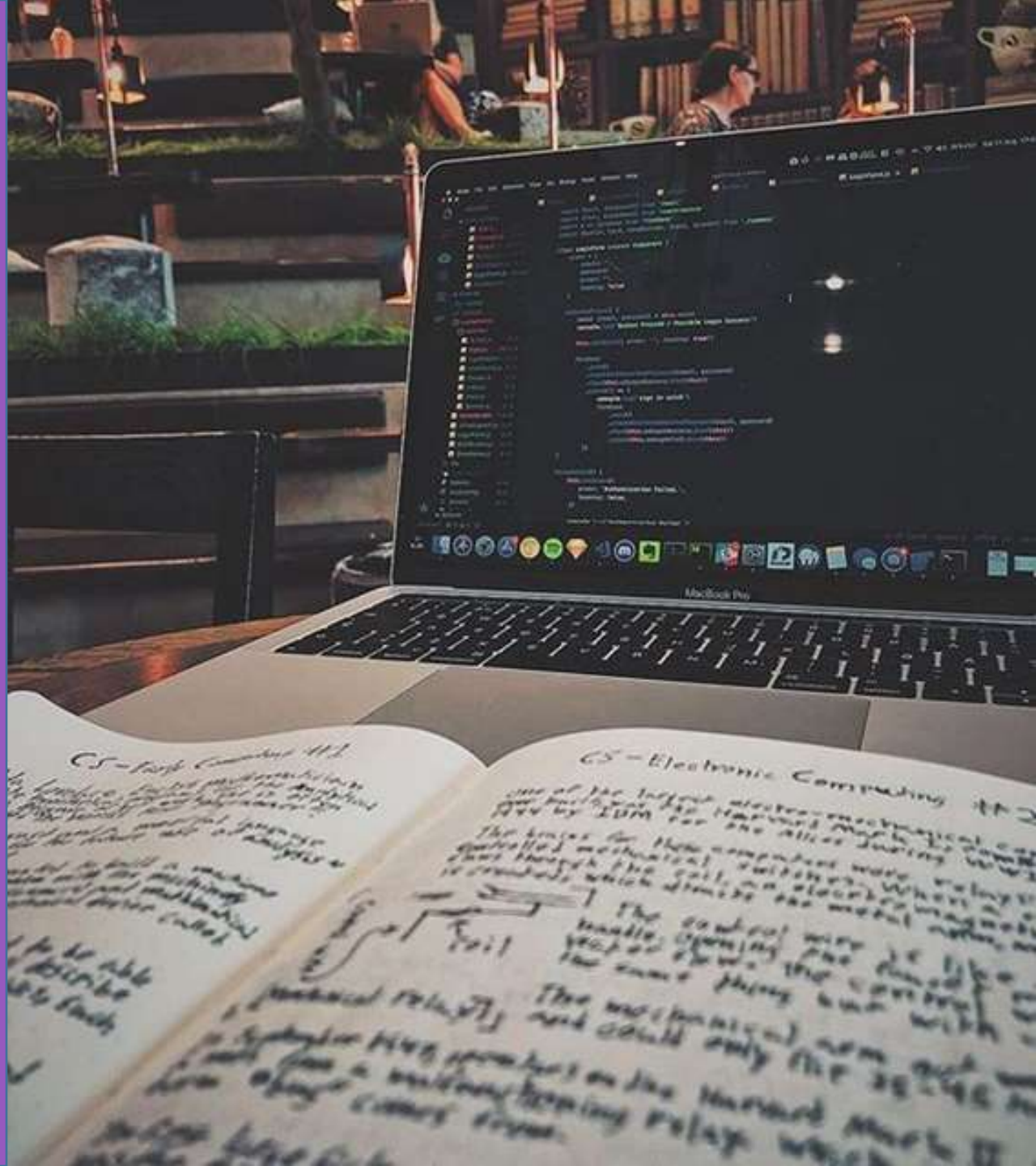
