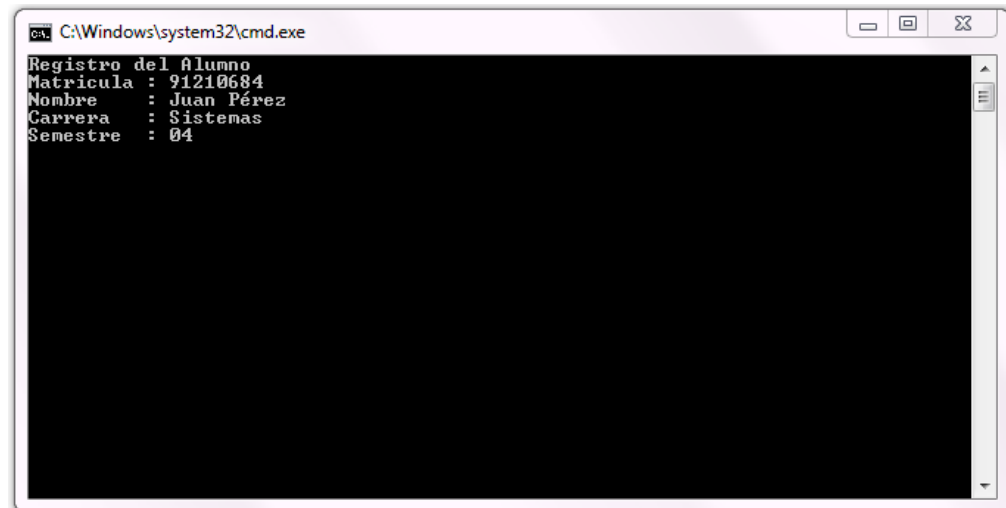
namespace Alumno
{
    class Program
    {
        static void Main(string[] args)
        {
            Console.WriteLine("Registro del Alumno");
            Alumno Juan = new Alumno();
            Console.Write("Matricula : ");
            Juan.Matricula = int.Parse(Console.ReadLine());
            Console.Write("Nombre : ");
            Juan.Nombre = Console.ReadLine();
            Console.Write("Carrera : ");
            Juan.Carrera = Console.ReadLine();
            Console.Write("Semestre : ");
            Juan.Semestre = int.Parse(Console.ReadLine());
            Juan.MostrarDatos();

            Console.WriteLine("\n\tOtro Alumno");
            Alumno Pedro = new Alumno("Pedro Juárez", "Sistemas",
423, 03);
            Pedro.MostrarDatos();

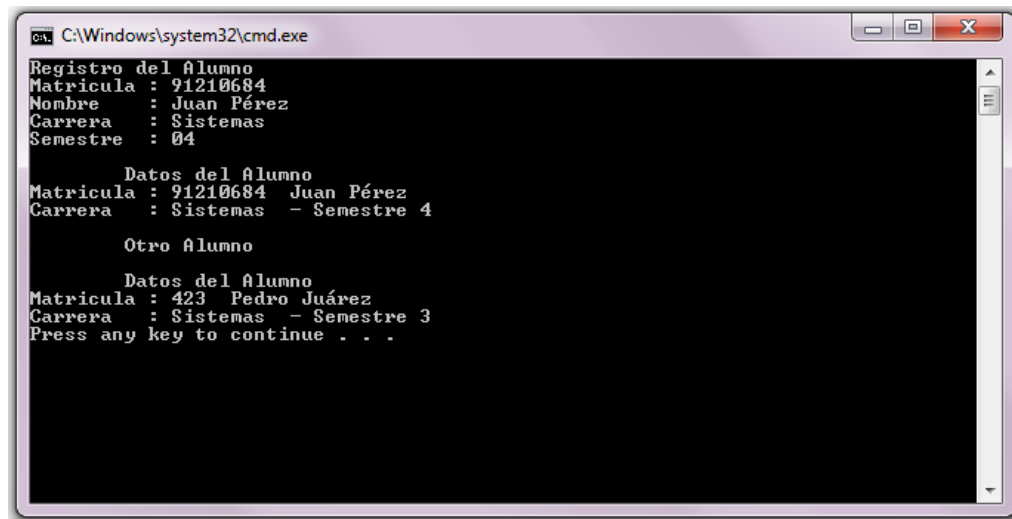
```

```
}  
}  
}
```

d) Corrida o Pantalla de captura



```
C:\Windows\system32\cmd.exe  
Registro del Alumno  
Matricula : 91210684  
Nombre : Juan Pérez  
Carrera : Sistemas  
Semestre : 04
```



```
C:\Windows\system32\cmd.exe  
Registro del Alumno  
Matricula : 91210684  
Nombre : Juan Pérez  
Carrera : Sistemas  
Semestre : 04  
  
Datos del Alumno  
Matricula : 91210684 Juan Pérez  
Carrera : Sistemas - Semestre 4  
  
Otro Alumno  
  
Datos del Alumno  
Matricula : 423 Pedro Juárez  
Carrera : Sistemas - Semestre 3  
Press any key to continue . . .
```

**Sugerencias
didácticas**

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.

Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML.

Identificar a que miembros se podrán acceder desde el exterior y cuales no.

Nombrar otros ejemplos de clases para identificar sus miembros y que modificadores de acceso son más adecuados para cada miembro.

Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y Objetos (C#)</i>. [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/classes-and-objects. Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	CUATRO
Nombre	Crea un objeto inicializándolo con el constructor por defecto
Competencia específica de la unidad 2	Comprende y aplica la estructura de clases para la creación de objetos y utiliza clases predefinidas para facilitar el desarrollo de aplicaciones.
Introducción	Los métodos constructores son aquellos métodos encargados de dar un estado inicial valido a la clase; inicializan variables, reservan memoria para el proceso, abren y leen archivos, etc. Este método se ejecuta automáticamente al instanciarse un objeto de la clase. En la mayoría de lenguajes de programación, el método (ó métodos) constructor tiene el mismo nombre de la clase. Un objeto de una determinada clase se crea cuando la ejecución del programa entra en el ámbito en que está definida y se destruye cuando se llega al final del ámbito. Esto es válido tanto para objetos globales como locales, ya que los objetos globales se crean al comenzar el programa y se destruyen al salir de él. Sin embargo, se puede crear un objeto también mediante el operador new, que asigna la memoria necesaria para alojar el objeto y devuelve su dirección, en forma de puntero, al objeto en cuestión. Si no proporciona un constructor para el objeto, C# creará uno de forma predeterminada que cree instancias del objeto y establezca las variables miembro en los valores predeterminados.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.6 Creación de objetos
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase

-
- Metodología**
1. Analiza el problema e identifica los elementos necesarios para resolver el problema utilizando el mecanismo de abstracción.
 2. Clasifica los datos y define la clase representándola a través del diagrama de clases en UML.
 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases.
 4. Compila el programa y corrige errores de sintaxis y lógica.
 5. Ejecuta el código y revisa los resultados.

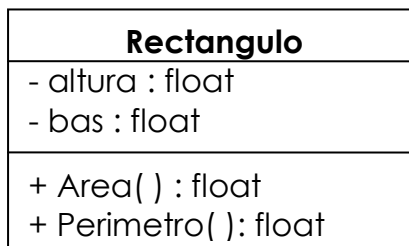
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Calcula el área y perímetro de la figura geométrica del Rectángulo.

a) Datos :

- base
- altura
- área
- perímetro

b) Diagrama de clases



c) Código C# en consola

CLASE RECTANGULO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Rectangulo
{
    class Rectangulo
    {
        // atributos
        float altura, bas;
    }
}
```

```

        // propiedades
        public float Altura
        {
            get { return altura; }
            set { altura = value; }
        }
        public float Bas
        {
            get { return bas; }
            set { bas = value; }
        }

        // métodos constructores
        public Rectangulo() { }
        public Rectangulo(float altura, float bas)
        {
            this.bas = bas;
            this.altura = altura;
        }

        // métodos
        public float Area()
        {
            return bas * altura;
        }

        public float Perimetro()
        {
            return 2*bas + 2*altura;
        }
    }
}

```

CLASE PRINCIPAL

```

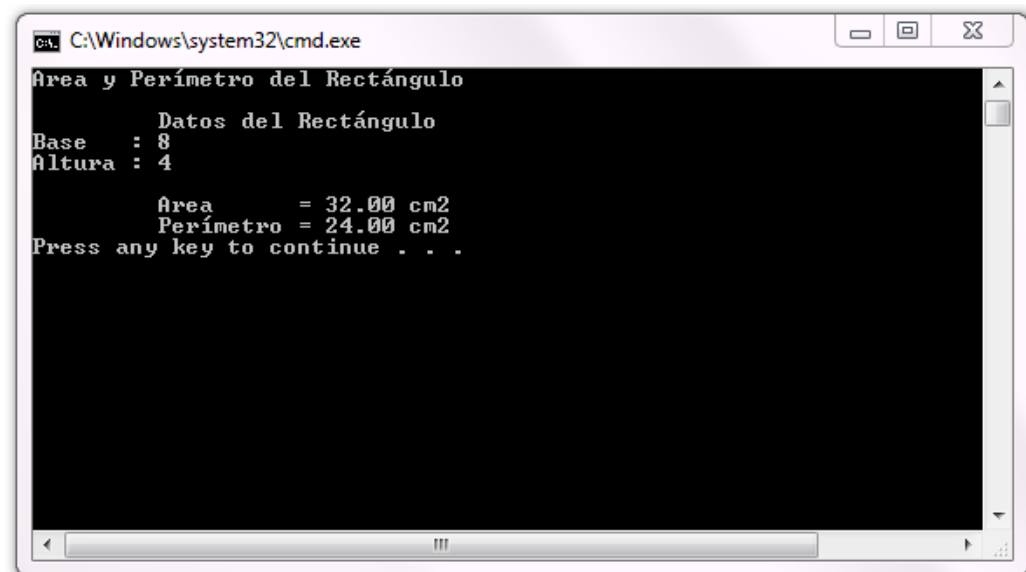
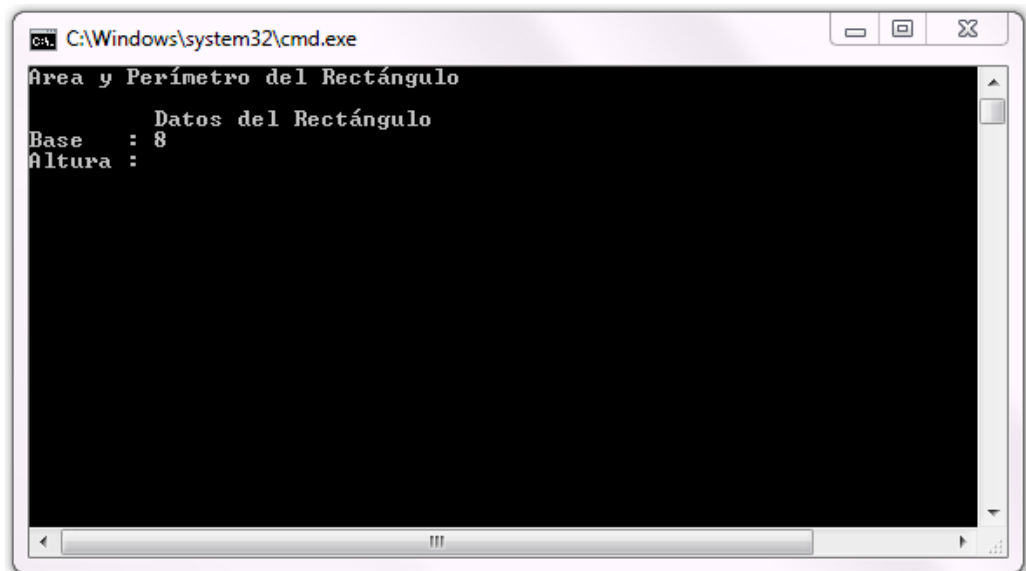
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Rectangulo
{
    class Program
    {
        static void Main(string[] args)
        {
            Rectangulo rect = new Rectangulo();
            Console.WriteLine("Área y Perímetro del
Rectángulo");
            Console.WriteLine("\n\t Datos del Rectángulo ");
            Console.Write("Base : ");
            rect.Bas = float.Parse(Console.ReadLine());
            Console.Write("Altura : ");
            rect.Altura = float.Parse(Console.ReadLine());

```

```
        Console.WriteLine("\n\t Área      = {0:N2} cm2 ",
rect.Area());
        Console.WriteLine("\t Perímetro = {0:N2} cm2 ",
rect.Perimetro());
    }
}
```

d) Corrida o Pantalla de captura



Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Inicializar el objeto utilizando el constructor por defecto. Utilizar métodos que calculen el área y perímetro y que regresen el resultado. Realiza otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y estructuras (Guía de programación de C#)</i> [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/ Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	CINCO
Nombre	Crea un objeto utilizando el constructor sobrecargado
Competencia específica de la unidad 2	Comprende y aplica la estructura de clases para la creación de objetos y utiliza clases predefinidas para facilitar el desarrollo de aplicaciones.
Introducción	Una clase puede tener múltiples constructores (utilizando polimorfismo). Los constructores son métodos y pueden estar sobrecargados • Mismo ámbito, mismo nombre, distintos parámetros • Permite inicializar objetos de distintas maneras
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.6 Creación de objetos
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan un objeto tipo Producto utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase Producto representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. Sobrecarga los métodos constructores. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es : I. Definición : Realiza el cálculo de una nota de venta del siguiente producto : Escobas, marca "La Mejor", descripción : escoba

ergonómica, precio unitario \$25.54. Inicializa el objeto con los datos indicados.

