

# PROGRAMACIÓN ORIENTADA A OBJETOS

1

<https://github.com/ramsoftware>

# DEFINICIÓN



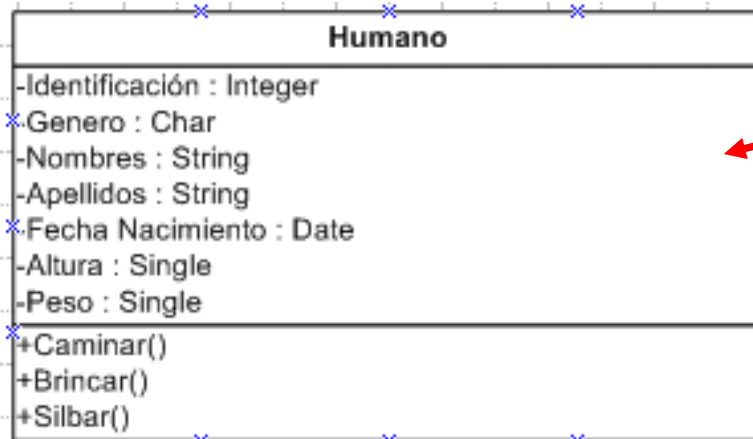
Humano es la clase



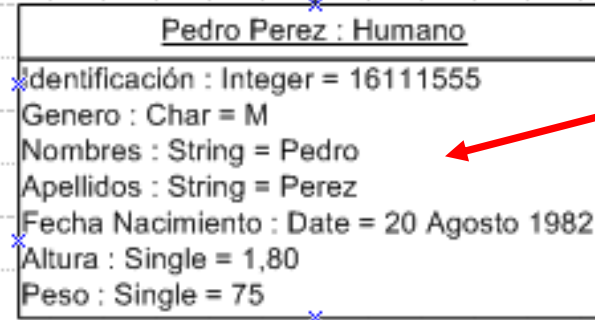
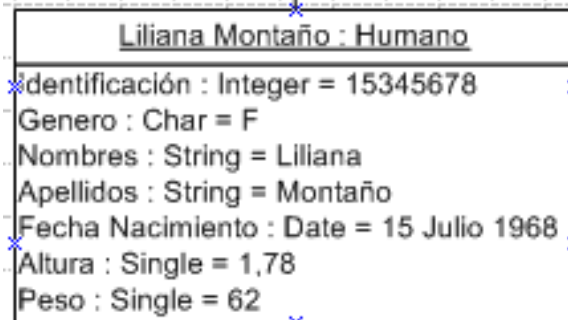
Cuatro personas.  
Cuatro instancias de la clase Humano.

# UML

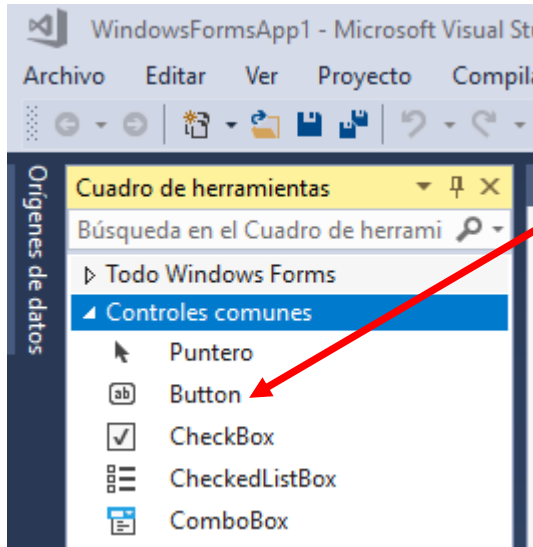
Humano es la clase



Dos personas.  
Dos instancias de la  
clase Humano.