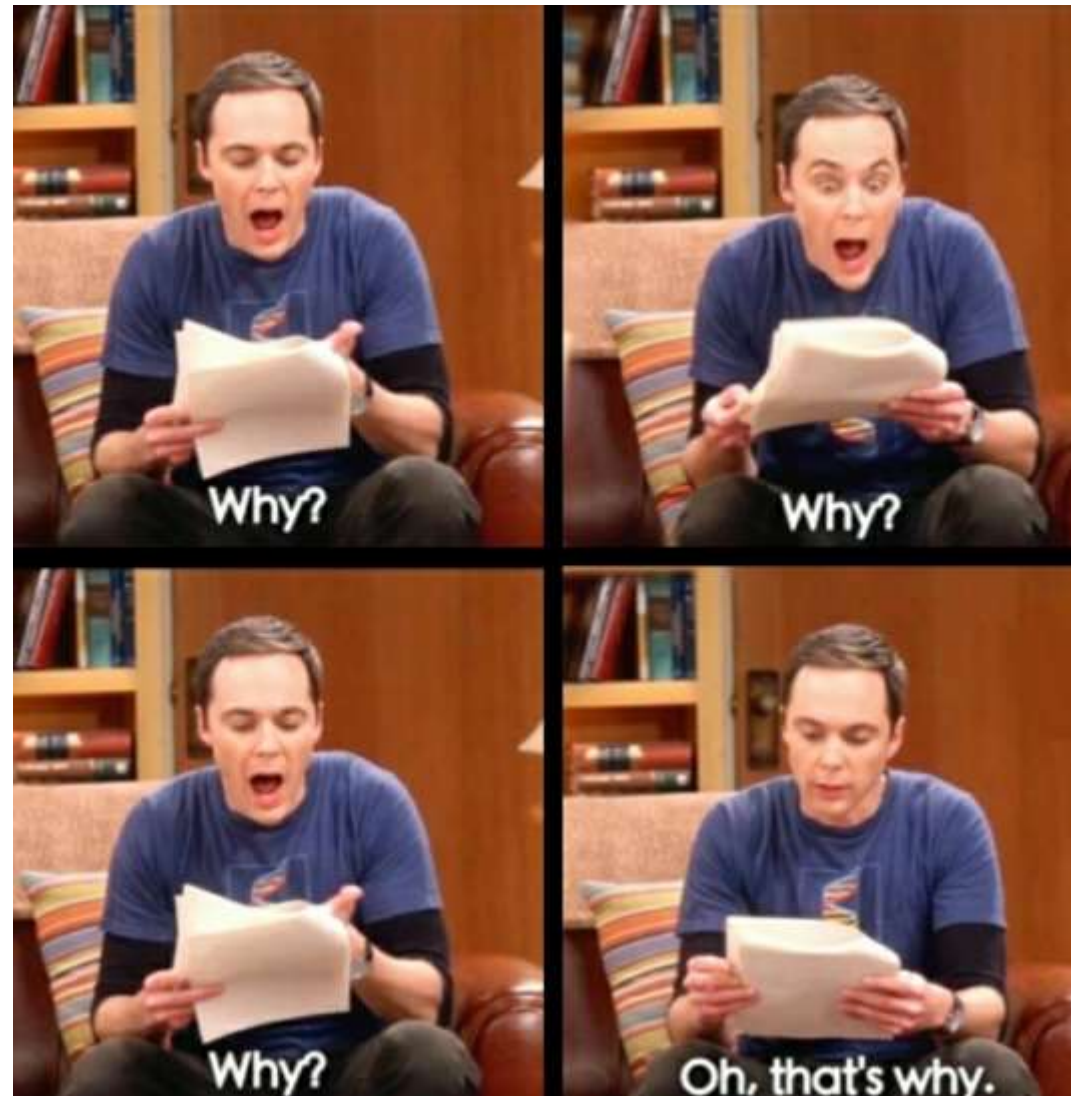


