Grado en Ingeniería Informática
Grado en Matemáticas e Informática

Asignatura: PROGRAMACIÓN II

Genéricos

Clara Benac Earle, Manuel Collado
Ángel Lucas González Martínez
Jaime Ramírez Rodríguez
Guillermo Román

DLSIIS - E.T.S. de Ingenieros Informáticos
Universidad Politécnica de Madrid

Noviembre 2013

Introducción

- Java permite definir clases con variables o parámetros que representan tipos ⇒ **Clases genéricas**
- Esto es posible, cuando el tipo de un dato no afecta al tratamiento que se le va a dar a ese dato.
  - Ejemplo: **clases contenedoras**
- Un mecanismo similar está disponible en lenguajes como C#, C++ o Ada

Introducción

- Surge a partir de Java 1.5 como una necesidad para mejorar la verificación de tipos
- Antes se utilizaba la clase Object como “comodín” para simular los genéricos, pero esto conllevaba:
  - Se podían mezclar objetos de las clases más dispares en una misma estructura contenedora
  - Había que realizar muy a menudo *castings* (conversión de Object a cualquier otra clase)

Solución: Genéricos

- Los genéricos permiten parametrizar el “tipo de contenido” de una clase contenedora
  - Ejemplo: List<T> permite crear una lista de elementos de tipo T
- De esta forma se verifica en tiempo de compilación el código
  - Evitamos mezclas en las clases contenedoras
  - Evitamos la realización de castings
- Permiten hacerlo “a la antigua usanza” pero el compilador nos avisa con un *warning* si no parametrizamos

Motivación

- Supongamos que en un programa necesitamos una clase ParEnteros:

```
public class ParEnteros {
  private int a, b;

  public ParEnteros(int a,int b) {
    this.a = a;
    this.b = b;
  }

  public ParEnteros swap () {
    return new ParEnteros(b, a);
  }
}
```

Motivación

- Pero también necesitamos un clase ParCaracteres con los mismos métodos que la anterior.

```
public class ParCaracteres {
  private char a, b;

  public ParCaracteres(char a,char b) {
    this.a = a;
    this.b = b;
  }

  public ParCaracteres swap () {
    return new ParCaracteres(b, a);
  }
}
```

- La implementación de los métodos es la misma, porque no depende del tipo del dato manipulado.

Definición de la clase genérica Par

- Abstraemos el tipo convirtiéndolo en un parámetro genérico T.

```
public class Par<T> {
  private T a, b;

  public Par(T a, T b){
    this.a = a;
    this.b = b;
  }

  public Par<T> swap (){
    return new Par<T>(b, a);
  }
}
```

T es el identificador que se asigna al tipo que se pasará cuando se instancie. Se puede poner cualquier identificador válido en Java. Por ejemplo **Datos**.

Ejemplo con clases genéricas Par

```
public class Prueba1 {
  public static void main(String[] args) {
    Par<String> p = new Par<String>("uno","dos");
    Par<Fecha> p1 = new Par<Fecha>(new Fecha(1,2,2003),
                                  new Fecha(4,5,2006));
    System.out.println ("Primer elemento de p: " + p.getA());
    System.out.println ("Primer elemento de p1: " + p1.getA());
    p = p.swap();
    p1 = p1.swap();
    System.out.println ("Primer elemento de p: " + p.getA());
    System.out.println ("Primer elemento de p1: " + p1.getA());
  }
}
```

¿Por qué no funciona?

```
public class Prueba2 {
  public static void main(String[] args) {
    Par<int> p = new Par<int>(1,2);
    Par<char> p1 = new Par<char>('a','b');
    System.out.println ("Primer elemento de p: " + p.getA());
    System.out.println ("Primer elemento de p1: " + p1.getA());
    p = p.swap();
    p1 = p1.swap();
    System.out.println ("Primer elemento de p: " + p.getA());
    System.out.println ("Primer elemento de p1: " + p1.getA());
  }
}
```

- El mecanismo de genéricos en Java requiere que el parámetro sea una clase, no un tipo básico.
- Solución ⇒ clases envoltorio (*wrappers*).

Clases envoltorio (*wrapper*)

- En Java existen tipos primitivos (`int`, `boolean`, `double`, etc.) para crear datos primitivos, y clases para poder crear objetos.
- En ocasiones (lo veremos más adelante) es muy conveniente poder tratar datos primitivos como objetos.
- Para ello se pueden crear las clases envoltorio (*wrapper classes*) que crean un “envoltorio” sobre los datos primitivos para poder tratarlos como objetos.

Ejemplo: envoltorio simple

- Ejemplo: podríamos definir la siguiente clase envoltorio para los enteros

```
public class Entero {
  private int valor;

  public Entero (int valor) {
    this.valor = valor;
  }

  public int intValue() {
    return valor;
  }
}
```

Clases asociadas a los tipos básicos

- Java proporciona **clases envoltorio** para los tipos básicos
- Por lo tanto, no tenemos que definirlas nosotros
- En muchos casos, Java puede realizar una conversión automática:

  Tipo Básico ↔ Clase

Clase	Tipo básico
Boolean	boolean
Byte	byte
Character	char
Double	double
Float	float
Integer	int
Long	long
Short	short

Uso de las clases envoltorio

- En general, se pueden mezclar valores básicos y objetos envoltorio en una expresión aritmética o asignación.
- La distinción entre el operador **==** y el método equals() se aplica también a los objetos de las clases envoltorio.
- Tanto los tipos básicos **int**, **char** como las correspondientes clases **Integer**, **Character** se pueden usar como selectores en sentencias **switch**.
- Proporcionan conversión String ↔ Tipo o Clase:

  Integer.valueOf(*string*) → *integer*

  integer.**toString**() → *string*

  Integer.toString(*int*) → *string*

Ahora ya funciona, usando *wrappers*

- Usamos Integer y Character en lugar de **int** y **char**.

```
public class Prueba2 {
  public static void main(String[] args) {
    Par<Integer> p = new Par<Integer>(1,2);
    Par<Character> p1 = new Par<Character>('a','b');
    System.out.println ("Primer elemento de p: " + p.getA());
    System.out.println ("Primer elemento de p1: " + p1.getA());
    p = p.swap();
    p1 = p1.swap();
    System.out.println ("Primer elemento de p: " + p.getA());
    System.out.println ("Primer elemento de p1: " + p1.getA());
  }
}
```

Crear *arrays* genéricos

- Java no permite crear *arrays* de elementos genéricos.

```
public class ParArrayGenerico<T> {
    private T[] par = new T[2]; // ERROR
    ....
}
```

- Solución: reflexión y la clase Array de la API

```
public class ParArrayGenerico<T> {
   private T[] par;
   public ParArrayGenerico (Class<T> clase) {
      this.par = (T[]) Array.newInstance(clase, 2);
   }
}
```