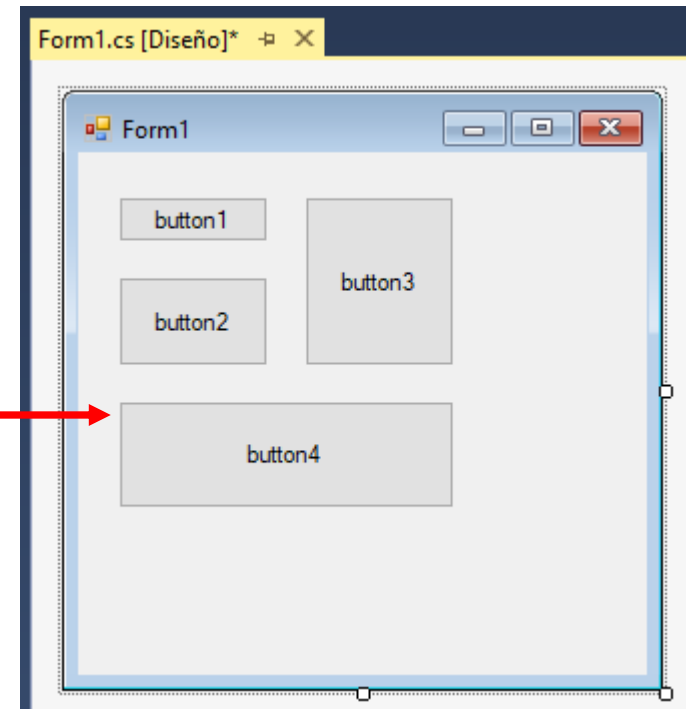


# ENTORNO GRÁFICO



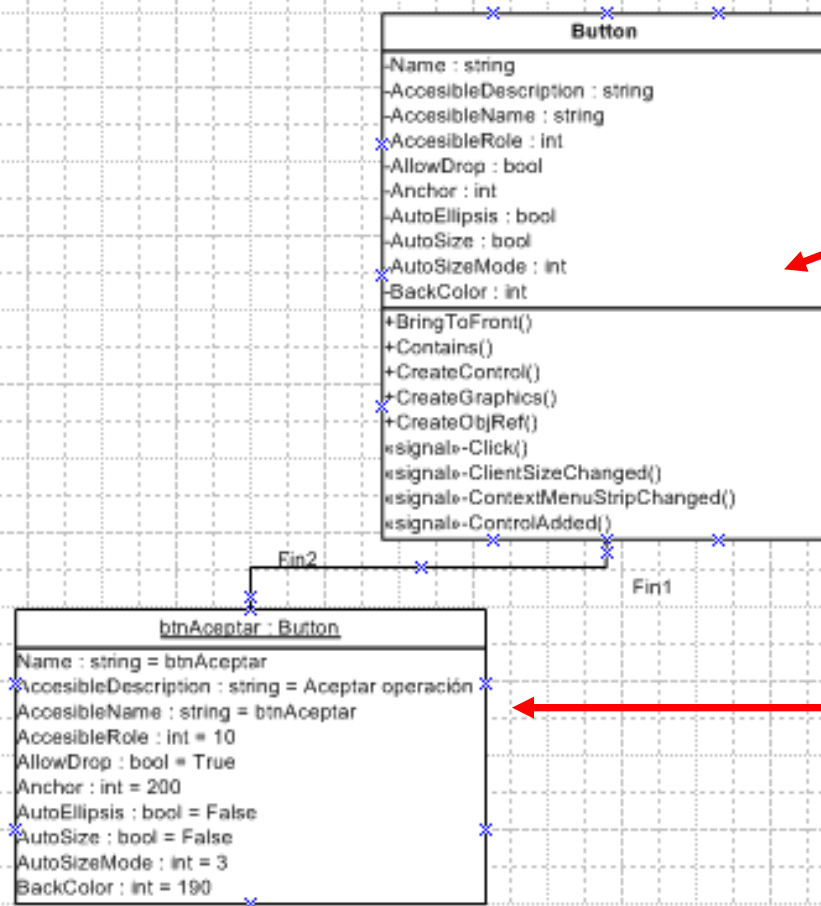
Button es la clase

Cuatro botones.  
Cuatro instancias de la clase Button



# ENTORNO GRÁFICO

Button es la clase



Un botón.  
Una instancia de la clase Button