# Clase Nro 6

- Clases
- Miembros de una Clase
- Métodos de una Clase
- Visibilidad de los miembros de una clase
- Ejemplo práctico
- Clase estática



# Sobre el aprendizaje de un paradigma y la mejor forma de usarlo



# Programación orientada a objetos

La programación Orientada a objetos (POO) es un paradigma de programación, que busca crear abstracciones del mundo real a través de objetos.

## Clase

La unidad atómica de un sistema orientado a objetos

Permite la creación de objetos.

Encapsula estado y comportamiento

Una estructura de datos

# Clase

## Declarando una clase

- Declarando una nueva clase que se llama MiClase

```
class MiClase
{
    //miembros y
    atributos de la clase
}
```



```
Typedef struct MiEstructura
{
    //miembros y de la
    estructura
}
```

# Clase

## Creando objetos

- Cada vez que generamos una variable del tipo de la clase decimos que estamos creando un **objeto**
- Cada objeto creado a partir de la clase se denomina **instancia** de la **clase**.
- El operador **new** nos permite reservar memoria de forma dinámica para crear una instancia de la clase

```
MiClase MiInstancia = new MiClase();
```

Objeto o instancia

Clase

# Clase

## Campos o atributos

- Llamamos campos de a una variable declarada dentro de una clase.

```
class MiClase  
{  
    Int x;  
    Int y;  
}
```



```
Typedef struct MiEstructura  
{  
    Int x;  
    Int y;  
}
```

```
MiClase MiInstancia = new MiClase();  
MiInstancia.x = 10;
```

# Clase

## Visibilidad de los miembro de una clase

Miembros Públicos y Miembros privados

```
public class Perro
{
    public string Nombre;
    public string Raza;
    public int Edad;
}
```

---

**Miembros públicos:** Son accesibles desde Afuera de la clase con el operador “.”

```
Perro MilInstancia = new Perro();
MilInstancia.Edad = 10;
```

```
public class Perro
{
    private string Nombre;
    private string Raza;
    private int edad;
}
```

**Miembros privados:** Son accesibles solo desde dentro de la clase

```
Perro schnauzer = new Perro();
schnauzer.edad = 10;
```

← **ERROR**

No puedes acceder a un miembro privado!

# Clase

## Propiedades (Getter y Setters)

- Llamamos propiedad a una función que nos permite acceder a las viables declaradas dentro de una clase
- Las propiedades permiten que una clase exponga una manera pública de obtener y establecer valores, a la vez que se oculta el código de implementación o verificación.

```
Pixel NuevoPixel = new Pixel();  
NuevoPixel.X = 10;
```

```
public class Perro  
{  
    private Int edad;  
  
    public int Edad  
    {  
        get => edad;  
        set => edad = value;  
    }  
  
    private string nombre;  
  
    public String Nombre  
    {  
        get => nombre;  
        set => nombre = value;  
    }  
}
```

} Propiedades

} Propiedades




# 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, agregue un punto, el nombre del método y paréntesis.

```
Perro Fangio = new Perro();  
Fangio. EdadEnAñosPerro();  
Console.WriteLine(Fangio.EdadEnAñosPerro());
```

```
public class Perro  
{  
    private string nombre;  
    private string Raza;  
    private int edad;  
  
    public string Nombre  
    {  
        get => nombre;  
        set => nombre = value;  
    }  
    public int Edad  
    {  
        get => edad;  
        set => edad = value;  
    }  
  
    Public void EdadEnAñosPerro()  
    {  
        return edad * 7;  
    }  
}
```



Método

# 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

Perro Fangio = new Perro("schnauzer");



Constructor



```
public class Perro
{
    private string nombre;
    private string raza;
    private int edad;

    public Perro(string raza)
    {
        this.raza = raza
    }

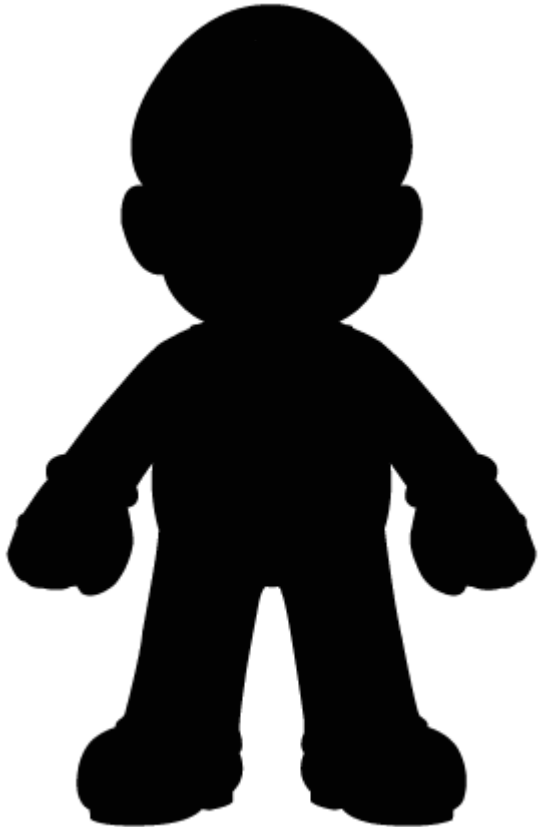
    public int Nombre
    {
        get => nombre;
        set => nombre = value;
    }

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

    Public void EdadEnAniosPerro()
    {
        return edad * 7;
    }
}
```

# Clase

Grupo de elementos de un conjunto que tiene características comunes.

