



CEI

Dr. Edye 656 y Rincón, Maldonado

Analista Programador

Programación II

Prof. Gonzalo Duarte

Valentina Benítez

Maldonado, 12 de septiembre de 2023

## Resumen

El actual proyecto se desarrolló empleando el lenguaje C# en el IDE *Visual Studio Community*. La propuesta consta de resolver una serie de 6 ejercicios de programación con la implementación de la Programación Orientada a Objetos. Estos ejercicios tratan acerca de buscar la manera de acceder a datos de atributos privados y poder manipularlos, sobre la suma, resta y distancia entre dos vectores, también de hallar la forma de sumar dos matrices y acerca de una empresa que quiere guardar sus documentos digitalmente para poder imprimirlos cuando desee. Para abordar cada uno de los problemas se utilizaron las siguientes herramientas: clases, atributos, métodos, objetos y *arrays*. Dentro de la categoría métodos se utilizó el método constructor y los métodos *getter* y *setter*.

# Contenido

<b>1. Introducción.....</b>	<b>1</b>
1.1 Qué es un IDE.....	1
1.2 POO - Programación Orientada a Objetos.....	1
1.3 Clases y Objetos.....	1
<b>2. Desarrollo.....</b>	<b>2</b>
2.1 Construcción de un auto.....	2
2.2 Impresión en pantalla.....	4
2.3 Suma y resta de vectores.....	6
2.4 Distancia entre los puntos de dos vectores.....	9
2.5 Suma de matrices.....	10
2.6 Repositorio digital.....	12
<b>3. Conclusiones.....</b>	<b>19</b>
<b>4. Referencias.....</b>	<b>20</b>

# 1. Introducción

A continuación se agrega la dirección en donde se encuentra el código resultado de cada problemática: <https://github.com/ValentiinaBenitez/primeraRepo>

## 1.1 Qué es un IDE

Un IDE (*Integrated Development Environment*) es un entorno de desarrollo para escribir código de manera eficiente con herramientas de construcción, compilación, depuración y gestión de proyectos. [1]

## 1.2 POO - Programación Orientada a Objetos

La Programación Orientada a Objetos (POO) es un paradigma, es decir, un modelo de programación que está basado en clases y objetos. La POO consiste en trasladar la naturaleza de los objetos de la vida real como su comportamiento, propiedades y estado a la programación, además incorporar el relacionamiento entre ellos. Una de las ventajas de este modelo de programación es la modularización del código, poder dividir el programa en partes o clases.

## 1.3 Clases y Objetos

Una clase es un modelo de código o plantilla donde se definen las características comunes de un grupo de objetos. Por otro lado los objetos son entidades creadas desde una clase que contienen atributos y métodos.

Los atributos y los métodos son las dos partes que componen una clase. Cuando se habla de atributos se hace referencia a las características de un objeto que funcionan como variables y en cuanto a los métodos, representan los comportamientos o acciones que puede emplear el objeto. Ambas estructuras se definen dentro de la clase.

Existe un modelo de método específico llamado método constructor que es el encargado de crear el objeto con sus atributos, no se puede prescindir de él y se lo destaca por tener el mismo nombre que la clase. Una característica de esta clase de método es que puede estar sobrecargado, es decir, pueden existir varios métodos con el mismo nombre sin embargo, todos deben recibir parámetros diferentes .

Además hay otros dos modelos de métodos denominados *getter* y *setter* que se utilizan para obtener o cambiar datos de un atributo privado. [2, 3, 4, 5]

## 2. Desarrollo

### 2.1 Construcción de un auto

Se pide realizar una clase “auto” que dentro lleve el método constructor sobrecargado. Además se deben incluir métodos *getter* y *setter* para ver y modificar el tipo de tapicería y tipo de climatizador.