# C#

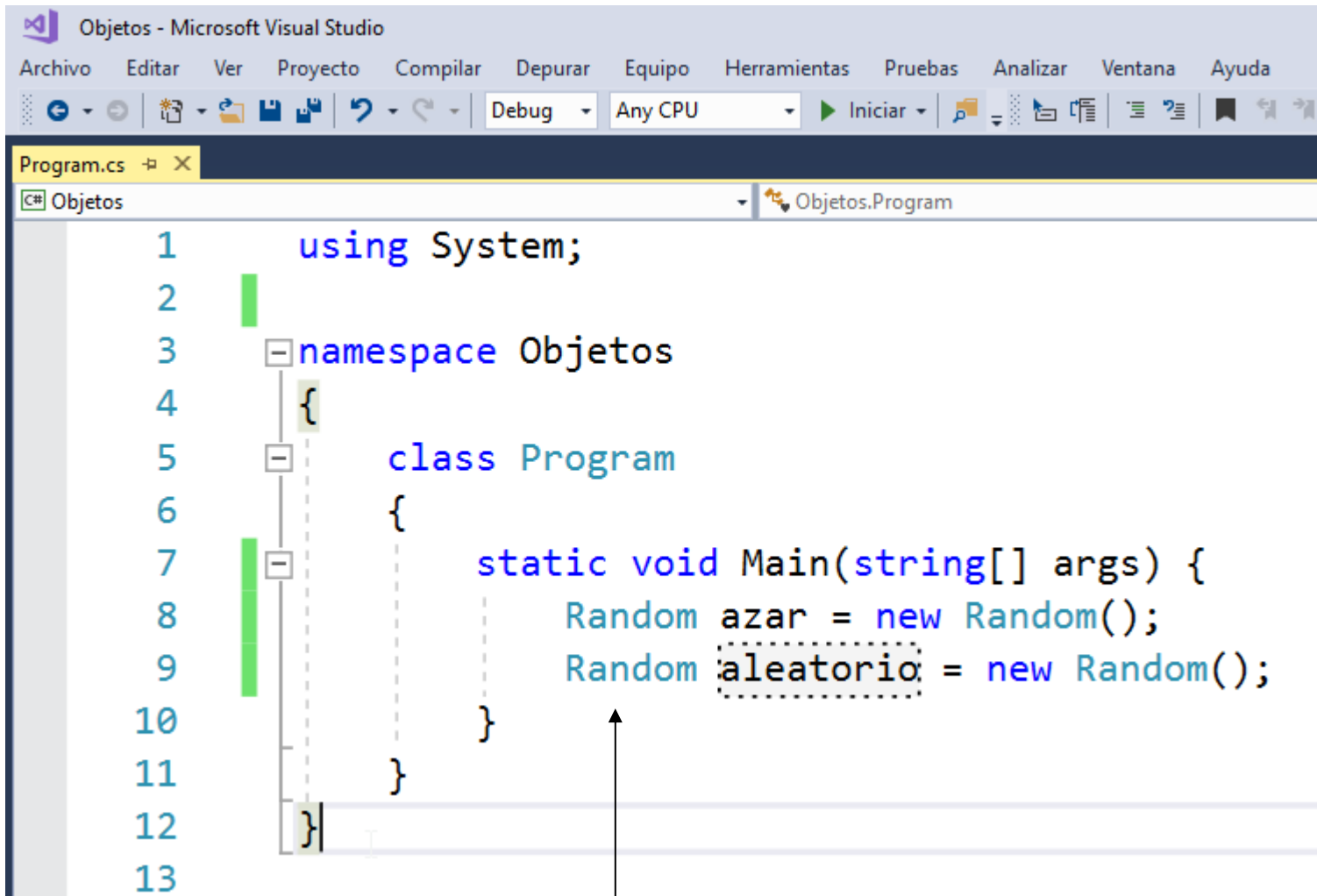
```
1  using System;
2
3  namespace Objetos
4  {
5      class Program
6      {
7          static void Main(string[] args) {
8              Random azar = new Random();
9          }
10     }
11 }
12
```