a) Datos :

- nombre
- marca
- descripcion
- precio
- calcular total

b) Diagrama de clases

Producto
- nombre : string - marca : string - descripcion: string - precio : float
+ Venta(cant: int):float +MostrarDatos(cant : int): void

c) Código C# en consola

CLASE PRODUCTO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

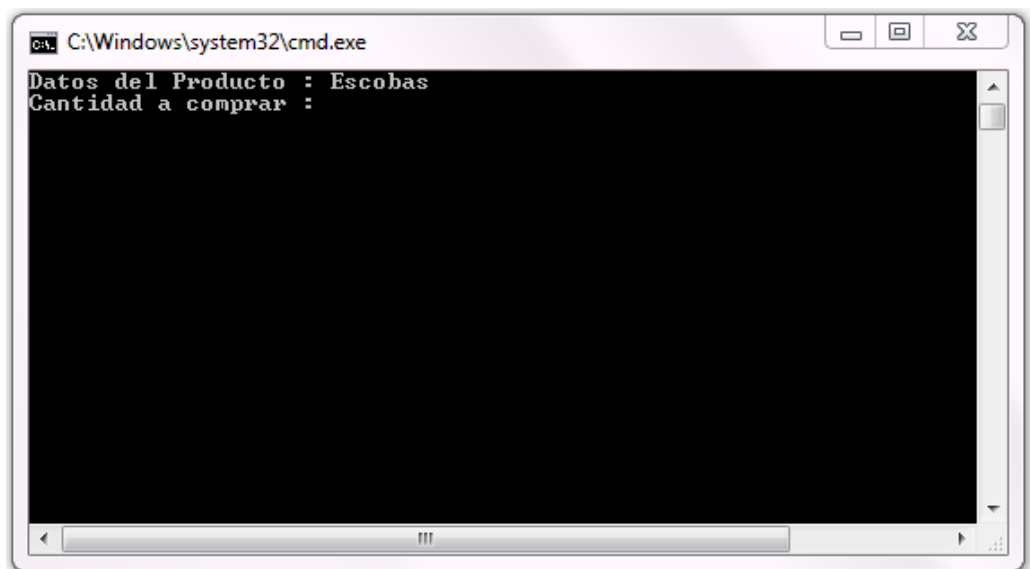
namespace Producto
{
    class Producto
    {
        //atributos
        string nombre, marca, descripcion;
        float precio;

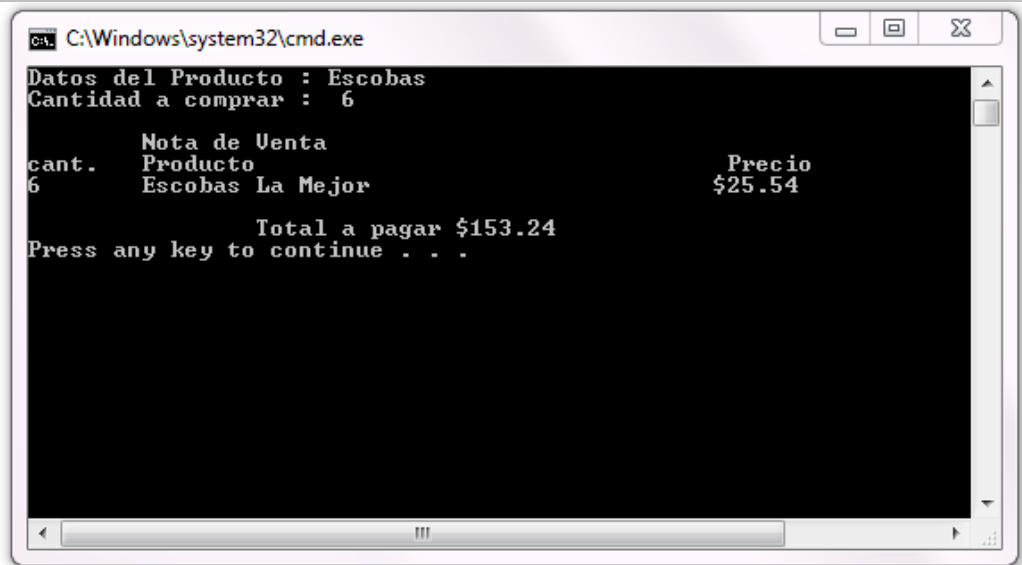
        // propiedades
        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
        public string Marca
        {
```

```
static void Main(string[] args)
{
    // inicializa el objeto con el constructor
    Producto prod = new Producto("Escobas", "La
Mejor", "Escoba ergonomica", 25.54f);
    Console.WriteLine("Datos del Producto : {0}",
prod.Nombre);
    Console.Write("Cantidad a comprar : ");
    int cant = int.Parse(Console.ReadLine());

    prod.MostrarDatos(cant);
}
}
```

d) Corrida o Pantalla de captura





```
C:\Windows\system32\cmd.exe
Datos del Producto : Escobas
Cantidad a comprar : 6

cant.  Producto  Precio
6      Escobas La Mejor  $25.54

Total a pagar $153.24
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Inicializar el objeto utilizando el constructor sobrecargado. Utilizar métodos que no regresan valor y métodos que regresan valor, utilizando el mecanismo de paso de parámetros por valor. Realiza otros ejercicios para practicar a partir del diagrama de clases.
-------------------------------	---

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
---------------------------	--

Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y estructuras (Guía de programación de C#)</i> [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/
---------------------	--

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	SEIS
Nombre	Elabora un programa utilizando manejo de excepciones
Competencia específica de la unidad 2	Comprende y aplica la estructura de clases para la creación de objetos y utiliza clases predefinidas para facilitar el desarrollo de aplicaciones.
Introducción	Las características de control de excepciones del lenguaje C# proporcionan una manera de afrontar cualquier situación inesperada o excepcional que se presente mientras se ejecuta un programa. En esta práctica se manejarán los tipos de cláusulas catch que se pueden utilizar como son: la cláusula catch específica, cláusula catch específica con identificador y cláusula catch general.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.9 Manejo de excepciones.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para el control de excepciones. 2. Clasifica los datos y define la clase principal. 3. Escribe el código de acuerdo a la definición del problema. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Elabora un programa donde realices un división entre dos números, utiliza las excepciones necesarias.

a) Datos :

- dividendo
- divisor
- cociente

c) Código C# en consola

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Excepciones
{
    class Program
    {
        static void Main(string[] args)
        {
            int divisor, dividendo, cociente;

            Console.WriteLine("Realiza la siguiente División\n");
            try
            {
                Console.Write("Dividendo: ");
                dividendo = int.Parse(Console.ReadLine());
                Console.Write("Divisor : ");
                divisor = int.Parse(Console.ReadLine());

                cociente = dividendo / divisor;

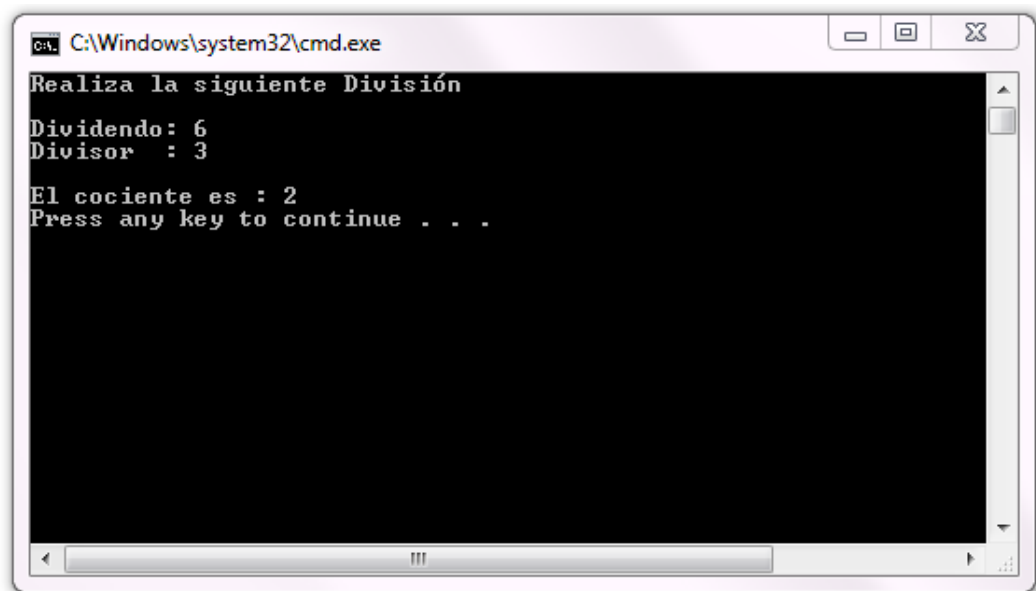
                Console.WriteLine("\nEl cociente es : {0} ",
cociente);
            }

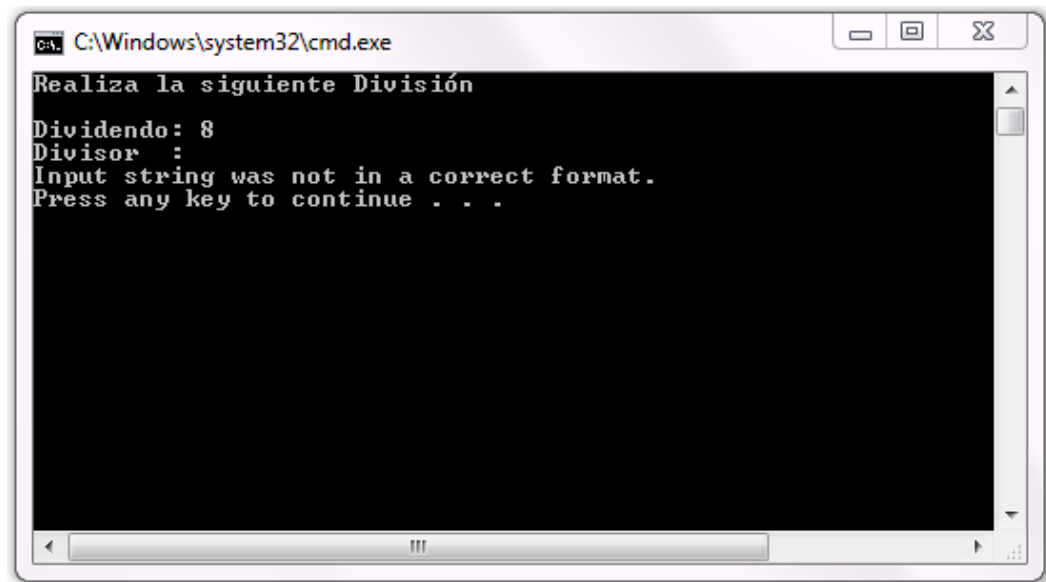
            catch (DivideByZeroException e)
            {
                Console.WriteLine(e.Message);
            }

            catch (FormatException e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}
```

```
    }  
    catch (OverflowException e)  
    {  
        Console.WriteLine(e.Message);  
    }  
}  
}
```

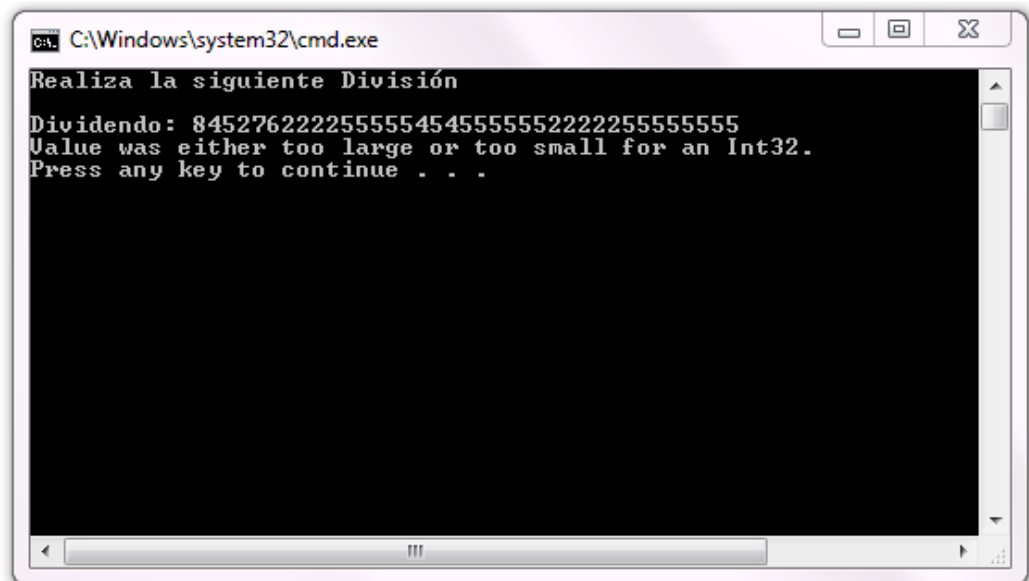
d) Corrida o Pantalla de captura





A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text: "Realiza la siguiente División", "Dividendo: 8", "Divisor :", "Input string was not in a correct format.", and "Press any key to continue . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```
C:\Windows\system32\cmd.exe
Realiza la siguiente División
Dividendo: 8
Divisor :
Input string was not in a correct format.
Press any key to continue . . .
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window contains the following text: "Realiza la siguiente División", "Dividendo: 845276222255555454555555222255555555", "Value was either too large or too small for an Int32.", and "Press any key to continue . . .". The window has standard Windows window controls (minimize, maximize, close) in the top right corner and a scrollbar on the right side.