#### Solución

Para realizar el código resultado de este problema se comenzó creando la clase Auto con sus determinados atributos, en donde se declaró específicamente como lo pide la letra climatizador y tapicería. A continuación se creó el método constructor base, sin ningún parámetro y se les asignó valores a los atributos, debido a que se solicitó un método constructor sobrecargado se pensó en realizar dos adicionales que reciban ambos diferentes parámetros. Luego se planteó el método *getter* sobre climatizador y tapicería por separado y el método *setter* en conjunto.

```
9 referencias
class Auto
{
    private int ruedas;
    private double largo;
    private double ancho;
    private bool climatizador;
    private string tapicería;
    private string color;
    private int año;

    1 referencia
    public Auto()
    {
        ruedas = 4;
        largo = 3;
        ancho = 2;
        climatizador = false;
        tapicería = "tela";
        color = "negro";
        año = 2021;
    }
}
```

Figura 2.1: La clase con sus atributos y el método constructor base.

```

1 referencia
public Auto(double largo, double ancho)
{
    ruedas = 4;
    this.largo = largo;
    this.ancho = ancho;
    climatizador = true;
    tapicería = "tela";
    color = "negro";
    año = 2021;
}

1 referencia
public Auto(string color, int año)
{
    ruedas = 4;
    largo = 3;
    ancho = 2;
    climatizador = false;
    tapicería = "tela";
    this.color = color;
    this.año = año;
}

```

Figura 2.2: Los métodos constructores con parámetros.

```

5 referencias
public bool getClimatizador()
{
    return climatizador;
}

5 referencias
public string getTapicería()
{
    return tapicería;
}

2 referencias
public void setClimatizadorTapicería(bool climatizador, string tapicería)
{
    this.climatizador = climatizador;
    this.tapicería = tapicería;
}

```

Figura 2.3: *Getter* y *setter*.

Una vez que se definió la clase y sus métodos se pasó a programar el sector *Program*. Se empleó la clase *Auto* para crear tres objetos con diferente identificador y por cada uno se utilizó un método constructor. Seguidamente se utilizó *Console.WriteLine()* y los métodos *getter* para imprimir en pantalla el tipo de climatizador y tapicería de cada objeto. Continuando con la solución a los requerimientos del problema se utilizó el método *setter* sobre dos de los objetos, en donde se cambian los valores de climatizador y tapicería. Finalizando el código se realizó la impresión del cambio en pantalla mediante *Console.WriteLine()* y el método *getter*.

```

0 referencias
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        Auto auto1 = new Auto();
        Auto auto2 = new Auto(3.6, 2.4);
        Auto auto3 = new Auto("blanco", 2022);

        Console.WriteLine("AUTO 1 - Climatizador: " + auto1.getClimatizador() + " / Tapicería: " + auto1.getTapicería());
        Console.WriteLine("AUTO 2 - Climatizador: " + auto2.getClimatizador() + " / Tapicería: " + auto2.getTapicería());
        Console.WriteLine("AUTO 3 - Climatizador: " + auto3.getClimatizador() + " / Tapicería: " + auto3.getTapicería());

        auto1.setClimatizadorTapicería(true, "algodón");
        auto3.setClimatizadorTapicería(true, "microfibra");

        Console.WriteLine("AUTO 1 - Climatizador: " + auto1.getClimatizador() + " / Tapicería: " + auto1.getTapicería());
        Console.WriteLine("AUTO 3 - Climatizador: " + auto3.getClimatizador() + " / Tapicería: " + auto3.getTapicería());
    }
}

```

Figura 2.4: La creación de objetos con diferente constructor y el uso de los métodos *getter* y *setter*.

## Solución en consola

```

C:\WINDOWS\system32\cmd. X + v
- □ X
AUTO 1 - Climatizador: False / Tapicería: tela
AUTO 2 - Climatizador: True / Tapicería: tela
AUTO 3 - Climatizador: False / Tapicería: tela
AUTO 1 - Climatizador: True / Tapicería: algodón
AUTO 3 - Climatizador: True / Tapicería: microfibra

```

Figura 2.5: Utilización del método *setter*.

## 2.2 Impresión en pantalla

Se solicita la implementación de un código en donde se imprima en pantalla el nombre de un alumno o el nombre y la edad según se prefiera. Se debe de crear una clase denominada alumno que contenga uno o varios métodos constructores, un método para imprimir el nombre y otro para el nombre y la edad. Se permite la utilización de los métodos *getter* y *setter* si es necesario.