Clase

Instancia

Reserve memoria y genera la  
instancia

# C#



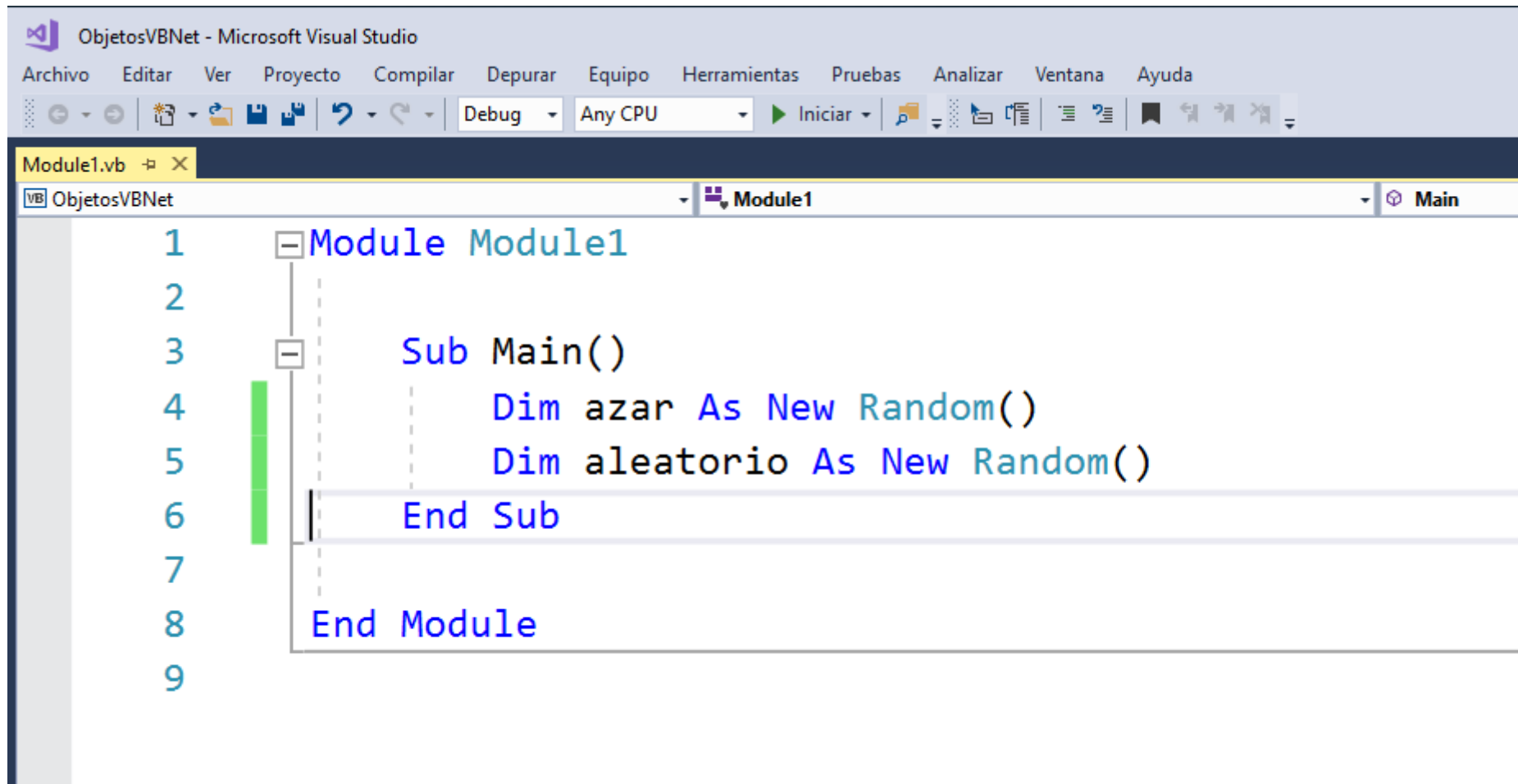
The screenshot shows the Microsoft Visual Studio IDE with a C# project named 'Objetos'. The file 'Program.cs' is open, displaying the following code:

```
1 using System;
2
3 namespace Objetos
4 {
5     class Program
6     {
7         static void Main(string[] args) {
8             Random azar = new Random();
9             Random aleatorio = new Random();
10        }
11    }
12 }
13
```

The code defines a namespace 'Objetos' containing a class 'Program'. Inside the 'Main' method, two instances of the 'Random' class are created: 'azar' and 'aleatorio'. The variable 'aleatorio' is highlighted with a dashed box, and an arrow points from the text below to it.

Misma clase y dos instancias de esa clase

# VISUAL BASIC .NET



ObjetosVBNet - Microsoft Visual Studio

Archivo Editar Ver Proyecto Compilar Depurar Equipo Herramientas Pruebas Analizar Ventana Ayuda

