Certified Tech Developer
The Ultimate Degree

Generics

Programación paramétrica

La programación paramétrica no es exclusiva de Java. De hecho, el lenguaje tardó bastante en incorporarla. Ada, un lenguaje orientado a objetos, lo contempla desde los años ochenta. Hay otros lenguajes que incorporan esta característica, como C++ (a través de *templates*), D, Eiffel, Delphi, TypeScript, etc. Hay muchos otros lenguajes que utilizan los "tipos paramétricos de datos". Algunos, lo llaman *genericity* dado que se establece un tipo de dato genérico para la estructura de dato o clase que se está definiendo, que se establecerá al momento de su uso. En Java, se lo denomina de manera similar: **Generics.**

En pocas palabras, Generics consiste en diferir la pregunta "¿de qué tipo es este objeto?". Entonces, el "tipo" del objeto se deja como un parámetro que el programador establecerá al momento de trabajar con ese objeto.

Sintaxis y uso

En Java, para definir un tipo paramétrico de datos, usamos los "< >". Supongamos un ejemplo sencillo: un balde. Este puede contener muchas cosas, tierra, arena, agua, combustible, etc.

Veamos cómo poner esto en código

```
public class Balde<T> {
   private T contenido;

   public Balde() {
   }

   public llenar(T contenido) {
       this.contenido = contenido;
   }

   public T obtenerContenido() {
       return contenido;
   }
}
```

Aquí, cuando definimos la clase balde, no establecimos el tipo de su contenido (el tipo de su atributo “contenido”), sino que “diferimos” esa decisión hasta el momento de usarlo. Es decir, dejamos el tipo como parámetro. Ese parámetro está definido por la letra T (puede ser cualquier letra del abecedario, menos Ñ, por supuesto). Normalmente se usa T por *type*. Esto también ocurre con las operaciones sobre el contenido: limitamos su uso de acuerdo con el tipo del atributo contenido.

Ahora, suponiendo que tenemos las clases agua, arena, combustible, etc., podríamos hacer:

```
Agua a = new Agua();
Combustible c = new Combustible();
Balde<Agua> b = new Balde<>();
b.setContenido(a);

// NO COMPILA! No puedo poner combustible en un balde de agua!
//b.setContenido(c);

// si el balde es de agua, siempre obtendré agua de él
System.out.println("Voy a tomar" + b.obtenerContenido());
```

Como podemos ver, no hubo *casteos* para operar con el contenido, y por tanto no hay peligros de errores al momento de la ejecución. Adicionalmente, no podemos poner algo que no corresponda dentro del balde.

¿Por qué no hacemos, entonces, el tipo del contenido "Object"? Eso nos permitiría poner agua, combustible o arena en el balde...

```
public class Balde {
   private Object contenido;

   public Balde() {
   }

   public llenar(Object contenido) {
       this.contenido = contenido;
   }

   public Object obtenerContenido() {
       return contenido;
   }
}
```

Es totalmente válido, pero no solo debemos *castear*, sino que podríamos mezclar el contenido del balde:

```
 Agua a = new Agua();
 Combustible c = new Combustible();
 Balde b = new Balde ();
 b.setContenido(a);
 b.setContenido(c);

//dudo que el contenido tenga un sabor agradable...
System.out.println("Voy a tomar" + b.obtenerContenido());
```

Es por esto por lo que el uso de tipos paramétricos, es decir, de Generics se vuelve interesante.