### Solución

La resolución que se ideó se basa en la creación de una clase con el identificador "Alumno" que contiene dos atributos denominados alumno y edad, dos métodos constructores también llamados Alumno que reciben diferentes parámetros, un *getter* para cada atributo, un método que con el uso de *Console.WriteLine()* imprima nombre y otro que imprima nombre y edad, ambos reciben un objeto como parámetro. En estos dos últimos métodos se

utilizó el método *getter* para obtener la información que corresponde al atributo del objeto que se quiere imprimir en pantalla.

```
10 referencias
class Alumno
{
    private string alumno;
    private int edad;
```

Figura 2.6: La clase con sus atributos.

```
1 referencia
public Alumno(string alumno)
{
    this.alumno = alumno;
}
2 referencias
public Alumno(string alumno, int edad)
{
    this.alumno = alumno;
    this.edad = edad;
}
```

Figura 2.7: Método constructor sobrecargado.

```
2 referencias
public string getAlumno()
{
    return alumno;
}
1 referencia
public int getEdad()
{
    return edad;
}
```

Figura 2.8: *Getter*.

```
2 referencias
public void imprimirEdadNombre(Alumno alumno)
{
    Console.WriteLine("El nombre del alumno es: " + alumno.getAlumno() + " y la edad es: " + alumno.getEdad());
}
1 referencia
public void imprimirNombre(Alumno alumno)
{
    Console.WriteLine("El nombre del alumno es: " + alumno.getAlumno());
}
```

Figura 2.9: Métodos para imprimir en pantalla el nombre o el nombre y la edad.



Con la clase junto a sus atributos y métodos ya creada se pasó a diseñar el sector *Program*. Para comenzar se llamó a la clase Alumno y se creó un objeto, se identificó como Alumno1 y se decidió usar el método constructor que utiliza dos parámetros por lo que a través de paréntesis se escribió un valor para nombre y otro para edad. Luego se realizó el mismo procedimiento pero cambiando el identificador de objeto a Alumno2 y con diferente constructor, entonces se escribió solo el valor de nombre. Para finalizar se escribió la porción de código que imprime en pantalla la información de los objetos mediante sus correspondientes métodos, para Alumno1 se utilizó el método que imprime el nombre y la edad y para Alumno2 se utilizó el método que imprime solo el nombre.

```
0 referencias
internal class Program
{
    0 referencias
    static void Main(string[] args)
    {
        Alumno alumno1 = new Alumno("Alaska", 1);
        Alumno alumno3 = new Alumno("Odin");

        alumno1.imprimirEdadNombre(alumno1);
        alumno3.imprimirNombre(alumno3);
    }
}
```

Figura 2.10: Creación de objetos con diferente constructor y los métodos para imprimir en pantalla.

## Solución en consola



```
C:\WINDOWS\system32\cmd. X
El nombre del alumno es: Alaska y la edad es: 1
El nombre del alumno es: Odin
```

Figura 2.11: Impresión en pantalla de los datos de los alumnos.

## 2.3 Suma y resta de vectores.

Se solicita realizar un código en donde se imprima la suma y la resta de dos vectores.

## Solución

Para abordar este problema se creó una clase identificada como Vector, con dos atributos X e Y, un método constructor también llamado Vector que recibe como parámetros los mencionados atributos y dos métodos *getter*, uno para cada atributo.

```
5 referencias
class Vector
{
    private int x;
    private int y;

    2 referencias
    public Vector (int x, int y)
    {
        this.x = x;
        this.y = y;
    }

    6 referencias
    public int getX ()
    {
        return x;
    }

    6 referencias
    public int getY ()
    {
        return y;
    }
}
```

Figura 2.12: La clase con atributos, método constructor y métodos *getter*.