```
C:\Windows\system32\cmd.exe
Realiza la siguiente División
Dividendo: 845276222255555454555555222255555555
Value was either too large or too small for an Int32.
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Utiliza los tipos de excepciones mostrando el mensaje de error correspondiente para identificar la excepción que se ejecutó. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Excepciones y control de excepciones (Guía de programación de C#)</i> [online] Available at : https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/exceptions/ Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	SIETE
Nombre	Elabora un programa donde relances tu propia excepción
Competencia específica de la unidad 2	Comprende y aplica la estructura de clases para la creación de objetos y utiliza clases predefinidas para facilitar el desarrollo de aplicaciones.
Introducción	En el lenguaje C#, los errores del programa se difunden en tiempo de ejecución a través del programa mediante un mecanismo denominado excepciones. Las excepciones se producen cuando el código encuentra un error y se detectan mediante el código que puede corregir el error. Las clases derivadas de <i>Exception</i> representan estas excepciones. Esta clase identifica el tipo de excepción y contiene las propiedades que albergan detalles acerca de ésta. Producir una excepción implica crear una instancia de una clase derivada de la excepción, configurar opcionalmente las propiedades de la excepción y, a continuación, iniciar el objeto con la palabra clave <i>throw</i>.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 2. Clases y Objetos 2.9 Manejo de excepciones.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para el control de excepciones. 2. Clasifica los datos y define la clase principal. 3. Escribe el código de acuerdo a la definición del problema. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : En este ejercicio escribirá un programa que lee de la consola un número entero de día (entre 1 y 365) y lo almacena en una variable entera. El programa convertirá este número en un mes y un día del mes e imprimirá el resultado en la consola. Por ejemplo, el resultado de introducir 40 debe ser “Febrero 9”. (En este ejercicio se ignora el efecto de los años bisiestos.)

1. Declare un tipo **enum** llamado **MesNombre** y asígnele los nombres de los doce meses, empezando con enero y terminando con diciembre.
2. Cambie el nombre de la clase a **QueDiaEs**.
3. Añada a **Main** una instrucción **Console.Write** que escriba un mensaje en la consola para pedir al usuario que introduzca un número de día entre 1 y 365.
4. Añada a **Main** una instrucción que declare una variable **string** llamada *linea* y la inicialice con una línea leída desde la consola con el método **Console.ReadLine**.
5. Añada a **Main** una instrucción que declare una variable **int** llamada *diaNum* y la inicialice con el entero devuelto por el método **int.Parse**.
6. Añada a **Main** una instrucción que declare una variable **int** llamada *mesNum* y la inicialice a cero.
7. Escriba una instrucción **foreach** que itere la colección **DiasMes** proporcionada. Para ello, añada la siguiente instrucción: `foreach (int diasEnMes in DiasMes) ...`
8. Añada un bloque de instrucciones como cuerpo de la instrucción **foreach**.
9. Añada una instrucción que declare una variable **enum MesNombre** llamada *temp*.
10. Inicialice *temp* desde la variable **int mesNum**. Para ello necesitará la siguiente expresión de cast: `(MesNombre)mesNum`.
11. Sustituya la inicialización de *mesNombre* por la expresión `temp.ToString()`.
12. Utilice el método **WriteLine** para obtener los valores de *diaNum* y *mesNombre*.
13. No olvide relanzar la excepción específica para este problema.
14. Agregue las cláusulas necesarias para control de excepciones.

a) Datos :
- número del día solicitante

c) Código C# en consola

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Relanzar
{
    public enum MesNombre { Enero, Feb, Marzo, abril, Mayo,
        Junio, Julio, agosto, sep, oct, nov, dic }
    class QueDiaEs
```

```

    {
        static void Main()
        {
            int[] DiasMes = { 31, 28, 31, 30, 31, 30, 31, 30,
30, 31, 30, 31 };

            try
            {
                Console.WriteLine("Por favor, escriba un número de
día entre 1 y 365: ");
                string linea = Console.ReadLine();
                int diaNum = int.Parse(linea);

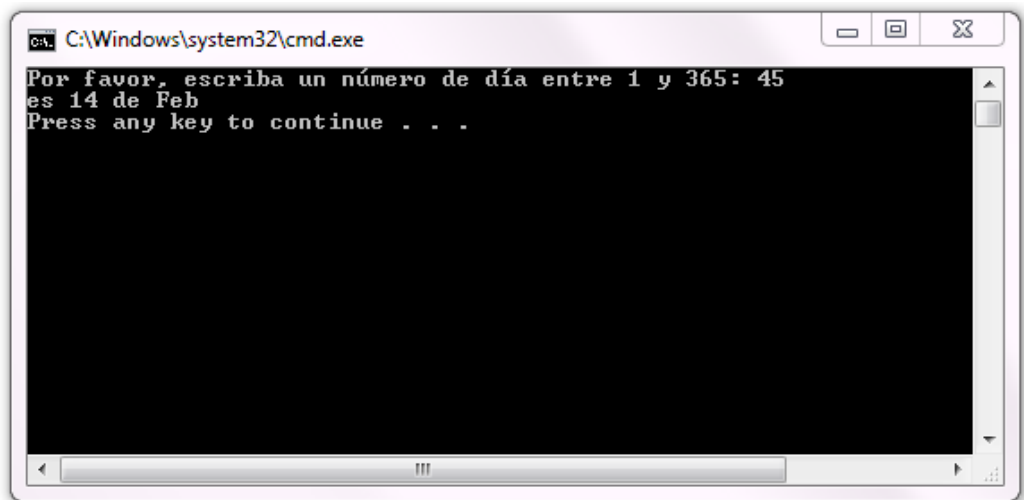
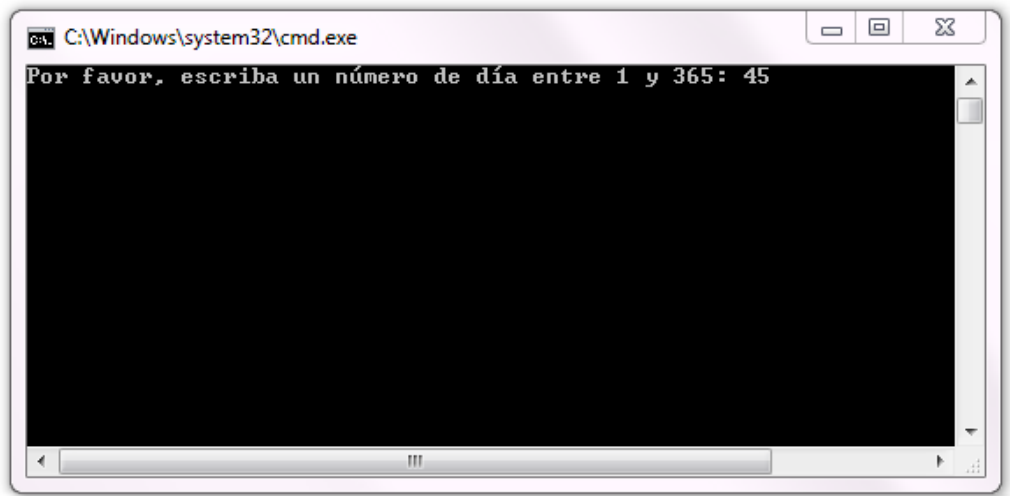
                if (diaNum < 1 || diaNum > 365)
                {
                    throw new ArgumentOutOfRangeException("Día
fuera de intervalo");
                }
                int mesNum = 0;
                foreach (int diasEnMes in DiasMes)
                {
                    if (diaNum <= diasEnMes)
                    {
                        break;
                    }
                    else
                    {
                        diaNum -= diasEnMes;
                        mesNum++;
                    }
                }
                MesNombre temp = (MesNombre)mesNum;
                string mesNombre = temp.ToString();
                Console.WriteLine("es {0} de {1}", diaNum,
mesNombre);
            }

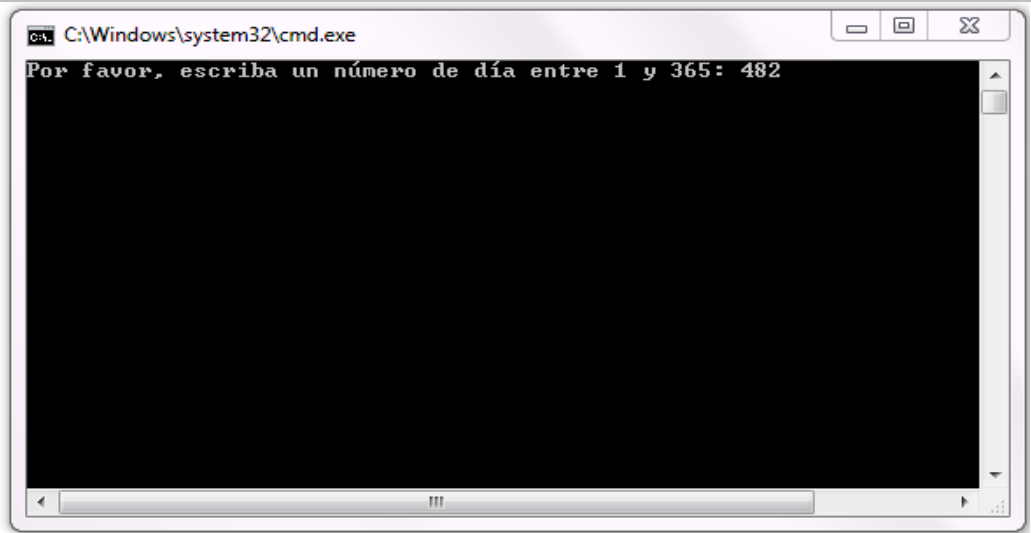
            catch (ArgumentOutOfRangeException capturada)
            {
                Console.WriteLine(capturada.ParamName);
            }
            catch (FormatException e)
            {
                Console.WriteLine(e.Message);
            }

            catch (OverflowException e)
            {
                Console.WriteLine(e.Message);
            }
        }
    }
}

```

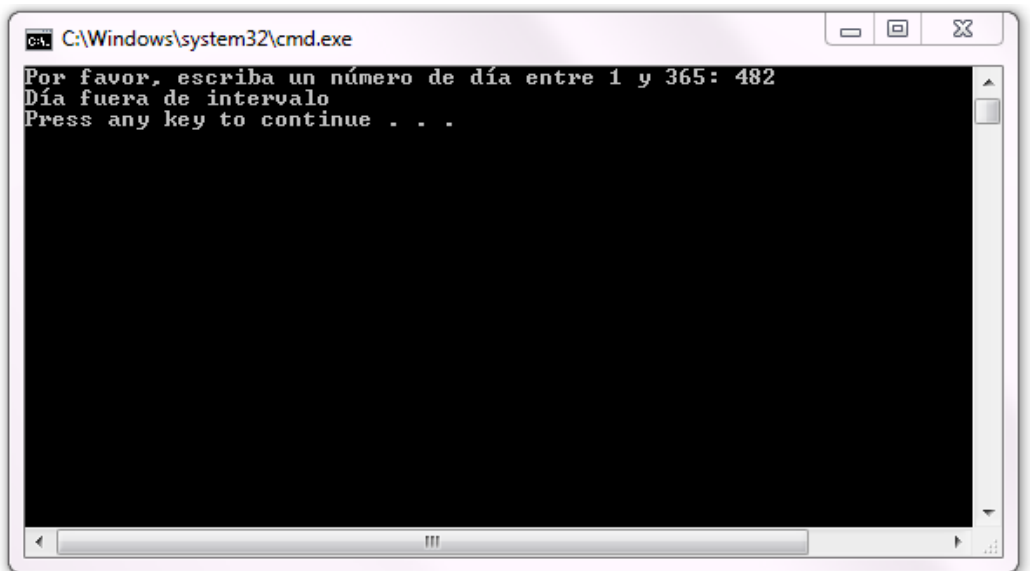
d) Corrida o Pantalla de captura





C:\Windows\system32\cmd.exe

```
Por favor, escriba un número de día entre 1 y 365: 482
```



C:\Windows\system32\cmd.exe

