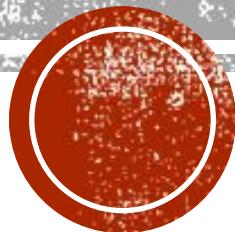


PROGRAMACIÓN

UNIDAD 6: Programación orientada a objetos. Clases



Curso / Ciclo Formativo: 1º Desarrollo de aplicaciones web

Apuntes Profesoras: Mari Cruz Sanz Zamarrón y María Navarro Elbal

Creación de paquetes

- Un **paquete** es un conjunto de clases relacionadas entre sí.
- Puede contener a su vez **subpaquetes**
- Java mantiene su biblioteca de clases en una estructura **jeráquica**
- Cuando nos referimos a una clase de un paquete (no importada) hay que referirse a la misma especificando el paquete y subpaquete al que pertenece (Ej: `java.io.File`)
- Permiten reducir los conflictos con los nombres puesto que dos clases que se llaman igual, si pertenecen a paquetes distintos, no debería dar problemas.
- Permiten proteger ciertas clases no públicas al acceso desde fuera del mismo.
- Si se crea un paquete, **package** se escribe al principio y si no da error.
- Uso de una clase en otro paquete:

```
import <nombre_clase>
```

Concepto de clase

- Permite **abstraer** el problema, ocultando datos y la implementación
- Acceso a los datos a través de los métodos **set** y **get**
- **Abstracción:** concepto clave en A&D cuya finalidad es crear un conjunto de clases que resuelvan el problema.
- Alto grado de **cohesión** (independencia)
- Niveles de **acceso** a los miembros de una clase:
 - **Public:** público
 - **Protected:** protegido
 - **Private:** privado.
 - **No especificado:** acceso en su paquete

Concepto de clase

- **Acceso público:** miembro que puede ser accedido desde **cualquier otra clase o subclase** que necesite utilizarlo. Una interfaz de una clase estará compuesta por todos los miembros públicos de la misma.
- **Acceso privado:** miembro que puede ser accedido **solamente desde los métodos internos de su propia clase**. Otro acceso será denegado
- **Acceso protegido:** acceso a miembros de este tipo **igual que el privado**. Para **subclases o clases del mismo paquete** a la que pertenece la clase, se considerarán estos miembros como públicos.
- **Acceso no especificado:** miembros accesibles por cualquier clase perteneciente al mismo paquete.

Acceso a los elementos de una clase según sus modificadores

	Clase	Subclase	Paquete	Todos
Private	X			
Protected	X	X	X	
Public	X	X	X	X

Ejemplo rectángulo

- Crear una **clase rectángulo** que tenga un nombre y dimensiones en centímetros.
 - Implementar los métodos básicos **set** y **get**.
 - Implementar un método que devuelva un objeto rectángulo en el que se ha **incrementado en un centímetro la altura**.
 - Implementar un método que devuelva un objeto rectángulo en el que se ha **incrementado en un centímetro el ancho**.

Conceptos de clase

- Referencia al objeto **this**
 - Objeto que se está ejecutando el método.
 - Resuelve **ambigüedades**.
- Clase **Object**
 - Raíz jerárquica de Java
 - Toda clase implementada en **Java es subclase de Object**: hereda todos los métodos de Object.

Método	Descripción
<code>clone()</code>	Permite "clonar" un objeto.
<code>equals()</code>	Permite comparar un objeto con otro.
<code>toString()</code>	Devuelve el nombre de la clase.
<code>finalize()</code>	Método invocado por el recolector de basura (garbage collector) para borrar definitivamente el objeto.