Luego se pasó al sector *Program* donde se realizó el código para la suma y resta de dos vectores. Se pensó en establecer una variable denominada *primerValor* en donde se le solicitó al usuario que ingrese un valor para X, a continuación se estableció otra variable con el nombre *segundoValor* para Y en la que también se pidió el valor. Al ingresar un dato desde consola este se guarda como un dato de tipo *string* y de esta manera no es posible realizar operaciones matemáticas entonces para solucionar esto se le añadió a las variables el concepto *Convert.ToInt32()* que convierte el dato de tipo *string* a uno tipo *int*. Después de que se solicitó los valores para el primer vector se pasó a crear el objeto “vectorUno” con el método *Vector* y como parámetros las variables anteriores. Para el segundo vector se realizó el mismo procedimiento pero con la diferencia de que al nombre del objeto se le denominó “vectorDos”.

```

0 referencias
static void Main(string[] args)
{
    Console.WriteLine("Cree el primer vector");
    Console.Write("Primer valor: ");
    int primerValor = Convert.ToInt32(Console.ReadLine());
    Console.Write("Segundo valor: ");
    int segundoValor = Convert.ToInt32(Console.ReadLine());
    Vector vectorUno = new Vector(primerValor, segundoValor);

    Console.WriteLine("Cree el segundo vector");
    Console.Write("Primer valor: ");
    primerValor = Convert.ToInt32(Console.ReadLine());
    Console.Write("Segundo valor: ");
    segundoValor = Convert.ToInt32(Console.ReadLine());
    Vector vectorDos = new Vector(primerValor, segundoValor);
}

```

Figura 2.13: Asignación de valores y creación de dos vectores.

Para la suma de los vectores se pensó en crear una variable llamada SumaX en donde se utilizó el método *getter* para obtener los valores de X de ambos vectores y sumarlos, del mismo modo ocurrió para los valores de Y en la variable denominada SumaY. En el caso de la resta se llevó a cabo de la misma manera pero cambiando el nombre de las variables por RestaX y RestaY. Como última instancia se escribió dos *Console.WriteLine()* para mostrar en pantalla la suma y la resta de ambos vectores.

```

int SumaX = (vectorUno.getX() + vectorDos.getX());
int SumaY = (vectorUno.getY() + vectorDos.getY());

int RestaX = (vectorUno.getX() - vectorDos.getX());
int RestaY = (vectorUno.getY() - vectorDos.getY());

Console.WriteLine("La suma de los vector es (" + SumaX + ", " + SumaY + ")");
Console.WriteLine("La resta de los vector es (" + RestaX + ", " + RestaY + ")");

```

Figura 2.14: Suma y resta de los vectores con su respectivo *ConsoleWriteLine()*.

## Solución en consola



```

C:\WINDOWS\system32\cmd. x + v
Cree el primer vector
Primer valor: 2
Segundo valor: 4
Cree el segundo vector
Primer valor: 6
Segundo valor: 8
La suma de los vector es (8, 12)
La resta de los vector es (-4, -4)

```

Figura 2.15: Elección de valores y la suma y resta de los vectores.

## 2.4 Distancia entre los puntos de dos vectores

Se pide añadir en el código anterior el cálculo de la distancia entre los puntos de los vectores creados.

### Solución

Este problema se resolvió como dice la letra, siguiendo el código anterior, por lo que se utilizó las mismas herramientas y se escribió en el mismo proyecto.

Cálculo de la distancia entre los puntos de dos vectores:

$$\sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

Figura 2.16: Cálculo de la distancia entre los puntos de dos vectores.

Para efectuar el cálculo de la distancia entre los puntos de los vectores se pensó en crear dos variables denominadas `distanciaVectorX` y `distanciaVectorY`, dentro de `distanciaVectorX` se restó el valor de X del segundo vector con el valor de X del primer vector y dentro de `distanciaVectorY` se restó el valor de Y del segundo vector con el valor de Y del primer vector. Luego se estableció una variable de tipo *double* denominada `moduloVector` para guardar el resultado final, dentro se definió el concepto *Math.Sqrt()*, que se utiliza para calcular la raíz cuadrada, y se le agregó la multiplicación de `distanciaVectorX` por sí mismo menos `distanciaVectorY` por sí mismo. Por último se imprimió en pantalla el resultado haciendo uso de *Console.WriteLine()*.

```
int distanciaVectorX = vectorDos.getX() - vectorUno.getX();  
int distanciaVectorY = vectorDos.getY() - vectorUno.getY();  
  
double moduloVector = Math.Sqrt((distanciaVectorX * distanciaVectorX) + (distanciaVectorY * distanciaVectorY));  
Console.WriteLine("La distancia entre los puntos de los vectores es de " + moduloVector);
```

Figura 2.17: Obtención de la distancia entre dos vectores mediante el concepto *Math.Sqrt()*.