```
Por favor, escriba un número de día entre 1 y 365: 482
Día fuera de intervalo
Press any key to continue . . .
```

**Sugerencias
didácticas**

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.

Utiliza los tipos de excepciones mostrando el mensaje de error correspondiente para identificar la excepción que se ejecuto.

Utiliza la instrucción correspondiente para crear tu propia excepción y relanzarla por el método solicitante.

Realizar otros ejercicios para practicar a partir del diagrama de clases.

**Reporte del
Alumno**

El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.

Bibliografía

Bell, D. and Parr, M. (2011). *C# para estudiantes*. Pearson Educación de México, S.A. de C.V.

Ceballos Sierra, F. (2012). *Microsoft C#*. México, D.F.: Alfaomega.

Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). *La Biblia de C#*. Anaya.

Gonzalez Seco, J. (2012). *El lenguaje de programación C#*.

Larman, C. (2003). *UML y patrones*. Naucalpan de Juárez: Pearson Educación.

Docs.microsoft.com. (2019). *Excepciones y control de excepciones (Guía de programación de C#)* [online] Available at :
<https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/exceptions/>

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	OCHO
Nombre	Define una clase con sus métodos
Competencia específica de la unidad 3	Comprende y aplica los diferentes tipos de métodos, tomando en cuenta el ámbito y tiempo de vida de los datos durante la ejecución de un programa.
Introducción	La biblioteca de C# proporciona métodos predefinidos, pero también podemos definir nuestros propios métodos e invocarlos de la misma forma que los predefinidos. Podemos definir métodos que pasan parámetros y métodos que retornan o no valores. En esta práctica se utilizarán métodos que retornan valor y métodos que no retornan valor.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Métodos 3.1 Definición de un método. 3.2 Estructura de un método.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan las clases utilizando el mecanismo de abstracción . 2. Clasifica los datos y define la clase Película representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Calcula cuantos horas y minutos dura una película dado la duración en minutos.

a) Datos :

- duración de la película

b) Diagrama de clases

Pelicula
- duracion : int
+ convertirMIN() : void

c) Código C# en consola

CLASE PELICULA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Metodos
{
    class Pelicula
    {
        int duracion;
        public int dura
        {
            get { return duracion; }
            set { duracion = value; }
        }

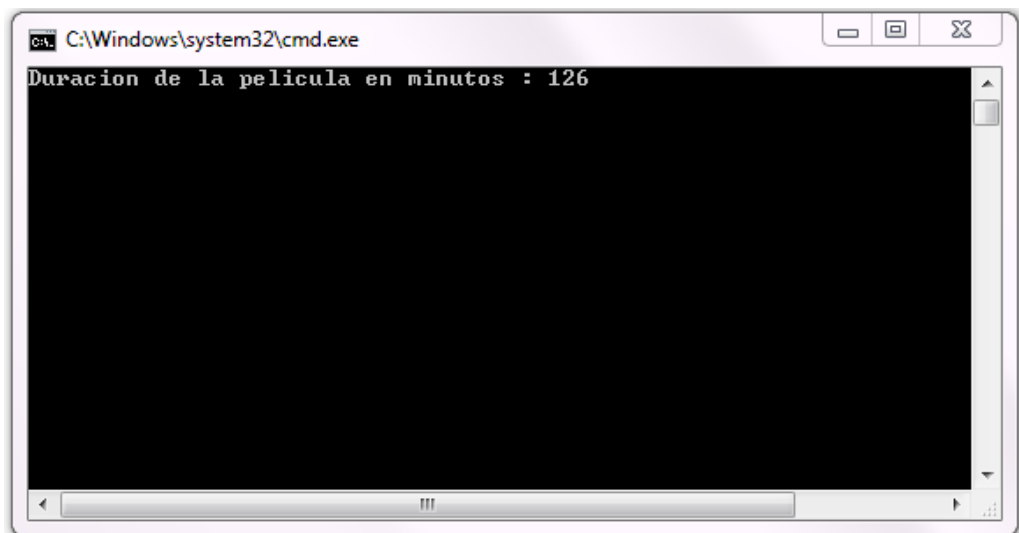
        public void convertirMIN()
        {
            int hr = duracion / 60;
            int min = duracion % 60;
            Console.WriteLine("La Pelicula dura {0}min. = {1} hr.
{2} min. ", dura, hr, min);
        }
    }
}
```

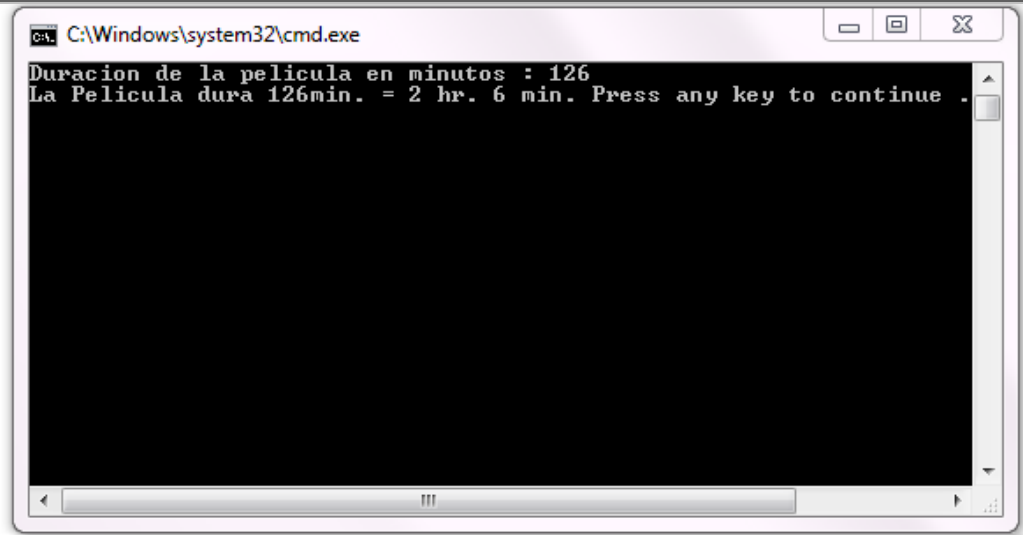
CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Metodos
{
    class Program
    {
        static void Main(string[] args)
        {
            Pelicula p = new Pelicula();
            Console.WriteLine("Duracion de la pelicula en minutos : 
");
            p.dura = int.Parse(Console.ReadLine());
            p.convertirMIN();
        }
    }
}
```

d) Corrida o Pantalla de captura





Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Utilizar métodos que no regresan valor. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y estructuras (Guía de programación de C#)</i> [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/ Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	NUEVE
Nombre	Define una clase donde defina un método que regrese valor
Competencia específica de la unidad 3	Comprende y aplica los diferentes tipos de métodos, tomando en cuenta el ámbito y tiempo de vida de los datos durante la ejecución de un programa.
Introducción	La biblioteca de C# proporciona métodos predefinidos, pero también podemos definir nuestros propios métodos e invocarlos de la misma forma que los predefinidos. Podemos definir métodos que pasan parámetros y métodos que retornan o no valores. En la definición de métodos también podemos utilizar el mecanismo de paso de parámetros, en este caso el paso de parámetros por valor. Cuando un método en C# invoca a otro método y le pasa un argumento de un tipo primitivo, pasa una copia de ese argumento. En esta práctica se utilizarán métodos que retornan valor y métodos que no retornan valor, además de paso de parámetros por valor.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Métodos 3.1 Definición de un método. 3.2 Estructura de un método. 3.3 Valor de retorno.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan la clase adecuada utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase Numeros representándola a través del diagrama de clases en UML.

-
3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases.
 4. Compila el programa y corrige errores de sintaxis y lógica.
 5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Definirá y usará parámetros de entrada en un método que devuelve un valor.

Crear un método llamado **MMayor** que aceptará como entrada dos parámetros enteros y devolverá el valor del que sea mayor.

Crear el método MMayor como se indica a continuación

1. Añada a la clase un método **public static** llamado **MMayor** .
2. El método aceptara dos parámetros **int** llamados **a** y **b** que se pasaran por valor, y devolverá un valor **int** que represente el mayor de los dos números.

Escribe el siguiente código en el método Main

1. Defina dos variables enteras llamadas **n1** y **n2**
2. Añada instrucciones que lean dos enteros desde la entrada por teclado y los asignen a **n1** y **n2**. Use los métodos **Console.ReadLine** e **int.Parse**.
3. Defina otro entero llamado **mayor**.
4. Haga una llamada al método **MMayor**, y asigne el valor devuelto a la variable **mayor**.
5. Muestre en consola cual de los dos números leídos fue el mayor.

a) Datos :

- primer número
- segundo número

b) Diagrama de clases

Numeros
- num1 : int - num2 : int
+ MMayor() : int

c) Código C# en consola

CLASE NUMEROS

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace RegresaValor
{
    class Numeros
    {
        // atributos
        int num1, num2;

        // propiedades
        public int N1
        {
            get { return num1; }
            set { num1 = value; }
        }

        public int N2
        {
            get { return num2; }
            set { num2 = value; }
        }

        // métodos constructores
        public Numeros() { }
        public Numeros(int n1, int n2)
        {
            num1 = n1;
            num2 = n2;
        }

        // método que regresa valor
        public int MMayor()
        {
            if (num1 > num2)
                return num1;
            else
                return num2;
        }
    }
}
```

CLASE PRINCIPAL

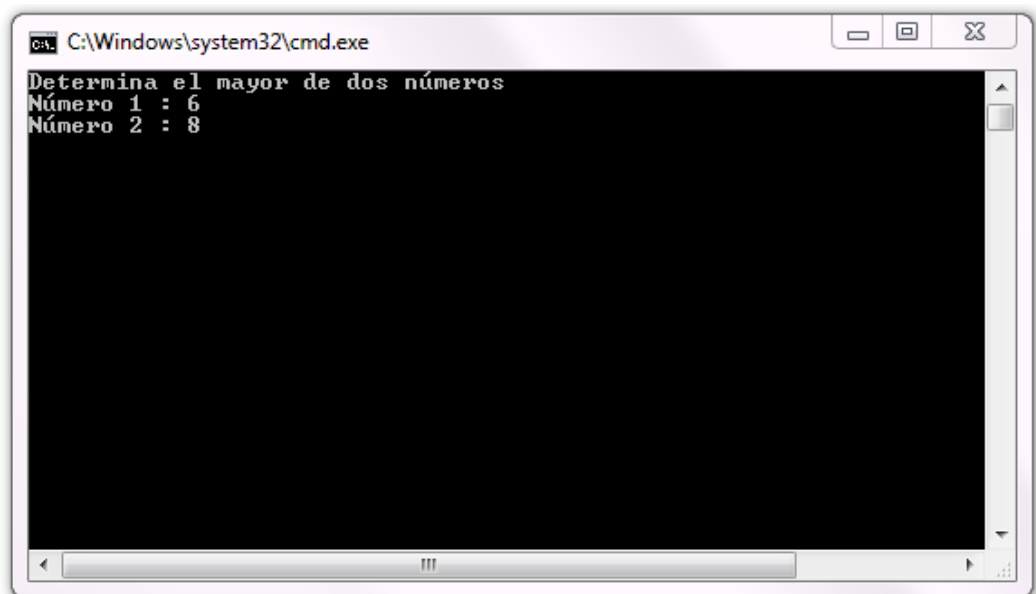
```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

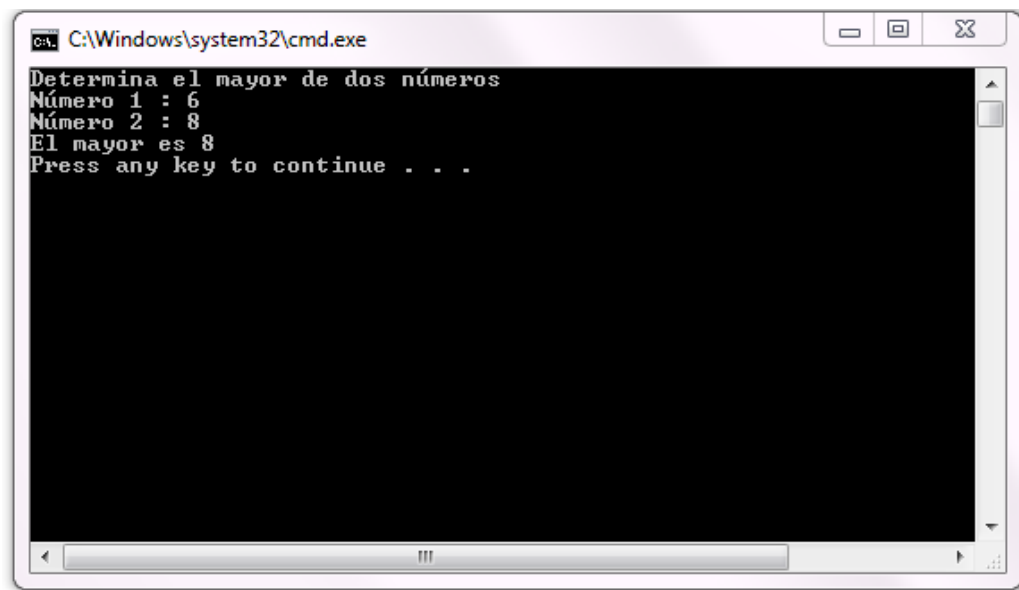
namespace RegresaValor
{
    class Program
    {
        static void Main(string[] args)
        {
            Numeros Num = new Numeros();
            Console.WriteLine("Determina el mayor de dos
números");
            Console.Write("Número 1 : ");
            Num.N1 = int.Parse(Console.ReadLine());
            Console.Write("Número 2 : ");
            Num.N2 = int.Parse(Console.ReadLine());

            int mayor= Num.MMayor();

            Console.WriteLine("El mayor es {0}", mayor);
        }
    }
}
```

d) Corrida o Pantalla de captura





Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Utilizar métodos que regresan valor. Realizar otros ejercicios para practicar a partir del diagrama de clases.
-------------------------------	--

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
---------------------------	--

Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y estructuras (Guía de programación de C#)</i> [online] Available at: https://docs.microsoft.com/es-
---------------------	--

es/dotnet/csharp/programming-guide/classes-and-structs/methods

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	DIEZ
Nombre	Crear un programa que sobrecargue un método
Competencia específica de la unidad 3	Comprende y aplica los diferentes tipos de métodos, tomando en cuenta el ámbito y tiempo de vida de los datos durante la ejecución de un programa.
Introducción	Un método es un bloque de código que contiene una serie de instrucciones. Un programa hace que se ejecuten las instrucciones al llamar al método y especificando los argumentos de método necesarios. En C#, todas las instrucciones ejecutadas se realizan en el contexto de un método. Los métodos se pueden sobrecargar utilizando polimorfismo en funciones, esto es utilizando la misma firma del método pero con diferente tipo y cantidad de parámetros y diferente implementación.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 3. Métodos 3.8 Sobrecarga de métodos.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios que definan la clase utilizando el mecanismo de abstracción. 2. Clasifica los datos y define la clase Suma representándola a través del diagrama de clases en UML. 3. Escribe el código de acuerdo a la clase mostrada en el diagrama de clases. 4. Compila el programa y corrige errores de sintaxis y lógica. 5. Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Escribe un programa que sobrecargue métodos.