Conceptos de clase

- **Método clone()**
 - Equivale a utilizar un constructor de copia
 - Puede ser necesario implementar un método clone, que sobrescribirá el método clone de su superclase, actuando de una forma más específica.
 - **Ejemplo:** Implementar el método clone en la clase Rectángulo y observar sus efectos.

```
public class Rectangulo implements Cloneable{
    private int ancho;
    private int alto;
    private String nombre;

    public Rectangulo(int an, int al){...4 lines}

    public void setNombre(String nombre){...3 lines}

    public int getAncho(){...3 lines}

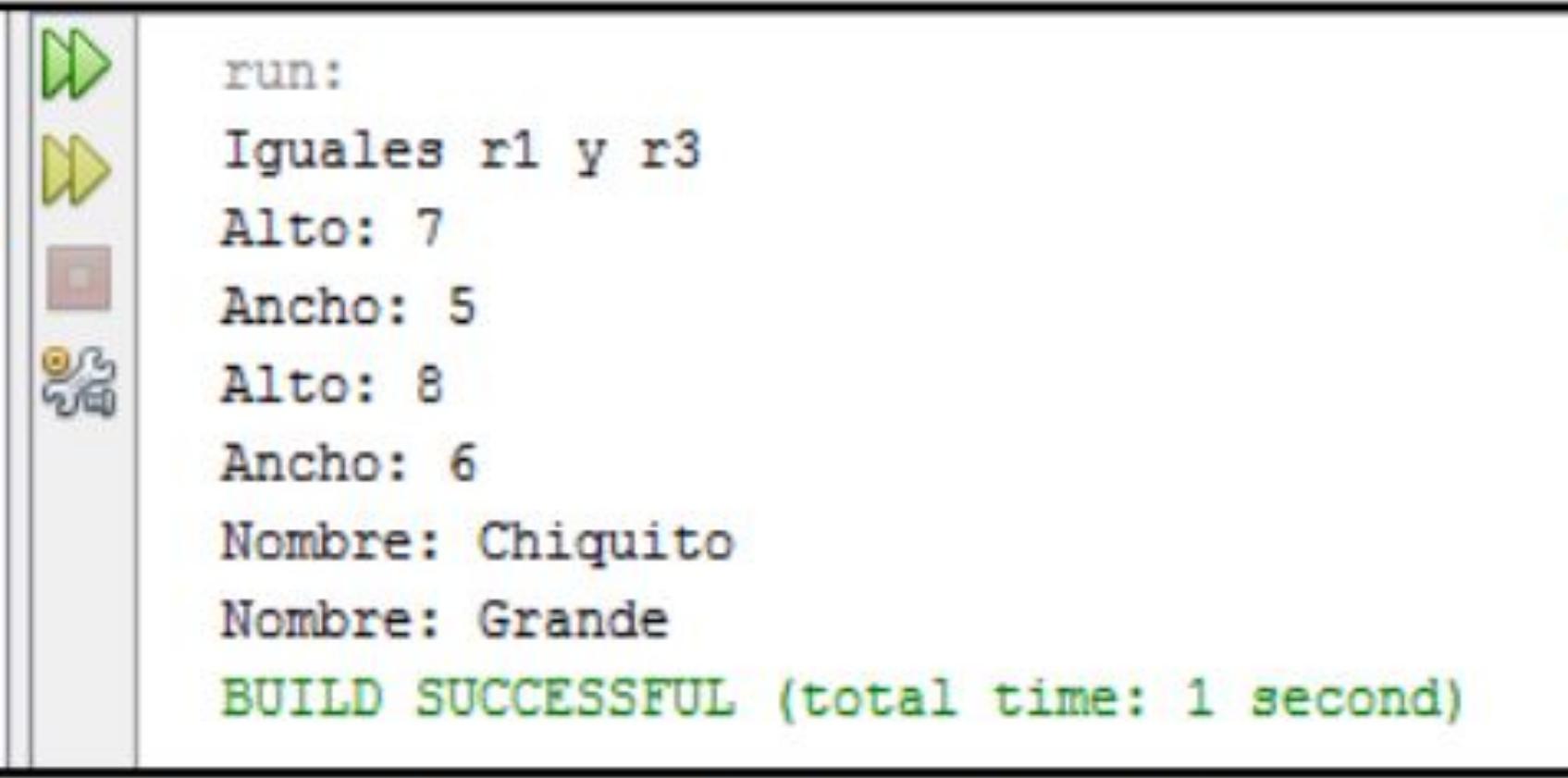
    public int getAlto(){...3 lines}

    public String getNombre(){...3 lines}

    public Rectangulo incrementarAncho(){...4 lines}

    public Rectangulo incrementarAlto(){...4 lines}
@Override
    public Object clone(){
        Object objeto = null;
        try{
            objeto = super.clone();
        }catch (CloneNotSupportedException ex){
            System.out.println("Error al duplicar");
        }
        return objeto;
    }
}
```

```
public static void main(String[] args) {
    Rectangulo r1 = new Rectangulo(5,7);
    Rectangulo r2 = (Rectangulo)r1.clone();
    Rectangulo r3 = r1;
    if(r1==r2){
        System.out.println("Iguales r1 y r2");
    }
    if(r1==r3){
        System.out.println("Iguales r1 y r3");
    }
    r2.incrementarAncho();
    r2.incrementarAlto();
    r1.setNombre("Chiquito");
    r2.setNombre("Grande");
    System.out.println("Alto: " + r1.getAlto());
    System.out.println("Ancho: " + r1.getAncho());
    System.out.println("Alto: " + r2.getAlto());
    System.out.println("Ancho: " + r2.getAncho());
    System.out.println("Nombre: " + r1.getNombre());
    System.out.println("Nombre: " + r2.getNombre());
}
```



```
run:  
Iguales r1 y r3  
Alto: 7  
Ancho: 5  
Alto: 8  
Ancho: 6  
Nombre: Chiquito  
Nombre: Grande  
BUILD SUCCESSFUL (total time: 1 second)
```

