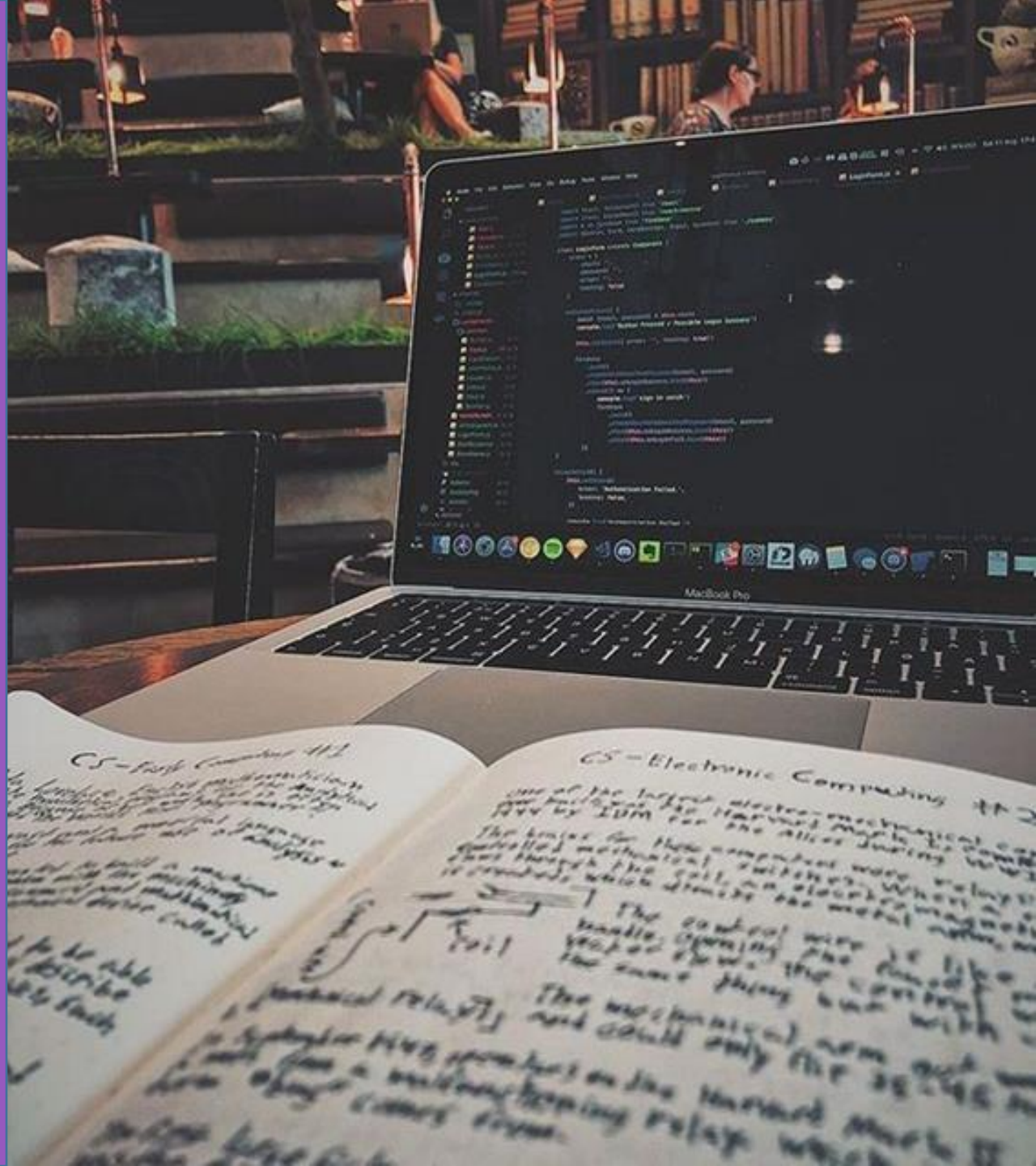


PОО primera parte

- Pilares de la programación orientada a objetos
- Abstracción
- Nivel de acceso
- Composición
- Agregación





Pilares de la Programación Orientada a Objetos (POO)



Pilares de la Programación Orientada a Objetos (POO)

- **Abstracción**

La abstracción encarada desde el punto de vista de la programación orientada a objetos expresa las características esenciales de un objeto, las cuales distinguen al objeto de | los demás

- **Encapsulamiento**

El encapsulamiento es la capacidad de controlar quien puede ver y utilizar los distintos módulos internos de nuestro sistema