1. Deberá pedir 4 números, para ello defina un método que regrese el número capturado.
2. Realice las sumas necesarias combinando los 4 números. Para ello Sobrecargue los métodos de suma y desplegar.
3. En el método Main pida los números, haga la llamada a los métodos suma y desplegar enviando los parámetros correspondientes y un string que me indique que números son los que se están sumando. Por ejemplo n1 + n2 :

a) Datos :

- numero 1
- numero 2
- numero 3
- numero4
- sobrecargar el método suma

b) Diagrama de clases

Suma
+ pedirnumero(n : string): int + suma(a : int, b : int) : int + suma(a : int, b : int, c : int) : int + suma(a : int, b : int, c : int, d : int): int + desplegar(m : string, a : int, b : int, c : int) : void + desplegar(m : string, a : int, b : int, c : int, d : int) : void + desplegar(m : string, a : int, b : int, c : int, d : int, e : int) : void

c) Código C# en consola

CLASE SUMA

```
using System;  
using System.Collections.Generic;  
using System.Linq;  
using System.Text;
```

```

namespace Sobre cargaSuma
{
    class Suma
    {
        public int pedirnumero(string n)
        {
            Console.WriteLine("\n\tIngresa el número {0} : ",n);
            return int.Parse(Console.ReadLine());
        }

        public Suma() { }

        public int suma(int a, int b)
        { return a + b; }

        public int suma(int a, int b, int c)
        { return a + b + c; }

        public int suma(int a, int b, int c, int d)
        { return a + b + c + d; }

        public void desplegar(string m, int a, int b, int c)
        { Console.WriteLine(m + " {0} + {1} = {2}", a, b, c); }

        public void desplegar(string m, int a, int b, int c, int
d)
        { Console.WriteLine(m + " {0} + {1} + {2} = {3}", a, b,
c, d); }

        public void desplegar(string m, int a, int b, int c, int
d, int e)
        { Console.WriteLine(m + " {0} + {1} + {2} + {3} = {4}",
a, b, c, d, e); }
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Sobre cargaSuma
{
    class Program
    {
        static void Main(string[] args)
        {
            // declarar variables
            int n1, n2, n3, n4, s;

            // pedir datos

```

```

Suma obj = new Suma();
Console.WriteLine("\n\t\tSuma de números");
n1 = obj.pedirnumero("n1");
n2 = obj.pedirnumero("n2");
n3 = obj.pedirnumero("n3");
n4 = obj.pedirnumero("n4");

// llamar a metodos sobrecargados y desplegar
s = obj.suma(n1, n2);
obj.desplegar("\nn1 + n2 : ", n1, n2, s);

s = obj.suma(n1, n3);
obj.desplegar("n1 + n3 : ", n1, n3, s);

s = obj.suma(n1, n4);
obj.desplegar("n1 + n4 : ", n1, n4, s);

s = obj.suma(n2, n3);
obj.desplegar("n2 + n3 : ", n2, n3, s);

s = obj.suma(n2, n4);
obj.desplegar("n2 + n4 : ", n2, n4, s);

s = obj.suma(n3, n4);
obj.desplegar("n3 + n4 : ", n3, n4, s);

s = obj.suma(n1, n2, n3);
obj.desplegar("\nn1 + n2 + n3 : ", n1, n2, n3, s);

s = obj.suma(n1, n3, n4);
obj.desplegar("n1 + n3 + n4 : ", n1, n3, n4, s);

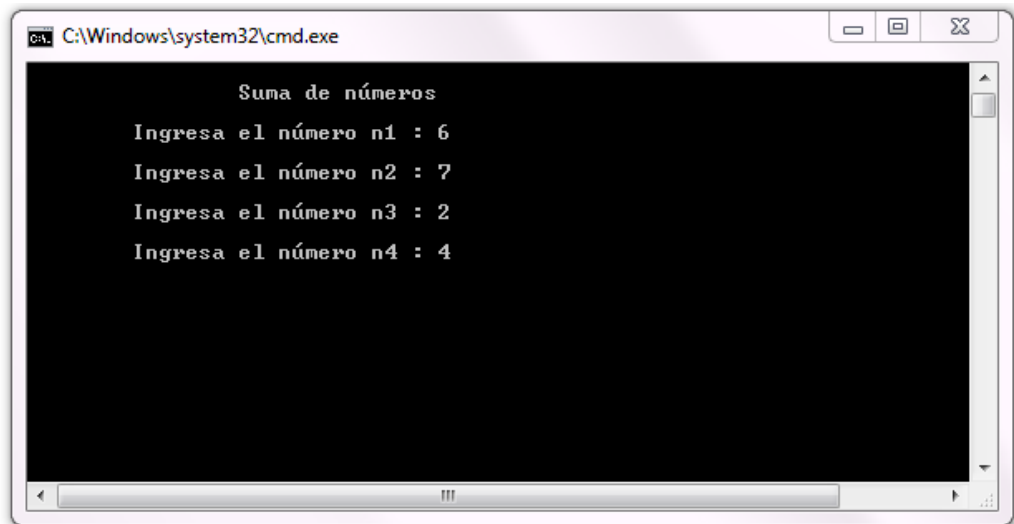
s = obj.suma(n1, n2, n4);
obj.desplegar("n1 + n2 + n4 : ", n1, n2, n4, s);

s = obj.suma(n2, n3, n4);
obj.desplegar("n2 + n3 + n4 : ", n2, n3, n4, s);

s = obj.suma(n1, n2, n3, n4);
obj.desplegar("\nn1 + n2 + n3 + n4 : ", n1, n2, n3,
n4, s);
    }
}

```

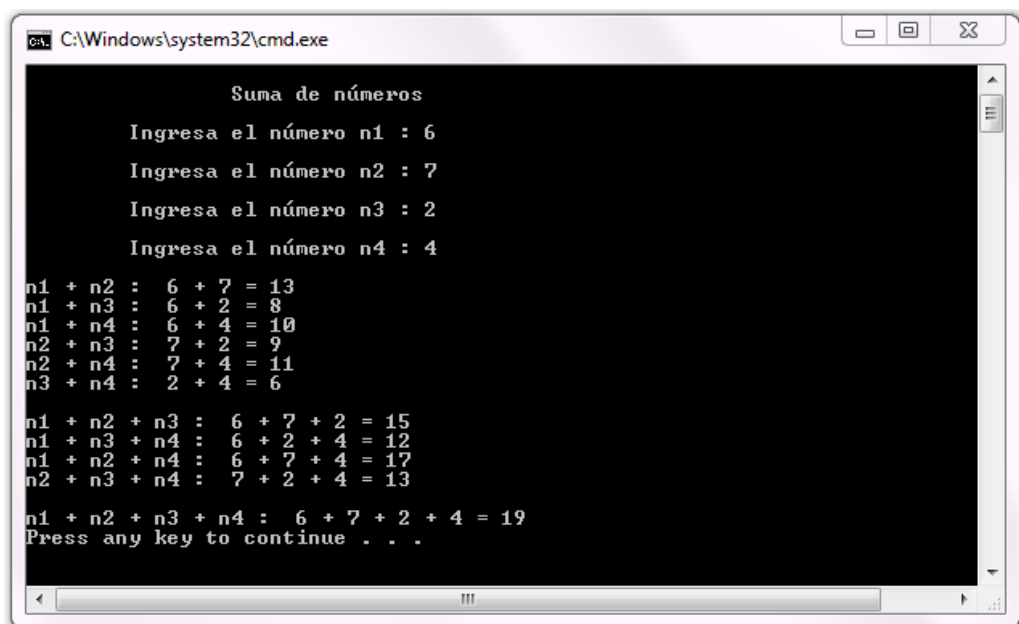
d) Corrida o Pantalla de captura



```
C:\Windows\system32\cmd.exe

Suma de números

Ingresa el número n1 : 6
Ingresa el número n2 : 7
Ingresa el número n3 : 2
Ingresa el número n4 : 4
```



```
C:\Windows\system32\cmd.exe

Suma de números

Ingresa el número n1 : 6
Ingresa el número n2 : 7
Ingresa el número n3 : 2
Ingresa el número n4 : 4

n1 + n2 : 6 + 7 = 13
n1 + n3 : 6 + 2 = 8
n1 + n4 : 6 + 4 = 10
n2 + n3 : 7 + 2 = 9
n2 + n4 : 7 + 4 = 11
n3 + n4 : 2 + 4 = 6

n1 + n2 + n3 : 6 + 7 + 2 = 15
n1 + n3 + n4 : 6 + 2 + 4 = 12
n1 + n2 + n4 : 6 + 7 + 4 = 17
n2 + n3 + n4 : 7 + 2 + 4 = 13

n1 + n2 + n3 + n4 : 6 + 7 + 2 + 4 = 19
Press any key to continue . . .
```

**Sugerencias
didácticas**

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.

Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML.

Identificar los métodos que se pueden sobrecargar y sus características.

Nombrar otros ejemplos de métodos que se pueden sobrecargar.

Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y estructuras (Guía de programación de C#)</i> [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-structs/methods Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	ONCE
Nombre	Elabora un ejercicio donde muestre la herencia entre las clases Persona y Estudiante
Competencia específica de la unidad 4	Comprende y aplica los conceptos de herencia y polimorfismo en programas que utilicen clases base, clases derivadas, clases abstractas e interfaces.
Introducción	El mecanismo de herencia de POO permite que una clase nueva herede todas las características de otra clase ya existente, garantizando la reutilización de código. Con la herencia todas las clases están clasificadas en una jerarquía, donde cada clase puede tener una o más subclases. El término heredar significa que las subclases disponen de todos los métodos y propiedades de su superclase. Este mecanismo proporciona una forma rápida y cómoda de extender la funcionalidad de una clase. En C# solo se puede heredar de una clase base, a esto se le llama herencia simple. Una clase puede redefinir cualquier método heredado de una clase padre siempre que sea necesario que su comportamiento en la subclase sea diferente. Es posible referirse a un miembro de la clase base que se necesita redefinir en la clase hija utilizando la palabra reservada base y después el miembro que se redefine. El modificador base hace referencia a los métodos y atributos pero en este caso perteneciente a la clase padre del objeto que se está invocando un método o propiedad también se utiliza para indicar cómo deseamos inicializar los campos heredados de la clase padre.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 4. Herencia y Polimorfismo 4.2 Definición de una clase base. 4.3 Definición de una clase derivada.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior

-
- Editor de código Visual Studio 2010 o posterior
 - Apuntes de la clase
-

- Metodología**
1. Analiza el problema e identifica los elementos necesarios que definan las clases utilizando el mecanismo de abstracción y herencia.
 2. Clasifica los datos y define la clase base y la clase derivada representándola a través del diagrama de clases en UML.
 3. Escribe el código de acuerdo a las clases mostradas en el diagrama de clases.
 4. Compila el programa y corrige errores de sintaxis y lógica.
 5. Ejecuta el código y revisa los resultados.