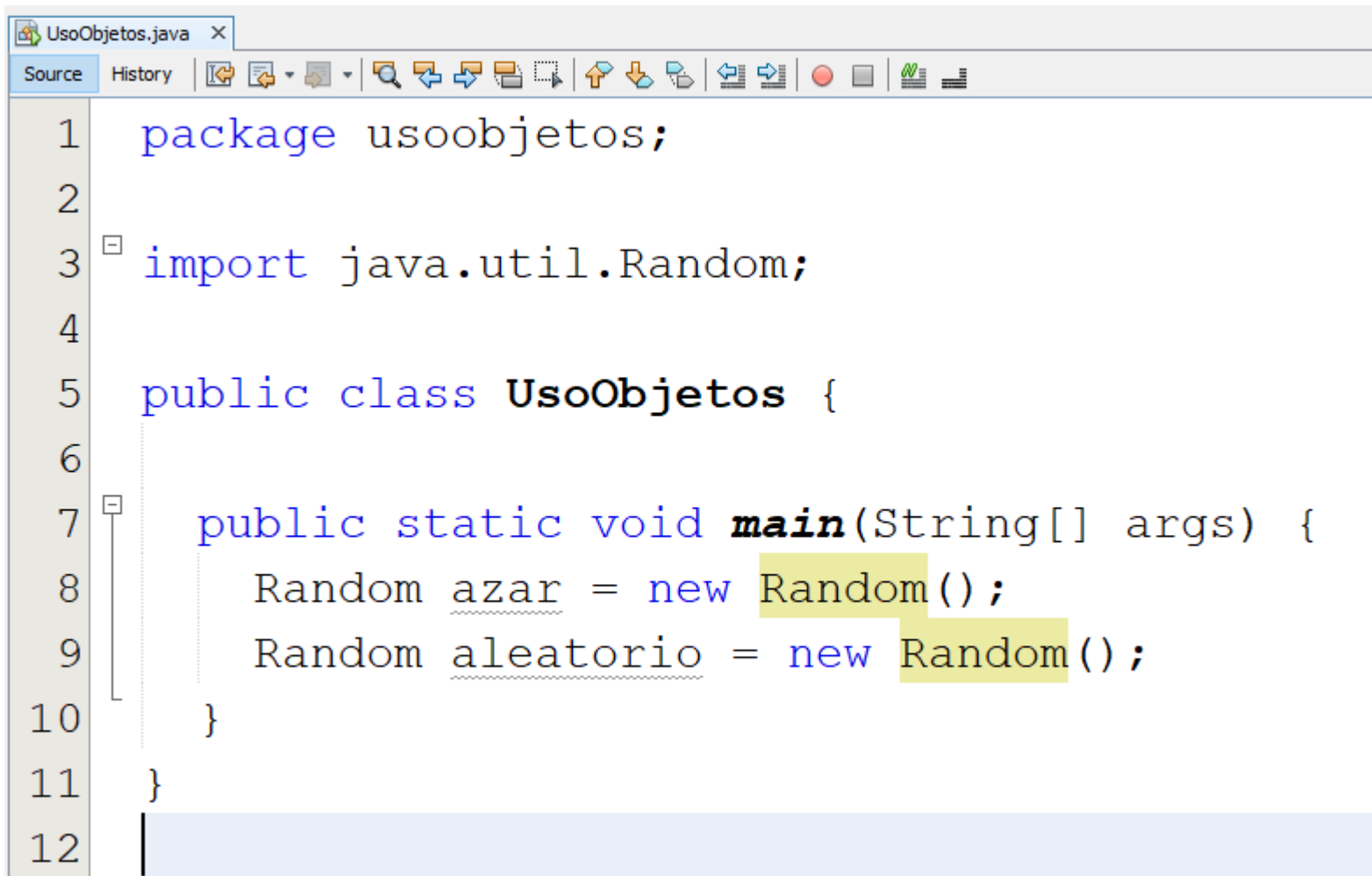
Module1.vb

ObjetosVBNet Module1 Main

```
1  Module Module1
2
3      Sub Main()
4          Dim azar As New Random()
5          Dim aleatorio As New Random()
6      End Sub
7
8  End Module
9
```



# JAVA



The screenshot shows a Java IDE window titled 'UsoObjetos.java'. The code is as follows:

```
1 package usoobjetos;
2
3 import java.util.Random;
4
5 public class UsoObjetos {
6
7     public static void main(String[] args) {
8         Random azar = new Random();
9         Random aleatorio = new Random();
10    }
11 }
12
```

The code is syntactically correct. The IDE interface includes a toolbar with various icons for file operations, editing, and running the program.

# CLASES, INSTANCIAS, MÉTODOS Y ATRIBUTOS

# JAVA. OBSERVE LAS 2 CLASES

```
package usoobjetos;
```

```
public class AreasFiguras {  
    public float Cuadrado(float Lado) {  
        return Lado * Lado;  
    }  
  
    public float Triangulo(float Base, float Altura) {  
        return (float) Base * Altura / 2;  
    }  
}
```

```
class Program {  
    public static void main(String[] args) {  
        AreasFiguras objFiguras = new AreasFiguras();  
        float AreaCuad = objFiguras.Cuadrado(12);  
        System.out.println(AreaCuad);  
    }  
}
```

Se instancia la clase  
AreasFiguras, la instancia  
se llama objFiguras

Se llama a un método  
público de la clase  
AreasFiguras

# C#

```
using System;
```

```
namespace Objetos{
```

```
    public class AreasFiguras{
```

```
        public float Cuadrado(float Lado) {
```

```
            return Lado * Lado;
```

```
        }
```

```
        public float Triangulo(float Base, float Altura) {
```

```
            return (float)Base * Altura / 2;
```

```
        }
```

```
    }
```

```
    public class Program{
```

```
        public static void Main(String[] args) {
```

```
            AreasFiguras objFiguras = new AreasFiguras();
```

```
            float AreaCuad = objFiguras.Cuadrado(12);
```

```
            Console.WriteLine(AreaCuad);
```

```
            Console.ReadKey();
```

```
        }
```

```
    }
```

```
}
```

# VB .NET

```
Public Class AreasFiguras
    Public Function Cuadrado(ByVal Lado As Single) As Single
        Return Lado * Lado
    End Function

    Public Function Triangulo(ByVal Base As Single, ByVal Altura As Single) As Single
        Return Base * Altura / 2
    End Function
End Class

Module Module1
    Public Sub Main()
        Dim objFiguras As AreasFiguras = New AreasFiguras()
        Dim AreaCuad As Single = objFiguras.Cuadrado(12)
        Console.WriteLine(AreaCuad)
        Console.ReadKey()
    End Sub
End Module
```

