Tema 4 –Colecciones y Genericidad

Programación Orientada a Objetos

Grado en Ingeniería Informática

DIS
Departamento de Informática y Sistemas

UNIVERSIDAD DE MURCIA

Contenido

- **Parte 1 – Colecciones**
  - Tipos `Collection`, `sequencedCollection`, `List` y `Set`
  - Tipos `Map`, `SequencedMap`, `HashMap`
  - Ordenación: `Comparable` y `Comparator`
    - `TreeSet` y `TreeMap`
  - Iteradores. Tipos `Iterable` e `Iterator`

- **Parte 2 – Genericidad**
  - Definición de clases genéricas
  - Declaración y construcción de tipos genéricos
  - Genericidad y tipos dinámicos
  - Métodos genéricos

Parte 2 - Genericidad

- Facilidad de un lenguaje de programación para definir **clases, interfaces y métodos parametrizados con tipos** de datos.
- Resultan de utilidad para la implementación de **tipos de datos contenedores** como las **colecciones** (`List<T>`, `HashSet<T>`)
- La genericidad sólo tiene sentido en **lenguajes con comprobación estática de tipos**, como Java.
- **La genericidad permite escribir código reutilizable.**

Definición de clase genérica

- Una **clase genérica** es una clase que en su declaración utiliza un tipo variable (**parámetro**), que será establecido cuando sea utilizada.

- Al **parámetro** de la clase genérica se le proporciona un nombre (`T`, `K`, `J`, etc.) que permite utilizarlo como **tipo de datos en el código de la clase**.

- Sobre las variables cuyo tipo sea el parámetro (`T`, `K`, `J`, etc.) **sólo es posible aplicar métodos de la clase `Object`**:
  - → dado que representan "cualquier dato" sólo podemos aplicar operaciones disponibles en todos los tipos de datos del lenguaje Java.

Clase genérica Contenedor

```
public class Contenedor<T> {

        private T contenido;

        public void setContenido(T contenido) {

                this.contenido = contenido;
        }

        public T getContenido() {

                return contenido;
        }
}
```

Operaciones disponibles

- Sobre una variable de tipo T, sólo podemos aplicar **métodos públicos de la clase `Object`**
  - **Nota**: el método `clone()` no es público en la clase Object.

- También podemos utilizar la **asignación** (=) y la comparación de **identidad** (== o !=).

- Dentro de la clase genérica, **NO es posible construir objetos de los tipos parametrizados**:

  ```
  T contenido = new T(); // No compila
  ```

Ejemplo: `hashCode` y `equals` en `Contenedor`

```
@Override
public int hashCode() {
    final int prime = 31;
    int result = 1;
    result = prime * result + ((contenido == null) ? 0 :
                                contenido.hashCode());
    return result;
}

@Override
public boolean equals(Object obj) {
      if (this == obj) return true;
      if (obj == null) return false;
      if (getClass() != obj.getClass()) return false;

      Contenedor<T> other = (Contenedor<T>) obj;

      return contenido.equals(other.contenido);
}
```

Uso de una clase genérica

- La **parametrización** de una clase genérica se realiza en la **declaración** de una variable y en la **construcción de objetos**.

```
Contenedor<String> contenedor =
            new Contenedor<String>();

contenedor.setContenido("hola");
```

- A partir de la versión 8 de Java se puede omitir el tipo al llamar al constructor.

Genericidad y tipos primitivos

- Las clases genéricas **no pueden ser parametrizadas a tipos primitivos**.
- Para resolver este problema el lenguaje define **clases envoltorio** de los tipos primitivos:
  - **`Integer`**, **`Float`**, **`Double`**, **`Character`**, **`Boolean`**, etc.
- El compilador transforma automáticamente tipos primitivos en clases envoltorio y viceversa: **autoboxing**.

```
Contenedor<Integer> contenedor =
                        new Contenedor<Integer>();
contenedor.setContenido(10);
int valor = contenedor.getContenido();
```

Uso de una clase genérica

- Es posible declarar un tipo genérico sin especificar el tipo (***tipo puro***).
- Es equivalente a parametrizar el tipo genérico con la clase `Object`.

```
Contenedor contenedor = new Contenedor();

contenedor.setContenido("hola");
contenedor.setContenido(7);
```

- **No es una práctica recomendable**. El compilador lo marca como un aviso.

