

Paquetes, clases y objetos

Objetivos

- 1) Establecer un espacio de trabajo y definir paquetes
- 2) Definir paquetes anidados
- 3) Entender que es una clase y un objeto
- 4) Saber definir una clase en un paquete
- 5) Saber definir atributos de una clase
- 6) Saber definir atributos constantes de una clase

Métodos get, set y constructor de una clase

Objetivos

- 1) Saber qué es y para qué sirven los métodos de una clase
- 2) Saber definir métodos get
- 3) Saber definir métodos set y uso de “this”
- 4) Saber definir constructores y destructores de confianza
- 5) Uso de “this()” y sobrecarga de constructores
- 6) Saber diferenciar entre el constructor implícito y por defecto.
- 7) Creación de objetos de una clase
- 8) Acceso a los métodos de una clase. Anidamiento de métodos
- 9) Control de errores en métodos set: Anticipación y captura

Métodos de cálculo, cambio de estado y conversión de cadenas

Objetivos

- 1) Saber definir métodos de cálculo
- 2) Uso de arrays y colecciones de objetos
- 3) Uso de métodos con parámetros un tipo de objeto
- 4) Saber definir métodos de consulta
- 5) Saber definir métodos de cambio de estado
- 6) Saber que todas las clases derivan de la clase Object
- 7) Métodos de conversión de cadenas. Reemplazamiento del método “toString”
- 8) Sobrecarga de métodos

Enumerados

Objetivos

- 1) Saber definir un enumerado
- 2) Saber declarar y definir variables de tipo enumerado.
- 3) Lectura, acceso y escritura de variables enumeradas
- 4) Comparación de variables enumeradas
- 5) Saber definir un enumerado en una clase
- 6) Métodos get, set y constructor para atributos enumerados.
- 7) Saber definir enumerados con constructor

Clonación , igualdad y comparación de objetos

Objetivos

- 1) Saber definir alias de objetos
- 2) Clonar superficialmente objetos
- 3) Saber distinguir entre mover y copiar objetos
- 4) Saber para que sirve el operador “instanceof”
- 5) Usar el operador de conversión de tipos de objetos
- 6) Saber cuándo dos objetos son alias
- 7) Reemplazar el método “equals”
- 8) Comparar dos objetos con el método “compareTo”

Atributos de tipo un objeto

Objetivos

- 1) Saber definir atributos de tipo un objeto o una colección
- 2) Saber definir métodos get y set moviendo atributos de objetos
- 3) Saber definir método get y set copiando atributos de objetos y colecciones de tipos básicos, envoltorio, cadena, BigDecimal o BigInteger
- 4) Saber definir métodos get y set copiando atributos de colecciones de objetos.
- 5) Saber definir métodos get(i), set(i), size(), add(i) y remove(i) para colecciones moviendo o copiando.

Atributos y métodos estáticos

Objetivos

- 1) Saber qué es un atributo estático o de clase
- 2) Saber definir constantes estáticas
- 3) Uso del constructor estático
- 4) Saber definir métodos estáticos: Métodos de fabricación y validación
- 5) Saber definir atributos estáticos variables
- 6) Saber definir métodos get y set para atributos estáticos
- 7) Uso de métodos y constantes estáticas en métodos de objeto.
- 8) Definir distintos modos para comparar objetos

Limpiadores

Objetivos

- 1) Saber la función del recolector de basuras
- 2) Conocer las formas de cómo se puede dejar memoria para ser liberada
- 3) Saber qué es un limpiador y cuándo se ejecuta
- 4) Saber definir un limpiador o finalizador asociado a cada objeto
- 5) Conocer la forma de pasar atributos al finalizador
- 6) Clonar el limpiador
- 7) Uso del método “close” para lanzar el finalizador

Eventos y documentación

Objetivos

- 1) Saber qué es un evento: tipos y controladores
- 2) Saber definir gestores de eventos asociados a un objeto
- 3) Saber definir métodos get de los gestores de eventos
- 4) Saber lanzar los eventos y ejecutar los controladores
- 5) Saber clonar los gestores de eventos
- 6) Saber que es un escuchador en una aplicación
- 7) Saber definir el controlador asociado a un escuchador
- 8) Saber asociar a un objeto un escuchador
- 9) Saber eliminar un escuchador asociado a un objeto
- 10) Documentar los miembros de una clase

UML

Objetivos

- | | |
|--|----------------------------------|
| 1) Paquetes | 7) Reemplazamiento de métodos |
| 2) Clases | 8) Enumerados |
| 3) Atributos de objetos | 9) Enumerados internos |
| 4) Atributos constantes de objetos | 10) Enumerados con constructor |
| 5) Métodos get, set y constructores | 11) Implementación de interfaces |
| 6) Métodos de cálculo, consulta, cambio de estado y de conversión de cadenas | 12) Miembros estáticos |
| | 13) Clases estáticas internas |
| | 14) Asociaciones |

Paquetes, clases y objetos

Objetivos

- 1) Establecer un espacio de trabajo y definir paquetes
- 2) Definir paquetes anidados
- 3) Entender que es una clase y un objeto
- 4) Saber definir una clase en un paquete
- 5) Saber definir atributos de una clase
- 6) Saber definir atributos constantes de una clase

Espacio de trabajo y paquetes

Un **espacio de trabajo** en java es un directorio donde se define un archivo java con el método principal o “main” y una serie de subdirectorios o **paquetes** donde se definen el resto de archivos .java necesarios para que funcione la aplicación

 **et** Espacio de trabajo Se definirá en este tema un único espacio de trabajo de nombre “**Tema 04**”

 **Aplicacion.java** Aplicaciones que contienen el método “main”

 **clases** Paquete “clases” En el paquete “clases” se definirán todas las clases.

 **fsg** Paquete “fsg”

 **in.java**

 **lib.java**

Los archivos “in.java” y “lib.java” son dos clases definidas en el paquete “fsg”. En tales archivos se definirá una primera línea indicando el paquete donde están definidas dichas clases:

```
package fsg;
```

Paquetes anidados

Dentro de un subdirectorio del espacio de trabajo se pueden definir otros subdirectorios o paquetes.

Tema 04

clases

Todas las clases definidas en el paquete “clases” tendrán como primera línea de código

package clases;

Paquetes
anidados

figuras

vehiculos

Todas las clases definidas en el paquete “enumerados” tendrán como primera línea de código

package clases.figuras;

Dentro de cada paquete se pondrán los siguientes elementos:

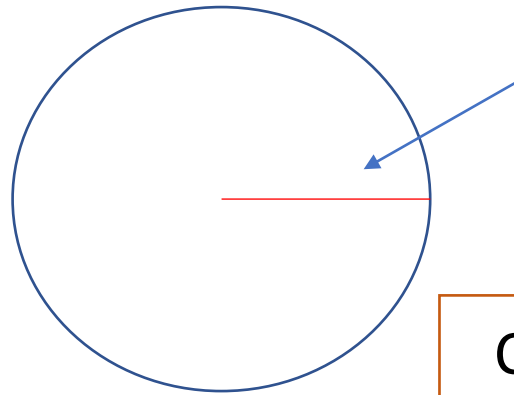
- Clases
- Enumerados
- Interfaces

Todas las clases definidas en el paquete “vehiculos” tendrán como primera línea de código

package clases.vehiculos;

Concepto de clase

Una clase será un tipo de datos que se define en un paquete con sus características (o atributos) y sus métodos.



Una circunferencia queda definida por su radio

Para cada circunferencia se puede obtener su longitud y su área a partir del valor del radio

Clase: Circunferencia
Paquete: clases.figuras

Atributos:

- radio

Métodos:

- longitud
- area

MIEMBROS DE LA CLASE
CIRCUNFERENCIA

Se define como **miembros de una clase** a todos sus atributos y métodos

Clase Persona

Una persona puede tener las siguientes características y métodos:

Atributos

nombre

edad

peso

altura

edad inicial

Primer valor que se le asigna a una persona como edad

Métodos

cumplir

engordar

adelgazar

crecer

indiceMasaCorporal



Clase: Persona
Paquete: clases

Atributos:

- nombre
- edad
- peso
- altura
- edad inicial

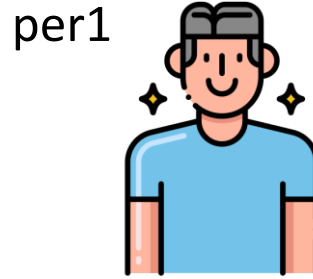
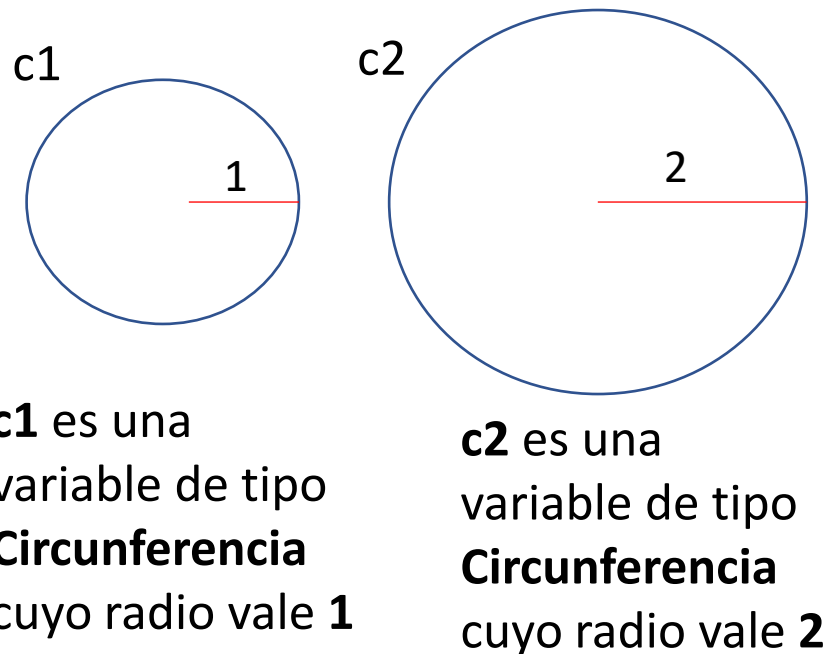
Métodos:

- cumplir
- engordar
- adelgazar
- crecer
- indiceMasaCorporal

MIEMBROS DE LA CLASE PERSONA

Objetos de una clase

Un objeto de una clase es una variable de tipo la clase en cuestión con unos valores determinados para cada uno de los atributos de dicha clase.



per1 es una variable de tipo **Persona** que se llama **Juan**, tiene **22** años, mide **1,76** m y pesa **69** kg

Como se verá después, los objetos se crean en una aplicación con el método del constructor




```
Circunferencia c1 = new Circunferencia(1);  
Circunferencia c2 = new Circunferencia(2);  
Persona per1 = new Persona("Juan", 22, 1.76, 69);
```

Definición de la clase Circunferencia

 `clases`

 `figuras`

 `Circunferencia.java`

```
package clases.figuras;  
public class Circunferencia
```

```
{  
}  
}
```

El nombre del archivo .java debe ser el mismo que el nombre de la clase

Modificadores de acceso de una clase:

- `public` → Se podrán definir objetos de la clase en cualquier archivo .java definido en el espacio de trabajo
- `package` (nada) → Sólo se pueden crear objetos de la clase en archivos definidos en su paquete

```
package clases.figuras;  
class Circunferencia
```

```
{  
}  
}
```

El nombre del archivo .java puede ser distinto del nombre de la clase

 **TODAS LAS CLASES SE DEFINIRAN POR DEFECTO COMO PÚBLICAS**

Definición de clases. Clase Persona

[Clases UML](#)




clases

 Persona.java



```
package clases;  
public class Persona  
{  
}
```

figuras

 Circunferencia.java



```
package clases.figuras;  
public class Circunferencia  
{  
}
```

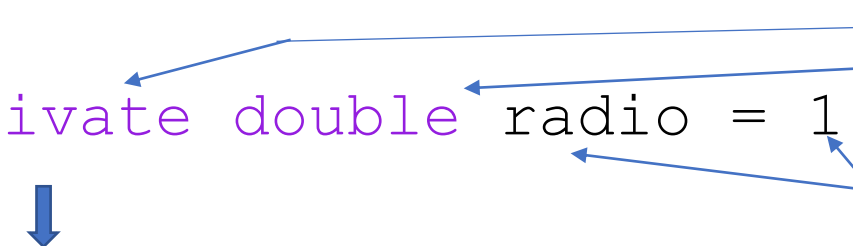
El nombre del archivo .java debe ser el mismo que el nombre de la clase

La notación “**clases.figuras.Circunferencia**” denota la clase **Circunferencia** que se define en el paquete **clases/figuras** del espacio de trabajo (subdirectorio **figuras** definido en el directorio **clases** del espacio de trabajo)

Atributos de una clase

Los atributos de una clase se definen dentro de la clase con las siguientes características:

```
package clases.figuras;  
public class Circunferencia  
{  
    private double radio = 1;  
}
```



El modificador de acceso “private” a un miembro de una clase significa que sólo se puede acceder a dicho miembro en el archivo donde se define o declara

- Modificador de acceso privado
- Tipo del atributo
- Nombre del atributo
- Valor inicial del atributo

Otros modificadores:

- protected → Sólo se puede acceder a ese miembro en la clase donde se define y en las clases que deriven de dicha clase.
- package → No se pone nada. Sólo se puede acceder a ese miembro en la clase donde se define, y en las clases de su mismo paquete.
- public → Se puede acceder a ese miembro desde cualquier clase.

Todos los atributos se deben definir como privados con el fin de que sea la clase la que se encargue de que dichos atributos tengan valores coherentes para todos los objetos de la clase. Por ejemplo, no es posible que una circunferencia tenga como radio un valor negativo o cero



Atributos de la clase Persona

En la clase Persona se definen los atributos de “edad”, “altura” y “peso” con los valores por defecto de 0, 0.5 y 2.5, respectivamente.

```
package clases;
public class Persona
{
    private int edad; ←
    private double altura = 0.5;
    private double peso = 2.5;
}
```

Los atributos (variables) si no se les asignan un valor toman como valor por defecto el nulo del tipo (El 0 para los tipos numéricos; el '\0' para los caracteres; el valor false, para los booleanos y "" para las cadenas)



Atributos constantes

Atributos constantes de objeto en UML

Un atributo constante es un atributo de una clase al cual se le asigna un valor que no se puede modificar durante la ejecución.

En la clase Persona los atributos “**nombre**” y “**edad inicial**” son constantes

```
package clases;  
public class Persona  
{  
    public final String NOMBRE;  
    public final int EDAD_INICIAL;  
}
```

Los atributos constantes suelen ser público

Un atributo constante se inicializa en su declaración o en el constructor de la clase

El atributo constante EDAD_INICIAL almacena el valor de la edad de la persona al ser creada. Depende del atributo “edad” ya que el primer valor que se asigne a dicho atributo, debe asignarse también a la constante

Este código da error ya que las constante no están inicializadas

Constantes independientes y dependientes

Métodos get, set y constructor de una clase

Objetivos

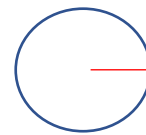
- 1) Saber qué es y para qué sirven los métodos de una clase
- 2) Saber definir métodos get
- 3) Saber definir métodos set y uso de “this”
- 4) Saber definir constructores y constructores de confianza
- 5) Uso de “this()” y sobrecarga de constructores
- 6) Saber diferenciar entre el constructor implícito y por defecto.
- 7) Creación de objetos de una clase
- 8) Acceso a los métodos de una clase. Anidamiento de métodos
- 9) Control de errores en métodos set: Anticipación y captura

Métodos de una clase. Clase Circunferencia

Los métodos de una clase sirven para acceder a los atributos, modificarlos, inicializarlos, convertirlos a cadena, hacer consultas y realizar cálculos

Se tienen los siguientes tipos de métodos:

- Get
- Set
- Constructor
- De cálculo
- De consulta
- De cadena
- De cambio de estado



Firma de un método:

Acceso Tipo Nombre(parámetros)

público	double	getRadio()
público	Circunferencia	setRadio(double radio)
público		Circunferencia(double radio)
público	double	longitud()
público	double	area()
público	boolean	esUnitaria()
publico	String	toString()
público	Circunferencia	duplicar()



Métodos get

Como todos los métodos se escribe dentro de la clase, a partir de ahora sólo se escribirá el método

Los métodos get son públicos y devuelven el valor de los atributos variables.

```
package clases.figuras;  
public class Circunferencia  
{  
    private double radio = 1;
```

```
    public double getRadio()  
    {  
        return radio;  
    }  
}
```

Características del método que accede al valor del atributo radio:

- Tiene el mismo tipo que el atributo
- El nombre del método es “get” seguido del nombre del atributo en mayúsculas
- Tiene una sola instrucción que devuelve el valor del atributo

EL ATRIBUTO **radio** SÓLO **DEBE** SER ACCEDIDO PARA CONSULTAR SU VALOR EN SU MÉTODO GET



En los demás métodos se accederá al valor del atributo **radio** con el método **getRadio()**

Métodos set. Clase Circunferencia.

Los métodos set son métodos públicos que se utilizan para asignar valores a los atributos y controlar que dichos valores son coherentes.

No se devuelve nada

Tipo del atributo a modificar

Nuevo valor del atributo "radio"

```
public void setRadio(double nuevoRadio)
{
    if (nuevoRadio <= 0)
        throw new IllegalArgumentException("Radio positivo");

    radio = nuevoRadio;
}
```

Se comprueba si el atributo es incoherente

Mensaje de error

Se lanza un error si el valor que se quiere asignar al atributo fuera incorrecto. "throw" termina la ejecución del método. Se pueden lanzar un mensaje de error para condición de error

Al atributo se le asigna el nuevo valor

EL ATRIBUTO **radio** SÓLO **DEBE** SER MODIFICADO EN SU MÉTODO SET

En los demás métodos se modifica el valor del atributo **radio** con el método **setRadio(valor)**

Métodos set. Uso de this

En los métodos set puede coincidir el nombre del atributo y el del parámetro del método. Para distinguirlos se usa **this**. También se puede devolver el objeto que se está modificando con **this**.

```
public Circunferencia setRadio(double radio)
{
    ...
```

El nombre del parámetros coincide con el del atributo

El método “set” devuelve un objeto de la clase en cuestión

```
    this.radio = radio;
```

```
    return this;
```

```
}
```

“this.radio” representa el atributo y “radio” el parámetro

Dependiendo de que el tipo del método sea “void” o la clase se utilizará una u otra versión

Los método “set” son de cambio de estado, ya que modifican el valor de un atributo

Se devuelve el objeto que se ha modificado



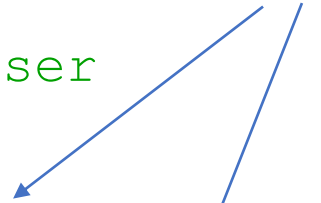
Métodos set

```
public Persona setAltura(double altura)
{
    if(altura<0.3)
        throw new IllegalArgumentException("La altura tiene que ser
mayor o igual que 0.3");
    if(altura>2.2)
        throw new IllegalArgumentException("La altura tiene que ser
menorr o igual que 2.2");
    if((int) (altura*100)/100.0 != altura)
        throw new IllegalArgumentException("La altura tiene que tener
como máximo 2 decimales");

    this.altura = altura;

    return this;
}
```

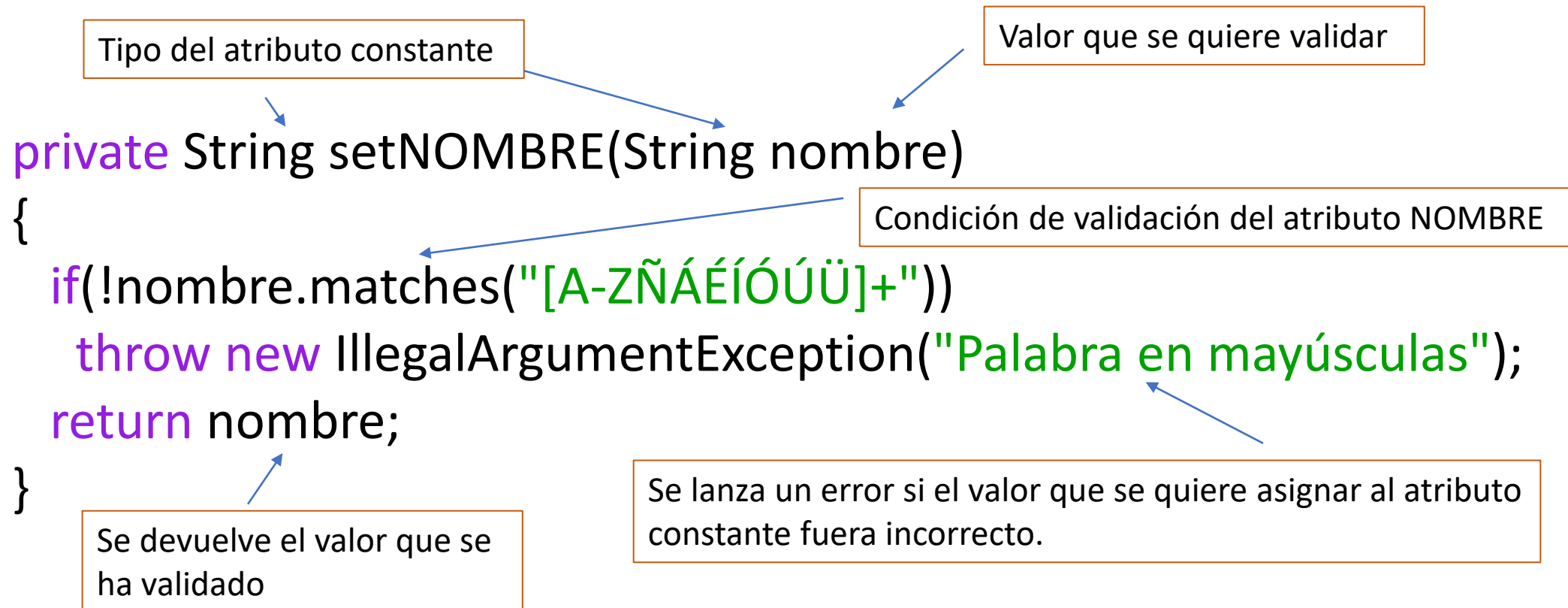
Se lanza un error específico para cada posible motivo de dato inválido





Métodos set de constantes independientes

Los métodos set de constantes son métodos privados que comprueban si el valor que se quiere asignar al atributo constante es o no correcto.