## Solución en consola

La distancia entre los puntos de los vectores es de 5,65685424949238

Figura 2.18: La distancia entre los vectores anteriormente creados.

## 2.5 Suma de matrices

Se requiere crear un programa en donde se sumen dos matrices de valores enteros elegidos de forma aleatoria.

### Solución

Para llevar a término este problema se creó la clase *Matriz* con un atributo denominado “d” que hace referencia a las dimensiones, el método constructor de la matriz y un método *getter* para obtener los valores guardados dentro de ella.

```
10 referencias
class Matriz
{
    private int d;
    3 referencias
    public Matriz(int d)
    {
        this.d = d;
    }

    3 referencias
    public string getInfo()
    {
        return $"{d}";
    }
}
```

Figura 2.19: La clase con su atributo, el método constructor y el método *getter*.

Una vez que se creó la clase se pasó a diseñar en la clase *Program* el funcionamiento. En un principio se determinó la variable “numero” y se creó la matriz de 2x2 denominada *matriz1*. Para definir los valores dentro de la matriz se le concedió la posibilidad al usuario de elegirlos, por lo que, a través de un bucle *for* se recorrió esa matriz y en cada vuelta haciendo uso de la variable anteriormente nombrada, se guardó el dato ingresado que luego se agregó a la matriz. De la misma manera sucedió en la creación de la segunda matriz la cual se le denominó *matriz2*.

```

0 referencias
static void Main(string[] args)
{
    int numero;

    Console.WriteLine("Primera matriz");
    Matriz[,] matriz1 = new Matriz[2,2];

    for (int i = 0; i < matriz1.GetLength(0); i++)
    {
        for (int j = 0; j < matriz1.GetLength(1); j++)
        {
            Console.WriteLine("Escriba el valor del número");
            numero = Convert.ToInt32(Console.ReadLine());
            matriz1[i, j] = new Matriz(numero);
        }
    }

    Console.WriteLine("Segunda matriz");
    Matriz[,] matriz2 = new Matriz[2, 2];

    for (int i = 0; i < matriz2.GetLength(0); i++)
    {
        for (int j = 0; j < matriz2.GetLength(1); j++)
        {
            Console.WriteLine("Escriba el valor del número");
            numero = Convert.ToInt32(Console.ReadLine());
            matriz2[i, j] = new Matriz(numero);
        }
    }
}

```

Figura 2.20: Creación de las dos matrices y asignación de valores.

Después de establecer las matrices se pasó a realizar la suma, para esto se creó otra matriz denominada `matrizResultado` del mismo tamaño y se declaró otra variable llamada `numero1`. Con el uso del bucle `for` se recorrió `matrizResultado` donde en cada instancia mediante el método `getter` para obtener valores y `numero1` para guardar el dato, se sumó el valor de `matriz1` y `matriz2` que se almacenó anteriormente en el mismo lugar que la presente vuelta del bucle y se le asignó el valor de `numero1` a `matrizResultado`.

```

Matriz[,] matrizResultado = new Matriz[2, 2];
int numero1;
for (int i = 0; i < matrizResultado.GetLength(0); i++)
{
    for (int j = 0; j < matrizResultado.GetLength(1); j++)
    {
        numero1 = int.Parse(matriz1[i, j].getInfo()) + int.Parse(matriz2[i, j].getInfo());
        matrizResultado[i, j] = new Matriz(numero1);
    }
}

```

Figura 2.21: Suma de matrices a través del bucle `for`, método `getter` y una variable.

