Unidad 7: anexo II

Diseño de tipos

Igualdad, representación, código, copia y orden

Contenidos

- Diseño de Tipos
- Igualdad, representación y código hash
- Clonación
- Orden natural

Diseño de tipos – Guía Interfaz

- Elección de propiedades:
  - **Simples**: sus valores no dependen de otras propiedades
  - **Derivadas**: dependen de alguna/s de las otras propiedades
  - **Compartidas**: son las propiedades de clase
- Elección del conjunto de métodos del tipo
- Establecer criterios para las siguientes propiedades:
  - Representación como cadena
  - Código como entero (Código Hash)
  - Criterio de igualdad
  - Orden natural
  - Clonación o copia

Tipos

- La clase Object
- Interfaz Cloneable
- Interfaz Copiable
- Interfaz Comparable

La clase Object

- La clase ***Object*** es la raíz de la jerarquía de clases en Java y debe ser extendida por cualquier nueva clase. Algunos de sus métodos públicos son:

```
                ...
boolean equals(Object o);
int hashCode();
String toString();

                ...
```

  - **equals**: se utiliza para comparar si dos objetos son *iguales*.
  - **hashCode**: devuelve el código *hash* del objeto. Dos objetos iguales tendrán el mismo código *hash*.
  - **toString**: devuelve una cadena de texto que representa al objeto.

La clase Object

- **equals:**
  - **Simetría**: si un objeto es igual a otro, el segundo también es igual al primero.
  - **Transitividad**: si un objeto es igual a otro, y este segundo es igual a un tercero, el primero también será igual al tercero.
- **equals/toString**:
  - Si dos objetos son iguales, sus representaciones en forma de cadena también lo serán.
- **equals/hashCode**:
  - Si dos objetos son iguales, sus códigos hash tienen que coincidir.

Equals

- Cuando se sobreescriba *Equals* no olvidar:
  - Comprobar que el argumento es del tipo adecuado y que no es nulo.
  - Si lo anterior se cumple:
    - Realizar la comprobación de equivalencia

Hashcode

- Un **hashCode** es un identificador de 32 bits que se almacena en la instancia de la clase.
- Toda clase debe incorporar un método **hashCode()** sobreescrito de forma más especifica con los datos contenidos.
- El **hashCode** tiene importancia para el rendimiento de estructura de datos que agrupan objetos en base al cálculo de los hashCode.
- Se sobrescribe para que se comporte de forma acorde a como lo hace **equals()**, es decir, si el método equals() dice que dos objetos son iguales, estos han de tener el mismo valor hash.

Hashcode: Ejemplo

- hashcode para un objeto Empleado basado en el hashcode de su DNI, su nombre y de su departamento. Cuanto más elaborada sea la función menos **sinónimos** o **colisiones** existirán en nuestro hash.

```
public class Empleado {
    String       DNI;
    int          IdEmpleado;
    String       nombre;

    //resto de atributos
    //Métodos de la Clase

    @Override
    public int hashCode() {
      int hash = 1;
      hash = hash * 15 + DNI.hashCode();
      hash = hash * 17 + IdEmpleado;
      hash = hash * 31 + nombre.hashCode();
      hash = hash * 13 + ((dept == null) ? 0 : dept.hashCode());
      return hash;
    }
}
```

La interface Clonable-Copia

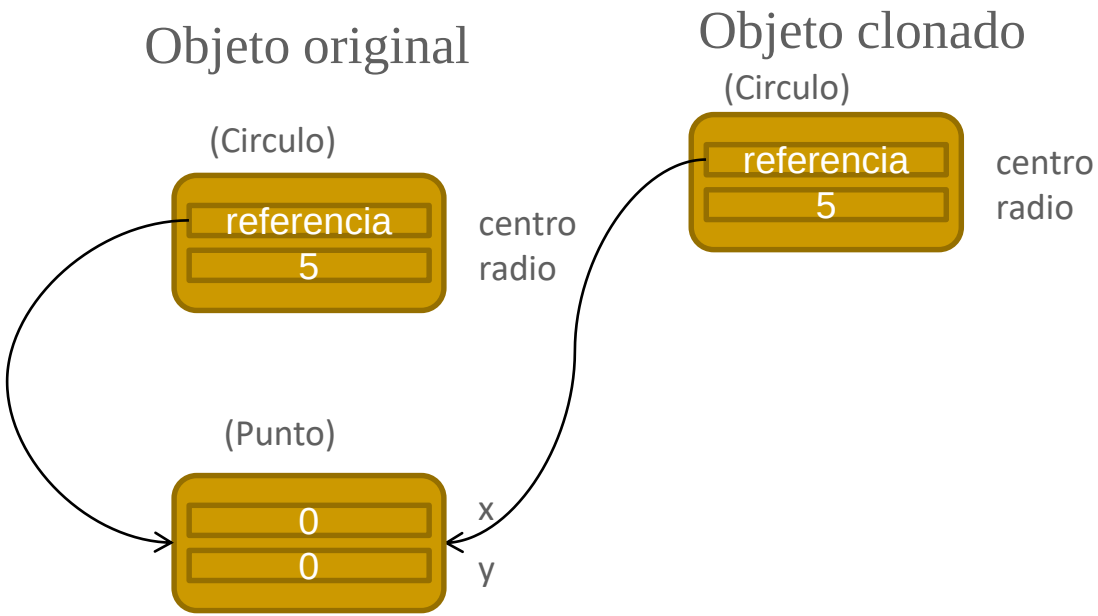
```
package java.lang;
public interface Cloneable {
}
```