Constructor

El constructor es un método especial que se utiliza para inicializar un objeto de la clase, asignando valores a sus atributos.

Características:

- Tiene el mismo nombre que el de la clase
- No tiene tipo de devolución (ni “void”)
- El modificador de acceso suele ser público.

Constructor en la clase Persona que inicializa todos sus atributos

```
public Persona(String nombre, int edad,  
double altura, double peso)
```

Valores de cada uno de los atributos

```
{  
    NOMBRE = setNOMBRE(nombre);  
    setEdad(edad);  
    setAltura(altura);  
    setPeso(peso);  
    EDAD_INICIAL = getEdad();  
}
```

Se ejecutan los métodos set de cada atributo variable y constante

Se establecen las constantes dependientes

ORDEN: 1. Constantes independientes 2. Variables 3. Constantes dependientes



Constructor de confianza

Un constructor de confianza o contrato es un constructor que asigna a los atributos de la clase su correspondiente parámetro sin validar el valor de tales parámetros

```
public Persona(String nombre, int edad, double altura, double peso)
{
    NOMBRE = nombre;
    this.edad = edad;
    this.altura = altura;
    this.peso = peso;
    EDAD_INICIAL = edad;
}
```

Se tiene la confianza o el contrato con aquellas aplicaciones o librerías que van a crear un objeto de tipo Persona que se pasarán como parámetros valores que cumplen las restricciones

- El nombre será una palabra en mayúsculas
- La edad estará entre 0 y 120, ambos inclusive;
- La altura tendrá como máximo 2 decimales y estará entre 0,3 y 2,2, ambos inclusive
- El peso tendrá como máximo 1 decimal y estará entre 1,7 y 140, ambos inclusive



Sobrecarga de constructores

Se pueden definir dos o más constructores. Sólo tienen que diferenciarse por el número de parámetros o por el tipo de los mismos.

Constructor en la clase Persona que inicializa el nombre, la altura y el peso

```
public Persona(String nombre, double altura , double peso)
{
    NOMBRE = setNOMBRE(nombre);
    setAltura(altura);
    setPeso(peso);
    EDAD_INICIAL = getEdad();
}
```

<pre>NOMBRE = nombre; this.edad = edad; this.altura = altura; EDAD_INICIAL = edad;</pre>
--

En todos los constructores se tienen que inicializar los atributos constantes que no se han inicializado en su declaración

Uso de this()

La expresión this() se utiliza en un constructor para referenciar otro constructor ya definido.



Definir constructores en la clase Persona para inicializar de todas las formas posibles el nombre, la altura y el peso

```
public Persona(String nombre,  
double altura , double peso)  
{  
    NOMBRE = nombre;  
    this.altura = altura;  
    this.peso = peso;  
    EDAD_INICIAL = edad;  
}
```

Este constructor podría redefinirse invocando el constructor que inicializa todos los atributos

La instrucción this() debe ser la primera instrucción

Sólo hay 3 posibilidades

```
public Persona(double altura, String nombre,  
double peso)  
{
```

```
    this(nombre,altura,peso);  
}
```

```
public Persona(double altura, double peso,  
String nombre)  
{
```

```
    this(nombre,altura,peso);  
}
```

```
public Persona(String nombre,  
double altura , double peso)  
{
```

```
    this(nombre,2,altura,peso);  
}
```

A la edad se le asigna su valor por defecto



Constructor implícito y por defecto

Si en una clase no se tiene constructores, se puede utilizar para crear los objetos el constructor implícito.

Constructor implícito en Persona

```
public Persona()  
{  
}  
}
```

Todos los atributos toman como valor el asignado en su declaración

El uso del constructor implícito de una clase, presupone que no se han definido constructores en esa clase y que todas las constantes se les ha asignado un valor en su declaración.

Si en la clase se ha definido algún constructor, para poder utilizar el constructor implícito hay que definirlo explícitamente (constructor por defecto)

Constructor por defecto en Persona

```
public Persona()  
{  
    NOMBRE = "DESCONOCIDO";  
    EDAD_INICIAL = edad;  
}
```

En el constructor por defecto hay que asignar un valor a todas las constantes.

Creación de objetos de una clase

Para crear un objeto en una aplicación es necesario:

- Importar la clase
- Declarar una variable de tipo dicha clase
- Inicializarla con un constructor

En una aplicación lo primero es importar las clases para poder definir objetos de dichas clases

`import` clases.Persona;

En el main (o en cualquier otro método) se declaran e inician objetos de una clase

`Persona p = new` Persona("LUIS",23,1.2,75);

El operador **"new"** tiene como argumento un constructor de una clase y como resultado un objeto de esa clase

Acceso a los métodos de una clase

Una vez creado un objeto de una clase

```
Persona p = new Persona("LUIS",23,1.2,75);
```

se puede acceder a los miembros públicos a partir del nombre del objeto de la siguiente manera:

p.EDAD_INICIAL ←

Se devuelve un entero con el valor inicial de la edad de “p”

p.getEdad() ←

Se devuelve un entero con el valor de la edad de “p”

p.setAltura(1.75) ←

Si el método “setAltura” es de tipo “void”, se asigna como nuevo valor de la altura de “p” es de 1.75 si es correcto; si no, se lanza un error. Además, si el método es de tipo “Persona”, se devuelve el objeto “p”

Control de errores: método set

Al asignar un valor a un atributo, que tiene restricciones, se pueden lanzar errores. Hay tres formas de tratar dichos errores:

- No hacer nada
- Anticiparse
- Capturar

No hacer nada

//edad es de tipo int
//p es de tipo Persona

Generar un número aleatorio entre 10 y 20

```
(int) (Math.random()*11)+10;
```

```
p.setEdad( (1) );
```



```
edad = (1)  
p.setEdad(edad);
```

Leer un entero entre 10 y 20

(1) El valor es válido

```
in.leerInt("EDAD: ",v->v>=10 && <=20);
```

Control de errores: Anticiparse

Leer un entero o generar un número aleatorio entre -10 y 10 y asignárselo, cuando sea válido, al atributo “edad”

//edad es de tipo int
//p es de tipo Persona

edad>0

Condición para
que la “edad”
sea válida

Anticiparse

```
while(true)
```

```
{
```

```
    edad = (1) ;
```

```
    if( (3) ) break;
```

```
    (2)
```

```
}
```

```
p.setEdad(edad);
```

Generar un número
aleatorio entre -10 y 10

```
(int) (Math.random()*21)-10;
```

Leer un entero
entre -10 y 10

```
in.leerInt(“EDAD: ”,v->v>=-10 && <=10);
```

```
System.out.println(“Edad incorrecta”);
```

(1) El valor puede ser inválido

(2) Mensaje de error

(3) Condición de validación



Control de errores: Capturar

Leer un entero o generar un número aleatorio entre -10 y 10 y asignárselo, cuando sea válido, al atributo “edad”

//edad es de tipo int
//p es de tipo Persona

p.setEdad((1));

Capturar el error

```
while(true) try {  
    edad = (1)  
    p.setEdad(edad);  
    break;  
}  
catch(Exception e)  
{  
    (2)  
}
```

Generar un número aleatorio entre -10 y 10
(int) (Math.random()*21)-10;

Leer un entero entre -10 y 10
in.leerInt("EDAD: ",v->v>=-10 && <=10);

System.out.println("Edad incorrecta");

(1) El valor puede ser inválido

(2) Mensaje de error

Instrucciones con métodos de clases

Crear una circunferencia de radio 3 y otra con valores por defecto

Circunferencia c = **new** Circunferencia(3), c1 = **new** Circunferencia();

Incrementar en una unidad el radio

//r es de tipo double

```
r = c.getRadio();  
r++;  
c.setRadio(r);
```



```
c.setRadio(c.getRadio()+1);
```

Duplicar el radio

//r es de tipo double

```
r = c.getRadio();  
r*=2;  
c.setRadio(r);
```



```
c.setRadio(2*c.getRadio());
```

Intercambiar los radios de c y c1

//r es de tipo double

```
r = c.getRadio();  
c.setRadio(c1.getRadio());  
c1.setRadio(r);
```

Reinicializar c

```
c = new Circunferencia(5);
```

Asignar a c el valor nulo c = **null**;

Instrucciones con métodos de clases (II)

Introducir por consola un radio válido

//c es una circunferencia

//r es de tipo double

```
r = in.leerDouble("RADIO: ",v->v>0);  
c.setRadio(r);
```



```
c.setRadio(in.leerDouble("RADIO: ",v->v>0));
```

Mostrar el radio por consola

//r es de tipo double

```
r = c.getRadio();  
System.out.println(r);  
System.out.printf("Radio:  
%1.2f\n",r);
```



```
System.out.println(c.getRadio());  
System.out.printf("Radio:  
%1.2f\n",c.getRadio());
```

Instrucciones con métodos de clases (III)

**Crear un rectángulo
leyendo un ancho y
alto entre 2 y 10**

//ancho y alto de tipo int

//rec de tipo Rectangulo

ancho = in.leerInt("ANCHHO: ",v>=2 && v<=10);

alto = in.leerInt("ALTO: ",v>=2 && v<=10);

rec = new Rectangulo(ancho,alto);

//rec de tipo Rectangulo

rec = new Rectangulo(
in.leerInt("ANCHHO: ",v>=2 && v<=10),
in.leerInt("ALTO: ",v>=2 && v<=10)
);

Anidamiento de métodos de clases

Los métodos que devuelven el objeto que invocan el método (return this) se pueden anidar.

//p es de tipo Persona y no es null

p.setEdad(24).setAltura(1.7).setPeso(86.7);



```
p.setEdad(24);  
p.setAltura(1.7);  
p.setPeso(86.7);
```

setEdad
devuelve "p"

└─ p.setAltura(1.7).setPeso(86.7);

setAltura devuelve "p"

└─ p.setPeso(86.7);

setPeso devuelve "p"

└─ p

Persona p = new Persona("LUIS",23,1.2,75);

Anidamiento de métodos de clases (II)

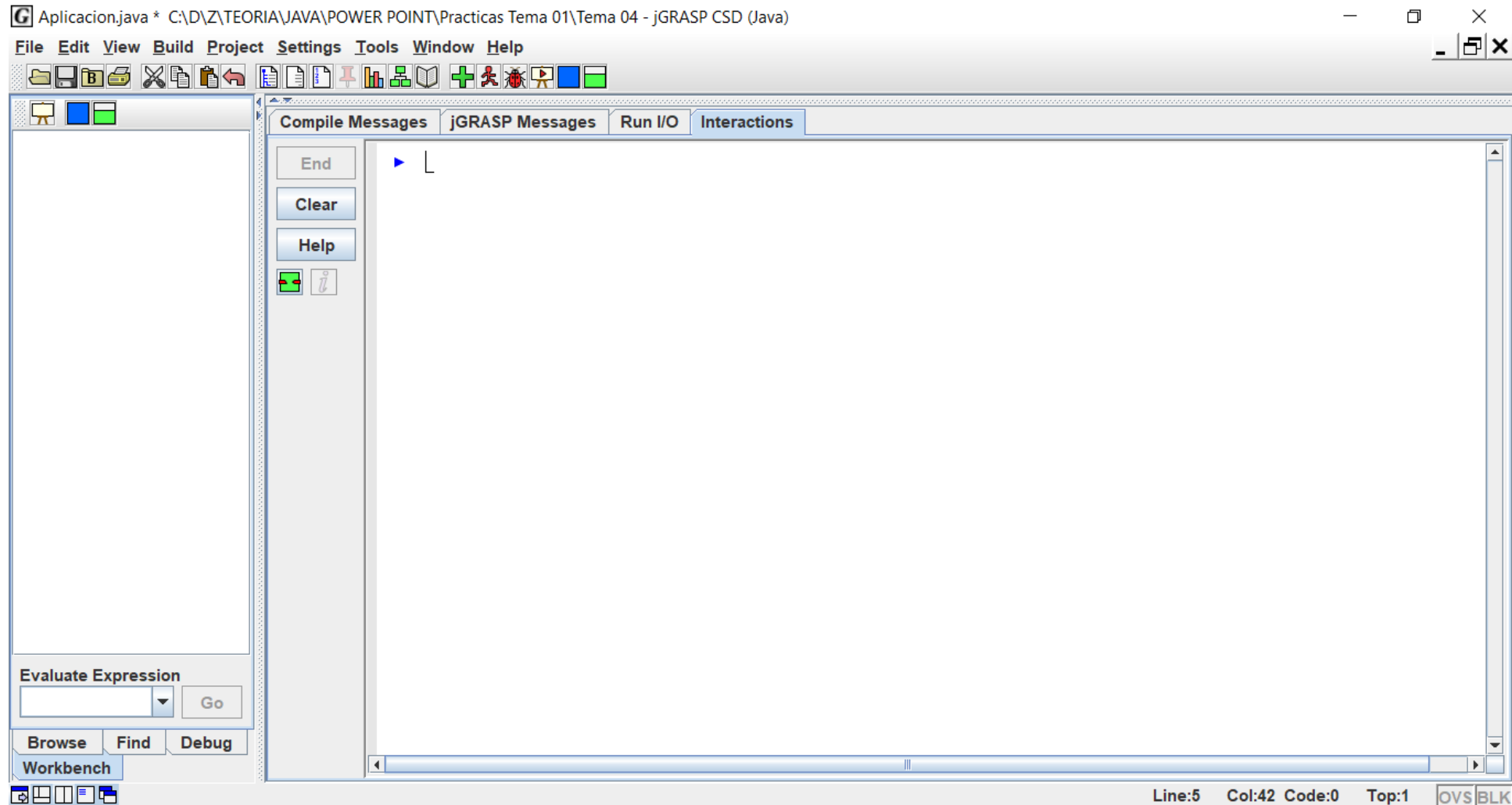
Modificar la edad a 80 y el peso a 65.6 y mostrar por consola la altura

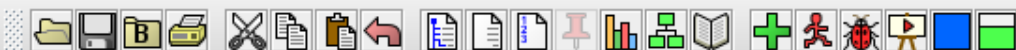
```
System.out.println(p.setEdad(80).setPeso(65.6).getAltura());
```

Crear “p” por defecto y leer la edad y el peso

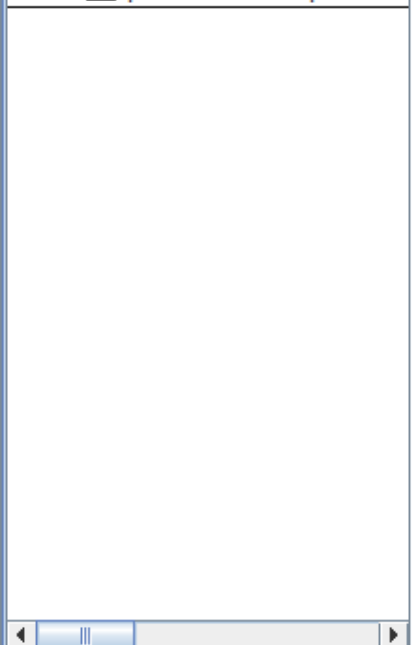
```
Persona p = new Persona().setEdad(in.leerInt("EDAD: ",v->v>0 && v<14))  
.setAltura(in.leerDouble("ALTURA: ",v-> v>=0.3 && v=2.2 && v == (int)  
(v*100)/100.0));
```

Uso de las ventanas Interactions y Workbench





p --> (obj 126 : clases.Per
 NOMBRE --> "LUIS" (o
 EDAD_INICIAL = 23 :
 edad = 23 : private int
 altura = 1.42 : private
 peso = 43.5 : private d



Evaluate Expression

Go

Browse Find Debug

Workbench

Status: interactions active

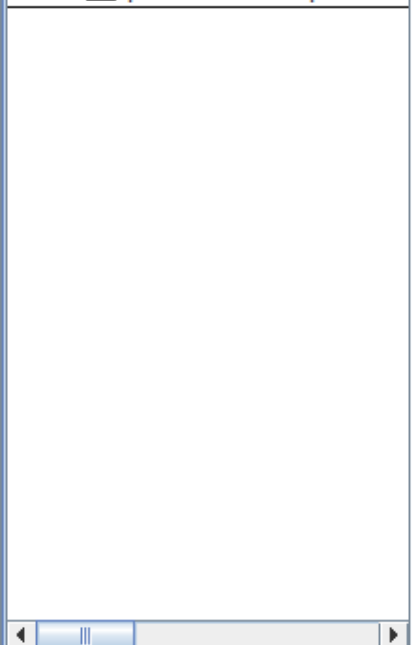
Compile Messages jGRASP Messages Run I/O Interactions

End
 Clear
 Help

```
▶ import clases.Persona;
▶ Persona p = new Persona("LUIS",23,1.42,43.5);
▶
```



p --> (obj 126 : clases.Per
 NOMBRE --> "LUIS" (o
 EDAD_INICIAL = 23 :
 edad = 23 : private int
 altura = 1.42 : private
 peso = 43.5 : private



Evaluate Expression

Go

Browse Find Debug

Workbench

Status: interactions active

Compile Messages jGRASP Messages Run I/O Interactions

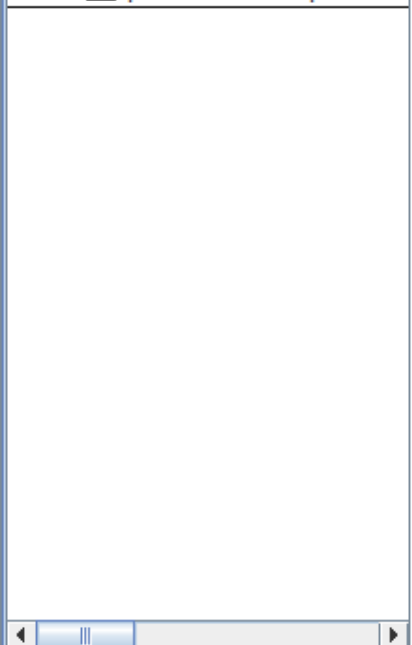
End
 Clear
 Help

```

▶ import clases.Persona;
▶ Persona p = new Persona("LUIS",23,1.42,43.5);
▶ System.out.println(p.getEdad());
▶ 23
▶ L
    
```



p --> (obj 126 : clases.Per
 NOMBRE --> "LUIS" (o
 EDAD_INICIAL = 23 :
 edad = 24 : private int
 altura = 1.42 : private
 peso = 43.5 : private