Genericidad – Tipos dinámicos

- En **tiempo de ejecución** se mantiene la información sobre el tipo utilizado para parametrizar la clase genérica.

```
if (contenedor instanceof Contenedor<Burbuja>) {
    …
}
```

Genericidad – Conversión de tipos

- Una conversión de tipos a un tipo genérico lo marca el compilador como un *warning*.
  - No se puede comprobar el tipo utilizado para la parametrización.

```
public static void main(String[] args) {
      LinkedList<Punto> puntos = new LinkedList<Punto>();

      // ... se crean y añaden objetos punto a la lista


      @SuppressWarnings("unchecked")
      LinkedList<Punto> copia =
                        (LinkedList<Punto>)puntos.clone();
}
```

Métodos genéricos

- Un método que declara una variable de tipo (por ejemplo, `<T>`) se denomina **método genérico**.
- Antes de la declaración del tipo de retorno del método se indica una variable que representa el tipo genérico (`<T>`).
- El alcance de la variable de tipo (`<T>`) es local al método, esto es, puede aparecer en la signatura del método y en el cuerpo del método.
- Es posible definir métodos genéricos incluso en clases que no son genéricas. Por ejemplo, la clase **`Collections`** tiene métodos genéricos y no es genérica.

Métodos genéricos - Ejemplo 1

- Método que acepta una secuencia de valores de cualquier tipo y lo convierte en una lista:

```
public static <T> List<T> asList (T... datos) {
        List<T> lista = new ArrayList<T>(datos.length);

        for (T elemento : datos)
                lista.add(elemento);

        return lista;
   }
```

Métodos genéricos - Ejemplo 1

- El método `asList` se podría invocar como sigue:

```
public static void main(String[] args) {

    List<Integer> listaEnteros = asList(1,2,3);

    String[] arrayPalabras = {"hola", "ciao", "hello"};

    List<String> listaPalabras = asList(arrayPalabras);
}
```

- **El tipo de `T` se infiere** a partir del tipo de los argumentos o la variable a la que se asigna el resultado.

Métodos genéricos - Ejemplo 2

- Añade una secuencia variable de elementos a una lista:

```
public static <T> void addAll (List<T> lista,
                               T... elementos) {
     for (T elemento : elementos)
             lista.add(elemento);
}
```

```
List<Integer> enteros = new ArrayList<Integer>();
addAll(enteros, 1, 2, 3);
```

Métodos genéricos - Ejemplo 3

- El siguiente ejemplo declara un método genérico que retorna un elemento aleatorio de cualquier lista:

```
public static <T> T getElementoAleatorio(List<T> lista){
    Random random = new Random();
    int index = random.nextInt(lista.size());
    return lista.get(index);
}

// Programa

List<Integer> enteros = new ArrayList<Integer>();
addAll(enteros, 1, 2, 3);

int entero = getElementoAleatorio(enteros);
```

Genericidad. Aspectos clave.

- Cualquier tipo (clase, interfaz) se puede definir como un tipo genérico (con parámetros).
- Un tipo genérico puede tener varios parámetros.
  - `HashMap<K,V>`
- Una clase genérica puede heredar de otra clase genérica.
- Una clase genérica puede implementar una interfaz genérica.
  - `LinkedList<T>` implementa `List<T>`
- Al heredar o implementar el tipo genérico se puede establecer el tipo.
  - `class OrdenInverso implements Comparator<String>`

Anexos

- Como **anexos** al tema 4 se han incluido los siguientes contenidos:
  - Limitaciones que impone la genericidad al sistema de tipos. Solución utilizando el **tipo comodín**.
  - **Genericidad restringida**: permite restringir el conjunto de tipos que se pueden utilizar para parametrizar un parámetro de tipo genérico.
  - **Genericidad y herencia**.
  - Genericidad en el tipo de datos **Pila**.