Para terminar se imprimió en pantalla con `Console.WriteLine()` `matrizResultado`, donde nuevamente se hizo uso del bucle `for` para recorrer la matriz y en cada cambio de vuelta con el método `getter` se obtuvo el valor guardado correspondiente.

```
Console.WriteLine("Suma de las matrices:");

for (int i = 0; i < matrizResultado.GetLength(0); i++)
{
    for (int j = 0; j < matrizResultado.GetLength(1); j++)
    {
        Console.Write($" {matrizResultado[i, j].getInfo()}");
    }
    Console.WriteLine("");
}
```

Figura 2.22: Impresión en pantalla del resultado de la suma de las matrices.

## Solución en consola

```
Primera matriz
Escriba el valor del número
3
Escriba el valor del número
5
Escriba el valor del número
1
Escriba el valor del número
7
Segunda matriz
Escriba el valor del número
3
Escriba el valor del número
6
Escriba el valor del número
9
Escriba el valor del número
0
Suma de las matrices:
6 11
10 7
```

Figura 2.23: Elección de valores de dos matrices y su suma.

## 2.6 Repositorio digital

Se requiere llevar a cabo un repositorio digital para agrupar documentos de una empresa y poder imprimirlos cuando se necesiten. Se solicitaron las siguientes clases con sus atributos y métodos:

Impresora	Remito	Factura	FacturaLuz	Municipal	ReciboSueldo
+ Imprimir(Factura): void + Imprimir(FacturaLuz): void + Imprimir(Municipal): void + Imprimir(ReciboSueldo): void + Imprimir(Remito): void	- CantBultos: int - Fecha: int - Numero: int	- Fecha: int - Importe: int	- CodigoPago: int - Importe: int	- Importe: int - Partida: int	- Legajo: int - Total: int

Figura 2.24: Clases y atributos.

## Solución

Para comenzar con la solución se creó la clase Remito, la clase Factura, la clase FacturaLuz, la clase Municipal y la clase ReciboSueldo con sus respectivos atributos privados como se observa en la figura 2.24, seguido a esto se estableció el método constructor de cada una.

```
4 referencias
internal class Remito
{
    private int cantBultos;
    private string fecha;
    private int número;

    1 referencia
    public Remito(int cantBultos, string fecha, int número)
    {
        this.cantBultos = cantBultos;
        this.fecha = fecha;
        this.número = número;
    }
}
```

Figura 2.25: Clase Remito con sus atributos y su método constructor.

```
4 referencias
internal class Factura
{
    private string fecha;
    private int importe;

    1 referencia
    public Factura(string fecha, int importe)
    {
        this.fecha = fecha;
        this.importe = importe;
    }
}
```

Figura 2.26: Clase Factura con sus atributos y su método constructor.

```
4 referencias
internal class FacturaLuz
{
    private int codigoPago;
    private int importe;

    1 referencia
    public FacturaLuz(int codigoPago, int importe)
    {
        this.codigoPago = codigoPago;
        this.importe = importe;
    }
}
```

Figura 2.27: Clase FacturaLuz con sus atributos y su método constructor.



```

4 referencias
internal class Municipal
{
    private int importe;
    private int partida;

    1 referencia
    public Municipal(int importe, int partida)
    {
        this.importe = importe;
        this.partida = partida;
    }
}

```

Figura 2.28: Clase Municipal con sus atributos y su método constructor.

```

4 referencias
internal class ReciboSueldo
{
    private int legajo;
    private int total;

    1 referencia
    public ReciboSueldo(int legajo, int total)
    {
        this.legajo = legajo;
        this.total = total;
    }
}

```

Figura 2.29: Clase ReciboSueldo con sus atributos y su método constructor.