Evaluate Expression

Go

Browse Find Debug

Workbench

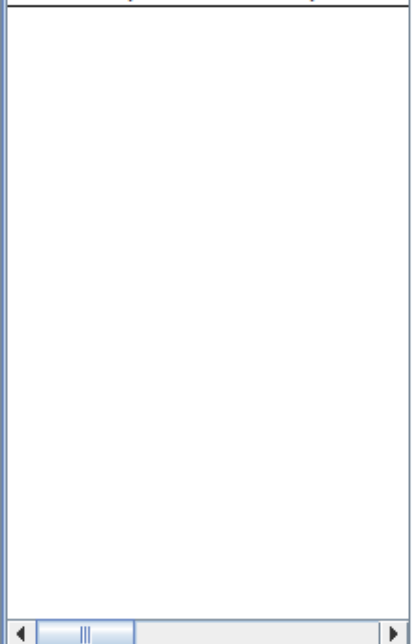
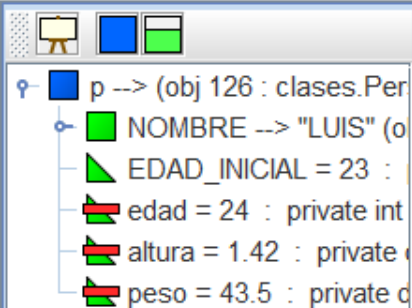
Status: interactions active

Compile Messages jGRASP Messages Run I/O Interactions

End
 Clear
 Help

```

▶ import clases.Persona;
▶ Persona p = new Persona("LUIS",23,1.42,43.5);
▶ System.out.println(p.getEdad());
    23
▶ p.setEdad(24);
▶
    
```



Evaluate Expression

Go

Browse Find Debug

Workbench

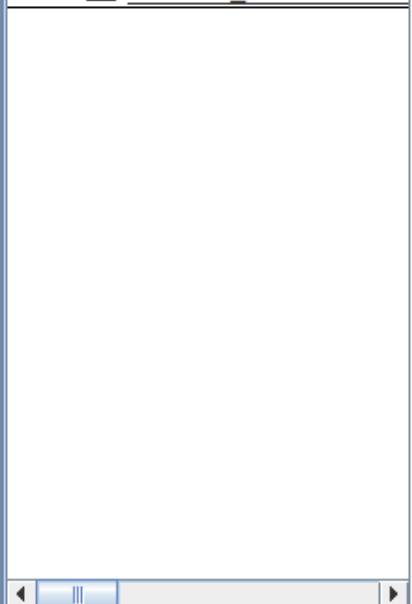
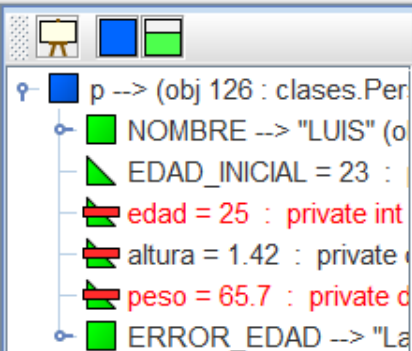
Status: interactions active

Compile Messages jGRASP Messages Run I/O Interactions

End
Clear
Help

```

▶ import clases.Persona;
▶ Persona p = new Persona("LUIS",23,1.42,43.5);
▶ System.out.println(p.getEdad());
  23
▶ p.setEdad(24);
▶ System.out.println(p.getEdad()+p.getAltura());
  25.42
▶
    
```



Evaluate Expression

 Go

Browse Find Debug

Workbench

Status: interactions active

Compile Messages jGRASP Messages Run I/O Interactions

End

Clear

Help



```
▶ import clases.Persona;
▶ Persona p = new Persona("LUIS",23,1.42,43.5);
▶ System.out.println(p.getEdad());
23
▶ p.setEdad(24);
▶ System.out.println(p.getEdad()+p.getAltura());
25.42
▶ p.setEdad(25).setPeso(65.7);
▶ |
```


Aplicaciones con objetos

Hágase un programa que cree dos circunferencias por defecto, lea en ambas un radio entre 0 y 2, intercambie los radios de las dos circunferencias y los muestre.

```
Circunferencia c1 = new Circunferencia();  
Circunferencia c2 = new Circunferencia();  
double aux = 0;
```

```
c1.setRadio(in.leerDouble("RADIO: ",v->v>0 && v<2));  
c2.setRadio(in.leerDouble("RADIO: ",v->v>0 && v<2));  
aux = c1.getRadio();  
c1.setRadio(c2.getRadio());  
c2.setRadio(aux);  
System.out.println(c1.getRadio());  
System.out.println(c2.getRadio());
```



Salida por
consola

```
RADIO: 0,6  
RADIO: 1,7  
1.7  
0.6
```

Métodos de cálculo, cambio de estado y conversión de cadenas

Objetivos

- 1) Saber definir métodos de cálculo
- 2) Uso de arrays y colecciones de objetos
- 3) Uso de métodos con parámetros un tipo de objeto
- 4) Saber definir métodos de consulta
- 5) Saber definir métodos de cambio de estado
- 6) Saber que todas las clases derivan de la clase Object
- 7) Métodos de conversión de cadenas. Reemplazamiento del método "toString"
- 8) Sobrecarga de métodos



Métodos de cálculo

Son métodos que a partir de los atributos obtienen un resultado numérico o de carácter

Definir el método “área” en la clase Circunferencia

```
public double area()  
{  
    return Math.PI*Math.pow(radio,2);  
}
```

```
public double area()  
{  
    double ar = 0;  
    ar = Math.PI*Math.pow(radio,2);  
    return ar;  
}
```

Uso de dicho método en una aplicación

```
System.out.println(c.area());  
double valor = c.area();  
if(c.area()>20) ...
```

Acceso al atributo

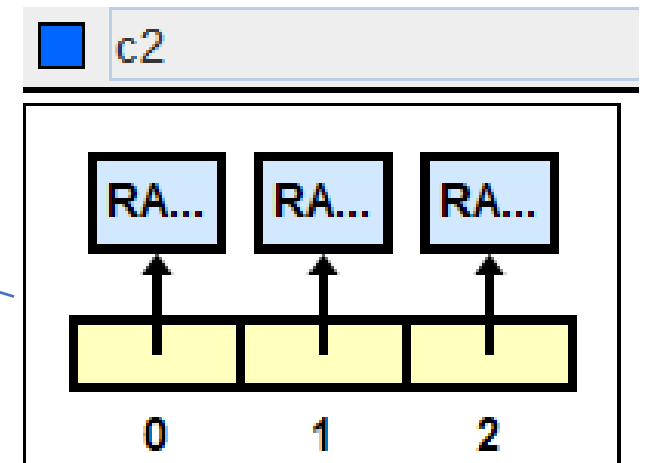
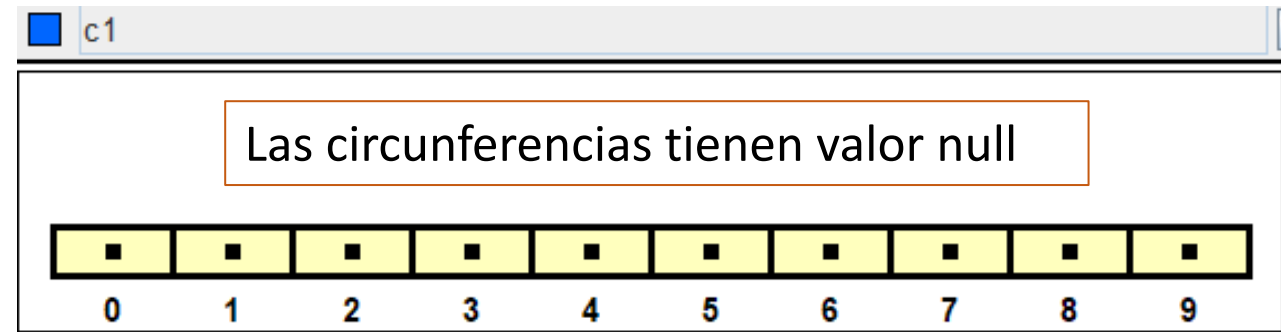
Arrays y colecciones de objetos

Definir un array de circunferencias

```
Circunferencia[] c1 = new Circunferencia[10];  
Circunferencia[] c2 = {  
    new Circunferencia(2),  
    new Circunferencia(3.5),  
    new Circunferencia(4)};
```

Procesar un array de circunferencias

```
for(int i=0; i<c2.length; i++)  
{  
    c2[i].duplicar();  
    System.out.println(c2[i]);  
}
```

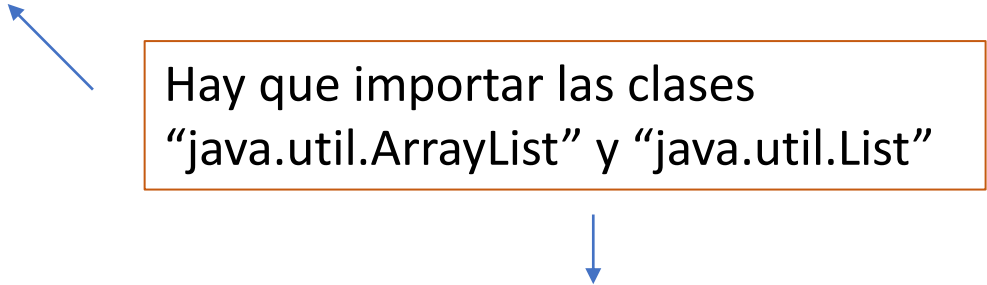


Arrays y colecciones de objetos (II)

Definir un ArrayList de Circunferencias e inicializarlas

```
ArrayList<Circunferencia> c1 = new ArrayList<Circunferencia>();  
c1.add(new Circunferencia(2));  
c1.add(new Circunferencia(3.5));  
c1.add(new Circunferencia(4));
```

Hay que importar las clases
"java.util.ArrayList" y "java.util.List"



```
ArrayList<Circunferencia> c2 = new ArrayList<Circunferencia>(List.of(  
    new Circunferencia(2),  
    new Circunferencia(3.5),  
    new Circunferencia(4)  
));
```

Procesar un ArrayList de circunferencias

```
for(int i=0; i<c2.size(); i++)  
{  
    c1.get(i).duplicar();  
    System.out.println(c1.get(i));  
}
```

Arrays y colecciones de objetos (III)

Ordenar un array de objetos

```
Arrays.sort(c1,0,c1.length, (a,b)-> (a.getRadio()>b.getRadio())?-1:  
(a.getRadio()<b.getRadio())?1:0 );
```

Hay que importar la clase "java.util.Arrays"

Criterio de comparación: Se ordena las circunferencias
descendentemente por el valor del radio

Ordenar un ArrayList de objetos

```
Collections.sort(c1,(a,b)-> (a.getRadio()>b.getRadio())?-1:  
(a.getRadio()<b.getRadio())?-1:0 );
```

Hay que importar la clase "java.util.Collections"



Métodos con parámetros de tipo un objeto

Se pueden definir métodos en una aplicación que permitan pasar como parámetros objetos.

```
public static void main(String[] args)
{
    Circunferencia c1 = new Circunferencia(3);

    mostrarRadio(c1);
}
```

Salida
consola



El radio vale: 3.0

En java los objetos se pasan por referencia. Es decir, que cualquier modificación que hagamos del objeto "c" se ve reflejada en el objeto "c1" que se pasa como argumento

```
static void mostrarRadio(Circunferencia c)
{
    System.out.println("El radio vale: "+ c.getRadio());
}
```

Si en el método "mostrarRadio" se pusiera la instrucción "c.setRadio(10)", entonces "c1" tendría como radio 10 después de ejecutarse "mostrarRadio(c1)"



Métodos de consulta

Son métodos que a partir de los atributos obtienen un resultado de verdadero/falso.

**Comprobar si una
circunferencia es unitaria**

```
public boolean esUnitaria()  
{  
    return radio==1;  
}
```

**Uso de dicho método
en una aplicación**

Acceso al atributo

```
public double area()  
{  
    if (radio==1) return true;  
    return false;  
}
```

```
System.out.println(c.esUnitaria());  
if(c.esUnitaria()) ...  
while(!c.esUnitaria()) ...
```




Métodos de cambio de estado

Son métodos que modifican uno o mas atributos de un objeto. En tales métodos se suele devolver el objeto que invoca el método.

Duplicar el radio de una circunferencia

```
public Circunferencia duplicar()
{
    radio*=2;
    return this;
}
```

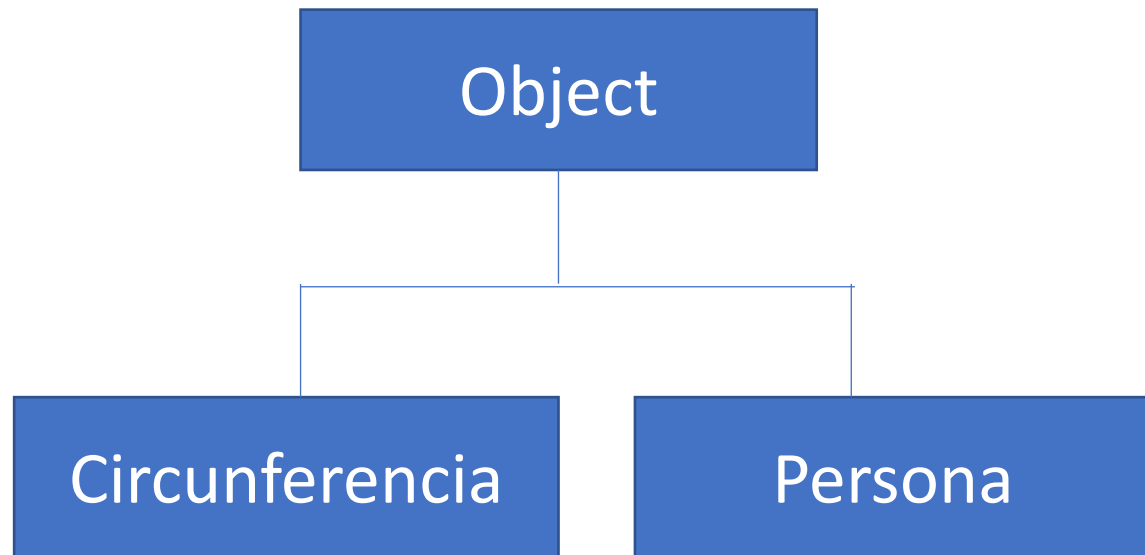
```
public Circunferencia duplicar()
{
    setRadio(radio*2);
    return this;
}
```

**Uso de dicho método
en una aplicación**

c.duplicar();

La clase Object

La clase Object es la clase primaria a partir de la cual se derivan las demás.



Las clases Circunferencia y Persona heredan los métodos definidos en la clase Object

La clase Object no tiene atributos y algunos de sus métodos son:

public		Object()
protected	Object	clone()
public	boolean	equals(Object o)
public	int	hashCode()
public	String	toString()
protected	void	finalize()
public	Class<?>	getClass()

El método toString. Uso en clases derivadas

El método toString convierte los objetos en cadenas.

```
Persona p1 = new Persona("PEDRO",23,1.5,63);  
System.out.println(p1);  
System.out.println(p1.toString());
```

Al mostrar un objeto por consola se invoca el método "toString"

Representación hexadecimal sin signo del código hash del objeto

Salida
consola

clases.Persona@728938a9
clases.Persona@728938a9

Nombre de la clase y su paquete



Métodos de conversión de cadena

Son métodos que devuelven una cadena

Defínase el método `toString` en `Circunferencia`

`@Override`

```
public String toString()
```

```
{
```

```
    return "RADIO_INICIAL: " + RADIO_INICIAL + "\n" +
```

```
    "RADIO: " + radio;
```

```
}
```

El método de cadena "`toString()`" es un método de cadena especial ya que está definido en la clase `Object` de la cual derivan todas las clases. Con `@Override` se indica que el método a definir se está reemplazando.

Acceso al atributo

Uso de dicho método
en una aplicación

```
System.out.println(c.toString());  
System.out.println(c);
```

Son
equivalentes



Sobrecarga de métodos de clase

En una clase se pueden definir métodos con el mismo nombre. Tales métodos se diferencian por el número de parámetros o por el tipo de los mismos.

dibujar(String):String → Sobrecarga del método dibujar() que a la representación en cadena del rectángulo le precede el mensaje que se indica como parámetro del método

```
public String dibujar(String mensaje)
{
    String sal = dibujar();
    sal = mensaje + sal;
    return sal;
}
```

```
public String dibujar(String mensaje)
{
    return mensaje + dibujar();
}
```

Se invoca una sobrecarga del método

Aplicaciones con objetos II

Hágase un programa que cree una circunferencia cuyo radio es aleatorio entre 1 y 2 (este último sin incluir). Se mostrarán todos los atributos, se duplicará el radio y se obtendrá el área.

```
double ale = Math.random()+1;  
double area = 0;  
Circunferencia c = new Circunferencia(ale);  
System.out.println(c.toString());  
c.duplicar();  
area = c.area();  
System.out.println("Area: " +area);
```

```
Circunferencia c = new Circunferencia(Math.random()+1);  
System.out.println(c);  
System.out.println("Area: " +c.duplicar().area());
```

Salida por
consola

```
RADIO_INICIAL: 1.3711608415709344  
RADIO: 1.3711608415709344  
Area: 23.625807869152542
```

Enumerados

Objetivos

- 1) Saber definir un enumerado
- 2) Saber declarar y definir variables de tipo enumerado.
- 3) Lectura, acceso y escritura de variables enumeradas
- 4) Comparación de variables enumeradas
- 5) Saber definir un enumerado en una clase
- 6) Métodos get, set y constructor para atributos enumerados.
- 7) Saber definir enumerados con constructor



Enumerados

Los enumerados son secuencia de valores con nombres.

