

# Atributos y Métodos

Prof. Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos  
Universidad de Granada

Programación y Diseño Orientado a Objetos

Doble Grado en Ingeniería Informática  
y Administración y Dirección de Empresas  
(Curso 2024-2025)

# Créditos

- Las siguientes imágenes e ilustraciones son libres y se han obtenido de:
  - ▶ Emojis, <https://pixabay.com/images/id-2074153/>
- El resto de imágenes e ilustraciones son de creación propia, al igual que los ejemplos de código

# Objetivos

- Aprender la utilidad, significado y uso de:
  - ▶ Atributos de instancia
  - ▶ Atributos de instancia de la clase
  - ▶ Atributos de clase
- Aprender a usar métodos de instancia y de clase
- Aprender las diferencias entre Java y Ruby en cuanto a:
  - ▶ Atributos de clase
  - ▶ Visibilidad privada
- Tomar nota de los errores más frecuentes que soléis cometer
- Usar correctamente las pseudovariables `this` y `self`
- Conocer los especificadores de acceso

# Contenidos

- 1 Atributos y métodos de instancia
- 2 Atributos y métodos de clase
  - Ejemplos
- 3 Pseudovariables
- 4 Especificadores de acceso. Visibilidad
  - Ejemplos

# Atributos de instancia

- La definición de las clases incluye los atributos de instancia **que tendrá cada objeto que sea instancia de esa clase**
- Los atributos de instancia **son variables** que están asociadas a cada objeto
- **Cada instancia tiene su propio espacio de atributos o variables de instancia.**
  - ▶ Así, cada instancia tendrá los mismos atributos que otra instancia de la misma clase, pero en zonas de memoria distintas
- El **estado** de cada instancia se describe mediante los **valores de estos atributos**

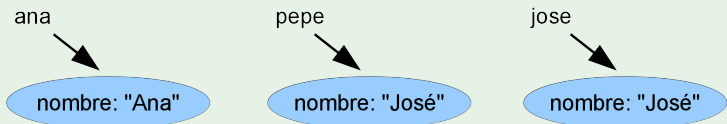
# Atributos de instancia

## Ejemplo: La clase Persona

```
1      class Persona {  
2          private String nombre;  
3          // . . .  
4      }
```

- Todas las instancias de la clase `Persona` tendrán un atributo denominado `nombre`
  - ▶ Existirá una variable denominada `nombre` para cada instancia
  - ▶ Dos instancias de `Persona` distintas almacenan el nombre en variables distintas (almacenadas en zonas de memoria distintas) aunque se llamen igual

```
1      Persona ana = new Persona ("Ana");  
2      Persona pepe = new Persona ("José");  
3      Persona jose = new Persona ("José");
```



# Métodos de instancia

- Son **funciones o métodos** definidos en una clase y que estarán asociados a los objetos de dicha clase
- Desde los **métodos asociados a un determinado objeto** son accesibles los **atributos de instancia de dicho objeto**.  
Tanto para lectura como para escritura.

## Ejemplo: La clase Persona

```

1 class Persona {
2     private String nombre;
3     // ...
4
5     String saludar () {
6         return "Hola, me llamo " + nombre;
7     }
8
9     void cambiaNombre (String otroNombre) {
10        nombre = otroNombre;
11    }
12 }

```

## Ejemplo: La clase Persona

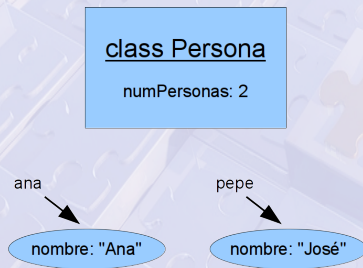
```

1 Persona pepe = new Persona ("José")
2 Persona ana = new Persona ("Ana");
3
4 System.out.println (ana.saludar ());
5 // Hola, me llamo Ana
6
7 ana.cambiaNombre ("Ana Belén");
8
9 System.out.println (ana.saludar ());
10 // Hola, me llamo Ana Belén
11
12 System.out.println (pepe.saludar ());
13 // Hola, me llamo José

```

# Atributos de clase

- Almacenan información que se considera **asociada a la propia clase y no a cada instancia**
- Son por tanto variables asociadas a la clase y **globales a todas las instancias de esa clase**
- Cada atributo de clase **existe de forma única**



## Ejemplo: Contador de instancias

- Tanto ana, como pepe tienen accesible el atributo de clase `numPersonas`
- Cuando su valor cambia, lo hace para todas las instancias
- El atributo es único, está en una zona de memoria asociada a la clase, no a las instancias



# Atributos de clase

→ **Diseño** ←

- ¿Cuándo usarlos?