Luego a cada clase se le agregó su *getter* y *setter*.

```

1 referencia
public string getFecha()
{
    return fecha;
}

1 referencia
public int getImporte()
{
    return importe;
}

0 referencias
public void setFecha(string fecha)
{
    this.fecha = fecha;
}

0 referencias
public void setImporte(int importe)
{
    this.importe = importe;
}

```

Figura 2.30: *Getter* y *setter* de la clase Factura.

```

public int getCantBultos()
{
    return cantBultos;
}

1 referencia
public string getFecha()
{
    return fecha;
}

1 referencia
public int getNúmero()
{
    return número;
}

0 referencias
public void setCantBultos(int cantBultos)
{
    this.cantBultos = cantBultos;
}

0 referencias
public void setFecha(string fecha)
{
    this.fecha = fecha;
}

0 referencias
public void setNúmero(int número)
{
    this.número = número;
}

```

Figura 2.31: *Getter* y *setter* de la clase Remito.

```

1 referencia
public int getCodigoPago()
{
    return codigoPago;
}

1 referencia
public int getImporte()
{
    return importe;
}

0 referencias
public void setCodigoPago(int codigoPago)
{
    this.codigoPago = codigoPago;
}

0 referencias
public void setImporte(int importe)
{
    this.importe = importe;
}

```

Figura 2.32: *Getter* y *setter* de la clase FacturaLuz.

```

1 referencia
public int getImporte()
{
    return importe;
}

1 referencia
public int getPartida()
{
    return partida;
}

0 referencias
public void setImporte(int importe)
{
    this.importe = importe;
}

0 referencias
public void setPartida(int partida)
{
    this.partida = partida;
}

```

Figura 2.33: *Getter* y *setter* de la clase Municipal.

```

1 referencia
public int getLegajo()
{
    return legajo;
}

1 referencia
public int getTotal()
{
    return total;
}

0 referencias
public void setLegajo(int legajo)
{
    this.legajo = legajo;
}

0 referencias
public void setTotal(int total)
{
    this.total = total;
}

```

Figura 2.34: *Getter* y *setter* de la clase ReciboSueldo.

Una vez se creó la clase de cada documento se pasó a realizar la clase “Impresora”, en donde se realizó el método “imprimir” sobrecargado y se le declaró a cada uno como parámetro la clase con el objeto de cada documento. Dentro de estos métodos haciendo uso del *Console.WriteLine()* y los métodos *getter* respectivos a la clase se obtuvo e imprimió en pantalla los datos del objeto.

```

internal class Impresora
{
    1 referencia
    public void imprimir(Remito remito)
    {
        Console.WriteLine("La cantidad de bultos es " + remito.getCantBultos() + ", la fecha es " + remito.getFecha() + " y el número es " + remito.getNúmero());
    }
    1 referencia
    public void imprimir(Factura factura)
    {
        Console.WriteLine("La fecha de la factura es " + factura.getFecha() + " y el importe es de " + factura.getImporte());
    }
    1 referencia
    public void imprimir(FacturaLuz factura)
    {
        Console.WriteLine("El código de la factura de la luz es " + factura.getCodigoPago() + " y el importe es de " + factura.getImporte());
    }
    1 referencia
    public void imprimir(Municipal municipal)
    {
        Console.WriteLine("El importe del municipal es " + municipal.getImporte() + " y la partida es " + municipal.getPartida());
    }
    1 referencia
    public void imprimir(ReciboSueldo reciboSueldo)
    {
        Console.WriteLine("El legajo del recibo de sueldo es " + reciboSueldo.getLegajo() + " y el total es de " + reciboSueldo.getTotal());
    }
}

```

Figura 2.35: Clase impresora con el método imprimir sobrecargado.