Razona los resultados obtenidos

Conceptos de clase

- **Método equals()**

- Permite realizar comparación entre un objeto y otro
- Comprueba que ambas referencias sean iguales (no tiene más ventajas que ==)
- Para una comprobación en profundidad, reescribir el método.

```
Rectangulo r1 = new Rectangulo(4,5);
Rectangulo r2 = new Rectangulo(3,8);
Rectangulo r3 = r1;
if(r1.equals(r2)){
    System.out.println("Iguales r1 y r2");
}
if(r1.equals(r3)){
    System.out.println("Iguales r1 y r3");
}
```

Conceptos de clase

- **Ejemplo:** Método **equal** que compruebe si dos **Rectangulos** son iguales.

Conceptos de clase

- **Método `toString()`**
 - Permite obtener el **nombre de la clase desde el cual fue invocado + @ + representación hexadecimal del código hash del objeto.**
 - Ejemplo
- **Método `finalize()`**
 - Permite **liberar la memoria** ocupada por un objeto sin referencias, invocada por el recolector de basura.

Estructura y miembros de una clase

- Miembros de una clase:
 - **static** (miembro estático de clase)
 - Todos los miembros de la clase compartirán los miembros estáticos.
- Una **variable estática**, es una variable única utilizada por todos los objetos de la clase.

Ejemplo de miembro de clase

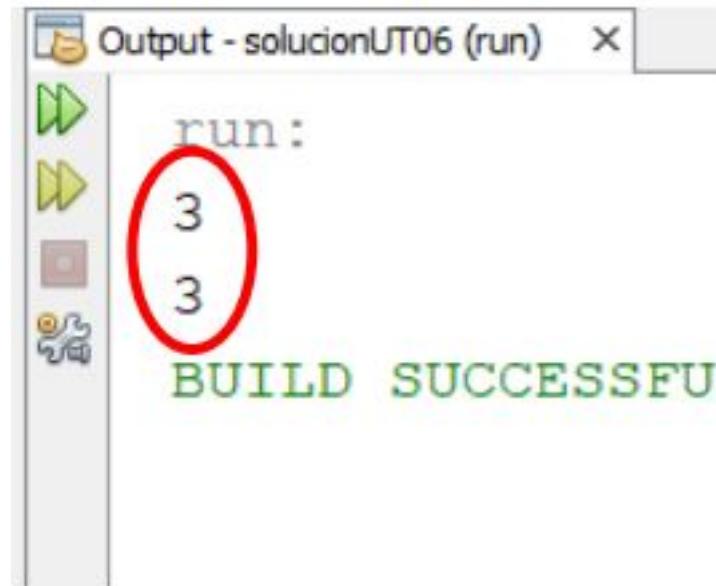
```
public class Cohete{  
    private static int numCohetes=0;  
  
    public Cohete(){  
        numCohetes++;  
    }  
  
    public int getCohetes(){  
        return numCohetes;  
    }  
}
```

```
public static void main(String[] args){  
    Cohete c1 = new Cohete();  
    Cohete c2 = new Cohete();  
    Cohete c3 = new Cohete();  
  
    System.out.println(c1.getCohetes());  
    System.out.println(c3.getCohetes());  
}
```