- ▶ Se debe pensar en ellos cuando se necesite almacenar información que **siempre** va a ser **común** a **todas** las instancias de la clase
- ▶ No tendría sentido que cada instancia de la clase (Persona en el ejemplo anterior) guardase una copia del valor almacenado en ese atributo (numPersonas)
- ▶ Esto además haría su actualización extremadamente costosa.

- Ejemplos típicos:

- ▶ Contador de instancias
- ▶ Constantes
- ▶ Para evitar el uso de *números mágicos*

★ ¿Quién sabría decirme qué es un número mágico?

Usar números mágicos en los exámenes será **penalizado**

# Ejemplo

## Java: Constante vs. *número mágico*

```
1 class Factura {
2     ...
3     float calculaIVA (float baseImponible) {
4         return baseImponible * 0.21;    // Número mágico, fuente de errores
5     }
6 }
7
8 // Otro modo de diseñarlo
9
10 class GestionTributaria {
11     static float IVA = 0.21;    // Variable de clase
12     ...
13 }
14
15 class Factura {
16     ...
17     float calculaIVA (float baseImponible) {
18         return baseImponible * GestionTributaria.IVA;    // Uso de la variable de clase
19     }
20 }
```

# Métodos de clase

- Son funciones y procedimientos **asociados a la propia clase**
- Es habitual que accedan/actualicen atributos de clase
- No se puede acceder *directamente* a atributos/métodos de instancia desde un método de clase
  - ▶ Sería necesario solicitar ese elemento a una instancia concreta

## Java: Contador de instancias

```

1 class Persona {
2   // atributo y método de clase
3   static private int numPersonas = 0;
4   static int getNumPersonas () {
5     return numPersonas;
6   }
7   // atributo de instancia
8   private String nombre;
9   // inicializador
10  Persona (String unNombre) {
11    nombre = unNombre;
12    numPersonas++;
13  }
14 }
```

## Ruby: Contador de instancias

```

1 class Persona
2   # atributo y método de clase
3   @@num_personas = 0
4   def self.num_personas
5     @@num_personas
6   end
7   # inicializador
8   def initialize (un_nombre)
9     # atributo de instancia
10    @nombre = un_nombre
11    @@num_personas += 1
12  end
13 end
14
```

★ ¿Cómo se mostraría el número de instancias creadas?

# Atributos de clase:

# Particularidad de Ruby

- Existen **dos tipos** de atributos de clase
  - ▶ Atributos de clase (`@@atributo_de_clase`)
  - ▶ Atributos de instancia de la clase (`@atributo_instancia_clase`)
- Los atributos de clase son accesibles directamente desde el ámbito de instancia.
  - ▶ Los atributos de instancia de la clase, no
- Los atributos de clase se comparten con las subclasses (herencia).  
**Esto puede ser muy peligroso**
  - ▶ Los atributos de instancia de la clase, no

# Errores frecuentes en Ruby


- **Confundir** atributos de instancia con atributos de instancia de la clase
  - ▶ Hay que tener en cuenta en qué ámbito se está
  - ▶ En una clase, cualquier punto dentro de un método de instancia está en ámbito de instancia, lo demás está en ámbito de clase
  - ▶ En un **ámbito de instancia**,  
`@variable` alude a un **atributo de instancia**
  - ▶ En un **ámbito de clase**,  
`@variable` alude a un **atributo de instancia de la clase**
- **Añadir atributos** de instancia, atributos de instancia de la clase o atributos de clase **cuando hay que usar variables locales**
  - ▶ Las variables locales y los parámetros de método no llevan `@`
- Estos errores, en los exámenes, serán **penalizados**

# Ejemplos

## Ruby: Confusión entre atributos

```
1 class Clase
2   @@variable = "De clase"
3   @variable = "De instancia de la clase"
4
5   def initialize
6     @variable = "De instancia"
7   end
8
9   def muestraValores (variable)
10    puts @@variable
11    puts @variable
12    puts variable
13  end
14
15  def self.muestraValores
16    variable = "Local"
17    puts @@variable
18    puts @variable
19    puts variable
20  end
21 end
22 objeto = Clase.new
23 objeto.muestraValores ("Parámetro")
24 Clase.muestraValores
```

→ *Diseño* ←

- Los nombres de las variables deben ser significativos
- Debe evitarse nombrar a cosas distintas con el mismo nombre
- En el ejemplo anterior  no se han seguido estas recomendaciones por motivos docentes

★ ¿Cuál es el resultado de ejecutar este programa?