Defínase en el paquete
“clases.enumerados” el
enumerado “Profesion”
con las siguientes
profesiones “albañil”,
“pintor”, “artista”,
“programador” y “físico”

Acceso público (otros acceso serían
privado e interno)

```
package clases.enumerados;  
public enum Profesion  
{  
    albañil,  
    pintor,  
    artista,  
    programador,  
    fisico  
}
```

Paquete donde se define el enumerado

Nombre del
enumerado

Palabra reservada que indica que se está
definiendo un tipo enumerado

Cada una de las profesiones son
identificadores que se separan
con comas y que se identifican
por un número: “albañil” toma
el valor de 0; “pintor”, 1;
“artista”, 2; “programador”, 3 y
“físico”, 4

Uso de enumerados en clases

Importa enumerados en una clase

```
import clases.enumerados.Profesion;
```

Declarar e inicializar variables de tipo un enumerado

```
Profesion pro1 = Profesion.pintor;
```

Mostrar un enumerado

```
System.out.println(pro1);
```

Salida
consola



pintor

Obtener el nombre del enumerado

```
s = pro1.name(); // s String = "pintor"
```

Obtener el ordinal del enumerado

```
n = pro1.ordinal(); // n int = 1
```

Obtener un array con los valores

```
a = pro1.values(); // a int[] = {0,1,2,3,4}
```

Obtener número de valores

```
n = pro1.values().length; // n int = 5
```

Convertir una cadena a enumerado

```
pro1 = Profesion.valueOf("fisico");
```

Se lanza error si la cadena no se puede convertir a enumerado



Uso de enumerados en clases (II)

Comparar enumerados

```
pro1 == Profesion.artista  
pro1 != Profesion.artista  
pro1.compareTo(Profesion.fisico)<0  
pro1.compareTo(Profesion.fisico)>0
```

Se comparan los valores ordinales

Lectura de enumerados

```
pro1 = Profesion.values()[in.leerInt("Entero entre 0 y 4",v->v>=0&&v<5)];
```

```
pro1 = Profesion.valueOf(in.leerString("Profesion: "));
```

Se lanza error si la cadena no se puede convertir a enumerado



Enumerados II

Se pueden definir enumerados dentro de una clase.

```
public class Persona
```

```
{
```

```
    public enum EstadoCivil
```

```
    {
```

```
        soltero,
```

```
        casado,
```

```
        dicorciado,
```

```
        separado,
```

```
        viudo
```

```
    }
```

```
    ...
```

```
}
```

Atributos de tipo un enumerado (externo o interno)

```
private EstadoCivil estadoCivil;
```

Por defecto se le asigna "soltero" (el 0)

Método get y set de un atributo enumerado

```
public EstadoCivil getEstadoCivil(){
```

```
    return estadoCivil;
```

```
}
```

```
public Persona setEstadoCivil(EstadoCivil estadoCivil){
```

```
    this.estadoCivil = estadoCivil;
```

```
    return this;
```

```
}
```

Acceso al enumerado en otras clases

```
Persona.EstadoCivil ec = Persona.EstadoCivil.soltero;
```

Sólo se tiene que importar la clase Persona

Enumerados con constructor

Enumerados con constructor en UML



```
public enum ColorPelo {  
    CASTAÑO,  
    MORENO,  
    RUBIO,  
    PELIRROJO  
}
```

```
public enum ColorPelo {  
    CASTAÑO(9,4),  
    MORENO(6,5),  
    RUBIO(2,1),  
    PELIRROJO(2,7);  
}
```

```
private int eumelanina = 0;  
private int feomelanina = 0;
```

```
public int getEumelanina()  
{  
    return eumelanina;  
}
```

```
public int getFeomelanina()  
{  
    return feomelanina;  
}
```

```
private ColorPelo(int eumelanina, int  
feomelanina)  
{  
    this.eumelanina = eumelanina;  
    this.feomelanina = feomelanina;  
}
```

No pueden ser públicos

El color del pelo se debe a las concentraciones de dos sustancias

Acceso a los atributos de un enumerado

```
ColorPelo pelo = ColorPelo.CASTAÑO;
```

```
pelo.getEumalanina(); //Devuelve el valor 9
```

```
pelo.getFeomelanina(); //Devuelve el valor 4
```

Clonación , igualdad y comparación de objetos

Objetivos

- 1) Saber definir alias de objetos
- 2) Clonar superficialmente objetos
- 3) Saber distinguir entre mover y copiar objetos
- 4) Saber para que sirve el operador “instanceof”
- 5) Usar el operador de conversión de tipos de objetos
- 6) Saber cuándo dos objetos son alias
- 7) Reemplazar el método “equals”
- 8) Comparar dos objetos con el método “compareTo”

Asignación de objetos

Es posible ejecutar la instrucción de ASIGNACIÓN entre variables de objetos que sean del mismo tipo.

```
Persona p1 = new Persona("PEDRO",23,1.5,63);
```

```
Persona p2 = null;
```

```
p2 = p1;
```

```
p2.setEdad(24);
```

```
System.out.println(p1.getEdad());
```

Salida
consola

24

Se dice que "p1" y "p2" son alias del mismo objeto (apuntan a la misma zona de memoria donde están los datos del objeto)

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

p2 → null

En "p1" se almacena la dirección donde están los datos del objeto

En "p2" no se almacena ninguna dirección

→ p2 = p1;

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

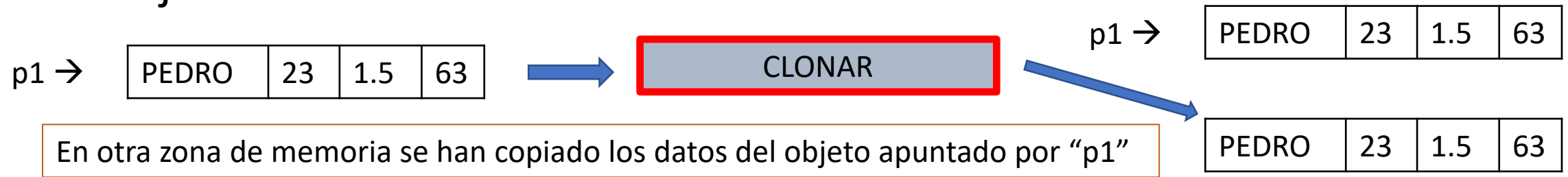
p2 →

"p1" y "p2" almacenan la misma dirección

NO OLVIDES NUNCA QUE UNA VARIABLE DE OBJETO SIEMPRE ALMACENA LA DIRECCIÓN DONDE COMIENZAN LOS DATOS DEL OBJETO O EL VALOR null

Clonar objetos

Clonar objetos consiste en hacer una replica en memoria de los datos del objeto



En java se tiene un método para clonar objetos definido en la clase Object (de la cual se derivan todas las clases).

Para clonar una persona se escribe el siguiente código en la clase Persona

@Override

public Persona clone()

{

return (Persona) super.clone();

}

protected Object clone()

Hay que capturar la excepción "CloneNotSuportedException" e implementar la interfaz Cloneable

Al compilar

unreported exception CloneNotSupportedException; must be caught or declared



Clonar objetos. Clonar en la clase Persona

```
@Override
public Persona clone()
{
    try
    {
        return (Persona) super.clone();
    }
    catch (CloneNotSupportedException e)
    {
        e.printStackTrace();
    }
    return null;
}
```

Para cualquier otra clase, se cambia "Persona" por el nombre de la clase

```
public class Persona implements Cloneable
```

```
{
}
```

```
Persona p1 = new Persona("PEDRO",23,1.5,63);
Persona p2 = null;
p2 = p1.clone();
p2.setEdad(24);
System.out.println(p1.getEdad());
```

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

p2 →

PEDRO	23	1.5	63
-------	----	-----	----

24

Salida
consola

23

ESTA FORMA DE CLONAR SE DICE QUE ES UNA COPIA SUPERFICIAL

Mover o copiar

Mover

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

p2 → null



p2 = p1;

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

p2 →

PEDRO	23	1.5	63
-------	----	-----	----

Copiar

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

p2 → null



p2 = p1.clone();

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

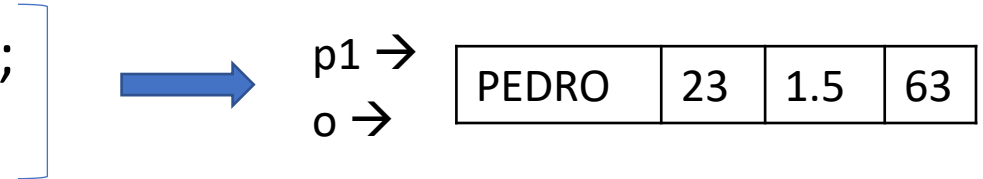
p2 →

PEDRO	23	1.5	63
-------	----	-----	----

Asignar objetos a la clase Object

Se puede mover una variable de tipo una Persona, Circunfencia, etc a una variable de tipo Object.

```
Persona p1 = new Persona("PEDRO",23,1.5,63);  
Object o = p1;
```




PEDRO	23	1.5	63
-------	----	-----	----

El operador instanceof

Con dicho operador se puede averiguar si una variable de tipo Object apunta a un objeto de otra clase

```
if(o instanceof Persona) {  
    Persona p = (Persona) o;  
}
```



La variable “o” aunque apunte a una Persona sólo puede acceder a métodos de la clase Object

El operador de conversión de tipos

Una vez que sepamos que una variable de tipo Object apunta a un objeto, por ejemplo, de tipo Persona, se recupera dicho objeto utilizando el operador de conversión de tipos

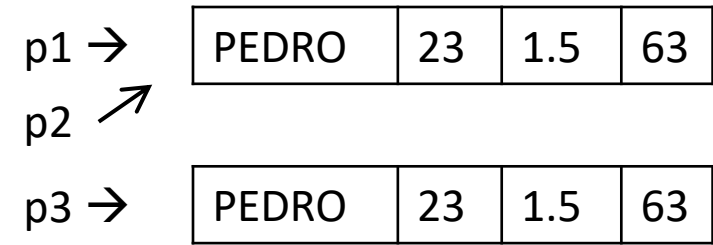
Igualdad de objetos

Dos objetos son iguales si son alias, es decir, apuntan a la misma zona de memoria

```
Persona p1 = new Persona("PEDRO",23,1.5,63);
```

```
Persona p2 = p1;
```

```
Persona p3 = new Persona("PEDRO",23,1.5,63);
```



```
System.out.println(p1==p2);
```

```
System.out.println(p1==p3);
```

Salida
consola



true

false

p1 y p2 son iguales ya que apuntan a la misma zona de memoria

p1 y p3 aunque tienen los mismos datos no son iguales ya que no apuntan a la misma zona de memoria

El método equals. Uso en clase derivadas.

El método “equals” compara dos objetos siguiendo el mismo criterio que el operador “==”

```
Persona p1 = new Persona("PEDRO",23,1.5,63);  
Persona p2 = p1;  
Persona p3 = new Persona("PEDRO",23,1.5,63);
```

El método “equals” sólo devuelve verdadero si ambos objetos apuntan a la misma zona de memoria.

```
System.out.println(p1.equals(p2));  
System.out.println(p1.equals(p3));
```

Salida consola → **true**
false



Sobrescribir el método equals

Se puede sobrescribir el método “equals” de la clase Object en una clase para modificar el criterio estándar de igualdad entre dos objetos.

@Override

```
public boolean equals(Object o)
```

```
{
```

```
    if(!(o instanceof Persona)) return super.equals(o);
```

```
    Persona v = (Persona) o;
```

```
    return NOMBRE.equals(v.NOMBRE) &&
```

```
        edad == v.getEdad() &&
```

```
        altura == v.getAltura() &&
```

```
        peso == v.getPeso();
```

```
}
```

Si el objeto “o” no es de tipo “Persona”, se invoca el método “equals” de la clase Object

En cualquier otra clase se modifica “Persona” por el nombre de la clase

Se define el criterio de comparación entre dos objetos del mismo tipo. En es caso, dos personas son iguales si todos sus atributos (nombre, edad, altura y peso) y estado civil son iguales

Comparar objetos

Para comparar dos objetos de una clase se utiliza el método “compareTo” definido en la interfaz “Comparable”

Una **interfaz** en java es una “clase” en la que se definen firmas de métodos y otros tipos de miembros.

Interfaz Comparable<T>



```
public interface Comparable<T>
{
    int compareTo(T o);
}
```

Una clase puede definir el método “compareTo” si implementa la interfaz Comparable

```
public class Persona implements Cloneable ,
    Comparable<Persona>
```

```
{
```

```
@Override
```

```
public int compareTo(Persona o)
```

```
{
```

```
    return ;
```

```
}
```

```
}
```

Sobreescribir el método definido en la interfaz

Devolverá -1 si “this < o”
Devolverá 1 si “this > o”
Devolverá 0 si “this = o”



Comparar objetos. Clase Persona

Impleméntese la interfaz “Comparable<T>” en la clase Persona, de forma que se comparé dos objetos por su edad

@Override

```
public int compareTo(Persona o)
{
    if(edad < o.getEdad()) return -1;
    if(edad > o.getEdad()) return 1;

    return 0;
}
```

Atributos de tipo un objeto

Objetivos

- 1) Saber definir atributos de tipo un objeto o una colección
- 2) Saber definir métodos get y set moviendo atributos de objetos
- 3) Saber definir método get y set copiando atributos de objetos y colecciones de tipos básicos, envoltorio, cadena, BigDecimal o BigInteger
- 4) Saber definir métodos get y set copiando atributos de colecciones de objetos.
- 5) Saber definir métodos get(i), set(i), size(), add(i) y remove(i) para colecciones moviendo o copiando.

Atributos de tipo un objeto

En la clase Persona se define los siguientes atributos de tipo un objeto.

`private Mascota mascota;`

Objeto

`private int[] puntos;`

Array de tipos básicos

`private Circunferencia[] cir;`

Array de objetos

`private ArrayList<Double> pesos;`

Lista de tipos básicos

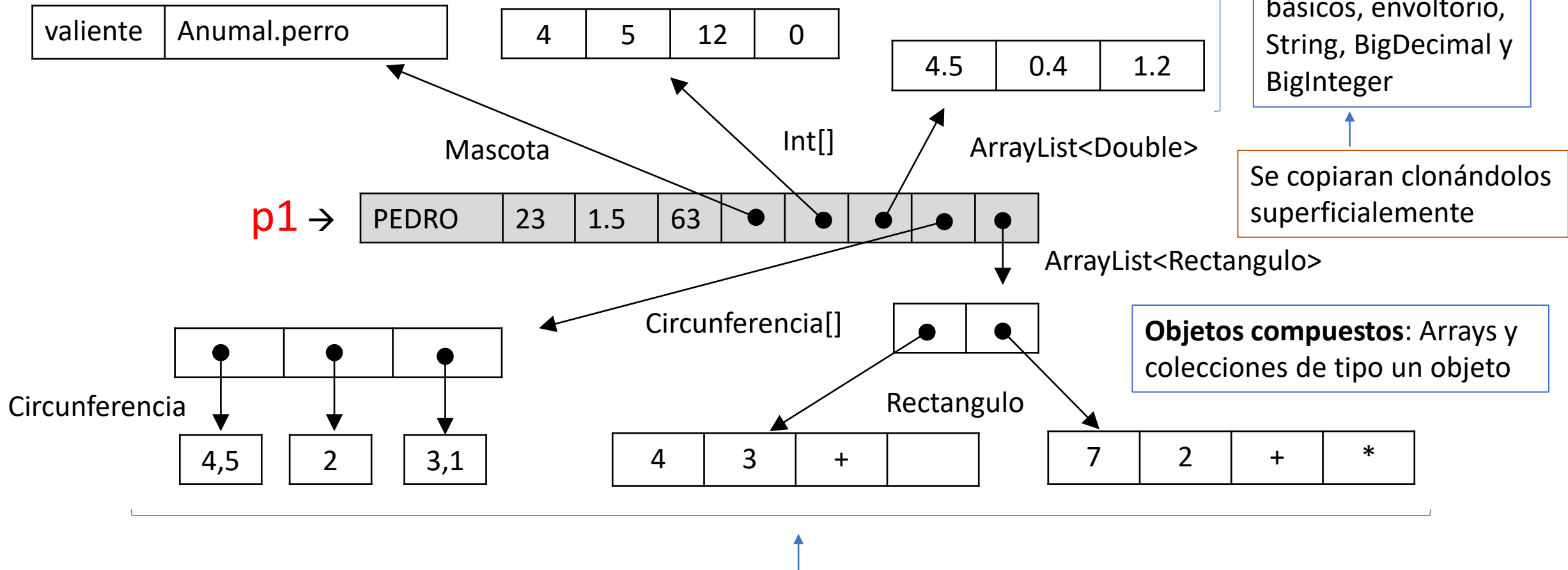
`private ArrayList<Rectangulo> rec;`

Lista de objetos

Por defecto a los atributos de tipo un objeto se les asigna el valor de null

Atributos de tipo objeto

Los atributos de tipo objeto se podrán mover o copiar



Se copiarán clonándolos superficialmente y recorriéndolos y clonando superficialmente cada una de las componentes

Métodos get y set de atributos objeto

Mover

```
private T atributo;
```

```
public T getAtributo()  
{  
    return atributo;  
}
```

Ventaja

- No se consume memoria

Inconveniente

- Se puede modificar directa o indirectamente el objeto sin utilizar el método set

```
public Persona setAtributo(T atributo)  
{  
    //Validación "atributo"  
    this.atributo = atributo;  
    return this;  
}
```

T	atributo	Atributo
Mascota	mascota	Mascota
int[]	puntos	Puntos
ArrayList<Double>	pesos	Pesos
Circunferencia[]	cir	Cir
ArrayList<Rectángulo>	rec	Rec

Atributos de objeto. Mover

Instrucciones que modifican los objetos de la variable “p1” sin utilizar los método set para dichos atributos

```
p1.getMascota().setAnimal(Animal.pájaro);
```

```
p1.getPuntos()[0] = -100;
```

```
p1.getPesos().add(34.4);
```

```
p1.getCir()[1].setRadio(2);
```

```
p1.getRec().get(0).setAncho(10);
```

Se modifican los objetos “p1.mascota”, “p1.puntos”, “p1.pesos”, “p1.cir” y “p1.rec” ya que “p1.getMascota()”, “p1.getPuntos()”, “p1.getPesos()”, “p1.getCir()” y “p1.getRec()” devuelven una referencia a dichos objetos, respectivamente

Modificar indirectamente los objetos definidos en “p1”

```
int[] pun = p1.getPuntos();
```

“pun” es un alias de “p1.puntos”

```
pun[1]++;
```

Se modifica la primera componente de p1.puntos

Métodos get y set de atributos objeto simple

Copiar

```
private T atributo;
```

```
@SuppressWarnings("unchecked")
```

```
public T getAtributo()
```

```
{
```

```
if(atributo==null) return null;
```

```
return (T) atributo.clone();
```

```
}
```

No es necesario si se valida que es distinto de null

```
@SuppressWarnings("unchecked")
```

```
public Persona setAtributo(T atributo)
```

```
{
```

```
//Validación "atributo"
```

```
if(atributo==null) this.atributo = null;
```

```
else this.atributo = (T) atributo.clone();
```

```
return this;
```

```
}
```

(T) y @ Sólo para ArrayList

Inconveniente

- Se consume memoria

Ventaja

- Sólo se puede modificar utilizando su método set

T	atributo	Atributo
Mascota	mascota	Mascota
int[]	puntos	Puntos
ArrayList<Double>	pesos	Pesos

Atributos de objeto simple. Copiar

Mascota m = p1.getMascota();

m.setAnimal(Animal.pajaro);

p1.setMascota(m);

“m” es una copia de “p1.mascota”

Se modifica “m” pero no “p1.mascota”

Se válida “m” y se copia en “p1.mascota”

Puntos p = p1.getPuntos();

p[0]=-100;

p1.setPuntos(p);

“p” es una copia de “p1.puntos”

Se modifica “p” pero no “p1.puntos”

Se válida “p” y se copia en “p1.puntos”

Métodos get y set de atributos objeto compuesto

Copiar

```
private T atributo;
```

No es necesario si se valida que es distinto de null

```
@SuppressWarnings("unchecked")
```

```
public T getAtributo()
```

```
{
```

```
if(atributo==null) return null;
```

```
T v = (T) atributo.clone();
```

Recorrer "v" y clonar las componentes que no sean null

```
return v;
```

```
}  
for(int i=0; i<v.length();i++)  
if(v[i]!=null)  
v[i] = v[i].clone();
```

```
@SuppressWarnings("unchecked")
```

```
public Persona setAtributo(T atributo)
```

```
{
```

```
//Validación "atributo"
```

```
if(atributo==null) this.atributo = null;
```

```
else { T v = (T) atributo.clone();
```

Recorrer "v" y clonar las componentes que no sean null

```
this.atributo = v; }
```

```
return this;
```

```
}
```

(T) y @ Sólo para ArrayList

```
for(int i=0; i<v.size();i++)  
if(v.get(i)!=null)  
v.set(i,v.get(i).clone());
```

T	atributo	Atributo
Circunferencia[]	cir	Cir
ArrayList<Rectangulo>	rec	Rec

Atributos de objetos complejo. Copiar

Circunferencia[] c = p1.getCir();

c[1].setRadio(2);

p1.setCir(c);

“c” es una copia de “p1.cir”

Se modifica “c[1]” pero no “p1.cir[1]”

Se válida “c” y se copia en “p1.cir”

ArrayList<Rectángulo> r = p1.getRec();

r.get(0).setAncho(10);

p1.setRec(r);

“r” es una copia de “p1.rec”

Se modifica “r.get(0)” pero no “p1.rec.get(0)”

Se válida “r” y se copia en “p1.rec”

Métodos de atributos de objetos

- Métodos `getAtributo(i)` y `setAtributo(i, nuevo)` para acceder y actualizar las componentes de un array o colección. En el caso de objetos compuestos se decidirá si mover o copiar el objeto devuelto o almacenado.
- Métodos `sizeAtributo()` que devuelven el número de componentes del array o de la colección.
- Métodos `addAtributo(valor)` y `removeAtributo(valor)` para añadir nuevos valores a la colección (en el caso de objetos compuestos se decidirá si mover o copiar el nuevo objeto a almacenar) y eliminar, respectivamente.