- **numCohetes** se crea con **c1**
- Se inicializa a 0
- Cuando se crean **c2** y **c3** no se vuelve a inicializar, ya existe y es estática, solo se incrementa.

Ejemplo de miembro de clase

```
public class TestCohete {  
  
    /**  
     * @param args the command line arguments  
     */  
  
    public static void main(String[] args) {  
        Cohete c1=new Cohete();  
        Cohete c2=new Cohete();  
        Cohete c3=new Cohete();  
  
        System.out.println(c1.getCohetes());  
        System.out.println(c3.getCohetes());  
    }  
}
```



Estructura y miembros de una clase

- Métodos de instancia y de clase
 - Métodos de **instancia**: utilizados por la instancia.
 - **Ejemplo**: implementar un método que calcule el área de la clase Rectángulo.
 - Pueden acceder a los miembros de instancia de los métodos de clase.
 - Métodos de **clase**: comunes a una clase.
 - Son métodos estáticos.
 - No tienen referencia **this**.
 - No pueden acceder a miembros que no sean **static**
 - Un método **no static** puede acceder a miembros **static** y miembros **no static**

Estructura y miembros de una clase

```
public class Test {  
    public int dato=0;  
    public static int datoStatico=0;  
  
    public void metodo(){  
        this.datoStatico++;  
    }  
  
    public static void metodoStatico(){  
        this.datoStatico++;  
        datoStatico++;  
    }  
  
    public static void main(String[] args){  
        dato++;  
        datoStatico++;  
        metodoStatico();  
        metodo();  
    }  
}
```

Un método no static sí puede tener referencia this a un atributo static

Error al compilar. Los métodos static no tienen referencia this

Error al compilar. Los métodos static (main es static) no puede acceder a miembros no static.

Error al compilar. Los métodos static no puede acceder a miembros no static.

Estructura y miembros de una clase

- Tabla resumen

Método	Llamada	Declaración	Acceso
Clase	Clase.metodo(parámetros)	static	Miembros de clase
Instancia	Instancia.metodo(parámetros)		Miembros de clase y de instancia

Estructura y miembros de una clase

- Ejemplo de **métodos de clase**: `java.lang.Math`
 - Se llaman anteponiendo el nombre de la clase `Math`

```
Math.cos(angulo);
```

Métodos Math	Descripción
static int abs (int a) static long abs (long a) static double abs (double a) static float abs (float a)	Devuelve el valor absoluto del parámetro pasado
static int max (int a, int b) static long max (long a, long b) static double max (double a, double b) static float max (float a, float b)	Devuelve el mayor de los valores a o b
static int min (int a, int b) static long min (long a, long b) static double min (double a, double b) static float min (float a, float b)	Devuelve el menor de los valores a o b
static double pow (double a, double b)	Potencia de un número. Devuelve el valor de a elevado a b
static double random()	Números aleatorios. Devuelve un número aleatorio de tipo double entre cero y uno (no incluido)
static int round (float a) static long round (double a)	Redondeo. Redondea el parámetro a al valor entero más cercano.