# JAVASCRIPT. ANTIGUO.

```
<!DOCTYPE HTML><html><body><script>
function AreasFiguras() {
    this.Cuadrado = function(Lado) {
        return Lado * Lado;
    }

    this.Triangulo = function(Base, Altura) {
        return Base * Altura / 2;
    }
}

//Instancia y llama a los métodos
var objFiguras = new AreasFiguras();
var AreaCuad = objFiguras.Cuadrado(12);
document.write(AreaCuad);
</script></body></html>
```

# JAVASCRIPT. NUEVO.

```
<!DOCTYPE HTML><html><body><script>
```

```
class AreasFiguras{
```

```
    Cuadrado(Lado) {
```

```
        return Lado * Lado;
```

```
    }
```

```
    Triangulo(Base, Altura){
```

```
        return Base * Altura / 2;
```

```
    }
```

```
}
```

```
//Instancia y llama a los métodos
```

```
var objFiguras = new AreasFiguras();
```

```
var AreaCuad = objFiguras.Cuadrado(12);
```

```
document.write(AreaCuad);
```

```
</script></body></html>
```

<https://developer.mozilla.org/es/docs/Web/JavaScript/Referencia/Classes>

# PHP

```
<?php
```

```
class AreasFiguras{  
    public function Cuadrado($Lado) {  
        return $Lado * $Lado;  
    }  
  
    public function Triangulo($Base, $Altura) {  
        return $Base * $Altura / 2;  
    }  
}
```

```
$objFiguras = new AreasFiguras();  
$AreaCuad = $objFiguras->Cuadrado(12);  
echo $AreaCuad;
```



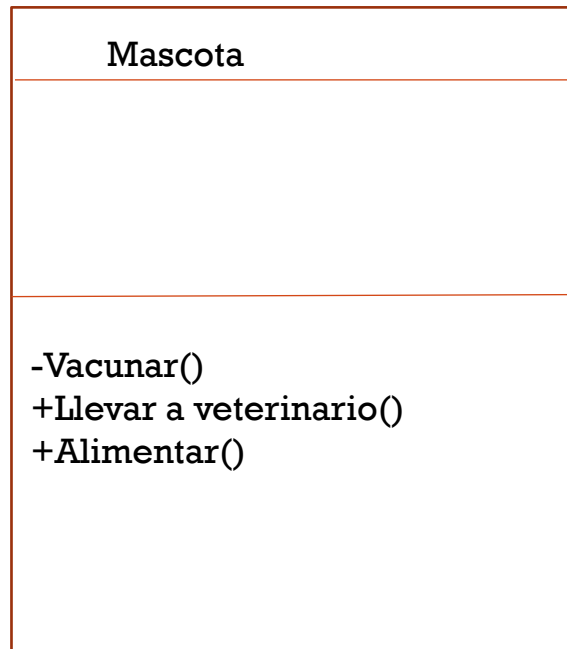
# MODIFICADORES DE MÉTODOS: PUBLIC, PRIVATE

- Cuando un método es “public” significa que la instancia puede llamarlo.
- Cuando un método es “private”, la instancia NO puede llamarlo.

```
class ValidandoFecha{  
    public bool EsFechaCorrecta(int Anyo, String Mes, int Dia){ }  
    private bool validaAnyo(int Anyo){ }  
    private bool validaMes(String Mes){ }  
}
```

```
ValidandoFecha instancia = new ValidandoFecha();  
bool EsCorrecto = instancia.EsFechaCorrecta(2018, "agosto", 25); //Correcto  
bool Otro = instancia.validaMes("septiembre"); //Incorrecto
```

# UNA CLASE EN UML. VISIBILIDAD (PUBLIC/PRIVATE)



Nombre de clase (en singular iniciando en mayúscula)

Atributos o propiedades (se estudia más adelante)

Métodos

+ Es public, - es private

# ATRIBUTOS

- Son variables declaradas dentro de la clase y no en los métodos

```
public class UnaClase{
    public int Valor;
    public double Cantidad;
    public bool EsValido;
    public char Estado;
    public String Descripcion;
}

public class Program{
    public static void Main(String[] args) {
        UnaClase Ejemplo = new UnaClase();
        Ejemplo.Valor = 156;
        Ejemplo.Cantidad = 5.8902;
        Ejemplo.EsValido = true;
        Ejemplo.Estado = 'T';
        Ejemplo.Descripcion = "Producto";
        Console.WriteLine(Ejemplo.Descripcion);
        Console.ReadKey();
    }
}
```

