

Clase Object y sus métodos

En lo que todos son iguales

Programación I

Grado en Ingeniería Informática

MDR, JCRdP y JDGD

Introducción

- ▶ La clase Object es la raíz de la jerarquía de clases en Java
- ▶ Pertenece al paquete java.lang
- ▶ Suministra las funcionalidades básicas de cualquier objeto
- ▶ No es posible crear objetos de esta clase pero sus referencias se usan como comodín para referenciar objetos de cualquier clase

Métodos principales

Método	Descripción
<code>Object clone()</code>	Crea un objeto copia del actual y lo devuelve
<code>boolean equals(Object r)</code>	Compara el objeto actual con el referenciado por "r" y devuelve verdadero si son iguales
<code>Class getClass()</code>	Devuelve un objeto de la clase "Class" que nos indica detalles de la clase a la que pertenece el objeto
<code>int hashCode()</code>	Se usa en algunos contenedores. Para el mismo estado se debe devolver el mismo valor.
<code>String toString()</code>	Devuelve la ristra que representa el objeto actual. Se invoca automáticamente cuando se requiere una String y se usa otro tipo de objeto

Comparando objetos

- ▶ Los operadores `==` y `!=` se pueden usar para comparar referencias no para comparar objetos
- ▶ Para comparar objetos se utiliza el método `equals`
- ▶ Al método `equals` se le pasa un objeto y devuelve un lógico
- ▶ Para comparar el objeto referenciado por `x` e `y` se usa:

`x.equals(y)`

Comparando objetos

- ▶ Todos los objetos disponen de **equals** heredado de la clase **Object**
- ▶ La mayoría de las clases de la biblioteca de clases implementan su método **equals**
- ▶ Si queremos que los objetos de las clases que creamos se puedan comparar adecuadamente debemos redefinir **equals**
- ▶ **equals** tiene el siguiente formato:
public boolean equals(Object obj) {...}

Comparando objetos

- ▶ El uso del parámetro `Object` permite comparar con cualquier objeto
- ▶ En caso de que la referencia pasada sea **null** se debe devolver falso
- ▶ En general, es necesario un **cast** para acceder al objeto de la clase correspondiente y hacer la comparación

Comparando objetos

```
public bool Cilindro {  
    ...  
    public boolean equals(Object obj){  
        if(obj instanceof Cilindro){  
            Cilindro cmp=(Cilindro) obj;  
            return altura==cmp.altura &&  
                radio ==cmp.radio;  
            //La comparación de reales es insegura  
        }else return false;  
    }  
    ...  
}
```

Copiando objetos

- ▶ Hay que tener presente que la operación de asignación "=" no podemos aplicarla a objetos, sólo a referencias.
- ▶ La forma en que se copia objetos en Java es mediante un método que duplica el objeto sobre el que actúa:
`Object clone()throws CloneNotSupportedException`
- ▶ El método **clone** se hereda de la clase **Object**

Copiando objetos

- ▶ Si se quiere permitir que se puedan clonar objetos de una clase, la clase debe implementar la interfaz vacía **Cloneable**
- ▶ Aunque se hereda clone se debe implementar un método clone público
- ▶ Por convención el objeto a devolver debe obtenerse al llamar a `super.clone()`

Copiando objetos

- ▶ La llamada a `super.clone()` nos devuelve un objeto nuevo al que se le han asignado atributo a atributo los valores del actual (copia superficial)
- ▶ La clonación de un objeto debería implicar la independencia del original por lo que se debe hacer una clonación de los objetos atributos que sean necesarios (copia en profundidad).
- ▶ Los array en Java realizan una clonación superficial.

Ejemplo de copia de objetos

```
class Alineación implements Cloneable{
    Futbolista[] futbolistas= new Futbolista[11] ;
    ...
    public Alineación clone()
        throws CloneNotSupportedException
    {
        Alineación r= (Alineación) super.clone();
        r.futbolistas= futbolistas.clone();
        for(int i=0; i<futbolistas.length; i++)
            r.futbolistas[i]=futbolistas[i].clone();
        return r;
    }
    ...
}
```

Mostrando objetos

- ▶ En muchas ocasiones se requiere obtener el estado de un objeto en un formato visible
- ▶ Esta forma legible es normalmente un String
- ▶ La clase **Object** suministra un método **toString** que devuelve una ristra con la clase y número de referencia del objeto sobre la que se aplica
- ▶ Si queremos personalizar este resultado debemos redefinir el método **toString**
- ▶ El compilador siempre que encuentra una referencia distinta de String donde debe ir una String genera una llamada al método **toString()**

Mostrando objetos

```
public class Conjunto{
    private int[] v= new int[10];
    ...
    public String toString() {
        String res="{ ";
        for(int i=0; i<v.length; i++){
            res += v[i];
            if(i<v.length-1){
                res +=", ";
            }
        }
        return res+"}";
    }
    static public void main(String[] args){
        Conjunto c;
        System.out.println(c);//Equivale a c.toString()
    }
}
```