Genéricos

Antonio Espín Herranz

Introducción

- Las clases genéricas se introdujeron en java 1.5
- Los *generics* permiten **usar tipos** para **parametrizar** las clases, interfaces y métodos al definirlas.
- Los beneficios son:
  - **Comprobación de tipos más fuerte** en tiempo de compilación.
  - **Eliminación de *casts*** aumentando la legibilidad del código.
  - Posibilidad de implementar **algoritmos genéricos**, con tipado seguro.

Introducción

- Podríamos partir de un código similar a este (con el objetivo de almacenar un tipo cualquiera):

```
public class ObjectWrapper {
    private Object ref;

    public ObjectWrapper(Object ref) {
        this.ref = ref;
    }
    public Object get() {
        return ref;
    }
    public void set(Object reference) {
        this.ref = ref;
    }
}
```

El uso de esta clase implicará Casting cuando vayamos a obtener un objeto

Introducción

- **Un uso de la clase podría ser:**

```
ObjectWrapper stringWrapper = new ObjectWrapper("Hello");
stringWrapper.set("another string");
String myString =(String)stringWrapper.get();
```

- ***Pero nada evita a nivel de compilación un código como este:***

```
ObjectWrapper stringWrapper = new ObjectWrapper("Hello");
stringWrapper.set(new Integer(101));
String myString =(String)stringWrapper.get();
```

  - **El código compila pero en ejecución podemos obtener una exception → ClassCastException**

Introducción

- La forma correcta de hacerlo es un con una clase genérica que en tiempo de compilación indicamos el tipo de objeto que queremos referenciar.

- En la declaración de la clase podemos indicar un tipo o varios genéricos:

Ejemplo

```
public class Wrapper<T> {

private T instancia;

public Wrapper(T instancia) {
	super();
	this.instancia = instancia;
}

public T getInstancia() {
	return instancia;
}

public void setInstancia(T instancia) {
	this.instancia = instancia;
}

@Override
public String toString() {
	return "Wrapper [instancia=" + instancia + "]";
}
}
```

Introducción

- Para utilizar el tipo anterior necesitamos declarar objeto e indicar el tipo en la clase al igual que hacemos con las colecciones.

Wrapper<Integer> entero = **new Wrapper<Integer>(123);**
System.***out.println(entero.getInstancia());***

Wrapper<String> texto = **new Wrapper<>("hola");**
System.***out.println(texto.getInstancia());***

- ***Si cometemos los errores de antes, de cambiar de tipo, el compilador nos avisa en tiempo de compilación.***

Introducción

- También es posible pasar más de un tipo:

```
public class MyClass<T, U, V, W> {}
```

- También podemos utilizar interfaces con tipos genéricos:

```
public interface Pair<K, V> {
        public K getKey();
        public V getValue();
}
```

- Es el **compilador el que genera la clase adecuada** para el tipo indicado en la declaración de la clase.
- **La JVM no conoce la clase genérica.**

Ejemplos

```
public class OrderedPair<K, V> implements Pair<K, V> {
    private K key;
    private V value;

    public OrderedPair(K key, V value) {
        this.key = key;
        this.value = value;
    }

    public K getKey() { return key; }
    public V getValue() { return value; }
}
```

Convenciones de nombres

- E: elemento de una colección.
- K: clave.
- N: número.
- T: tipo.
- V: valor.
- S, U, V etc: para segundos, terceros y cuartos tipos.

En los métodos

- **Los métodos también pueden tener su propia definición de tipos genéricos** y no tiene porque ser la clase:

```
public class MiClase {
    public static <K, V> boolean compare(Pair<K, V> p1,
        Pair<K, V> p2){
        return p1.getKey().equals(p2.getKey()) &&
            p1.getValue().equals(p2.getValue());
    }
}
```