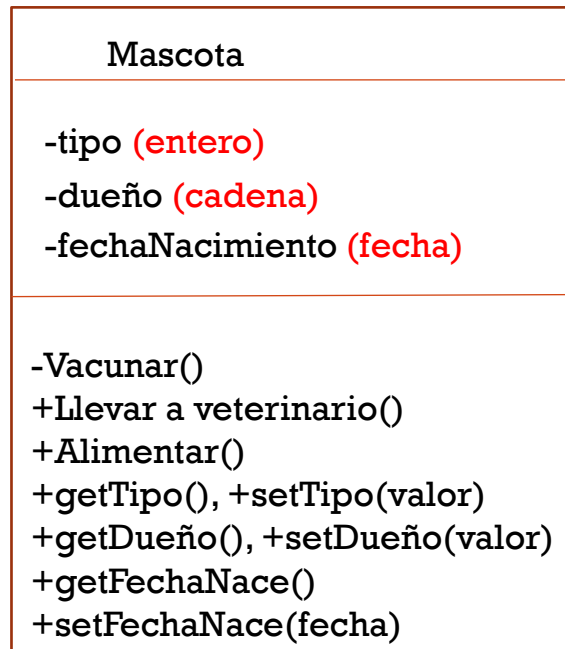
# ATRIBUTOS

- Se recomienda hacer “private” los atributos.

```
public class UnaClase {  
    private int Valor;  
  
    public void setValor(int Valor) { this.Valor = Valor; }  
    public int getValor() { return this.Valor; }  
}  
  
public class Program {  
    public static void Main(String[] args) {  
        UnaClase Ejemplo = new UnaClase();  
        Ejemplo.setValor(45);  
        Console.WriteLine(Ejemplo.getValor());  
        Console.ReadKey();  
    }  
}
```

- Los “getters” y “setters” son, en cierto modo, polémicos. Usted juzgará con el tiempo: <http://michelletores.mx/para-que-sirven-los-setters-y-getters/> ,  
<http://www.chuidiang.org/clinux/sobrecarga/ocultacion.php>

# UNA CLASE EN UML. VISIBILIDAD (PUBLIC/PRIVATE)



← Nombre de clase (en singular iniciando en mayúscula)

← Atributos o propiedades

← Métodos

+ Es public, - es private

# ATRIBUTOS ESTÁTICOS

- Son atributos que son definidos con un valor dentro de la clase. Si son públicas, la clase no necesita instanciarse para acceder a estos.

```
public class UnaClase {  
    public static int Valor = 15;  
}  
  
public class Program {  
    public static void Main(String[] args) {  
        Console.WriteLine(UnaClase.Valor);  
        Console.ReadKey();  
    }  
}
```

# ATRIBUTOS ESTÁTICOS

Sirven para definir constantes. Es un valor global a todos los objetos instanciados.

```
public class CalculaPrecios {  
    private static int IVA = 19;  
    public int PrecioTotal(int valor) {  
        return IVA * valor / 100 + valor;  
    }  
}  
  
public class Program {  
    public static void Main(String[] args) {  
        CalculaPrecios costo = new CalculaPrecios();  
        int cuesta = 100000;  
        int total = costo.PrecioTotal(cuesta);  
        Console.WriteLine(total);  
        Console.ReadKey();  
    }  
}
```

# MÉTODOS ESTÁTICOS

```
class Matematicas{
    public static boolean EsPrimo(int num) {
        if (num <= 1) return false;
        if (num == 2) return true;
        if (num % 2 == 0) return false;
        for (int divide = 3; divide <= Math.sqrt(num); divide += 2)
            if (num % divide == 0) return false;
        return true;
    }
}

class Main {
    public static void main(String[] args) {
        int numero = 78901;
        boolean UnPrimo = Matematicas.EsPrimo(numero);
        System.out.println(UnPrimo);
    }
}
```

No necesita instanciarse la clase “Matemáticas”



# CONSTRUCTOR

- Es un método que se ejecuta cuando la clase es instanciada. Sólo ocurre esa vez, no más.

# JAVA. CONSTRUCTOR SIN ARGUMENTOS.

```
package usoobjetos;
//Generador congruencial lineal de números aleatorios
public class Generador {
    private int valX0, valA, valB, valN;
    //Inicializa las semillas del generador. Constructor.
    public Generador(){
        valX0 = 111;
        valA = 78;
        valB = 45;
        valN = 77;
    }
    //Retorna número aleatorio
    public double numAleatorio(){
        valX0 = (valX0 * valA + valB) % valN;
        double valR = (double) valX0 / valN;
        return valR;
    }
}

class Program {
    public static void main(String[] args) {
        Generador azar = new Generador();
        for (int cont=1; cont<=10; cont++)
            System.out.println(azar.numAleatorio());
    }
}
```