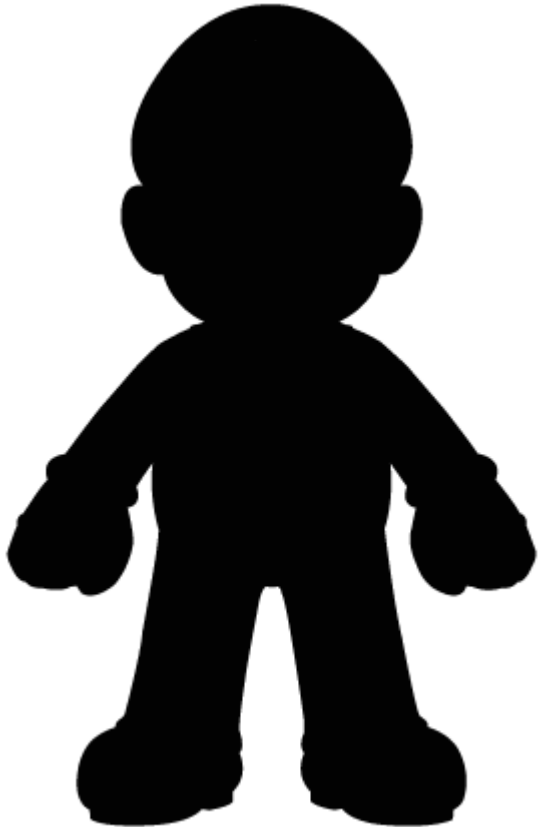


```
public class Player
{
    float PosicionX;
    float PosicionY;
    int vidas

    public void Saltar()
    {
        PosicionX += 5;
        PosicionY += 10;
    }
}
```

# Clase

Creando instancias de una clase Player



```
Player MARIO = new Player();
```

```
MARIO. PosicionX = 20px;
```

```
MARIO. PosicionY = 10px;
```

```
MARIO.Vidas = 3;
```

# Clase

Creando instancias de una clase Player



```
Player MARIO = new Player();
```

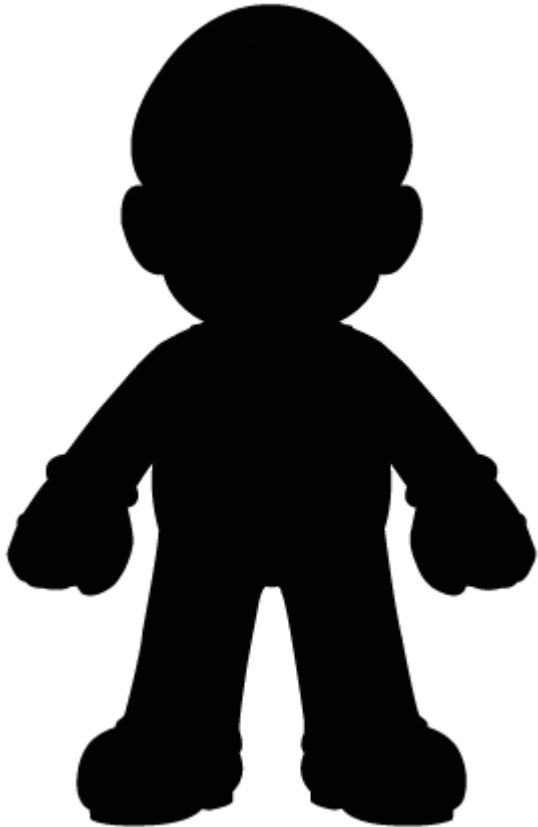
```
MARIO.PosicionX = 20px;
```

```
MARIO.PosicionY = 10px;
```

```
MARIO.Vidas = 3;
```