Después de crear las herramientas necesarias se pasó a producir el código en el sector *Program*. Para comenzar se creó un objeto con la clase Remito denominado remito1 y se le añadió valores, a continuación se creó un objeto con la clase Impresora denominado i1 y luego a i1 se le asignó el método imprimir con el objeto remito1 como parámetro. Esto se hizo con todos los tipos de documentos como se observa en la figura 2.36. Al término del código se utilizó el método *setter* en el objeto remito1 y se le cambió los valores, esto llevó a crear otro objeto con la clase Impresora denominado i6, al cual se le asignó el método imprimir con el parámetro remito1 para confirmar su cambio.

```

static void Main(string[] args)
{
    Remito remito1 = new Remito(12, "13 de mayo", 10);
    Impresora i1 = new Impresora();
    i1.imprimir(remito1);

    Factura factural = new Factura("5 de septiembre", 300);
    Impresora i2 = new Impresora();
    i2.imprimir(factural);

    FacturaLuz facturaluz1 = new FacturaLuz(4117, 1200);
    Impresora i3 = new Impresora();
    i3.imprimir(facturaluz1);

    Municipal municipal1 = new Municipal(2000, 65);
    Impresora i4 = new Impresora();
    i4.imprimir(municipal1);

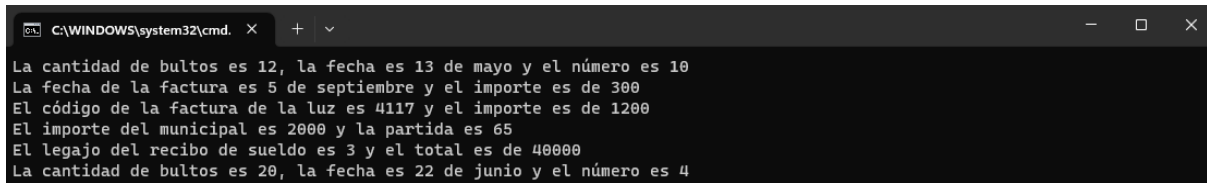
    ReciboSueldo reciboSueldo1 = new ReciboSueldo(3, 40000);
    Impresora i5 = new Impresora();
    i5.imprimir(reciboSueldo1);

    remito1.setCantBultos(20);
    remito1.setFecha("22 de junio");
    remito1.setNúmero(4);
    Impresora i6 = new Impresora();
    i6.imprimir(remito1);
}

```

Figura 2.36: Creación de objetos de cada tipo de documento y utilización del método imprimir de la clase Impresora.

## Solución en consola



```
C:\WINDOWS\system32\cmd. X + v
La cantidad de bultos es 12, la fecha es 13 de mayo y el número es 10
La fecha de la factura es 5 de septiembre y el importe es de 300
El código de la factura de la luz es 4117 y el importe es de 1200
El importe del municipal es 2000 y la partida es 65
El legajo del recibo de sueldo es 3 y el total es de 40000
La cantidad de bultos es 20, la fecha es 22 de junio y el número es 4
```

Figura 2.37: Impresión de cada documento.

### 3. Conclusiones

Como se observa en el desarrollo de cada solución a través de la ejecución de cada código se comprobó que su funcionamiento se adecua perfectamente a lo esperado, además se cumplió con los requerimientos individuales de cada problema y se logró llevar a cabo la implementación de la Programación Orientada a Objetos de manera eficiente. Una desventaja de algunos de los códigos y aspecto a mejorar es su escasa propuesta hacía el usuario de ofrecerle la oportunidad de elegir valores para los atributos, por ejemplo cuando se imprime en pantalla el nombre y la edad de alumnos, en las dimensiones de las matrices del ejercicio de suma de matrices y también en los valores de los documentos del repositorio. Otros puntos a mejorar son en la construcción del auto donde se puede agregar la construcción de una moto con setters en otros atributos y en las operaciones con vectores en donde se puede agregar la división y multiplicación. En conclusión los códigos realizados solucionan los problemas presentados sin embargo, se pueden ampliar para ofrecer un mejor y más sustancioso producto.

## 4. Referencias

[1] - <https://immune.institute/blog/que-es-un-ide/>

[2] - <https://www.youtube.com/watch?v=xz7gn9UXdrA>

[3] - <https://tecno-simple.com/que-es-un-atributo-en-poo-definicion-y-ejemplos/>

[4]

<https://www.fundamentosprogramacion.site/2023/01/que-es-un-metodo-en-programacion.html>

[5]

<https://soka.gitlab.io/csharp/basico/poo/desc-acceso-get-set/desc-acceso-get-set/#:~:text=Los%20descriptores%20de%20acceso%20get,valores%20de%20entrada%20por%20ejemplo.>