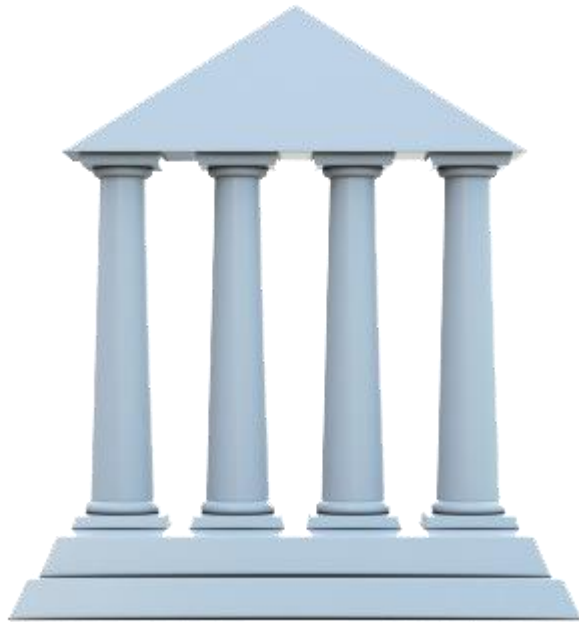
- **Herencia**

La herencia es una relación especial entre dos clases, la clase base y la clase derivada, en donde la clase derivada obtiene la habilidad de utilizar ciertas propiedades y funcionalidades de la clase base, incluso pudiendo sustituir funcionalidad de la clase base.

- **Polimorfismo**

Es la capacidad de decidir de forma dinámica a qué método recurrir si hay varios con igual cabecera, no atendiendo a sus parámetros sino al objeto sobre el que es invocado

Pilares de la POO



Abstracción

Encapsulamiento

Herencia

Polimorfismo

Abstracción

Al realizar una abstracción queremos omitir detalles que no son relevantes para representar solo aquellos relevantes.



¿Qué es?

¿Qué representa?

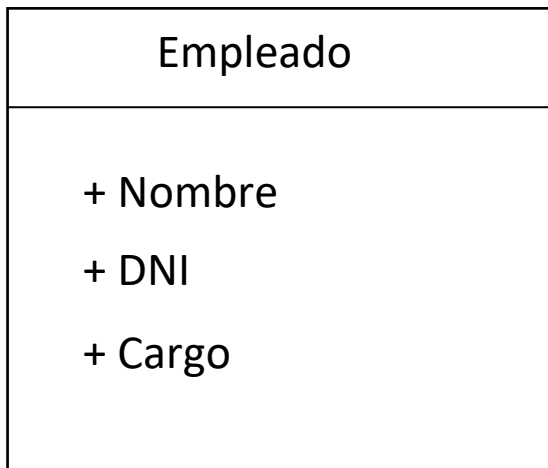
¿En qué contexto?

Abstracción

Problema

Supongamos que nuestro sistema identifica empleados de una empresa y necesitamos presentar una abstracción de estos

Representando la Abstracción



Utilizando la Abstracción

```
Class Empleado
{
    // miembros públicos, privados, etc
    // métodos
}

Empleado EmpleadoA = new Empleado();
Var empleadoB = new Empleado();
```

Miembros de una clase

Principio de ocultamiento de información

Atributo

Los atributos son las características individuales que diferencian un objeto de otro y determinan su apariencia, estado u otras cualidades. Los atributos se guardan en variables denominadas de instancia, y cada objeto particular puede tener valores distintos para estas variables.