# Clase

Creando instancias de una clase Player



```
Player LUIGI = new Player();
```

```
LUIGI. PosicionX = 30px;
```

```
LUIGI. PosicionY = 10px;
```

```
LUIGI.Vidas = 3;
```

# Clase

Creando instancias de una clase Player



```
Player LUIGI = new Player();
```

```
LUIGI. PosicionX = 30px;
```

```
LUIGI. PosicionY = 10px;
```

```
LUIGI.Vidas = 3;
```

# Clase

Utilizando un método de la clase Player();

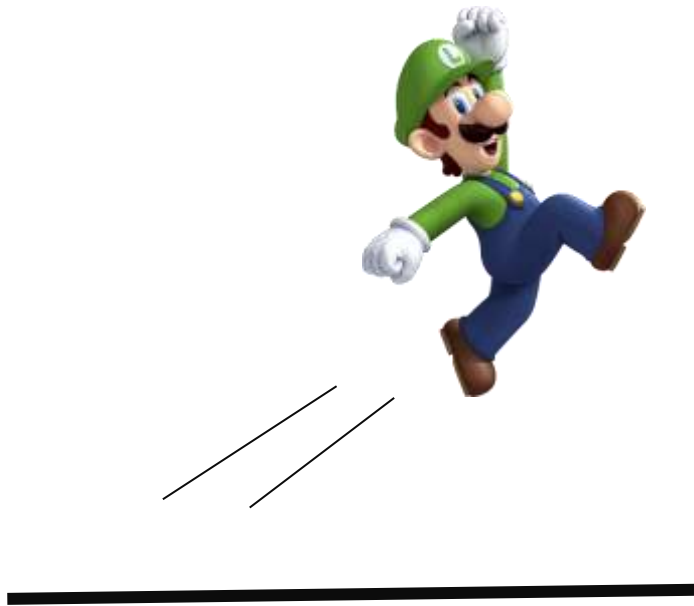


Mario.Saltar();



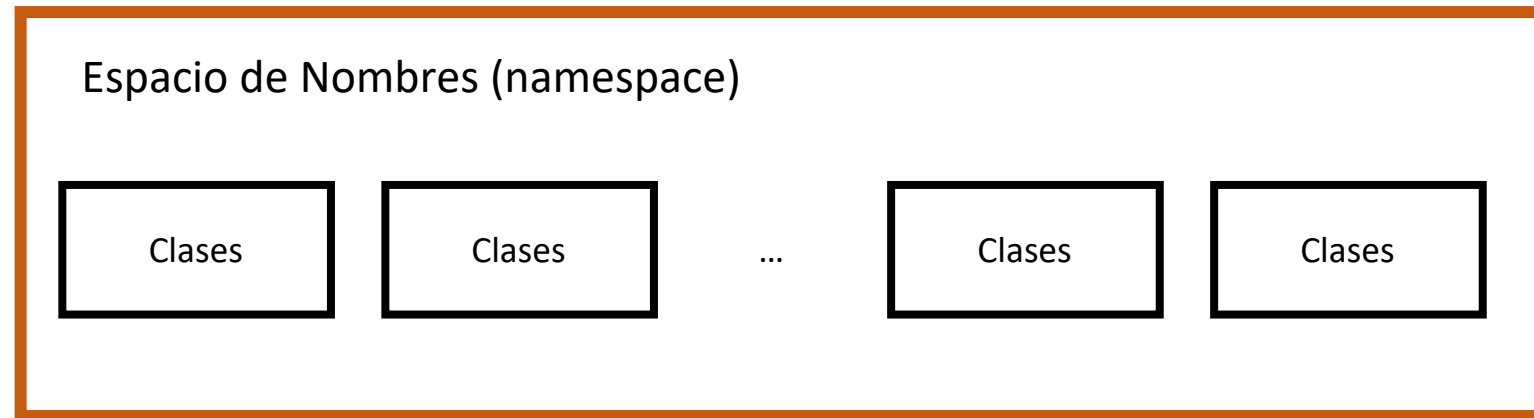
# Clase

Utilizando un método de la clase Player();



LUIGI.Saltar();

# Clase – Organización de las clases



# Clase – Organización de las clases

Proyecto

Espacio de Nombres (namespace)

Clases

Clases

...

Clases

Clases

Espacio de Nombres (namespace)

Clases

Clases

...

Clases

Clases