Más de Java 2

Comparator, Comparable, Acotamiento de tipos y Clases anónimas

Comparator

En Java hay una interfaz que permite comparar elementos de otra clase.

```
public interface Comparator<T> {

        public int compare (T o1, T o2 );
        .
        .
        .

}
```

Si el valor que devuelve el método compare...

- Es negativo, entonces o1 < o2
- Es cero, entonces o1 = o2
- Es positivo, entonces o1 > o2

Comparable

En Java hay una interfaz que permite a la clase que la implemente comparar instancias entre sí.

```
public interface Comparable<T> {

        public int compareTo (T o);

}
```

Sea x el objeto invocante, si el valor que devuelve el método compareTo...

- Es negativo, entonces x < o
- Es cero, entonces x = o
- Es positivo, entonces x > o

ED
ESTRUCTURAS DE DATOS

Ejemplo

Usando la clase MiComparador que implementa **Comparator**:

```
MiComparador comparador = new MiComparador( … );
int x = comparador.compare(objeto1, objeto2);
if ( x == 0) System.out.println(“Son iguales”);
if ( x < 0 ) System.out.println(“objeto1 es menor”);
if ( x > 0 ) System.out.println(“objeto1 es mayor”);
```

Usando la clase ElementoOrdenado que implementa **Comparable**:

```
ElementoOrdenado objeto1 = new ElementoOrdenado( … );
ElementoOrdenado objeto2 = new ElementoOrdenado( … );
int x = objeto1.compareTo(objeto2);
if ( x == 0) System.out.println(“Son iguales”);
if ( x < 0 ) System.out.println(“objeto1 es menor”);
if ( x > 0 ) System.out.println(“objeto1 es mayor”);
```

Acotamiento de tipos

Al usar clases genéricas podemos restringir el tipo parametrizado para que sea de algún tipo específico. Se hace de la siguiente manera:

$$\text{class MiClase<T extends } B_1 \text{ \& } B_2 \text{ \& ... \& } B_n\text{> \{ ... \}}$$

En este caso, la palabra reservada **extends** se usa indistintamente tanto para clases como para interfaces.

ED
ESTRUCTURAS DE DATOS

```
Class A { ... }
Interface B { ... }
Interface C { ... }
```

Si un tipo parametrizado es acotado con varios tipos, entonces las clases deben ser especificadas antes que las interfaces.

```
class MiClase<T extends A & B & C> { ... } // Completamente válido
```

```
class MiClase<T extends B & A & C> { ... } // Dará error de compilación
```

ED
ESTRUCTURAS DE DATOS

Ejemplo

Podemos crear una clase genérica y asegurarnos que sus elementos sean comparables entre sí.

```
public class ColeccionOrdenada <T extends Comparable> { … }
```

También podemos hacer alguna clase genérica que solo funcione con números.

```
public class ColeccionNumerica <T extends Number> { … }
```

Clases anónimas

Son clases que no tienen nombre y solo se usan una vez.

Mientras que las clases normalmente son declaraciones, las clases anónimas son expresiones.

Para usar una clase anónima se llama a un constructor y a continuación de este se agrega la definición de la clase en un bloque de código.

ED
ESTRUCTURAS DE DATOS

Sintáxis de una clase anónima

- El operador **new**
- El nombre de una interfaz a implementar o una clase a extender.
- Paréntesis que contienen los argumentos de un constructor.
  En caso de implementar una interfaz se usan paréntesis sin argumentos.
- Un bloque de código entre llaves con la declaración de una clase.
  *Se pueden declarar tantos elementos como en una clase normal, con la excepción de que **no se pueden declarar constructores.***

ED
ESTRUCTURAS DE DATOS

Ejemplo

Tenemos la clase ColeccionGatos con el siguiente método:

```
public void reordena (Comparator<Gato> comparador);
```

Podemos reordenar por nombre:

```
miColeccion.reordena(new Comparator<Gato> ( ) {

   @Override
   public int compare(Gato g1, Gato g2){
      return g1.getNombre( ).compareTo(g2.getNombre( ));
   }
});
```

También podemos reordenar por edad:

```
miColeccion.reordena(new Comparator<Gato> ( ) {

   @Override
   public int compare(Gato g1, Gato g2){
      return g1.getEdad( ) - g2.getEdad( );
   }
});
```

Continuará...