# Ejemplos

## Java: Atributos y métodos de clase y de instancia, variables locales

```
1  public class Persona {
2
3      private static final int MAYORIAEDAD=18; // Atributo de clase
4      private LocalDateTime fechaNacimiento;    // Atributo de instancia
5
6      Persona(LocalDateTime fecha) {
7          fechaNacimiento=fecha;
8      }
9
10     public boolean mayorDeEdad() { // Método de instancia
11         LocalDateTime ahora= LocalDateTime.now(); //Llamada a método de clase
12         // "ahora" es una variable local
13
14         //Años completos transcurridos
15         long edad=ChronoUnit.YEARS.between(fechaNacimiento , ahora);
16
17         return (edad>=MAYORIAEDAD);
18     }
19 }
```

★ ¿Qué efecto tiene la palabra `final` en la declaración de `MAYORIAEDAD`?

# Ejemplos

## Ruby: Atributos y métodos de clase y de instancia, variables locales

```
1  require 'date'
2
3  class Persona
4    @@MAYORIA_EDAD = 18 # Atributo de clase
5
6    def initialize (fecha)
7      @fecha_nacimiento=fecha # Atributo de instancia
8    end
9
10   def mayor_de_edad # Método de instancia
11     ahora = Date.today # "ahora" es una variable local
12     edad = ahora.year - @fecha_nacimiento.year - 1
13     if (ahora.month > @fecha_nacimiento.month)
14       edad += 1
15     else
16       if (ahora.month == @fecha_nacimiento.month)
17         if (ahora.day >= @fecha_nacimiento.day)
18           edad += 1
19         end
20       end
21     end
22     return (edad >= @@MAYORIA_EDAD)
23   end
24 end
```

★ ¿Qué significa la línea 1?



# Ejemplos

## Ruby: Atributos y métodos de clase y de instancia, variables locales

```
1  class Persona
2    @MAYORIA_EDAD=18 # Atributo de instancia de la clase
3
4    def self.edad_legal # Método de clase (Persona.edad_legal)
5      @MAYORIA_EDAD
6    end
7
8    def initialize (fecha)
9      @fecha_nacimiento = fecha # Atributo de instancia
10    end
11
12    def mayor_de_edad # Método de instancia
13      ahora = Date.today
14      edad = ahora.year - @fecha_nacimiento.year - 1
15      if (ahora.month > @fecha_nacimiento.month)
16        edad += 1
17      else
18        if (ahora.month == @fecha_nacimiento.month) && (ahora.day >= @fecha_nacimiento.day)
19          edad += 1
20        end
21      end
22      return (edad >= self.class.edad_legal) # (Persona.edad_legal)
23    end
24  end
```

★ ¿Se puede prescindir del método `edad_legal`?

★ ¿Cómo quedaría la línea 22?

# Ejemplos

## Ruby: Atributos y métodos de clase y de instancia, variables locales

```
1  class Producto
2    @@iva = 21
3
4    def initialize(precio, nombre)
5      @precio = precio
6      @nombre = nombre
7    end
8
9    def instanciaSetIVA(iv)
10     # Acceso directo a un atributo de clase desde un método de instancia
11     @@iva = iv
12     # Esto no es posible con atributos de instancia de la clase
13   end
14
15   def self.claseSetIVA(iv)
16     # Acceso directo a un atributo de clase desde un método de clase
17     @@iva = iv
18   end
19
20   def to_s
21     "nombre: #{@nombre}, precio: #{@precio}, iva: #{@iva}"
22   end
23 end
```

★ ¿Qué diferencia hay entre los métodos de las líneas 9 y 15?

# Ejemplos

## Ruby: Atributos y métodos de clase y de instancia, variables locales

```
1  # Usando la clase anterior
2
3  p = Producto.new(2, "cosa")
4  puts p.to_s
5
6  p.instanciaSetIVA(25)
7  puts p.to_s
8
9  Producto.classSetIVA(27)
10 puts p.to_s
11
12 # Lo siguiente no funciona en Ruby
13 # En cualquier caso NO es recomendable
14
15 p.classSetIVA(50) # esto no funciona en Ruby
16 puts p.to_s
```

★ ¿Qué salida producen las líneas 4, 7 y 10?

# Pseudovariables

- Existen palabras reservadas que referencian al propio objeto, o a la clase
  - Java: **this** (también en C++, C#, etc.)
  - Ruby: **self** (también en Python, Rust, etc. )

## Java: this

```

1  class Persona {
2    private String nombre;
3
4    Persona (String nombre) {
5      this.nombre = nombre;
6    }
7
8    Persona () {
9      this ("Anónimo");
10   }
11 }
```

★ Significado de **this** en las líneas 5 y 9

★ Significado de **self** en las líneas 3 y 14

## Ruby: self

```

1  class Persona
2    @@MAYORIA_EDAD = 18
3
4    def self.mayoria_edad
5      return @@MAYORIA_EDAD
6    end
7
8    def initialize (nombre)
9      @nombre = nombre
10   end
11
12   def nombre
13     return @nombre
14   end
15
16   def prueba (nombre)
17     puts nombre
18     puts self.nombre
19   end
20 end
```

