Genéricos

En pocas palabras, los genéricos permiten que los tipos (clases e interfaces) sean parámetros al definir clases, interfaces y métodos. Al igual que los parámetros formales más conocidos que se utilizan en las declaraciones de métodos, los parámetros de tipo proporcionan una forma de reutilizar el mismo código con diferentes entradas. La diferencia es que las entradas de los parámetros formales son valores, mientras que las entradas de los parámetros de tipo son tipos.

El código que utiliza genéricos tiene muchos beneficios sobre el código no genérico:

- Comprobaciones de tipos más estrictas en tiempo de compilación. Un compilador de Java aplica una comprobación de tipos estricta al código genérico y emite errores si el código viola la seguridad de tipos. Reparar errores en tiempo de compilación es más fácil que reparar errores en tiempo de ejecución, que pueden ser difíciles de encontrar.
- Eliminación de conversiones. El siguiente fragmento de código sin genéricos requiere conversión:

```
List list = new ArrayList();
list.add("hello");
String s = (String) list.get(0);
```

Cuando se reescribe para usar genéricos, el código no requiere conversión:

```
List<String> list = new ArrayList<String>();
list.add("hello");
String s = list.get(0);   // no cast
```

- Permitir a los programadores implementar algoritmos genéricos. Al utilizar genéricos, los programadores pueden implementar algoritmos genéricos que funcionan en colecciones de diferentes tipos, se pueden personalizar, son seguros en cuanto a tipos y más fáciles de leer.

Tipos genéricos

Una clase de caja sencilla

Un tipo *genérico* es una clase o interfaz genérica que se parametriza sobre tipos. La siguiente Clase Box se modificará para demostrar el concepto.

```
public class Box {
    private Object object;

    public void set(Object object) { this.object = object; }
    public Object get() { return object; }
}
```

Dado que sus métodos aceptan o devuelven un **Object**, usted es libre de pasar lo que quiera, siempre que no sea uno de los tipos primitivos. No hay forma de verificar, en tiempo de compilación,

cómo se utiliza la clase. Una parte del código puede colocar un **Integer** en el cuadro y esperar obtener objetos de tipo **Integer** de él, mientras que otra parte del código puede pasar por error un **String**, lo que da como resultado un error de ejecución.

Una versión genérica de la clase Box

Una *clase genérica* se define con el siguiente formato:

```
class name<T1, T2, ..., Tn> { /* ... */ }
```

La sección de parámetros de tipo, delimitada por corchetes angulares (<>), sigue al nombre de la clase. Especifica los parámetros de tipo (también llamados variables de tipo) T1, T2, ..., y Tn.

Para actualizar la clase Box para que use genéricos, crea una declaración de tipo genérico cambiando el código " public class Box" por " public class Box<T>". Esto introduce la variable de tipo, T, que se puede usar en cualquier lugar dentro de la clase.

Con este cambio la clase Box pasa a ser:

```
/**
 * Generic version of the Box class.
 * @param <T> the type of the value being boxed
 */
public class Box<T> {
    // T stands for "Type"
    private T t;

    public void set(T t) { this.t = t; }
    public T get() { return t; }
}
```

Como puede ver, todas las apariciones de **Object** se reemplazan por **T**. Una variable de tipo puede ser cualquier tipo no primitivo que especifique: cualquier tipo de clase, cualquier tipo de interfaz, cualquier tipo de matriz o incluso otra variable de tipo.

Esta misma técnica se puede aplicar para crear interfaces genéricas.

Convenciones de nomenclatura de parámetros de tipo

Por convención, los nombres de los parámetros de tipo son letras mayúsculas simples. Esto contrasta marcadamente con las convenciones de nombres de variables que ya conoces, y con razón: sin esta convención, sería difícil distinguir entre una variable de tipo y un nombre de clase o interfaz común.

Los nombres de parámetros de tipo más utilizados son:

- E - Elemento (ampliamente utilizado por el marco de colecciones de Java)
- K - Key
- N - Número
- T - Tipo
- V - Valor

- S, U, V, etc. - 2º, 3º, 4º tipo
- Verá estos nombres utilizados en toda la API de Java SE.

Invocación y creación de una instancia de un tipo genérico

Para hacer referencia a la clase genérica `Box` desde su código, debe realizar una invocación de tipo genérico, que la reemplaza **`T`** con algún valor concreto, como **Integer**:

```
Box<Integer> integerBox;
```

Puedes pensar en una invocación de tipo genérico como similar a una invocación de método ordinario, pero en lugar de pasar un argumento a un método, estás pasando un argumento de tipo (**Integer** en este caso) a la clase `Box` misma.

Terminología de parámetros de tipo y argumentos de tipo: muchos desarrolladores usan los términos "parámetro de tipo" y "argumento de tipo" indistintamente, pero estos términos no son lo mismo. Al codificar, se proporcionan argumentos de tipo para crear un tipo parametrizado. Por lo tanto, ***T*** *in* ***Foo<T>*** *es un parámetro de tipo y* ***String*** *in* ***Foo<String>*** *es un argumento de tipo. Esta sección observa esta definición al usar estos términos.*

Al igual que cualquier otra declaración de variable, este código no crea realmente un nuevo objeto **Box**. Simplemente declara que **`Box`** contendrá una referencia a **Integer** una "**caja de números enteros**", que es como **`Box<Integer>`** se lee.

Una invocación de un tipo genérico generalmente se conoce como un tipo parametrizado.

Para instanciar esta clase, utilice la palabra clave **`new`**, como de costumbre, pero coloque **`<Integer>`** entre el nombre de la clase y el paréntesis:

```
Box<Integer> integerBox = new Box<Integer>();
```

El diamante

En Java SE 7 y versiones posteriores, puede reemplazar los argumentos de tipo necesarios para invocar el constructor de una clase genérica con un conjunto vacío de argumentos de tipo (`<>`) siempre que el compilador pueda determinar o inferir los argumentos de tipo a partir del contexto. Este par de corchetes angulares, `<>`, se denomina informalmente diamante. Por ejemplo, puede crear una instancia de `Box<Integer>`con la siguiente declaración:

```
Box<Integer> integerBox = new Box<>();
```

Bibliografía: https://dev.java/learn/generics/intro/