Métodos get(i) y set(i) de atributos objeto simple

Copiar

Para una tabla los índices serían i, j

```
private T atributo;
```

```
public S getAtributo(int i)
{
    return atributo(i);
}
```

atributo[i]

atributo.get(i)

```
@SuppressWarnings("unchecked")
public Persona setAtributo(int i, S valor)
{
    //Validación "valor"
    atributo = (T) atributo.clone();
    Asignar valor a atributo(i);
    return this;
}
atributo[i] = valor;
atributo.set(i,valor);
```

(T) y @ Sólo
para ArrayList

T	S	atributo	Atributo
int[]	int	puntos	Puntos
ArrayList<Double>	Double	pesos	Pesos

Métodos get(i) y set(i) de atributos objeto compuesto

Copiar

Para una tabla los índices serían i, j

@SuppressWarnings("unchecked")

public Persona setAtributo(int i, S valor)

{

//Validación "valor"

atributo = (T) atributo.clone();

if(valor== null) atributo(i)=null;

else { S v = (S) valor.clone(); (*)

Asignar "v" a atributo(i);

return this;

}

(T) (S) y @ Sólo para ArrayList

private T atributo;

public S getAtributo(int i)

{

if(atributo(i)==null) return null;

S v = (S) atributo(i).clone(); (*)

return v;

}

atributo[i]

atributo.get(i)

atributo.set(i,valor);

atributo[i] = valor;

(*) Si "v" de tipo Array o ArrayList, recorrer "v" y clonar las componentes que no sean null

T	S	atributo	Atributo
Circunferencia[]	Circunferencia	cir	Cir
ArrayList<Rectangulo>	Rectangulo	rec	Rec

Método clone para atributos de objeto. **Mover**

```
public Persona clone()
{
    try
    {
        return (Persona) super.clone();
    }
    catch(CloneNotSupportedException e)
    {
        e.printStackTrace();
    }

    return null;
}
```

En el caso de que se haya decidido copiar (en vez de mover) no será necesario modificar el método "clone"

```
Persona p = (Persona) super.clone();
```

```
if(mascota!=null) p.setMascota(mascota.clone());
if(puntos!=null) p.setPuntos(puntos.clone());
if(pesos!=null) p.setPesos(pesos.clone());
if(cir!=null){
    Circunferencia[] c = cir.clone();
    for(int i=0;i<c.length;i++)
        if(c[i]!=null) c[i] = c[i].clone();
    p.setCir(c);
}
if(rec!=null){
    ArrayList<Rectangulo> r = rec.clone();
    for(int i=0;i<r.size();i++)
        if(r.get(i)!=null) r.set(i,r.get(i).clone());
    p.setRec(r);
}
```

Clonar cada atributo de tipo objeto

```
return p;
```



Atributos y métodos estáticos

Objetivos

- 1) Saber qué es un atributo estático o de clase
- 2) Saber definir constantes estáticas
- 3) Uso del constructor estático
- 4) Saber definir métodos estáticos: Métodos de fabricación y validación
- 5) Saber definir atributos estáticos variables
- 6) Saber definir métodos get y set para atributos estáticos
- 7) Uso de métodos y constantes estáticas en métodos de objeto.
- 8) Definir distintos modos para comparar objetos

Atributos estáticos de una clase

Un atributo estático de una clase es un atributo cuyo valor es el mismo para todos los objetos de esa clase.

Cada vez que se crea una persona, se incrementa el contador de “numero de personas creadas”

Atributos estáticos constantes

Atributos estáticos (variables)

numero de ojos

Todas las personas tienen dos ojos



numero de personas creadas

total edad

Cada vez que se modifica la edad de una persona, se actualiza el totalizador del total de las edades de todas las personas

esperanza de vida hombre

esperanza de vida mujer

Cada cierto tiempo se actualiza la esperanza de vida de los habitantes de un país

Declaración e inicialización de un atributo estático constante

```
static public final int NUMERO_DE_OJOS = 2;  
static public final int ESPERANZA_HOMBRE;  
static public final int ESPERANZA_MUJER;
```

ACCESO A MIEMBROS ESTÁTICOS FURA DE LA CLASE

Persona.NUMERO_DE_OJOS

Constructor estático

Las constantes estáticas se inicializan en su declaración o en el constructor estático

static

{

//Se podría acceder a un fichero para obtener tales datos

ESPERANZA_HOMBRE = 85;

ESPERANZA_MUJER = 90;

}

Métodos estáticos

En una clase se definirán métodos para procesar atributos estáticos. Tales métodos tendrán las siguientes características:

- Deben llevar el modificador “static”.
- Pueden tener cualquier modificador de acceso: private, package, protected o public.
- Pueden devolver cualquier tipo de dato o no devolver nada.
- Tendrán un nombre.
- Pueden tener parámetros.
- En su código, se pueden declarar variables, crear objetos y lanzar excepciones.
- En su código se puede hacer referencia a otros métodos estáticos.
- En su código, no se puede utilizar la palabra “this” ni acceder directamente a los métodos de la clase.
- Se puede sobrecargar el método.

```
static public int calculo(int a, double b) {
```

```
}
```


Métodos estáticos de fabricación

Son métodos que devuelven un objeto de la clase

En la clase Rectángulo se define un método estático para sumar dos rectángulos

```
static public Rectangulo sumar(Rectangulo a, Rectangulo b)
{
    if(a==null || b==null) return null;
    int ancho = a.getAncho()+b.getAncho();
    int alto = a.getAlto()+b.getAlto();
    Rectangulo r = new Rectangulo(ancho,alto,'+', ' ');
    return r;
}
```

Cuando hay parámetros de tipo objeto, debe tenerse en cuenta la posibilidad de que tengan el valor de null



Métodos estáticos de validación

En una clase se pueden definir métodos estáticos para validar el valor asignado a un atributo.

Método estático de validación del atributo radio en Circunferencia

```
public static boolean validoRadio(double radio)
{
    if(radius<=0) return false;
    return true;
}
```

El radio es incorrecto si es menor o igual que 0

El radio es correcto si es mayor que 0

Constante estática con la condición que debe cumplir el radio

```
static public final String ERROR_RADIO = "El radio tiene que ser positivo";
```

Métodos estáticos de validación (II)

[Miembros estáticos en UML](#)

Método estático de validación del atributo altura en Persona

```
public static boolean validoAltura(double altura)
```

```
{
```

```
    if(altura<0.3) return false;
```

La altura es incorrecta si es menor que 0.3

```
    if(altura>2.2) return false;
```

La altura es incorrecta si es mayor que 2.2

```
    if((int) (altura*100)/100.0 != altura) return false;
```

```
    return true;
```

La altura es incorrecta si tiene mas de dos decimales

```
}
```

La altura es correcta ya que está entre 0.3 y 2.2, ambos inclusive, y tiene como máximo dos decimales

Constante estática con la condición que debe cumplir la altura

```
static public final String ERROR_ALTURA = "La tiene que estar entre 0,3 y 2,2  
y tener como máximo dos decimales";
```

Métodos set

```
public Circunferencia setRadio(double radio)
{
    if(!validoRadio(radio))
        throw new IllegalArgumentException(ERROR_RADIO);

    this.radio = radio;
    return this;
}
```

**Método set del atributo
altura en Persona**

```
public void setAltura(double altura)
{
    if(!validoAltura(altura))
        throw new IllegalArgumentException(ERROR_ALTURA);
    this.altura = altura;
    return this;
}
```

**Método set del atributo
radio en Circunferencia**

Lectura de atributos válidos

Leer un radio válido y asignárselo a una circunferencia

```
//r int, c Circunferencia
```

```
r = in.leerInt("Radio: ",v->Circunferencia.validoRadio(v),Circunferencia.ERROR_RADIO);  
c.setRadio(r);
```

Leer una altura válida y asignárselo a una persona

```
//p Persona
```

```
p.setAltura(in.leerDouble("Altura: ",v->Persona.validoAltura(v),Persona.ERROR_ALTURA));
```

Atributos estáticos variables

Los atributos estáticos variables se definen del siguiente modo:

`static private int` creados = 0; • Por defecto, a todos los atributos estáticos
`static private int` totalEdad; ← se les asigna el valor nulo del tipo

- Se pueden definir varias variables estáticas del mismo tipo en su declaración → `static private int` creados = 0, total_edad;

- Las varias variables estáticas también se pueden inicializar en el constructor estático →

```
static
{
    totalEdad = 0;
}
```

Métodos get y set estáticos

Para las variables estáticas se pueden definir métodos get y set

```
static public int getCreados()  
{  
    return creados;  
}
```

```
static public int getTotalEdad()  
{  
    return totalEdad;  
}
```

Se definen como privados ya que la clase se encarga de actuizarlos

```
static private void setCreados(int creados)  
{  
    //Validar "creados"  
    Persona.creados = creados;  
}
```

Forma de referenciar el atributo estático si su nombre coincide con una variable o parámetro

```
static private void setTotalEdad(int totalEdad)  
{  
    //Validar "totalEdad"  
    Persona.totalEdad = totalEdad;  
}
```

Uso de métodos y constantes estáticas en métodos de objetos

En cada constructor que no tenga “this()” se incrementa el número de personas creadas en uno

```
public Persona(...)  
{  
    ...  
    setCreados(getCreados()+1);  
}
```



```
public Persona(...)  
{  
    ...  
    creados++;  
}
```


Uso de métodos y constantes estáticas en métodos de objetos (II)



En el método “setEdad” se actualiza el “totalEdad”

```
public Persona setEdad(int edad)
{
    ...
    setTotalEdad(getTotalEdad()-getEdad());
    this.edad = edad;
    setTotalEdad(getTotalEdad()+getEdad());
    return this;
}
```

Descontar la
edad que tenía

```
public Persona
setEdad(int edad)
{
    ...
    totalEdad-= this.edad;
    this.edad = edad;
    totalEdad+= this.edad;
    return this;
}
```

Acumular la nueva
edad que tiene



Es importante que todo método que modifique la edad invoque el método setEdad

Clase Comparator



Se pueden definir constantes estáticas con métodos de comparación entre dos objetos de la clase

```
static public Comparator<Persona> EDAD = (a,b)->
```

```
    (a==null && b==null)? 0:  
    (a==null && b!=null)?-1:  
    (a!=null && b==null)?1:  
    (a.getEdad()==b.getEdad())?0:  
    (a.getEdad()<b.getEdad())?-1:1;
```

La clase Comparator está en el paquete java.util

Uso del operador ternario para comparar dos objetos de tipo Persona por su edad y teniendo en cuenta si uno de ellos o los dos son null

Uso para ordenar objetos Persona almacenados en una colección

```
ArrayList<Persona> per = new ArrayList<Persona>(List.of(  
    new Persona("PEDRO",23,1.5,67),  
    new Persona("JUAN",11,1.6,66),  
    new Persona("PEDRO",18,1.5,67)  
));
```

ArrayList "per" ordenado por EDAD

Collections.sort(per,Persona.EDAD);

ArrayList inicializado con 3 objetos de tipo Persona

Limpiadores

Objetivos

- 1) Saber la función del recolector de basuras
- 2) Conocer las formas de cómo se puede dejar memoria para ser liberada
- 3) Saber qué es un limpiador y cuándo se ejecuta
- 4) Saber definir un limpiador o finalizador asociado a cada objeto
- 5) Conocer la forma de pasar atributos al finalizador
- 6) Clonar el limpiador
- 7) Uso del método “close” para lanzar el finalizador

Recolector de basura

El recolector de basura se encarga de liberar la memoria que no es referenciada o apuntada por ninguna variable

PEDRO	23	1.5	63
-------	----	-----	----



Los datos de una persona ya no pueden referenciarse desde ninguna variable



El recolector de basura comprueba cada cierto tiempo si hay zonas inaccesibles, liberando dicha memoria

- La comprobación y liberación de la memoria no es inmediata
- Se puede llamar al recolector de basura con la siguiente instrucción.

```
System.gc();
```

- Después de llamar al recolector de basura debe detenerse el control para que se de tiempo al recolector a liberar la memoria

```
in.dormir(1000);
```

Formas de dejar memoria para ser liberada

Asignar null

Persona p = new Persona("PEDRO",23,1.5,63); → p →

PEDRO	23	1.5	63
-------	----	-----	----

...

p = null;

Es importante que no hayan alias de p

PEDRO	23	1.5	63
-------	----	-----	----

System.gc();
in.dormir(1000);

Una vez liberada la memoria se ejecuta el método "limpiador" asociado a dicho objeto

PEDRO	23	1.5	63
------------------	---------------	----------------	---------------

run()
No se tiene acceso a los datos

Instrucción try-with-resources

Se pueden declarar mas variables separándolas con ";"

try(Persona p = new Persona("PEDRO",23,1.5,63))

p →

PEDRO	23	1.5	63
-------	----	-----	----

{
La clase Persona debe implementar la interfaz AutoCloseable

PEDRO	23	1.5	63
-------	----	-----	----

}

Es importante que no hayan alias de p

close() → Se tiene acceso a los datos

Al ser inaccesible la variable "p" se ejecuta su método "close"

Limpiador o Finalizador

El limpiador o finalizador es un método que se ejecuta cuando el área de memoria ocupada por un objeto se libera de memoria al no ser apuntada por ninguna variable

```
static class Finalizador implements Runnable
```

```
{  
    public void run()  
    {  
    }  
}
```

- Es una clase estática que se define dentro de cada clase
- Se asocia un objeto Finalizador para cada objeto de la clase, ejecutándose el método “run” cuando se libera la memoria de dicho objeto
- En el método “run” sólo se puede acceder a miembros estáticos de la clase.

Finalizador en la clase Persona

```
import java.lang.ref.Cleaner;
public class Persona ...
{
    /* FINALIZADOR */
    private Finalizador fin = new Finalizador(this);
    static class Finalizador implements Runnable
    {
        public void run()
        {
            En el método "run" sólo se puede acceder a todos los miembros
            estáticos de la clase Persona
        }
    }

    private Cleaner.Cleanable cleanable;
    private Finalizador(Persona p)
    {
        cleanable = Cleaner.create().register(p, this);
    }
}
```

Objeto que permite lanzar el método "run" y romper la asociación entre un objeto Persona y su Finalizador

Método para registrar o asociar a un objeto Persona (p) su objeto finalizador (this) y almacenarlo en "cleanable"

Paso de atributos al finalizador

static class Finalizador **implements** Runnable

```
{  
    private int edad;  
    public void run()  
    {  
        Persona.totalEdad-=edad;  
    }  
}
```

Se declara el atributo “edad” cuyo valor del objeto persona asociado se quiere mantener

Al liberar la memoria se quita del total de las edades, la edad última que tenía el objeto persona asociado

Se asigna al atributo privado “edad” del Finalizador el nuevo valor del atributo “edad” del objeto persona asociado

```
public Persona setEdad(int edad)  
{  
    ...  
    this.edad = edad;  
    ...  
    fin.edad = this.edad;  
    return this;  
}
```

Observa que tanto en Persona como en el Finalizador se puede acceder a los atributos privados de la otra clase

Finalizador en la clase Persona

Hay que modificar el método “clone” del siguiente modo:

Registrar el objeto Finalizador asociado al objeto persona clonado

Copiar valores de los atributos de la persona clonada a su finalizador

@Override

public Persona clone()

{

try

{

Persona p = (Persona) super.clone();

totalEdad+=p.getEdad();

p.fin = new Finalizador(p);

p.fin.edad = p.getEdad();

return p;

}

catch(CloneNotSupportedException e)

{

e.printStackTrace();

}

return null;

}

Modificar los atributos estáticos como el de el total de las edades de todos los objetos personas.

Finalizador y método close. Clase Persona

```
public class Persona implements AutoCloseable
{
    @Override
    public void close()
    {
        fin.cleanable.clean();
    }
}
```

Se ejecuta el método “clean” de la asociación finalizador-persona, de modo que se lanza el método “run” y se rompe dicha asociación

Se accede al objeto “cleanable” del finalizador del objeto persona asociado



Eventos y documentación

Objetivos

- 1) Saber qué es un evento: tipos y controladores
- 2) Saber definir gestores de eventos asociados a un objeto
- 3) Saber definir métodos get de los gestores de eventos
- 4) Saber lanzar los eventos y ejecutar los controladores
- 5) Saber clonar los gestores de eventos
- 6) Saber que es un escuchador en una aplicación
- 7) Saber definir el controlador asociado a un escuchador
- 8) Saber asociar a un objeto un escuchador
- 9) Saber eliminar un escuchador asociado a un objeto
- 10) Documentar los miembros de una clase

Eventos

Un evento es una notificación que envía un objeto indicando que se va a cambiar o ha cambiado el valor de un atributo

```
public Persona setEdad(int edad)
```

```
{
```

```
//Validacion edad
```

```
...
```

```
this.edad = edad;
```

```
...
```

```
return this;
```

```
}
```

EVENTO CAMBIANDOSE Se lanza el evento de que la edad se va a cambiar y se captura la excepción de valor incorrecto para cancelar el cambio si fuera preciso

Modificaciones de atributos estáticos que dependan del valor anterior de la edad (this.edad)

Modificaciones de atributos estáticos y del finalizador que dependan del nuevo valor de la edad (this.edad)

EVENTO CAMBIANDO Se lanza el evento de que la edad se ha cambiado



Se ha cambiado el valor de la edad

Gestor de eventos de cambio y cambiándose

Se asocia un gestor de eventos de cambio y cambiándose para cada objeto de persona

Atributo privado para gestionar los eventos de cambio del valor de cada atributo

```
private PropertyChangeSupport cambios = new PropertyChangeSupport(this);
```



Atributo privado para gestionar los eventos de cambiándose del valor de cada atributo

```
private VetoableChangeSupport cambiandose = new VetoableChangeSupport(this);
```




Las clases VetoableChangeSupport y PropertyChangeSupport están en el paquete **java.beans**

Métodos get de los gestores de eventos de cambio y cambiándose

Se definirán los métodos get de los gestores de eventos para que se puedan añadir métodos (o controladores) que se ejecutarán cuando el atributo vaya a cambiar de valor o ya haya cambiado de valor. En el primer caso, será posible cancelar la asignación del nuevo valor del atributo

```
public PropertyChangeSupport getCambios()  
{  
    return cambios;  
}  
  
public VetoableChangeSupport getCambandose()  
{  
    return cambiandose;  
}
```



Método get del gestor de eventos de cambio

Método get del gestor de eventos de cambiándose

Lanzar eventos de cambios

Se ejecutan los métodos (o controladores) de los eventos de cambio pasándoles como datos el nombre del atributo, el valor anterior del atributo y el nuevo valor del atributo

```
public Persona setEdad(int edad)
```

```
{  
    //Validacion edad  
    ...  
    int edadAnterior = this.edad;
```

```
    ...
```

```
    this.edad = edad;
```

```
    ...
```

```
    return this;
```

```
}
```

Después de validar la edad, se almacena como edad anterior el valor actual de la edad

```
cambios.firePropertyChange("edad",edadAnterior,this.edad);
```

Lanzar eventos de cambiándose

Se ejecutan los métodos (o controladores) de los eventos de cambiándose pasándoles como datos el nombre del atributo, el que se convertirá en el valor anterior y el que será el nuevo valor del atributo

```
public Persona setEdad(int edad)
{
    //Validacion edad
    ...
    this.edad = edad;
    ...
    return this;
}
```

La clase `PropertyVetoException` está en el paquete **java.beans**

```
try
{
    cambiandose.fireVetoableChange("edad",this.edad,edad);
}
catch(PropertyVetoException pve)
{
    return this;
}
```

Si alguno de tales métodos lanza la excepción `PropertyVetoException`, se cancela la asignación de la nueva edad

Asociar los gestores de eventos de cambio y cambiándose del objeto clonado



El método “clone” se modifica del siguiente modo:

```
Persona p = (Persona) super.clone();  
totalEdad+=p.getEdad();  
p.fin = new Finalizador(p);  
p.fin.edad = p.getEdad();
```

```
return p;
```

Los gestores de eventos se asocian al nuevo objeto clonado



```
p.cambios = new PropertyChangeSupport(p);  
p.cambiandose = new VetoableChangeSupport(p);
```

Eventos en una aplicación

La definición de eventos en una aplicación consta de las siguientes partes:

1. Definir el método o controlador que se ejecutará cada vez que cambia o va a cambiar el valor de un atributo de un objeto
2. Definir el escuchador que está pendiente de lo que le sucede a un atributo de un objeto con el fin de ejecutar su controlador asociado
3. Asociar a un objeto un escuchador de un atributo