Método

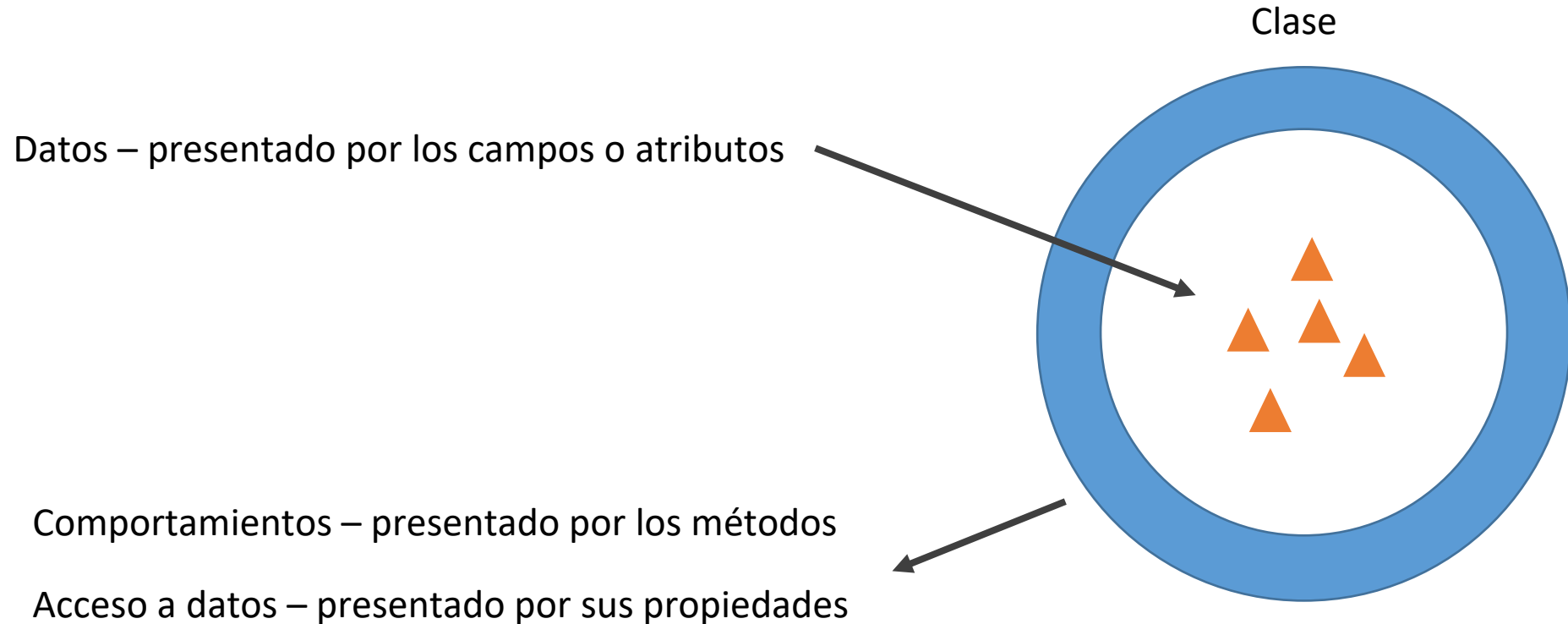
Un método es una función que se define dentro de una clase y se utiliza para representar el comportamiento de un objeto. Los métodos se utilizan para realizar tareas específicas en un objeto, como cambiar el estado de un objeto, calcular un valor, interactuar con otros objetos, etc.

Propiedades

Una propiedad es un identificador con un determinado tipo de dato que accede normalmente a un campo en forma directa o a través de un método.

Miembros de una clase

Principio de ocultamiento de información



Anatomía de una Clase

Encapsulamiento

Principio de ocultamiento de información

Encapsulamiento es un mecanismo que permite envolver elementos juntos como una unidad exponiendo los métodos y propiedades y ocultando sus atributos.

- datos (Atributos)
- Las acciones (Métodos)



Ocultamos/Bloqueamos características que no queremos que sean visibles y Presentamos/Exponemos las que sí queremos que sean visibles/utilizables por el resto de las clases.



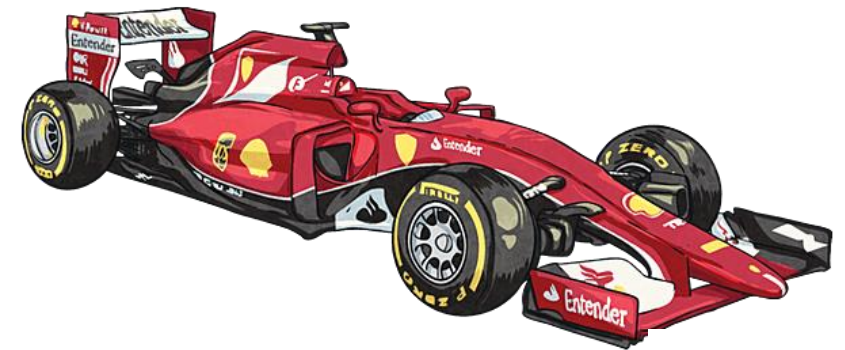
Principio de ocultamiento de la información

Encapsulamiento

Principio de ocultamiento de información

Encapsulamiento es un mecanismo que permite envolver elementos juntos como una unidad

- datos (Atributos)
- Las acciones (Métodos)