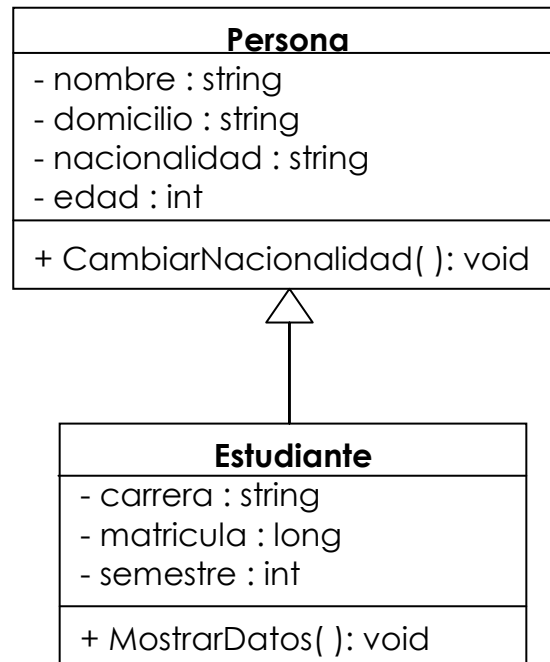
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Hereda de la clase Persona a la clase Estudiante y muestra todos sus datos.

a) Datos :

- nombre
- domicilio
- nacionalidad
- edad
- carrera
- matricula
- semestre
- cambiar de nacionalidad

b) Diagrama de clases



c) Código C# en consola

CLASE BASE PERSONA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Herencia
{
    class Persona
    {
        // atributos
        string nombre, domicilio, nacionalidad;
        int edad;

        // propiedades
        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }

        public string Domicilio
```

```

        {
            get { return domicilio; }
            set { domicilio = value; }
        }

        public string Nacionalidad
        {
            get { return nacionalidad; }
            set { nacionalidad = value; }
        }

        public int Edad
        {
            get { return edad; }
            set { edad = value; }
        }

        // Constructores
        public Persona() { }
        public Persona(string n, string d, string nac, int e)
        {
            nombre = n;
            domicilio=d;
            nacionalidad=nac;
            edad = e;
        }

        // métodos
        public void CambiarNacionalidad()
        {
            Console.WriteLine("\nNueva Nacionalidad : ");
            nacionalidad = Console.ReadLine();
        }
    }
}

```

CLASE DERIVADA ESTUDIANTE

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Herencia
{
    class Estudiante : Persona
    {
        // atributos
        string carrera;
        long matricula;
        int semestre;

        // propiedades

        public string Carrera

```

```

        {
            get { return carrera; }
            set { carrera = value; }
        }
        public long Matricula
        {
            get { return matricula; }
            set { matricula = value; }
        }
        public int Semestre
        {
            get { return semestre; }
            set { semestre = value; }
        }

        // Constructor por defeto
        public Estudiante(): base() { }

        // Constructor sobrecargado
        public Estudiante(string n, string c, long m, int s,
        string d, string nac, int e):base(n,d,nac,e)
        {
            carrera = c;
            matricula = m;
            semestre = s;
        }

        // método publico
        public void MostrarDatos()
        {
            Console.WriteLine("\n\tDatos del Estudiante\n");
            Console.WriteLine("Matricula : {0} \t {1}\t\t Edad :
{2}", matricula, Nombre, Edad);
            Console.WriteLine("Domicilio : {0} \t\t\t
Nacionalidad : {1}", Domicilio, Nacionalidad);
            Console.WriteLine("Carrera : {0} \t\t\t Semestre
: {1}", carrera, semestre);
        }
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

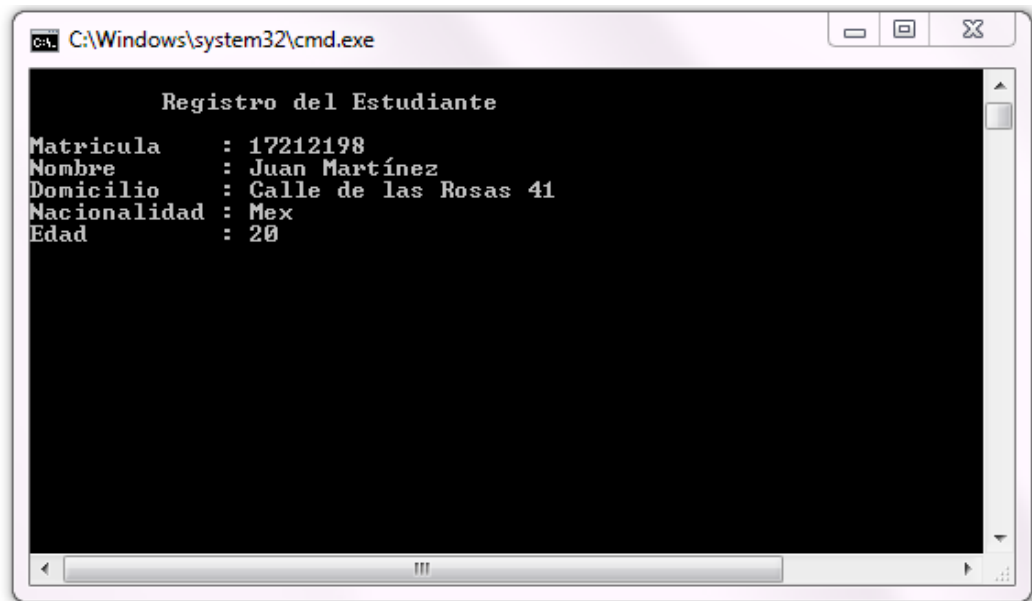
namespace Herencia
{
    class Program
    {
        static void Main(string[] args)
        {

```

```
        Console.WriteLine("\n\t Registro del Estudiante\n");
        Estudiante Est = new Estudiante();
        Console.Write("Matricula      : ");
        Est.Matricula = int.Parse(Console.ReadLine());
        Console.Write("Nombre        : ");
        Est.Nombre = Console.ReadLine();
        Console.Write("Domicilio     : ");
        Est.Domicilio = Console.ReadLine();
        Console.Write("Nacionalidad  : ");
        Est.Nacionalidad = Console.ReadLine();
        Console.Write("Edad          : ");
        Est.Edad = int.Parse(Console.ReadLine());
        Console.Write("Carrera       : ");
        Est.Carrera = Console.ReadLine();
        Console.Write("# Semestre     : ");
        Est.Semestre = int.Parse(Console.ReadLine());

        Est.CambiarNacionalidad();
        Est.MostrarDatos();
    }
}
```

d) Corrida o Pantalla de captura

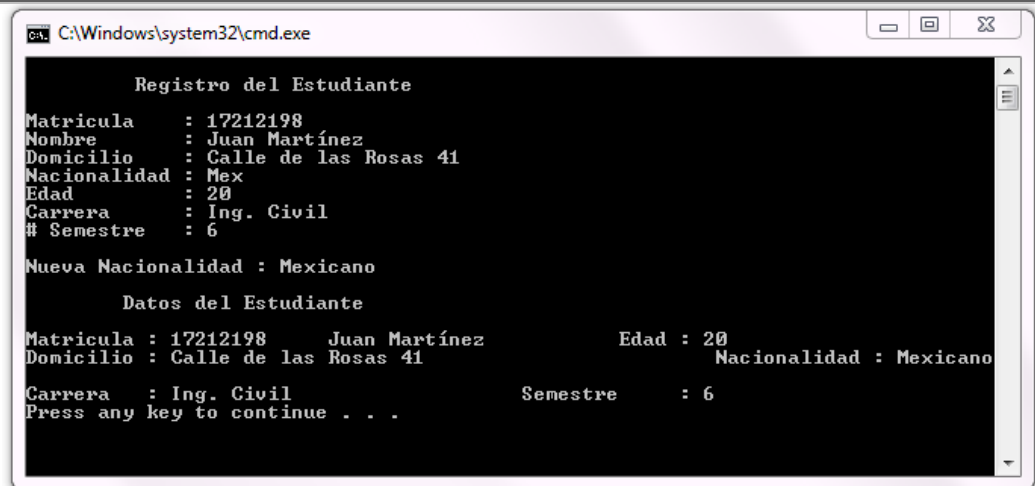


```
C:\Windows\system32\cmd.exe

Registro del Estudiante
Matricula      : 17212198
Nombre         : Juan Martínez
Domicilio      : Calle de las Rosas 41
Nacionalidad   : Mex
Edad           : 20
Carrera        : Ing. Civil
# Semestre     : 6
```

```
C:\Windows\system32\cmd.exe

Registro del Estudiante
Matricula      : 17212198
Nombre         : Juan Martínez
Domicilio      : Calle de las Rosas 41
Nacionalidad   : Mex
Edad           : 20
Carrera        : Ing. Civil
# Semestre     : 6
Nueva Nacionalidad :
```



```
C:\Windows\system32\cmd.exe

Registro del Estudiante
Matricula : 17212198
Nombre : Juan Martínez
Domicilio : Calle de las Rosas 41
Nacionalidad : Mex
Edad : 20
Carrera : Ing. Civil
# Semestre : 6

Nueva Nacionalidad : Mexicano

Datos del Estudiante
Matricula : 17212198      Juan Martínez      Edad : 20
Domicilio : Calle de las Rosas 41      Nacionalidad : Mexicano
Carrera : Ing. Civil      Semestre : 6
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Identificar los miembros de una clase e identificarlos en el diagrama de clases de UML. Identificar la clase base y la clase derivada con sus miembros heredados, definir el constructor por defecto y los constructores sobrecargados. Nombrar otros ejemplos de clases para identificar sus miembros. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Clases y estructuras (Guía de programación de C#)</i> [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/programming-guide/classes-and-

structs/inheritance

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	DOCE
Nombre	Elabora un programa donde implementes de manera implícita una interface
Competencia específica de la unidad 4	Comprende y aplica los conceptos de herencia y polimorfismo en programas que utilicen clases base, clases derivadas, clases abstractas e interfaces.
Introducción	Podemos definir las interfaces como la definición de un conjunto de miembros que serán comunes entre clases que serán (o no) completamente distintas. La interface conoce cómo será la funcionalidad de cualquier clase que la implemente pero, lógicamente, no podrá conocer los detalles de esa implementación en cada una de esas clases, de modo que una interface no puede implementar nada de código, sino que, únicamente, puede describir un conjunto de miembros. Las interfaces sirven para poder agrupar funcionalidades. Recordar que los lenguajes .NET soportan únicamente herencia simple, es decir, una clase se puede derivar de otra clase, pero no de varias; sin embargo, sí podemos implementar en una misma clase tantas interfaces como sean necesarias.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 4. Herencia y Polimorfismo 4.5 Interfaces. 4.5.1 Definición 4.5.2 Implementación
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase

-
- Metodología**
1. Analiza el problema e identifica los elementos necesarios para resolver el problema.
 2. Escribe el código utilizando las interfaces implícitas.
 3. Compila el programa y corrige errores de sintaxis y lógica.
 - 4 Ejecuta el código y revisa los resultados.

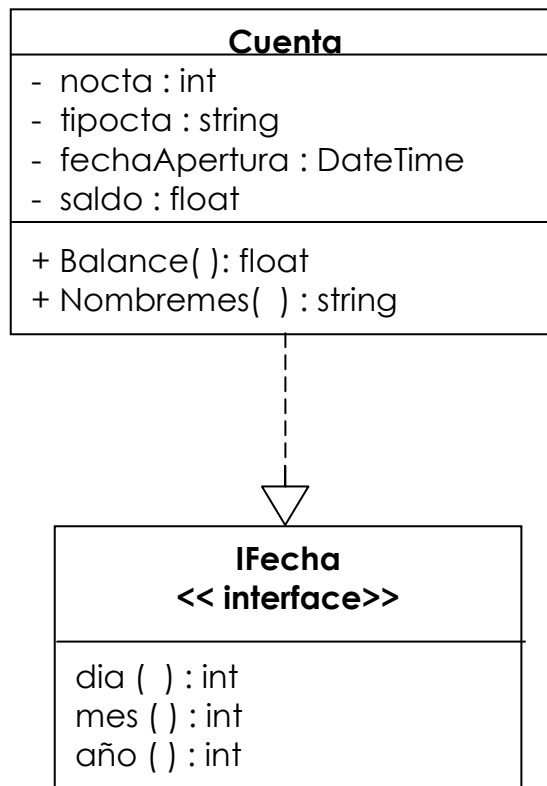
Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Define los datos de una cuenta bancaria y utiliza interfaces implícitas para mostrar la fecha. En este ejercicio utilizarás la referencia cultural para manejo de fechas.

a) Datos :

- no. de cuenta
- tipo de cuenta
- fecha de Apertura
- saldo de la cuenta

b) Diagrama de clases



c) Código C# en consola

CLASE CUENTA

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Globalization;

namespace InterfaceImp
{
    class Cuenta : IFecha
    {
        int nocta;
        string tipocta;
        DateTime fechaApertura;
        float saldo;

        public int NoCta
        {
            get { return nocta; }
            set { nocta = value; }
        }
        public string Tipo
        {
            get { return tipocta; }
            set { tipocta = value; }
        }
        public DateTime FechaApertura
        {
            get { return fechaApertura; }
            set { fechaApertura = value; }
        }
        public float Saldo
        {
            get { return saldo; }
            set { saldo = value; }
        }

        public Cuenta() { }
        public Cuenta(int n, string t, DateTime A, float s)
        {
            nocta = n;
            tipocta = t;
            fechaApertura = A;
            saldo = s;
        }

        public float Balance()
        {
            return saldo;
        }
    }
}
```

```

        // implementación implícita de los métodos de la
interface

    public int dia()
    {
        return FechaApertura.Day;
    }
    public int mes()
    {
        return FechaApertura.Month;
    }
    public int año()
    {
        return FechaApertura.Year;
    }

    public string Nombremes()
    {
        DateTimeFormatInfo formatoFecha =
CultureInfo.CreateSpecificCulture("es-MX").DateTimeFormat;
//CurrentCulture.DateTimeFormat;
        string nombreMes = formatoFecha.GetMonthName(mes());
        return nombreMes;
    }
    }
}