Controlador cambio

VeotableChangeListener
si cambiandose

atributo

clase

Descripción del
controlador

Parámetros

```
static PropertyChangeListener edadPersonaCambioUno(...)
```

```
{
```

```
return e ->{
```

```
Persona p = (Persona) e.getSource();
```

```
int edadAnterior = (int) (e.getOldValue());
```

```
int edadActual = (int) (e.getNewValue());
```

```
Código del controlador accediendo a la referencia del objeto  
"p", al valor anterior y actual del atributo y a los parámetros
```

```
};
```

```
}
```

Se recupera el objeto que lanzó el evento

Se recupera el valor
anterior del atributo

Se recupera el valor
actual del atributo

Acceso a los parámetros

Controlador cambiándose

VetoableChangeListener
si cambiándose

atributo

clase

Descripción del
controlador

Parámetros

```
static VetoableChangeListener edadPersonaCambandoseUno(...)
```

```
{
```

```
return e ->{
```

```
Persona p = (Persona) e.getSource();
```

```
int edadAnterior = (int) (e.getOldValue());
```

```
int edadActual = (int) (e.getNewValue());
```

Código del controlador accediendo a la referencia del objeto
"p", al valor anterior y actual del atributo y a los parámetros

```
};
```

```
}
```

```
if(edadAnterior>edadActual)
```

```
throw new PropertyVetoException("Cancelar",e);
```

Se recupera el objeto que lanzó el evento

Se recupera el que
será el valor anterior

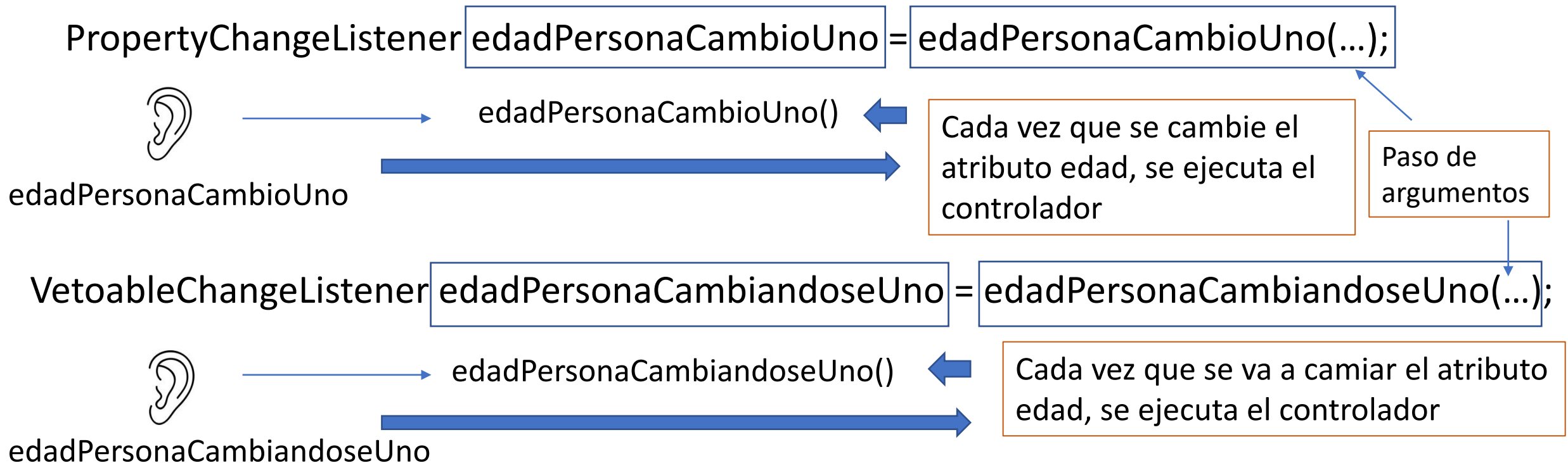
Se recupera el que
será el nuevo valor

Acceso a los parámetros

Se cancela la asignación de la nueva edad si fuera menor que la actual (la que será la anterior)

Escuchadores

En la aplicación principal se define variables de tipo escuchador (cambio y cambiándose) del siguiente modo:



Por convenio, el nombre del escuchador coincide con el del controlador al que se asocia

Asociar objetos a escuchadores

Para asociar un objeto a un escuchador en una aplicación se hace lo siguiente:

objeto	Gestor de eventos	Añadir escuchador	atributo	escuchador
p1	getCambios()	addPropertyChangeListener	"edad"	edadPersonaCambioUno();
p1	getCambiados()	addVoteableChangeListener	"edad"	edadPersonaCambiadosUno();
p2	getCambios()	addPropertyChangeListener	"edad"	edadPersonaCambioUno();

Cada vez que se vaya a cambiar el valor del atributo edad del objeto "p1" se ejecuta el controlador



edadPersonaCambiandoseUno()

p1 →

PEDRO	23	1.5	63
-------	----	-----	----

 → edadPersonaCambiandoseUno

p2 →

JUAN	20	1.6	58
------	----	-----	----

 → edadPersonaCambioUno



edadPersonaCambioUno()

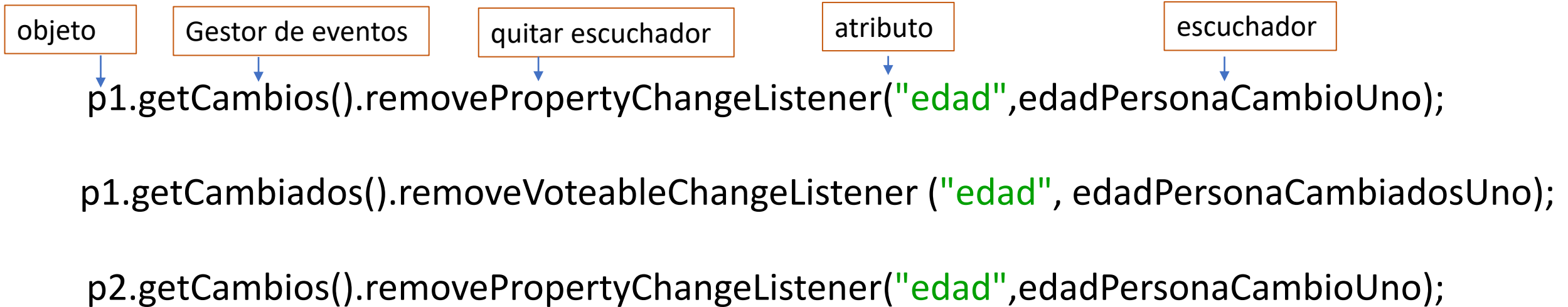
Se puede asociar el mismo escuchador a varios objetos

Cada vez que se cambie el valor del atributo edad del objeto "p1" o "p2" se ejecuta el controlador



Eliminar escuchadores asociados a objetos

Para quitar un escuchador asociado a un objeto en una aplicación se hace lo siguiente:



Documentación

Una clase y sus miembros se pueden documentar y generar un archivo web con dicha documentación. Para la obtención de dicha documentación se utilizará una herramienta de java de nombre “javadoc”. Dicha herramienta exige poner unos comentarios con el formato

```
/**  
 * Parte descriptiva.  
 * Que puede consistir de frases, párrafos o etiquetas  
 *  
 * @etiqueta texto específico de la etiqueta  
 */
```

que deben
aparecer justo
antes de la
declaración de la
clase, el campo o
método que se
pretende
documentar.



Documentar los miembros de una clase

Para documentar una clase se utilizan las siguientes etiquetas

@author	nombre del autor
@version	identificación de la versión y fecha
@see	referencia a otras clases y métodos

Para documentar un atributo no es obligatoria ninguna etiqueta

Para documentar un constructor o un método se utilizan las siguientes etiquetas

@param	nombre del parámetro	descripción de su significado y uso
@return		descripción de lo que se devuelve
@exception	nombre de la excepción	excepciones que pueden lanzarse
@throws	nombre de la excepción	excepciones que pueden lanzarse

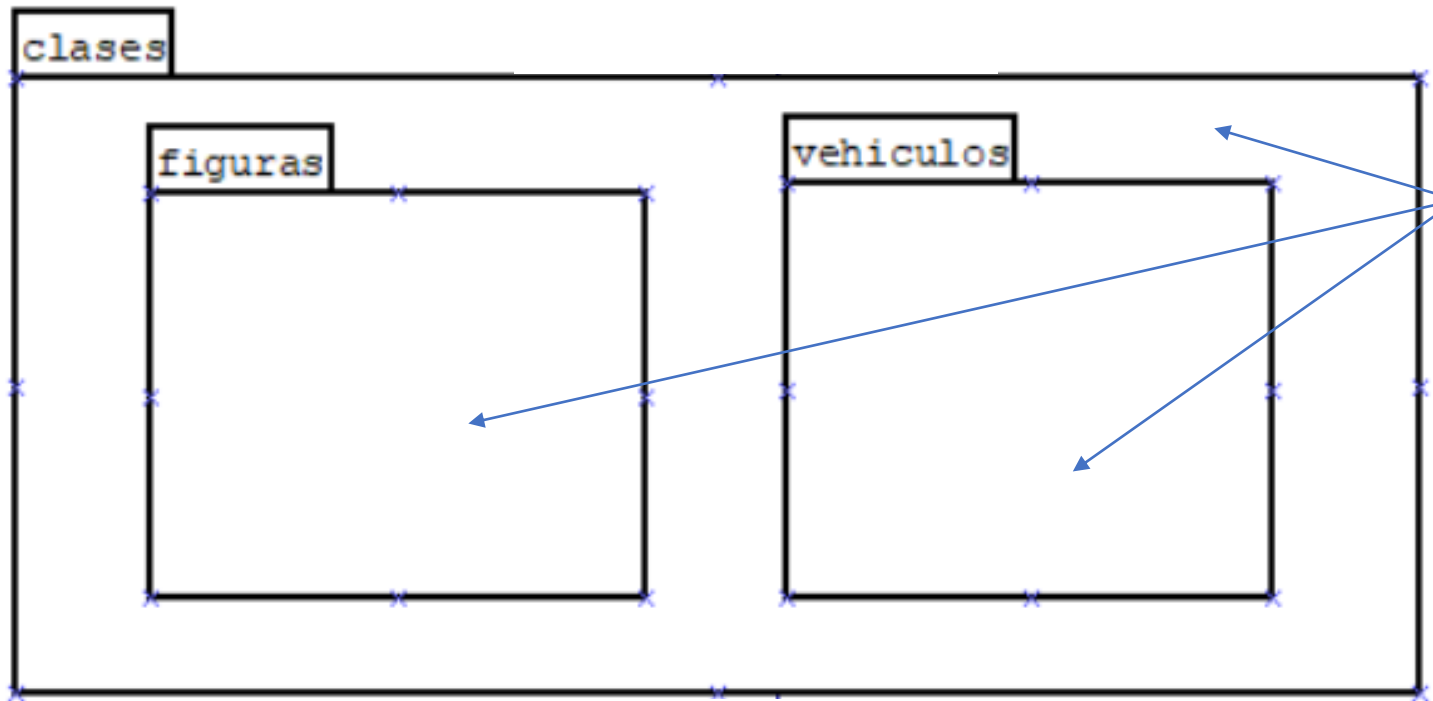
UML

Objetivos

- 1) Paquetes
- 2) Clases
- 3) Atributos de objetos
- 4) Atributos constantes de objetos
- 5) Métodos get, set y constructores
- 6) Métodos de cálculo, consulta, cambio de estado y de conversión de cadenas
- 7) Reemplazamiento de métodos
- 8) Enumerados
- 9) Enumerados internos
- 10) Enumerados con constructor
- 11) Implementación de interfaces
- 12) Miembros estáticos
- 13) Clases estáticas internas
- 14) Asociaciones

Paquetes en UML

En un diagrama UML los paquetes se indicarían del siguiente modo



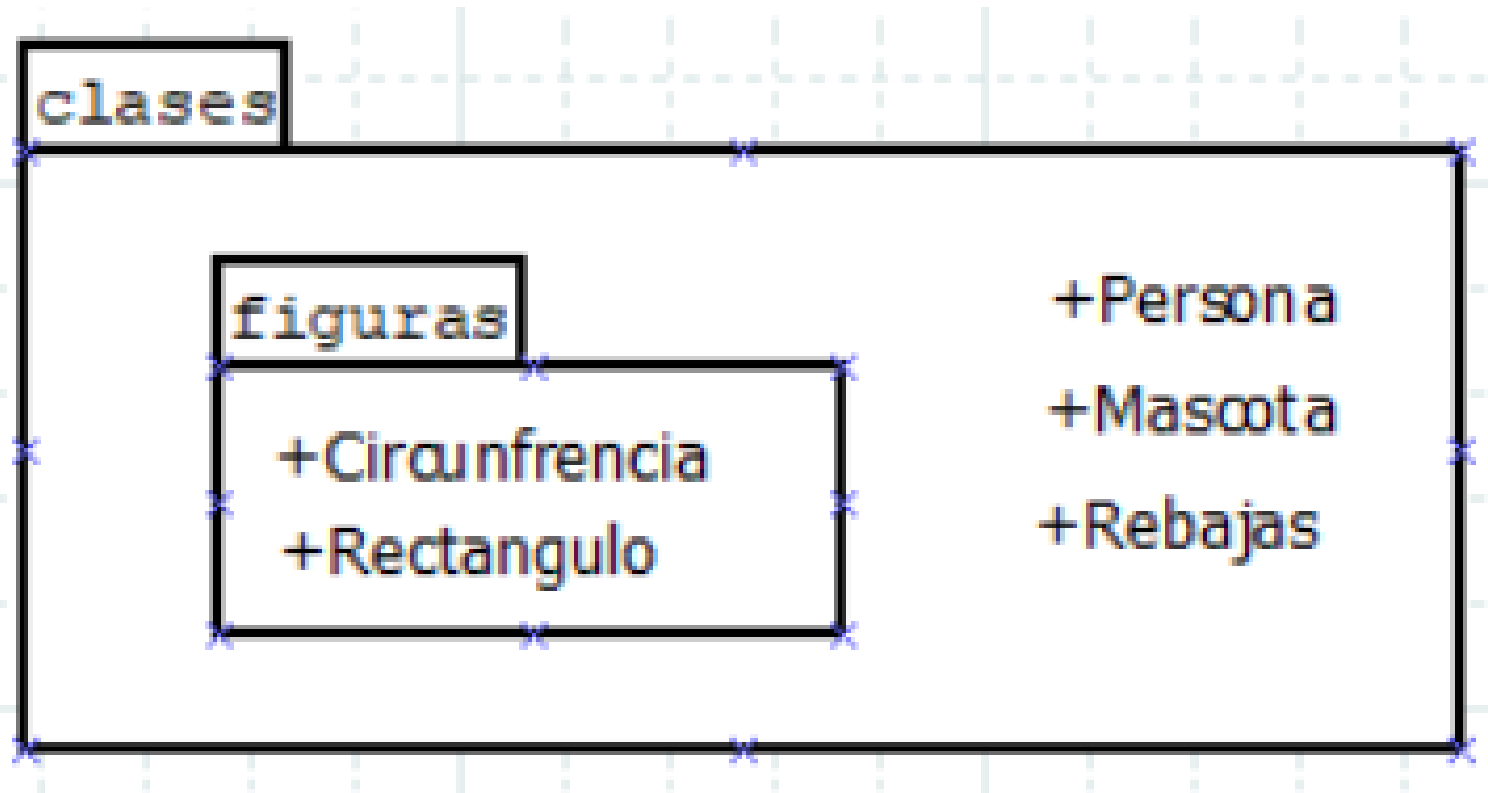
Dentro de cada paquete se pondrán los siguientes elementos con su visibilidad:

- Clases
- Enumerados
- Interfaces



Ejemplo de paquetes UML

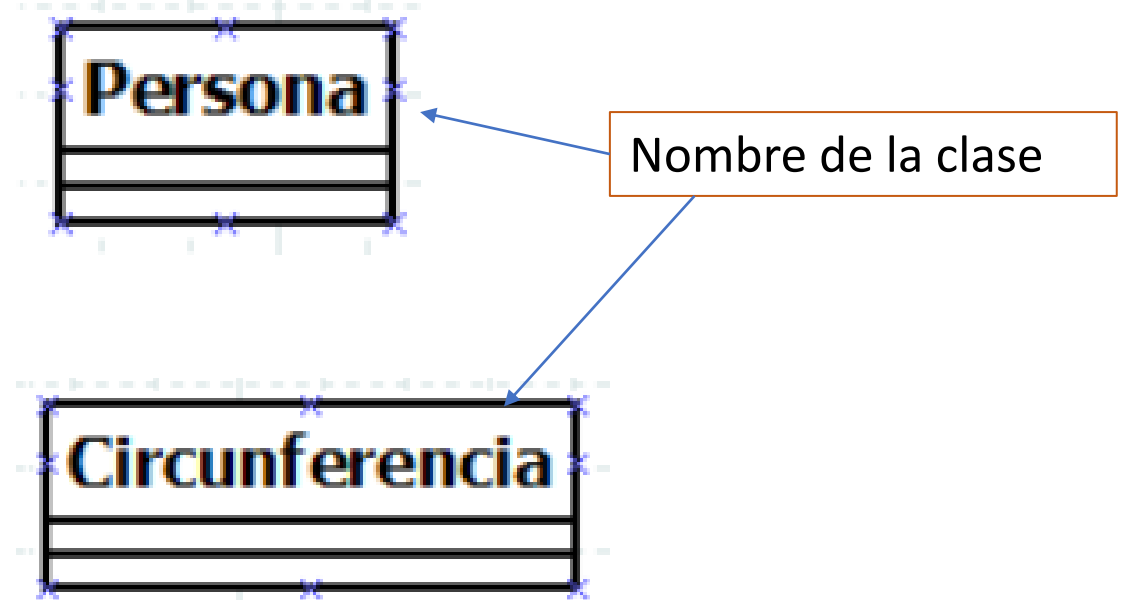
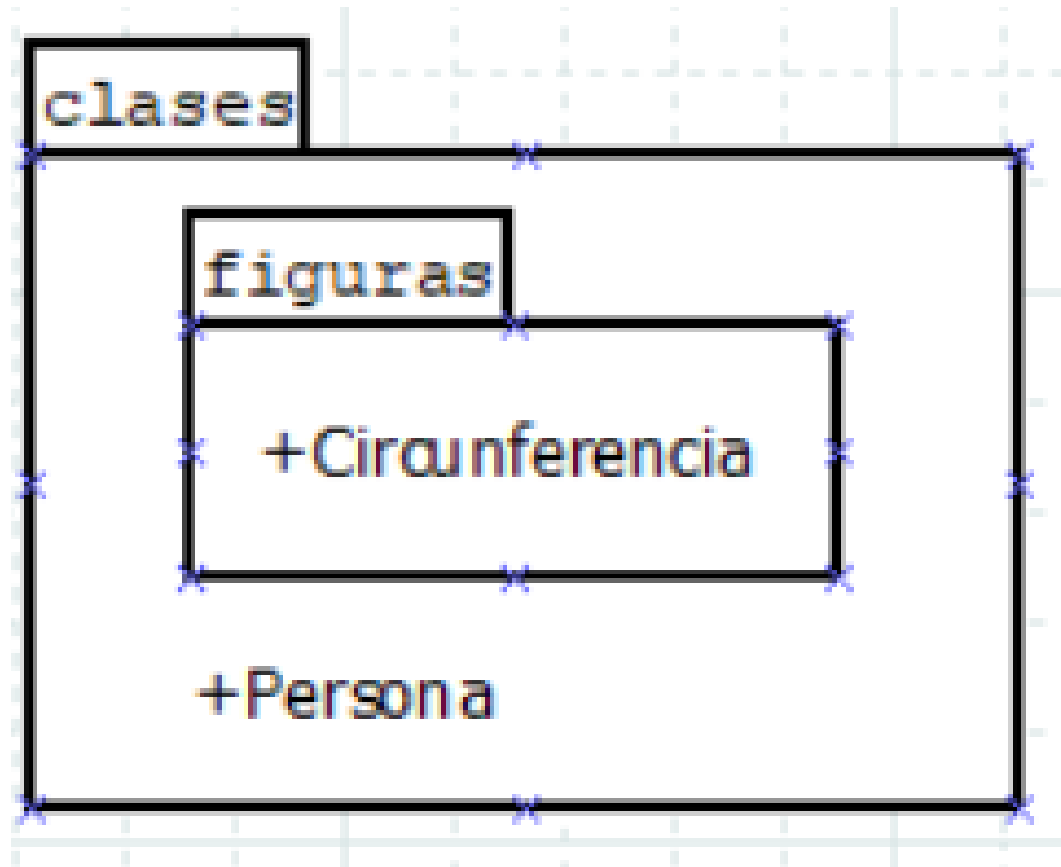
[Paquetes anidados](#)