# Especificadores de acceso (Visibilidad)

- Existen distintos niveles de acceso a atributos y métodos
- A este respecto hay diferencias importantes entre lenguajes
- En general:
  - ▶ **Privado:** sin acceso desde otra clase y/o desde otra instancia
  - ▶ **Paquete:** sin restricciones dentro del mismo paquete (no procede en Ruby)
  - ▶ **Público:** sin restricciones de acceso
- Este tema se abordará con detalle en otra lección

# Acceso privado: Diferencias entre Java y Ruby

## ● Java

se puede acceder a elementos **privados** (métodos y atributos)

- ▶ Desde una instancia a otra instancia de la misma clase
- ▶ Desde el ámbito de clase a una instancia de esa clase
- ▶ Desde el ámbito de instancia a la clase de la que se es instancia

## ● Ruby

- ▶ Todo lo anterior no está permitido en Ruby
- ▶ Los atributos siempre son privados

# Ejemplos de visibilidad

## Ruby: Visibilidad

```
1 class Prueba
2
3   def self.metodoClasePublico
4     puts "público de clase"
5   end
6
7   private # solo afecta a los métodos de instancia
8   def self.metodoClasePrivado # sigue siendo público
9     puts "privado de clase"
10  end
11
12  def metodoInstanciaPrivado
13    puts "privado de instancia"
14  end
15
16  # Así también se hace privado el método de instancia
17  private :metodoInstanciaPrivado
18
19  # Así se hacen privados los métodos de clase
20  private_class_method :metodoClasePrivado
21 end
22
23 Prueba.metodoClasePublico
24 #Prueba.metodoClasePrivado           # Error, es privado
25 #Prueba.metodoInstanciaPrivado        # Error, es de instancia
26 #Prueba.new.metodoInstanciaPrivado    # Error, privado
27 #Prueba.new.metodoClasePublico        # Error, es de clase
```

# Ejemplos de visibilidad

## Java: Visibilidad

```
1 public class UnaClase {
2
3     public void metodoPublico() { System.out.println("Público"); }
4
5     private void metodoPrivado() { System.out.println("Privado"); }
6
7     // Todo esto funciona en Java aunque llame la atención
8     public void usoDentroDeClase() {
9         metodoPrivado();
10    }
11
12    public void usoConOtroObjeto() {
13        UnaClase obj2 = new UnaClase();
14        obj2.metodoPrivado();
15    }
16
17    public static void main(String [] args) { // Seguimos en UnaClase
18        UnaClase obj = new UnaClase();
19        obj.metodoPublico();
20        obj.metodoPrivado();
21        obj.usoDentroDeClase();
22        obj.usoConOtroObjeto();
23    }
24 }
```



# Ejemplos de visibilidad

## Ruby: Visibilidad

```
1 class UnaClase
2
3   def metodoPublico
4     puts "Publico"
5   end
6
7   def usoDentroDeClase
8     metodoPrivado
9   end
10
11  def usoConOtroObjeto
12    obj2 = UnaClase.new
13    # obj2.metodoPrivado # error, privado de otra instancia
14  end
15
16  private
17  def metodoPrivado
18    puts "Privado"
19  end
20 end
21
22 obj = UnaClase.new
23 obj.metodoPublico
24 # obj.metodoPrivado # error, privado
25 obj.usoDentroDeClase
26 obj.usoConOtroObjeto
```

# Visibilidad



- ¿Qué visibilidad asignar a atributos y métodos?
  - ▶ Por regla general, **la más restrictiva**
  - ▶ **Privada para los atributos**
    - ★ Para los que necesiten ser leídos desde fuera de la clase, se creará un método con visibilidad de paquete o público (según corresponda) que proporcionará dicho atributo *al exterior* (**consultor**)
    - ★ **¡Cuidado!** Si lo que se proporciona es una **referencia**, el atributo podría ser modificado desde fuera de la clase
    - ★ Para los que necesiten ser modificados desde fuera de la clase, se creará un método con la visibilidad adecuada que reciba los parámetros necesarios y, tras realizar las comprobaciones pertinentes, realice la modificación (**modificador**)
    - ★ Los atributos de un objeto no deberían ser modificados por métodos distintos de los propios de dicho objeto (o de su clase)
    - ★ Solo se crearán los consultores y modificadores necesarios, y con la visibilidad más restrictiva que sea posible

# Atributos y Métodos

Prof. Francisco Velasco Anguita

Dpto. Lenguajes y Sistemas Informáticos  
Universidad de Granada

Programación y Diseño Orientado a Objetos

Doble Grado en Ingeniería Informática  
y Administración y Dirección de Empresas  
(Curso 2024-2025)