```

INTERFACE IFECHA

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace InterfaceImp
{
    interface IFecha
    {
        int dia();
        int mes();
        int año();
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace InterfaceImp

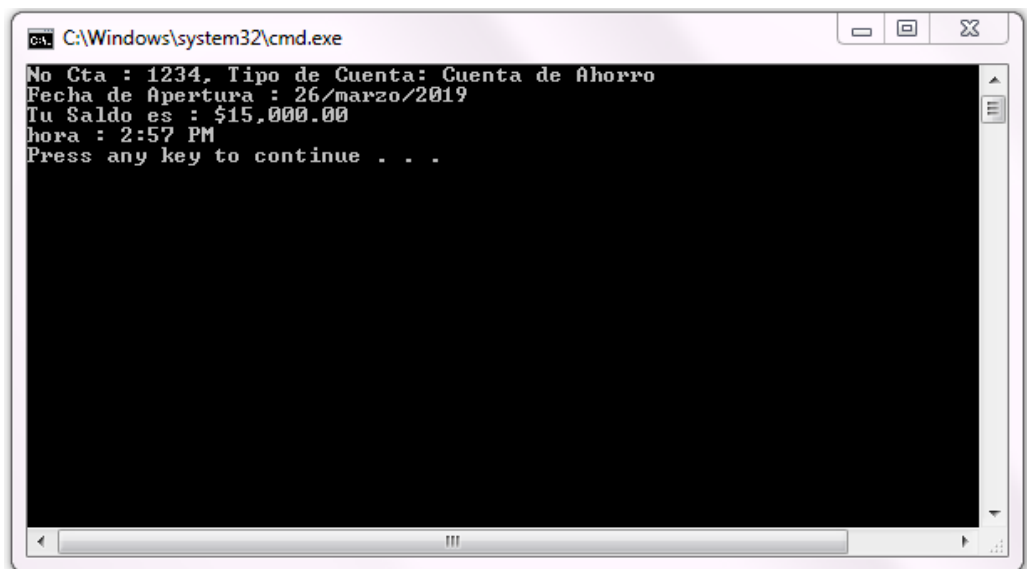
```

```

{
    class Program
    {
        static void Main(string[] args)
        {
            Cuenta c = new Cuenta(1234, "Cuenta de Ahorro",
DateTime.Today, 15000);
            Console.WriteLine("No Cta : {0}, Tipo de Cuenta:
{1}", c.NoCta, c.Tipo);
            Console.WriteLine("Fecha de Apertura : {0}/{1}/{2}",
c.dia(), c.Nombremes(), c.año());
            Console.WriteLine("Tu Saldo es : {0:c2}",
c.Balance());
            Console.WriteLine("hora : {0}",
DateTime.Now.ToShortTimeString());
        }
    }
}

```

d) Corrida o Pantalla de captura



**Sugerencias
didácticas**

Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase.
 Utiliza la definición de interfaces implícitas e impléméntala en la clase.
 Nombrar otros ejemplos de clases para implementar una o más interfaces.
 Realizar otros ejercicios para practicar a partir del diagrama de clases.

Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Interfaces (Guía de Programación C#)</i>. [online] Available at: https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/ Putier, S. (2018). <i>C# 7 y Visual Studio 2017</i>. Cornellà de Llobregat (Barcelona): ENI. Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	TRECE
Nombre	Elabora un programa donde implementes de manera explícita una interface
Competencia específica de la unidad 4	Comprende y aplica los conceptos de herencia y polimorfismo en programas que utilicen clases base, clases derivadas, clases abstractas e interfaces.
Introducción	Una interfaz no encapsula datos, sólo define cuales son los métodos que han de implementar los objetos de aquellas clases que implementen la interfaz. En la declaración de una interfaz, lo único que puede aparecer son declaraciones de métodos (su nombre y asignatura sin su implementación). Por convención, los nombres de las interfaces comienzan con una I mayúscula. Si una clase implementa dos interfaces que contienen un miembro con la misma firma, entonces la implementación de ese miembro en la clase hará que ambas interfaces usen ese miembro como su implementación. Es posible implementar un miembro de la interfaz explícitamente, creando un miembro de clase que solo se llama a través de la interfaz y es específico de esa interfaz. Esto se logra al nombrar al miembro de la clase con el nombre de la interfaz y un punto.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 4. Herencia y Polimorfismo 4.5 Interfaces. 4.5.1 Definición 4.5.2 Implementación
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase

-
- Metodología**
1. Analiza el problema e identifica los elementos necesarios para resolver el problema.
 2. Escribe el código utilizando las interfaces explícitas.
 3. Compila el programa y corrige errores de sintaxis y lógica.
 - 4 Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Define los datos para cada una de las figuras geométricas implementa una interface de manera explícita que tenga como miembro un método para calcular el área.

a) Datos

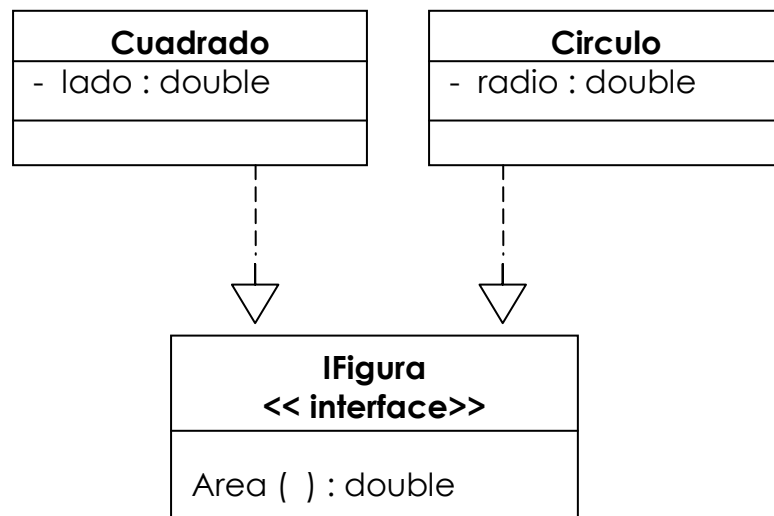
Figura Circulo

- radio
- área

Figura Cuadrado

- lado
- área

b) Diagrama de clases



c) Código C# en consola

CLASE CIRCULO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace InterfaceExp
{
    public class Circulo : IFigura
    {
        double radio;
        public Circulo(double c)
        {
            radio = c;
        }

        // implementación explícita del metodo de la interface

        #region Miembros de IFigura

        double IFigura.Area()
        {
            return Math.PI * Math.Pow(radio, (double)2);
        }

        #endregion
    }
}
```

CLASE CUADRADO

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace InterfaceExp
{
    public class Cuadrado : IFigura
    {
        double lado;
        public Cuadrado(double l)
        {
            lado = l;
        }

        // implementación explícita del metodo de la interface

        #region Miembros de IFigura
```

```

        double IFigura.Area()
        {
            return lado * lado;
        }

        #endregion
    }
}

```

INTERFACE

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace InterfaceExp
{
    public interface IFigura
    {
        double Area();
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

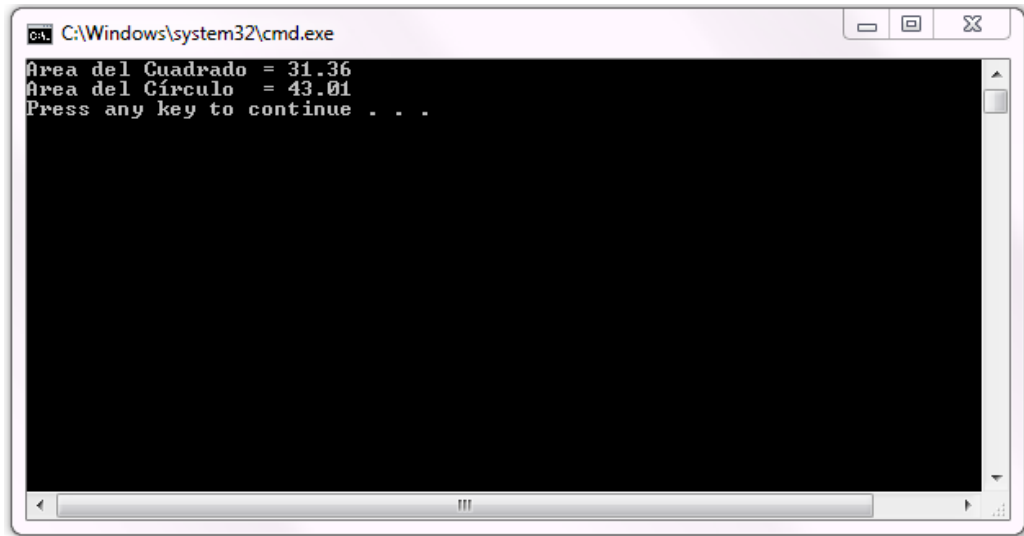
namespace InterfaceExp
{
    class Program
    {
        static void Main(string[] args)
        {
            Cuadrado c=new Cuadrado(5.6);
            // se declara una instancia del tipo de la
            interface y se ejecuta el metodo del tipo
            IFigura area = (IFigura)c;
            Console.WriteLine("Area del Cuadrado = {0:n2}",
            area.Area());

            Circulo ci = new Circulo(3.7);
            IFigura areaC = (IFigura)ci;
            Console.WriteLine("Area del Círculo = {0:n2}",
            areaC.Area());

        }
    }
}

```

d) Corrida o Pantalla de captura



```
C:\Windows\system32\cmd.exe
Area del Cuadrado = 31.36
Area del Círculo = 43.01
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Utiliza la definición de interfaces explícitas e impléméntala en las clases. Nombrar otros ejemplos de clases para implementar una o más interfaces. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i> . Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i> . México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i> . Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i> . Larman, C. (2003). <i>UML y patrones</i> . Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). <i>Implementación de interfaz explícita (guía de programación C#)</i> . [online] Available at:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/interfaces/explicit-interface-implementation>

Putier, S. (2018). *C# 7 y Visual Studio 2017*. Cornellà de Llobregat (Barcelona): ENI.

Walker, L. (2018). *CSharp, Courseware Development Tools*.

No. de Práctica	CATORCE
Nombre	Elabora programa que capture 10 estaturas y las convierta a cm
Competencia específica de la unidad 5	Conoce y aplica programas que implementen el uso de arreglos para reconocerlos como una herramienta para agrupar datos y facilitar la solución de problemas.
Introducción	Un arreglo es una estructura de datos que contiene un número de variables a las que se accede mediante índices calculados. Las variables contenidas en un arreglo, denominadas también elementos del arreglo, son todas del mismo tipo y este tipo se conoce como tipo de elemento del arreglo.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 5. Arreglos 5.1 Unidimensional.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para resolver el problema. 2. Escribe el código utilizando arreglos unidimensionales. 3. Compila el programa y corrige errores de sintaxis y lógica. 4 Ejecuta el código y revisa los resultados. Una solución a esta práctica siguiendo la metodología indicada es : I. Definición : Convierte 10 estaturas de metros a cm. Muestra cada una de las estaturas si mide más de 180 cm indica que es alto, si mide entre 160 a 180 cm es promedio o si mide menos de 160 cm es bajo. al final indica cuantas estaturas son altos, promedio o bajos.

-
- a) Datos
- estaturas en metros

- c) Código C# en consola

CLASE PRINCIPAL

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Estaturas
{
    class Program
    {
        static void Main(string[] args)
        {
            float[] estatura = new float[10];
            float[] estCM = new float[10];
            for (int p = 0; p < 10; p++)
            {
                Console.Write("{0}.- Estatura : ", p + 1);
                estatura[p] = float.Parse(Console.ReadLine());
            }
            for (int p = 0; p < 10; p++)
            {
                estCM[p] = estatura[p] * 100;
            }

            Console.WriteLine("\n\t Estaturas");
            Console.WriteLine("Metros\t\t\t\t\t Centímetros");
            // for (int p = 0; p < 10; p++)
            // {
            //
            // }

            int alto=0,prom=0,bajo=0;
            for (int p = 0; p < 10; p++)
            {
                if (estCM[p] >= 180)
                {
                    Console.Write(estatura[p] + "\t\t" +
estCM[p]);Console.WriteLine("\t es alto");
                    alto++;
                }
                else
                {
                    if (estCM[p] >= 160 && estCM[p] < 180)
                    {
                        Console.Write(estatura[p] + "\t\t" +
+ estCM[p]);Console.WriteLine("\t es promedio");
                    }
                }
            }
        }
    }
}
```

```

        prom++;
    }
    else
    {
        Console.Write(estatura[p] + "\t\t"
+ estCM[p]); Console.WriteLine("\t es bajo");
        bajo++;
    }
}
} // llave de for

Console.WriteLine("\nhay {0} altos",alto);
Console.WriteLine("hay {0} promedio",prom);
Console.WriteLine("hay {0} bajos",bajo);
}
}
}

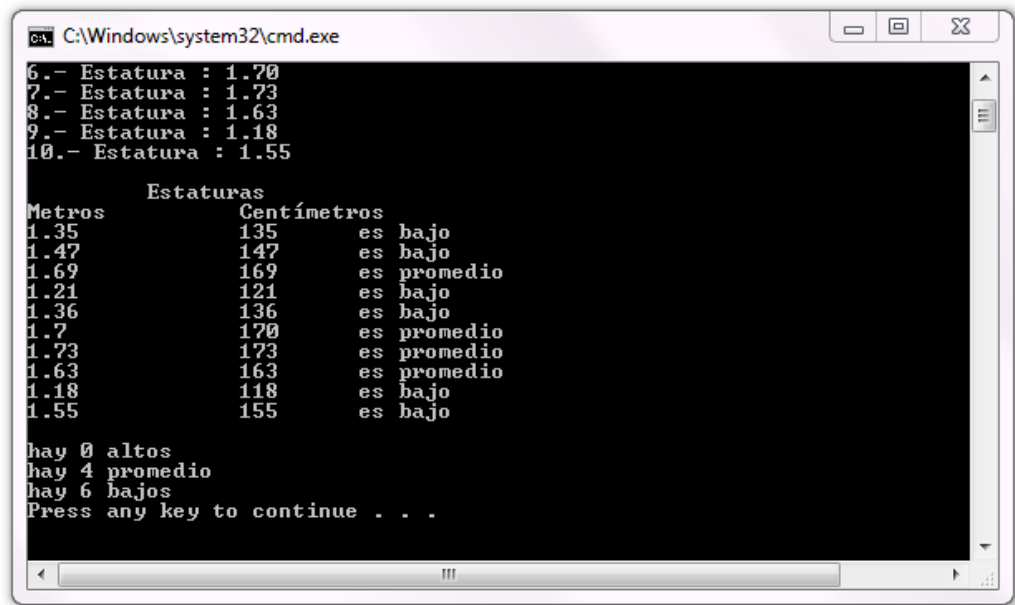
```