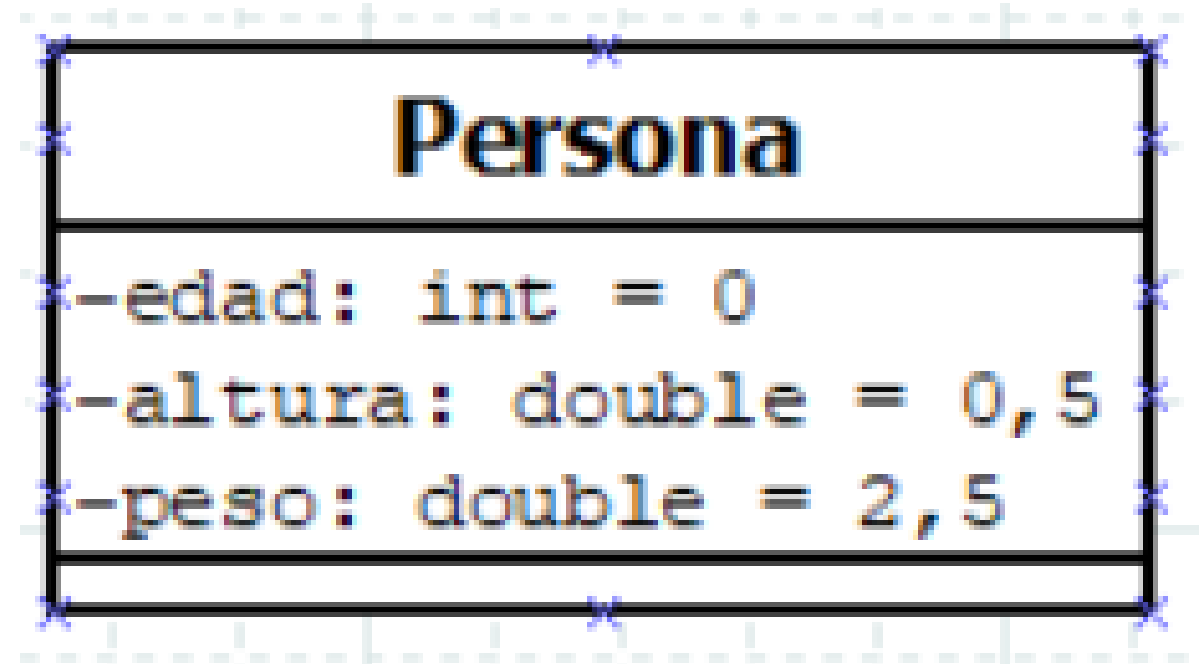
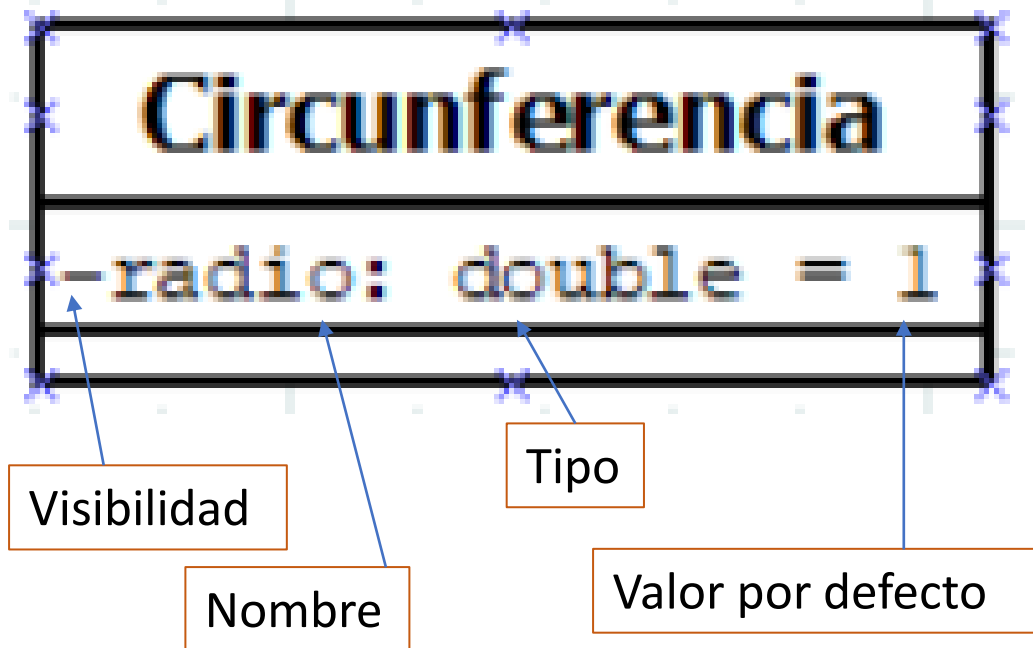
La visibilidad de una clase puede ser pública (+) o interna/paquete (~)

Definición de clases en UML

[Definición de clases](#)

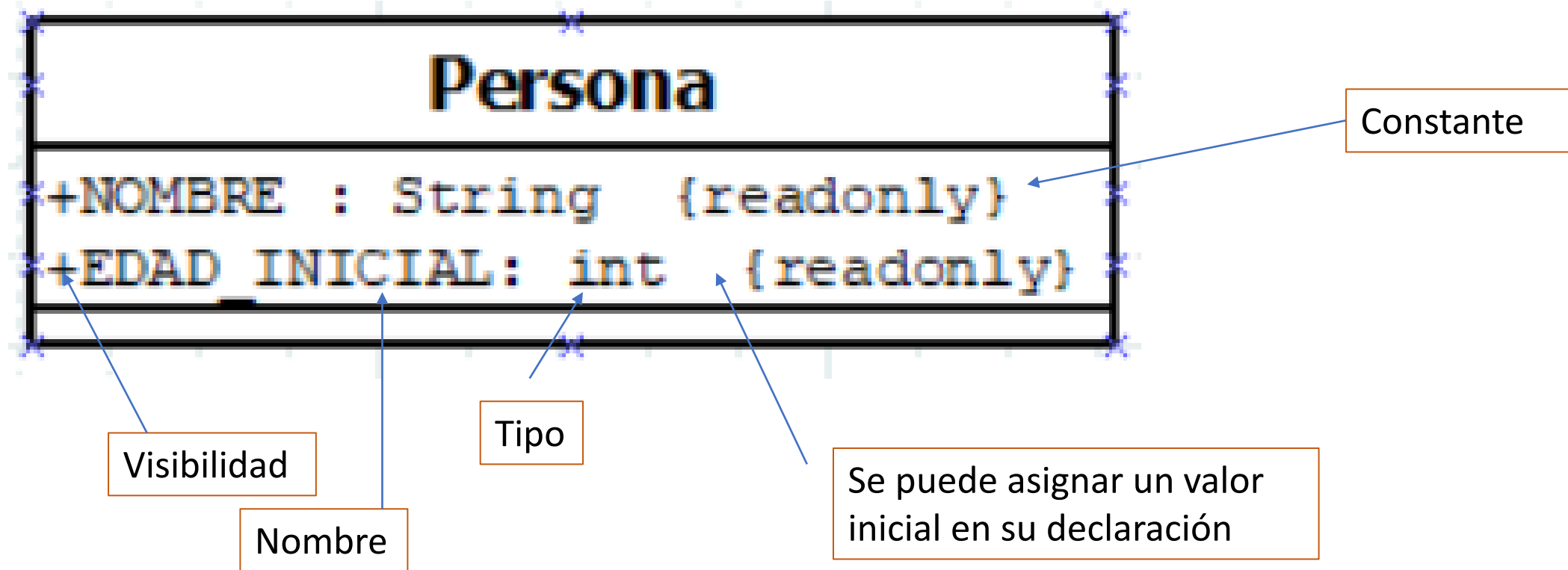


Atributos de objeto en UML

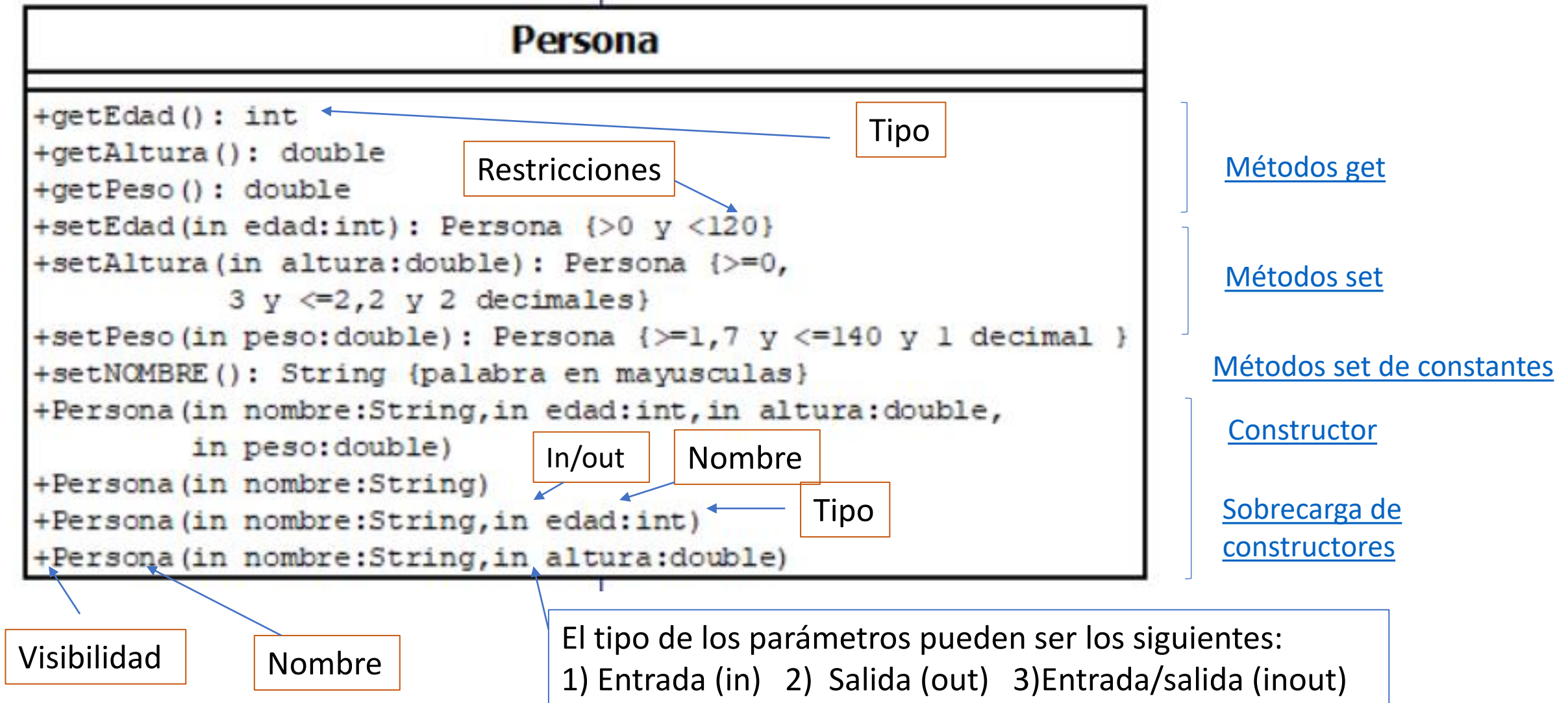


La visibilidad de un atributo o método puede ser pública (+), protegida (#), interna/paquete (~) y privada (-)

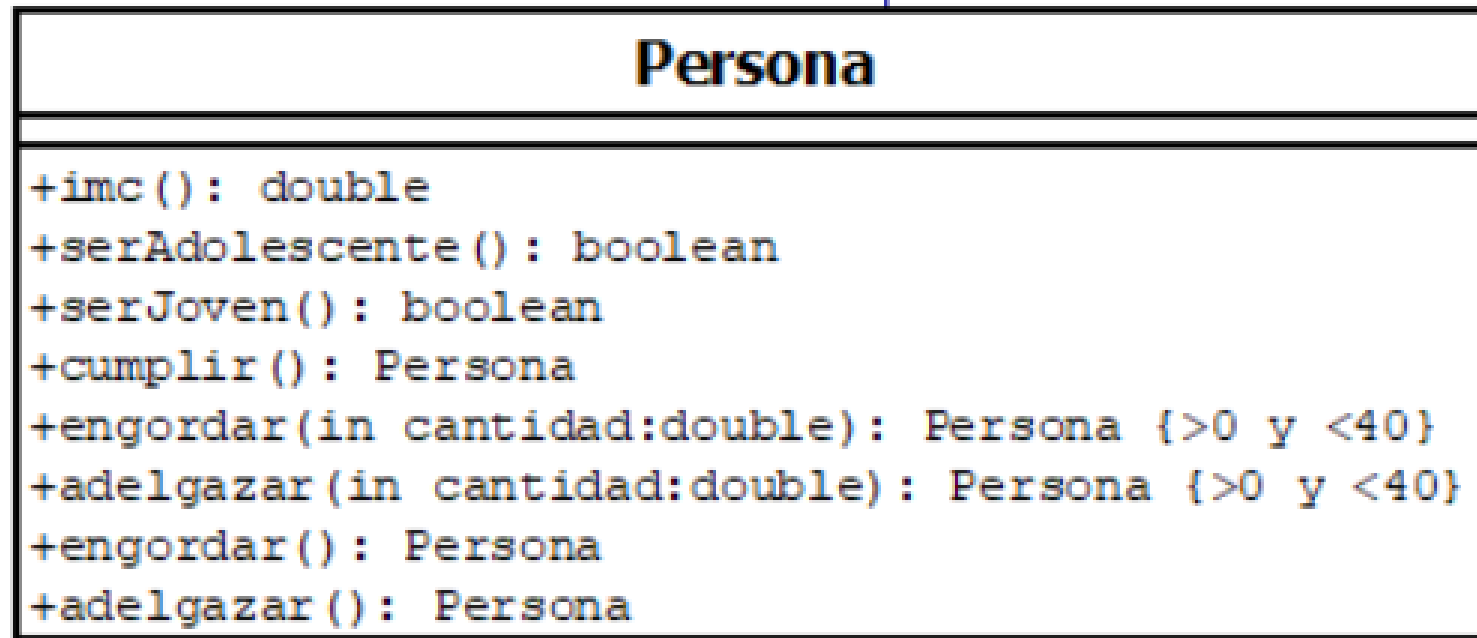
Atributos constantes de objetos en UML



Métodos get, set y constructores en UML



Métodos de cálculo, consulta, cambio de estado y conversión de cadenas en UML

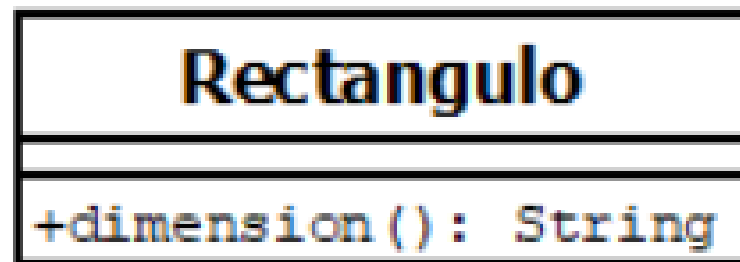


[Métodos de cálculo](#)

[Métodos de consulta](#)

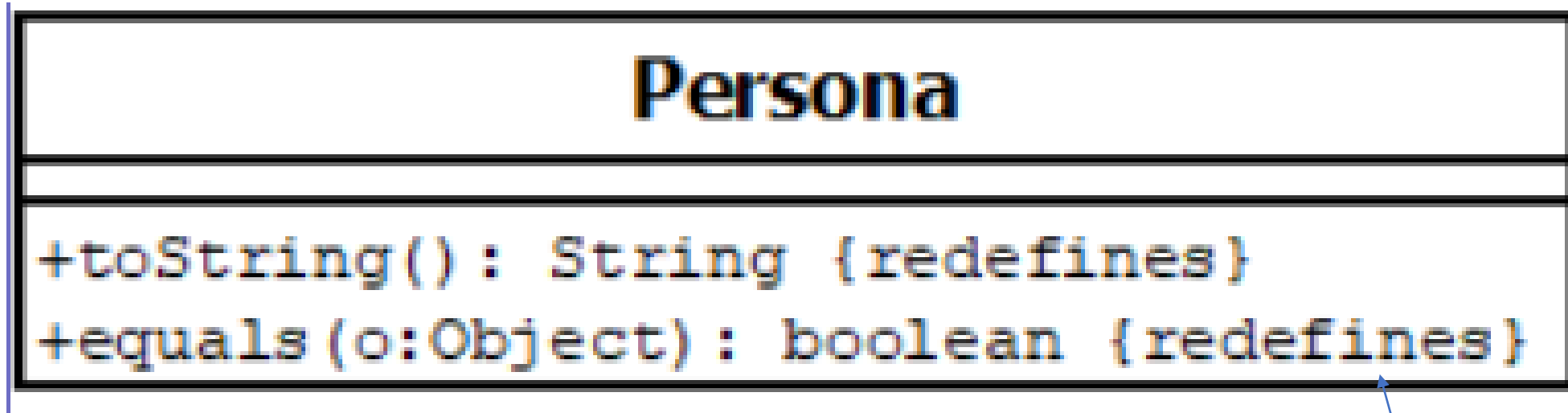
[Métodos de cambio de estado](#)

[Sobrecarga de métodos](#)



[Métodos de conversión a cadenas](#)

Reemplazamiento de métodos en UML

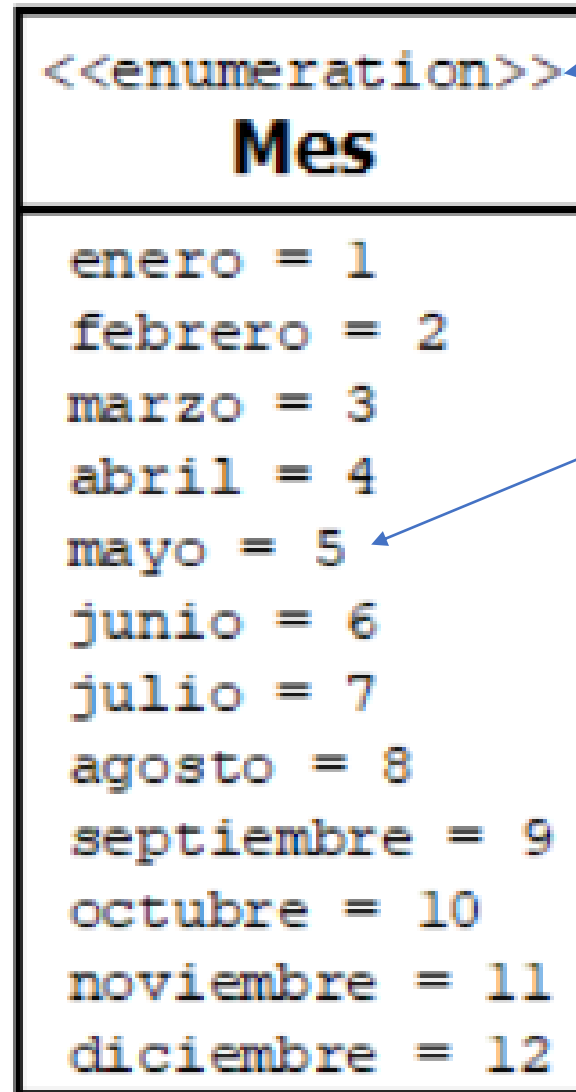
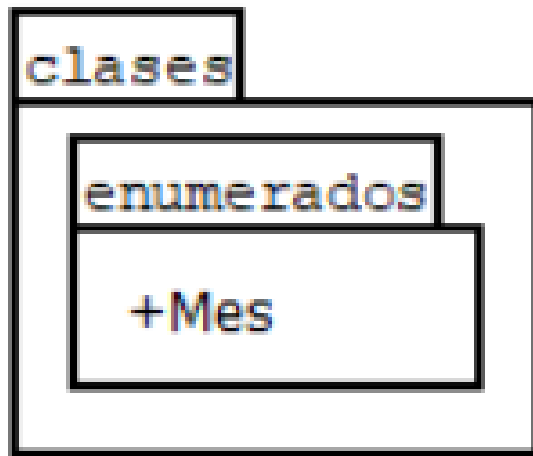