- Esta interfaz no añade métodos, pero habilita que los objetos puedan invocar el método *clone* de *Object*.
- Si una clase no la implementa y hace uso de *clone,* se lanzará la excepción *CloneNotSupportedException*.
- Propiedad **clone**/**equals**:
  - Si un objeto se obtiene a partir de otro haciendo uso del método **clone**, ambos objetos serán iguales pero no idénticos.

Copia superficial

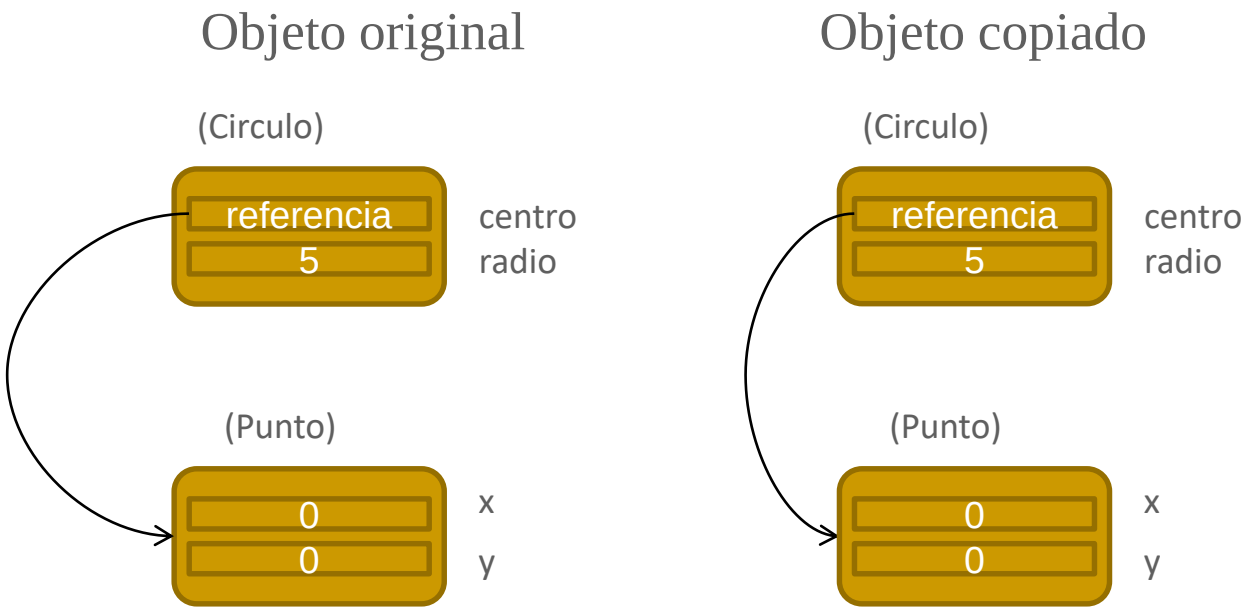
- Es la clonación por defecto
- **clone**: si el objeto tiene atributos de tipo objeto se realiza una copia de sus referencias, no se crean nuevos objetos.

  Ejemplo: clonación superficial de objetos de tipo Circulo



Copia en profundidad

- **copia**: si el objeto tiene atributos de tipo objeto se realiza una copia de dichos atributos en nuevos objetos. Es necesario generar el código en cada caso.
  Ejemplo: copia en profundidad de objetos de tipo Circulo



Interfaz Comparable

```
package java.lang;
public interface Comparable<T>{
     int compareTo(T ob);
}
```

- **compareTo** establece un orden natural para los objetos.
- Debe coincidir en criterio con el método *equals*.
- Valor entero devuelto:
  - Negativo si *this* es menor que *ob*, es decir, está antes en el orden.
  - Cero si *this* es igual a *ob*, es decir, el método *equals* devuelve true entre ellos.
  - Positivo si *this* es mayor que *ob*, es decir, está después en el orden.

Interfaz Comparable. Propiedades

- **compareTo**:
  - **Simetría**: si el resultado de comparar un objeto con otro es cero, el resultado coincide a la inversa.
  - **Transitividad**: si el signo de comparar un objeto con otro coincide con el signo de comparar el segundo con un tercero, entonces también coincide con el signo de comparar el primero con el tercero.
- **equals/compareTo**:
  - Si dos objetos son iguales, el resultado de su comparación será igual a cero, y viceversa.