d) Corrida o Pantalla de captura

```

C:\Windows\system32\cmd.exe
1.- Estatura : 1.35
2.- Estatura : 1.47
3.- Estatura : 1.69
4.- Estatura : 1.21
5.- Estatura : 1.36
6.- Estatura : 1.70
7.- Estatura : 1.73
8.- Estatura : 1.63
9.- Estatura : 1.18
10.- Estatura :

```



```
C:\Windows\system32\cmd.exe
6.- Estatura : 1.70
7.- Estatura : 1.73
8.- Estatura : 1.63
9.- Estatura : 1.18
10.- Estatura : 1.55

Estaturas
Metros      Centímetros
1.35        135      es bajo
1.47        147      es bajo
1.69        169      es promedio
1.21        121      es bajo
1.36        136      es bajo
1.7         170      es promedio
1.73        173      es promedio
1.63        163      es promedio
1.18        118      es bajo
1.55        155      es bajo

hay 0 altos
hay 4 promedio
hay 6 bajos
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Utiliza las instrucciones de iteración para recorrer arreglos. Realiza operaciones con los elementos de los arreglos. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i>. Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i>. México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i>. Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i>. Larman, C. (2003). <i>UML y patrones</i>. Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). Arreglos (C#). [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/arrays Walker, L. (2018). <i>CSharp, Courseware Development Tools</i>.

No. de Práctica	QUINCE
Nombre	Elabora un programa que capture 5 objetos tipo Estudiante y los almacene en un arreglo
Competencia específica de la unidad 5	Conoce y aplica programas que implementen el uso de arreglos para reconocerlos como una herramienta para agrupar datos y facilitar la solución de problemas.
Introducción	Un arreglo es una estructura de datos que contiene un número de variables a las que se accede mediante índices calculados. Las variables contenidas en un arreglo, denominadas también elementos del arreglo, son todas del mismo tipo y este tipo se conoce como tipo de elemento del arreglo. Los elementos del arreglo también pueden contener objetos del mismo tipo de dato abstracto.
Especificar la correlación con los temas y subtemas del Programa de estudio vigente.	Unidad 5. Arreglos 5.1 Unidimensional.
Material y equipo necesario	▪ Cuaderno y Lápiz ▪ Computadora personal ya sea de escritorio o laptop ▪ Lenguaje de Programación instalado de C# versión 3.0 o posterior ▪ Editor de código Visual Studio 2010 o posterior ▪ Apuntes de la clase
Metodología	1. Analiza el problema e identifica los elementos necesarios para resolver el problema. 2. Escribe el código utilizando arreglos unidimensionales. 3. Compila el programa y corrige errores de sintaxis y lógica. 4 Ejecuta el código y revisa los resultados.

Una solución a esta práctica siguiendo la metodología indicada es :

I. Definición : Elabora un programa que registre los datos de 5 Estudiantes como nombre, carrera, matricula y semestre. Muestra la lista de los estudiantes.

a) Datos

- nombre
- carrera
- matricula
- semestre
- Mostrar los datos

b) Diagrama de clases

Estudiante
- nombre : string - carrera : string - matricula : long - semestre : int
+ MostrarDatos(): void

c) Código C# en consola

CLASE ESTUDIANTE

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Arreglos
{
    class Estudiante
    {
        // atributos
        string nombre, carrera;
        long matricula;
    }
}
```

```

        int semestre;

        // propiedades
        public string Nombre
        {
            get { return nombre; }
            set { nombre = value; }
        }
        public string Carrera
        {
            get { return carrera; }
            set { carrera = value; }
        }
        public long Matricula
        {
            get { return matricula; }
            set { matricula = value; }
        }
        public int Semestre
        {
            get { return semestre; }
            set { semestre = value; }
        }

        // Constructor por defeto
        public Estudiante() { }

        // Constructor sobrecargado
        public Estudiante(string n, string c, long m, int s)
        {
            nombre = n;
            carrera = c;
            matricula = m;
            semestre = s;
        }

        // método publico
        public void MostrarDatos()
        {
            Console.WriteLine("\n\tLista de Estudiantes\n");
            Console.WriteLine("Matricula : {0} \t {1}",
matricula, nombre);
            Console.WriteLine("Carrera   : {0} \t\t\tSemestre :
{1}", carrera, semestre);
        }
    }
}

```

CLASE PRINCIPAL

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

```

```
namespace Arreglos
{
    class Program
    {
        static void Main(string[] args)
        {
            // definir y crear la lista
            Estudiante[] listaAlum = new Estudiante[5];
            Console.WriteLine("Registro del Alumno");

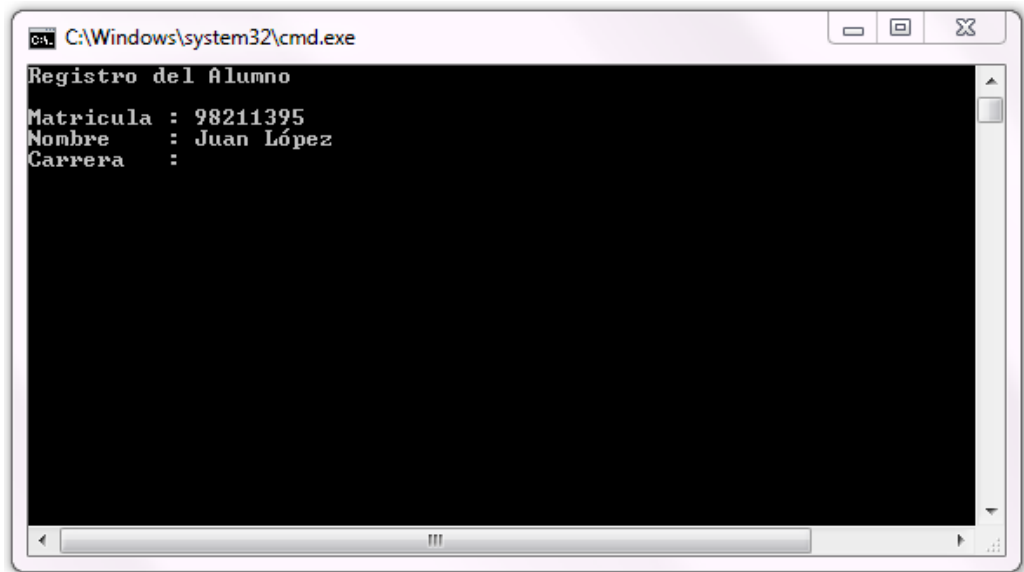
            // Captura de los objetos
            for (int p = 0; p < 5; p++)
            {

                Estudiante Est = new Estudiante();
                Console.Write("\nMatricula : ");
                Est.Matricula = int.Parse(Console.ReadLine());
                Console.Write("Nombre      : ");
                Est.Nombre = Console.ReadLine();
                Console.Write("Carrera    : ");
                Est.Carrera = Console.ReadLine();
                Console.Write("Semestre   : ");
                Est.Semestre = int.Parse(Console.ReadLine());
                listaAlum[p] = Est;    // almacenar en el arreglo
            }

            Console.Clear();

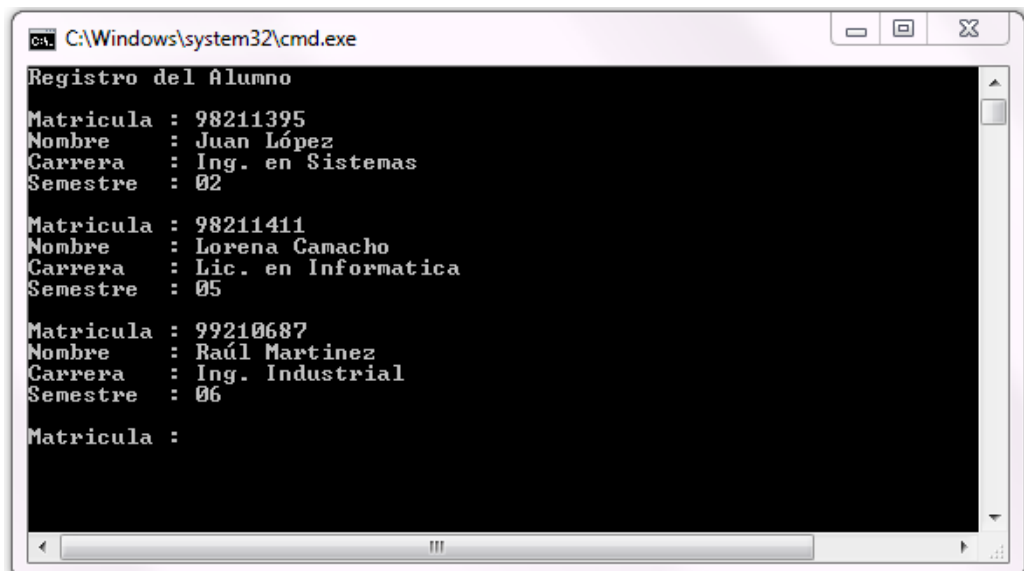
            // mostrar los elementos del arreglo
            foreach (Estudiante e in listaAlum)
            {
                e.MostrarDatos();
            }
        }
    }
}
```

d) Corrida o Pantalla de captura



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the following text:

```
Registro del Alumno
Matricula : 98211395
Nombre    : Juan López
Carrera   :
```



A screenshot of a Windows command prompt window titled "C:\Windows\system32\cmd.exe". The window displays the following text:

```
Registro del Alumno
Matricula : 98211395
Nombre    : Juan López
Carrera   : Ing. en Sistemas
Semestre  : 02

Matricula : 98211411
Nombre    : Lorena Camacho
Carrera   : Lic. en Informatica
Semestre  : 05

Matricula : 99210687
Nombre    : Raúl Martínez
Carrera   : Ing. Industrial
Semestre  : 06

Matricula :
```

```
C:\Windows\system32\cmd.exe

Lista de Estudiantes
Matricula : 98211395 Juan López
Carrera : Ing. en Sistemas Semestre : 2
Matricula : 98211411 Lorena Camacho
Carrera : Lic. en Informatica Semestre : 5
Matricula : 99210687 Raúl Martinez
Carrera : Ing. Industrial Semestre : 6
Matricula : 99215632 Rosa Toledo
Carrera : Ing. en TICS Semestre : 3
Matricula : 98211746 Carlos Pratt
Carrera : Ing. en Sistemas Semestre : 5
Press any key to continue . . .
```

Sugerencias didácticas	Para realizar esta práctica es necesario conocer y entender los conceptos vistos en clase. Utiliza las instrucciones de iteración para recorrer arreglos. Realiza operaciones con los elementos de los arreglos. Almacena elementos de tipos de datos abstractos. Realizar otros ejercicios para practicar a partir del diagrama de clases.
Reporte del Alumno	El Alumno entregará el diagrama de clases en UML, el código del programa y la captura de pantallas de los resultados del programa.
Bibliografía	Bell, D. and Parr, M. (2011). <i>C# para estudiantes</i> . Pearson Educación de México, S.A. de C.V. Ceballos Sierra, F. (2012). <i>Microsoft C#</i> . México, D.F.: Alfaomega. Ferguson, J., Patterson, B., Beres, J., Boutquin, P. and Gupta, M. (2018). <i>La Biblia de C#</i> . Anaya. Gonzalez Seco, J. (2012). <i>El lenguaje de programación C#</i> . Larman, C. (2003). <i>UML y patrones</i> . Naucalpan de Juárez: Pearson Educación. Docs.microsoft.com. (2019). Arreglos (C#). [online] Available at: https://docs.microsoft.com/es-es/dotnet/csharp/tour-of-csharp/arrays . Walker, L. (2018). <i>CSharp, Courseware Development Tools</i> .