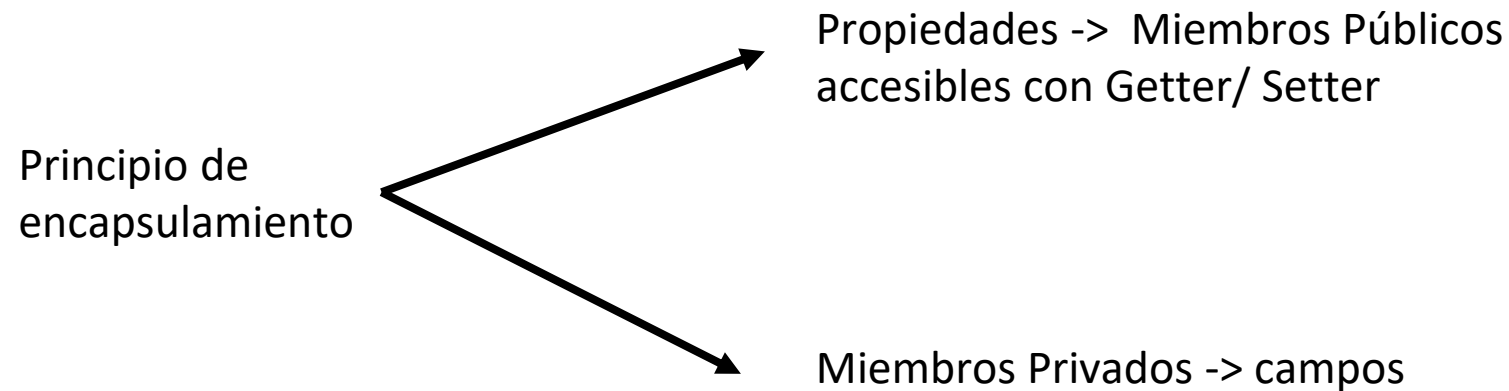


Encapsulamiento

Modificadores de acceso

Los campos son privados y son accedidas mediante métodos públicos o protegidos, permitiendo así consistencia en la información y el aumento de la cohesión.

- Publico, utilizando la palabra reservada ***public*** en los miembros de una clase
- Privado, utilizando la palabra reservada ***private*** en los miembros de una clase



Miembros de una clase

Nivel de Acceso

Una clase no estática puede contener métodos, campos, propiedades o eventos estáticos. El miembro estático es invocable en una clase, incluso si no se ha creado ninguna instancia de la clase. Siempre se tiene acceso al miembro estático con el nombre de clase, no con el nombre de instancia.

Solo existe una copia de un miembro estático, independientemente del número de instancias de la clase que se creen.

Miembros
de instancia → Accesible
desde un **objeto**

Miembros
estáticos → Accesible
desde la **clase**

```
public class Posicion
{
    public int X { get; set; }
    public int Y { get; set; }
    public Posición()
    {
        [...]
    }
}

public class Rectangulo
{
    public Posición MiPosicion { get; set; }
    public int Alto { get; set; }
    public int Ancho { get; set; }
    public Rectangulo()
    {
        [...]
    }

    Static public int CaluloArea(Rectangulo rect)
    {
        return rect.Alto * rect.Ancho;
    }
}
```


Clase

Métodos

Firma de un método

La firma de un método está constituida por un nombre y sus parámetros.

Nombre Parámetros = Firma



```
public void Mover (int NuevoX, int NuevoY)
{
    x = NuevoX;
    y = NuevoY;
}
```

Clase

Métodos

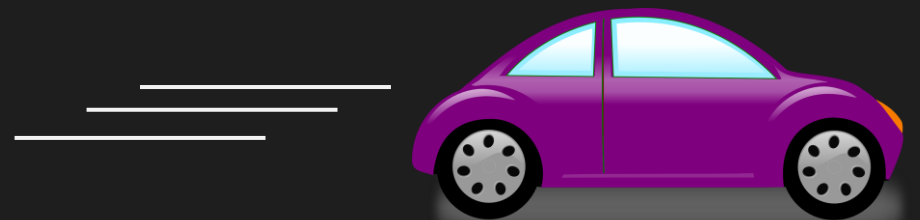
- Llamamos Método a una función que nos permite acceder a las viables declaradas dentro de una clase.
- Por lo tanto: Un método es un bloque de código que contiene una serie de instrucciones. Un programa hace que se ejecuten las instrucciones al llamar al método.
- Llamar a un método en un objeto es como acceder a un campo. Después del nombre del objeto, se agrega un punto, el nombre del método y los paréntesis.

```
public class Auto
{
    Int x;
    Int Y;

    Public Mover(int distancia)
    {
        x += distancia;
    }

    Public Frenar()
    {
        x = 0;
    }
}
```

```
[...]
Auto Auto = new Auto(); //inicio en (0,0)
Auto.Mover(10);
Auto.Frenar();
```