Sobrescritura de métodos

Métodos de conversión a cadenas

Sobrescribir el método equals

Enumerados en UML

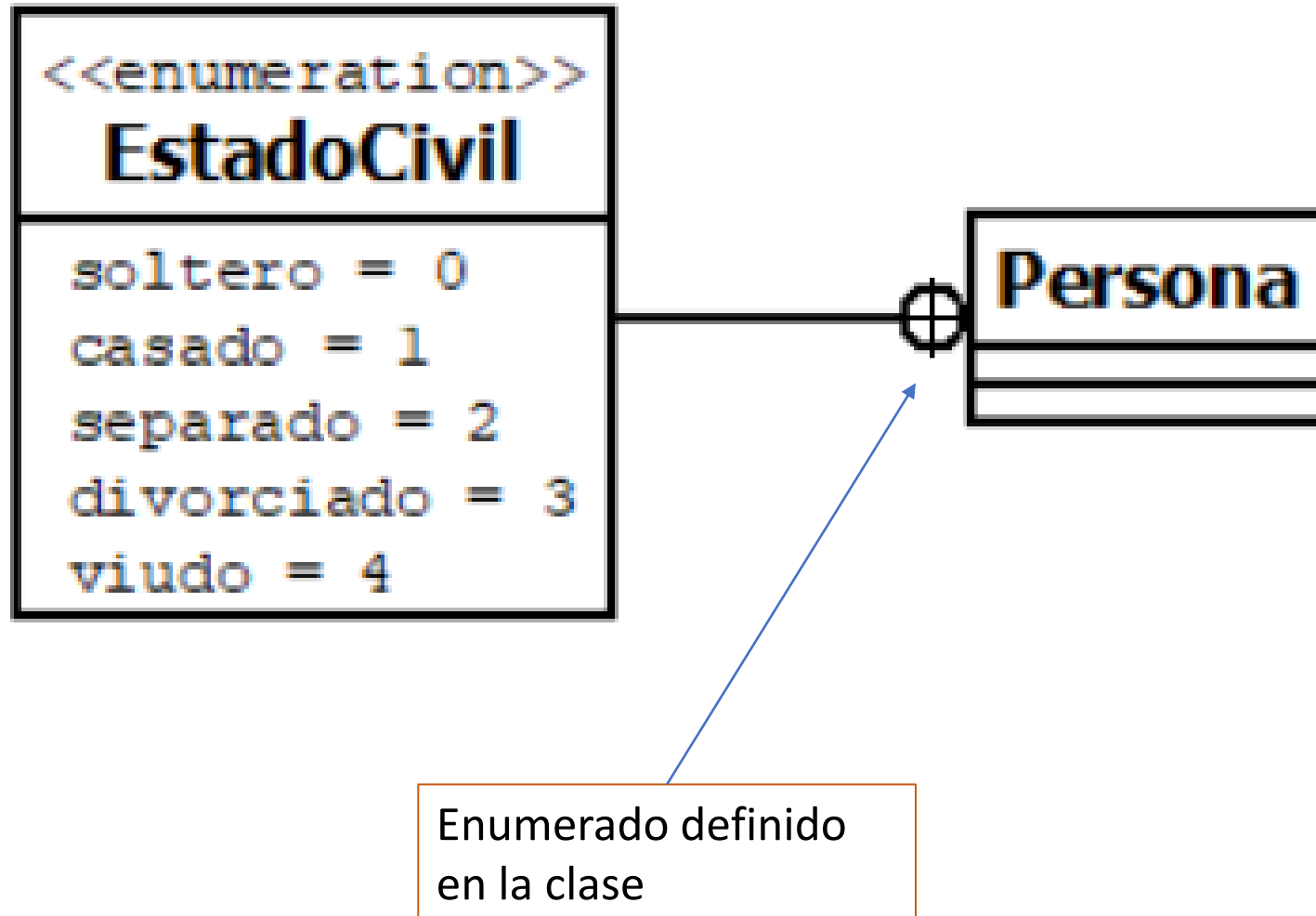


Estereotipo

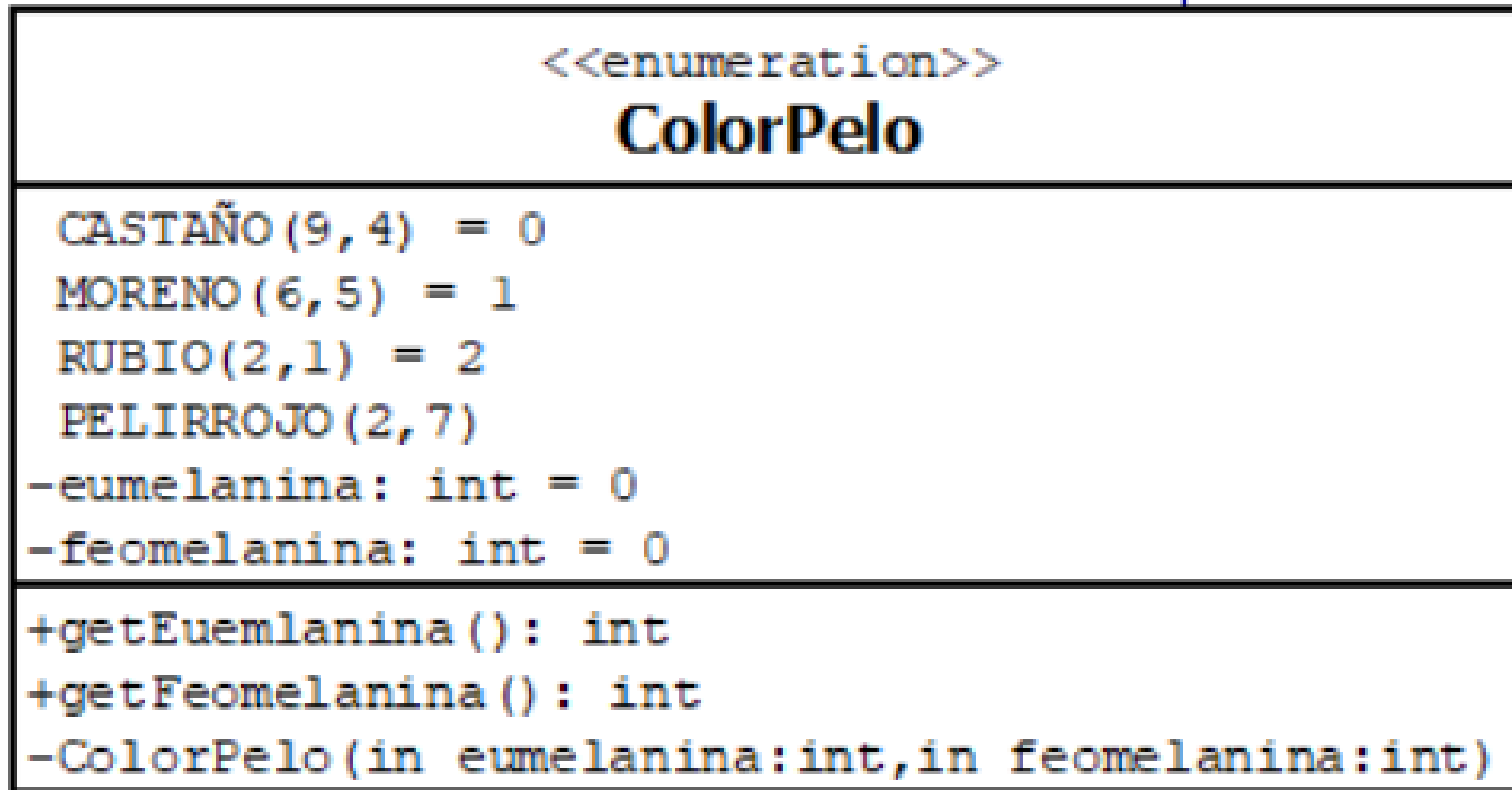
Valor del enumerado

Enumerados internos en UML

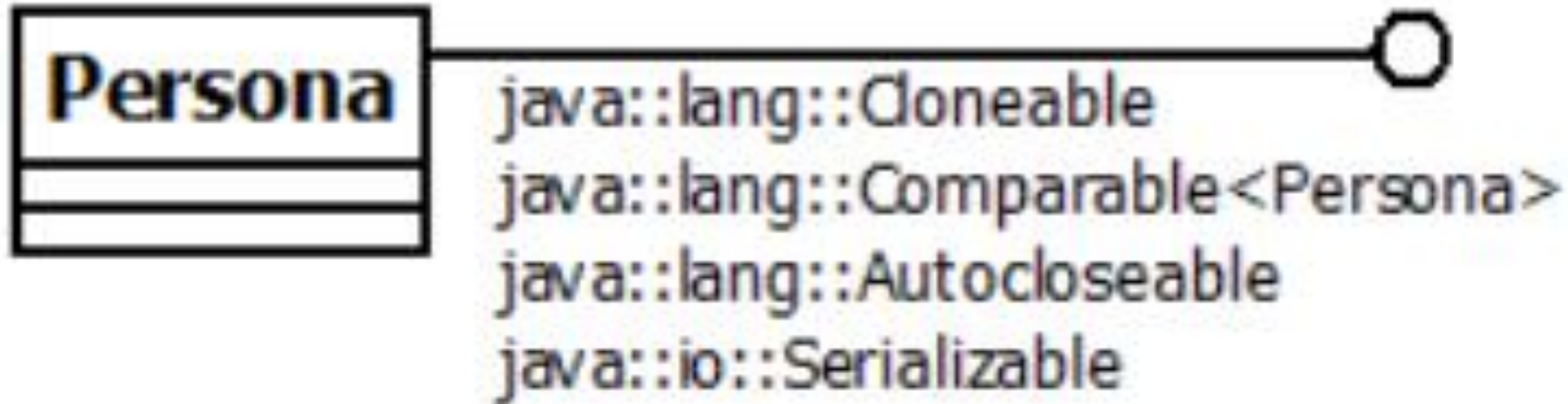
Enumerados internos



Enumerados con constructor en UML



Implementación de interfaces en UML

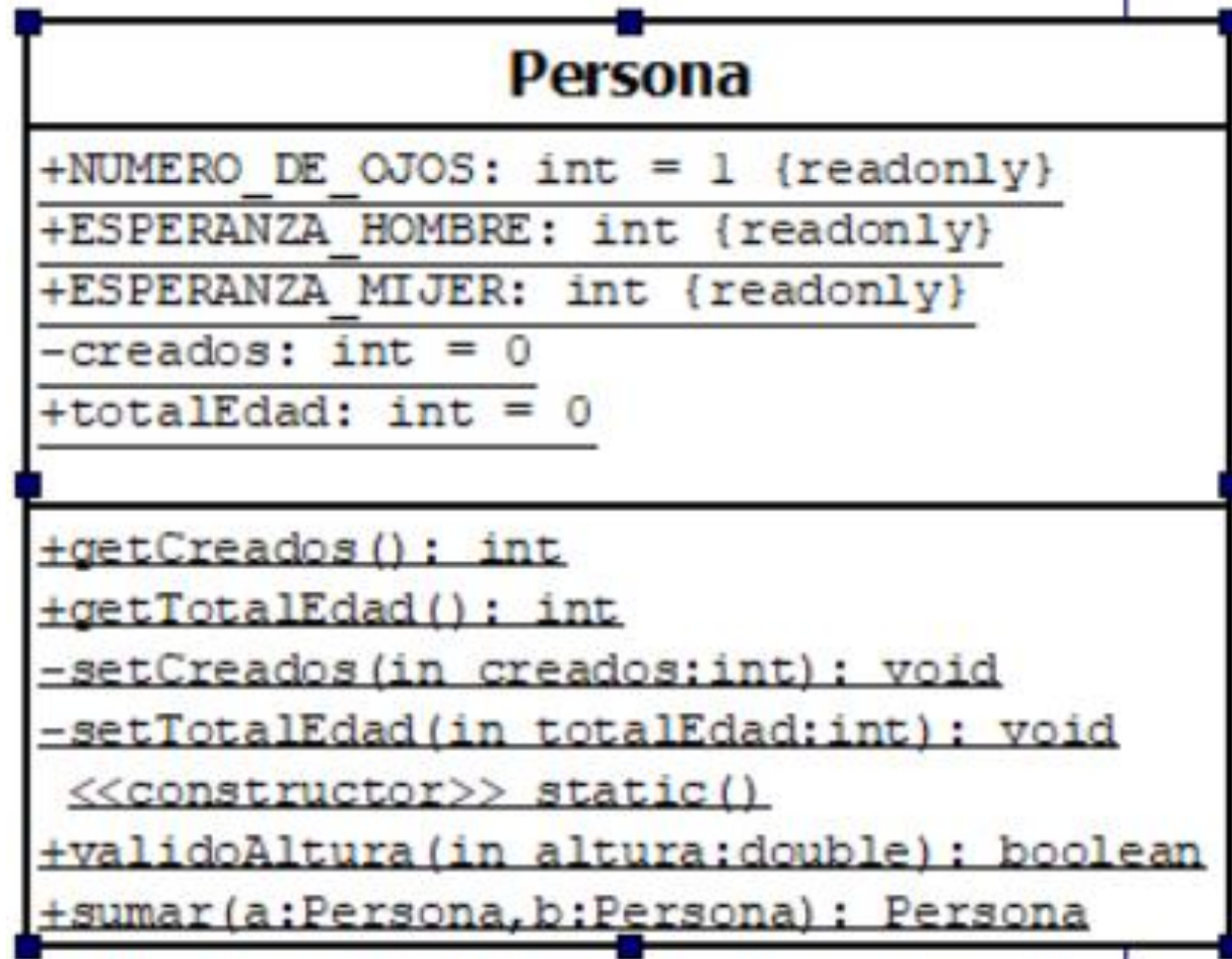


[Clonar objetos](#)

[Comparar objetos](#)

Miembros estáticos en UML

Todos los miembros estáticos se subrayan



[Atributos estáticos constantes](#)

[Atributos estáticos variables](#)

[Métodos get y set estáticos](#)

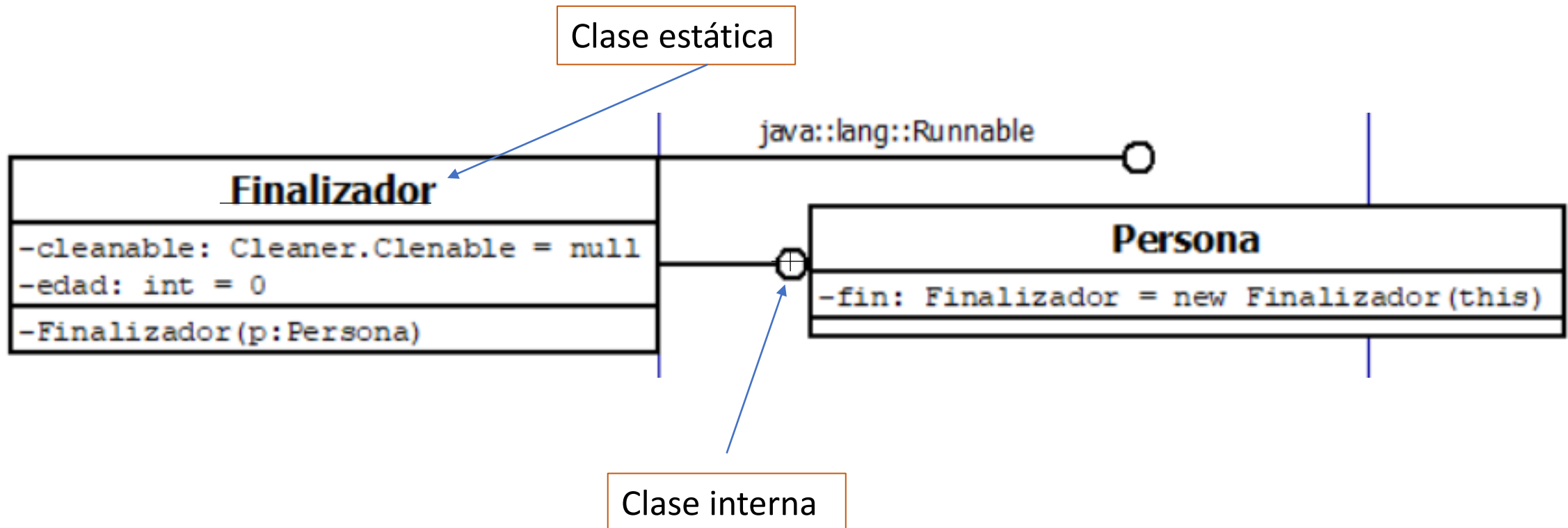
[Constructor estático](#)

[Métodos estáticos de validación](#)

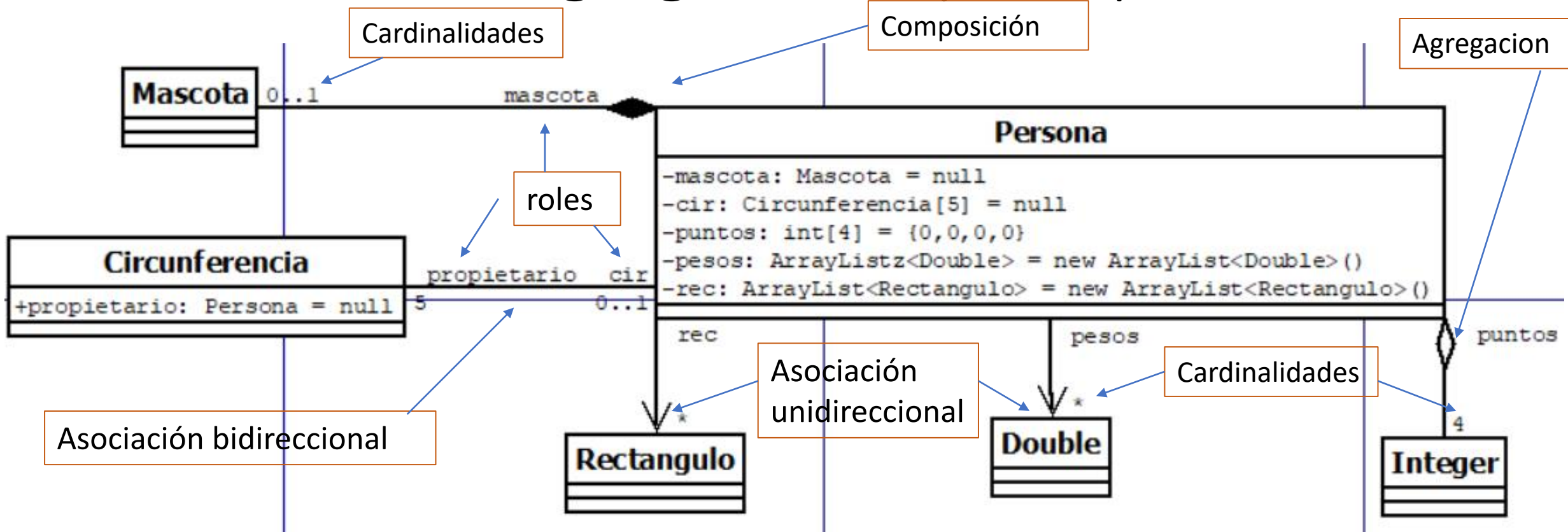
[Métodos estáticos de fabricación](#)

Clases estáticas internas en UML

[Finalizador](#)



Asociaciones: Agregaciones y composiciones



Cardinalidades: N .. M, N..*(muchos)
0..1, 0..* (o *),
1..1 (o 1), 1..*,
4..10, 5..5 (o 5)

↑ mínimo ↑ máximo

Composición: La destrucción del objeto persona supone la destrucción del objeto mascota (card max. de 1)

Agregación: La destrucción del objeto persona no supone la destrucción del objeto puntos (card max. puede ser N)