Atributos privados  
(muy recomendado)



Constructor



# JAVA. CONSTRUCTOR CON ARGUMENTOS

```
package usoobjetos;
//Generador congruencial lineal de números aleatorios
public class Generador {
    private int valX0, valA, valB, valN;
    //Inicializa las semillas del generador. Constructor.
    public Generador(int X0, int A, int B, int N){
        valX0 = X0;
        valA = A;
        valB = B;
        valN = N;
    }
    //Retorna número aleatorio
    public double numAleatorio(){
        valX0 = (valX0 * valA + valB) % valN;
        double valR = (double) valX0 / valN;
        return valR;
    }
}

class Program {
    public static void main(String[] args) {
        Generador azar = new Generador(901, 673, 843, 2851);
        for (int cont=1; cont<=10; cont++)
            System.out.println(azar.numAleatorio());
    }
}
```

# **SOBRECARGA DE MÉTODOS**

**28**

# Programación orientada a Objetos

## JAVA

```
public class Arranque {  
    public static void main (String argv[]){  
        ImprimeFlotante((float) 4.678);  
        ImprimeString("Hola Mundo");  
        ImprimeEntero(89);  
        ImprimeChar('J');  
        ImprimeBoolean(true);  
        ImprimeDouble(90.12731);  
    }  
}
```

Obsérvese que existe un método diferente para hacer lo mismo (imprimir) y todo porque los tipos de datos son distintos. El problema es que usted tiene que aprenderse de memoria todos esos métodos, luego la posibilidad de que se equivoque es alta.

# Programación orientada a Objetos

## JAVA

```
public class Arranque {  
    public static void main (String argv[]){  
        Imprime((float) 4.678);  
        Imprime("Hola Mundo");  
        Imprime(89);  
        Imprime('J');  
        Imprime(true);  
        Imprime(90.12731);  
    }  
}
```

Obsérvese ahora que el método se llama igual (Imprime), solo debe aprenderse uno, se equivoca menos, pero ¿Cómo puede ser eso posible? ¿No daría un mensaje de error al compilar?

# JAVA

```
class Program {  
  
    public static void main(String[] args) {  
        Imprime((float) 4.678);  
        Imprime("Hola Mundo");  
    }  
  
    static void Imprime(float Numero) {  
        System.out.println("Flotante es: " + Numero);  
    }  
  
    static void Imprime(String Cadena) {  
        System.out.println("Cadena es: " + Cadena);  
    }  
  
}
```

Así se implementa el método Imprime, uno es para números flotantes y el otro para cadenas, se llaman igual, pero se diferencian en que los parámetros tienen tipos de parámetros distintos. Más fácil de recordar.

# PRIMER EXPERIMENTO DE SOBRECARGA DE MÉTODOS

- ¿Cómo se comportará JavaScript con este código?

```
<!DOCTYPE HTML><html><body><script>  
Imprime(8.7);
```

```
function Imprime(Número) {  
    document.write("Flotante es: " + Número);  
}
```

```
function Imprime(Cadena) {  
    document.write("Cadena es: " + Cadena);  
}  
</script></body></html>
```



# JAVA

```
class Program {  
  
    public static void main(String[] args) {  
        System.out.println("Area cuadrado es: " + Area(5));  
        System.out.println("Area triángulo es: " + Area(7, 8));  
    }  
  
    static float Area(float fLado) {  
        return fLado * fLado;  
    }  
  
    static float Area(float fBase, float fAltura) {  
        return fBase * fAltura / 2;  
    }  
  
}
```

La primera retorna el área de un cuadrado, la segunda el área de un triángulo. La diferencia es que la primera recibe un solo parámetro y la segunda recibe dos parámetros.

# SEGUNDO EXPERIMENTO DE SOBRECARGA DE FUNCIONES

- ¿Cómo se comportará JavaScript con este código?

```
<!DOCTYPE HTML><html><body><script>
```

```
Imprime("a", "b");
```

```
function Imprime(numA, numB) {
```

```
    document.write("Son dos argumentos: " + numA + " y " + numB);  
}
```

```
function Imprime(numA) {
```

```
    document.write("Es un argumento: " + numA);  
}
```

```
</script></body></html>
```

# CUANDO NO FUNCIONA LA SOBRECARGA DE FUNCIONES

- Funciona si y solo si los parámetros son distintos (en tipo de dato o en número de parámetros)
- NO funciona cuando retorna tipos de datos distintos.

```
float Valor(float parametro){ return parametro*2; }  
int Valor(float parametro){ return (int) parametro/2; }
```

# PROGRAMAS GRANDES

- El paradigma POO nació para enfrentarse a problemas de programación complejos o programas muy extensos.