Clase

Métodos

Sobrecarga de un método

*La sobrecarga consiste en crear **más** de un procedimiento, constructor de instancia o propiedad en una clase con el mismo nombre y distintos tipos de argumento.*

Se utiliza el número y tipo de argumentos para seleccionar qué definición de método ejecutar, esta tarea la resuelve directamente el compilador.

```
public class FiguraGeometrica
{
    private int x;
    private int Y;

    //Metodo a
    public void mover(int NuevoX, int NuevoY)
    {
        x = NuevoX;
        y = NuevoY;
    }

    //Metodo b
    public void mover (string NuevoX,string NuevoY)
    {
        x = Convert.ToInt(NuevoX);
        y = Convert.ToInt(NuevoY);
    }

    //Metodo c
    public void mover (float NuevoX,float NuevoY)
    {
        x = Convert.ToInt(NuevoX);
        y = Convert.ToInt(NuevoY);
    }
}

[...]
```

`FiguraGeometrica Figura = new FiguraGeométrica();`

`Figura.mover(10,10); //Método a`

`Figura.mover(12.5f, 13.5f); //Método c`

`Figura.moverPunto(Console.ReadLine(), Console.ReadLine()); //Método b`

Clase

Métodos - Constructor

Un constructor es un método cuyo nombre es igual que el nombre de su tipo. La declaración del método incluye solo el nombre del método y su lista de parámetros; no incluye un tipo de valor devuelto

```
public class FiguraGeometrica
{
    Int x;
    Int Y;

    //Constructor a
    Public FiguraGeometrica()
    {
        x = 0;
        y = 0;
    }

    //Constructor b
    Public FiguraGeometrica(string NuevoX,string NuevoY)
    {
        x = Convert.ToInt(NuevoX);
        y = Convert.ToInt(NuevoY);
    }

    // Constructor c
    Public FiguraGeometrica(int NuevoX,int NuevoY)
    {
        x = NuevoX;
        y = NuevoY;
    }
}

[...]
```

FiguraGeometrica Figura = new FiguraGeométrica(); //inicio en (0,0)

FiguraGeometrica Figura2 = new FiguraGeométrica(10,10);

FiguraGeometrica Figura3 = new FiguraGeométrica
(Console.ReadLine(), Console.ReadLine()); // constructor b

Clase

Métodos - Constructor

Sobrecarga de un Constructor

Los constructores también pueden ser sobrecargados lo que permite Instanciar un objeto pasando distintos parámetros. Permitiendo ofrecer al desarrollador distintas formas de inicializar un objeto. Esto provee una gran versatilidad y comodidad a la hora de programar. Además, que muchas veces reduce código.

```
public class FiguraGeometrica
{
    Int x;
    Int Y;

    //Constructor a
    Public FiguraGeometrica()
    {
        x = 0;
        y = 0;
    }

    //Constructor b
    Public FiguraGeometrica(string NuevoX,string NuevoY)
    {
        x = Convert.ToInt(NuevoX);
        y = Convert.ToInt(NuevoY);
    }

    // Constructor c
    Public FiguraGeometrica(int NuevoX,int NuevoY)
    {
        x = NuevoX;
        y = NuevoY;
    }
}

[...]

FiguraGeometrica Figura = new FiguraGeométrica(); //inicio en (0,0)

FiguraGeometrica Figura2 = new FiguraGeométrica(10,10);

FiguraGeometrica Figura3 = new FiguraGeométrica (Console.ReadLine(),
Console.ReadLine()); // constructor b
```

Abstracción y Ocultamiento de información



Representando la Abstracción

Auto
<ul style="list-style-type: none">- Marca- Posición- Velocidad
<ul style="list-style-type: none">+ GetMarca()+ Frenar()+ Acelerar(aceleración: int)+ Acelerar(aceleración:string)

```
public class Auto
{
    private 2dVector posicion;
    private string marca;
    private int velocidad;

    Public string Marca
    { get {return marca;}
    }

    Public Acelerar(int aceleración)
    {
        posición.x = v + aceleracion;
    }

    Public Acelerar(string aceleración)
    {
        posición.x = v +
convert.toInt(aceleración);
    }

    Public Frenar()
    {
        velocidad = 0;
    }

}
```