Generics

# Introducción

El JDK introdujo muchas mejoras al lenguaje JAVA. Uno de estas nuevas características es Generics.

Los genéricos te permiten abstraerte sobre los tipos. El ejemplo más común son los contenedores, como los que se encuentran en la jerarquía de Collections de java.

Aquí una muestra del uso típico de ordenamiento

List myIntList = new LinkedList(); // 1

myIntList.add(new Integer(0)); // 2

Integer x = (Integer) myIntList.iterator().next(); // 3

El casteo en la línea 3 es un poco molesto. Típicamente, el programador sabe qué tipo de dato va a ser colocado en una lista en particular. Sin embargo, el casteo es esencial. El compilador puede solo garantizar que un “Object” será retornado por el método iterator().next(). Para asegurar la asignación a la variable de tipo Integer el casteo es requerido.

Por supuesto, el casteo no solo genera desorden, también genera la posibilidad de errores en tiempo de ejecución, ya que el programador puede cometer errores.

¿Qué pasaría si el programador pudiera expresar su intención y marcar la lista para restringirla para contener un tipo particular de elementos? Esta es la principal idea de tras de los genéricos. Aquí está una versión del programa anterior, usando genéricos.

List<Integer> myIntList = new LinkedList<Integer>(); // 1

myIntList.add(new Integer(0)); // 2

Integer x = myIntList.iterator().next(); // 3

Nota la declaración del tipo de variable de myIntList. Esta especifica que no es una List arbitraria, pero una lista de Integer se escribe List<Integer>. Decimos que esta List es una interface genérica que toma un tipo de dato como parámetro; en este caso Integer. También especificamos el tipo de parámetro cuando creamos este objeto.

Nota también que el casteo de la línea 3 se fue.

Ahora, podrás pensar que se alborotó más el asunto. En lugar de hacer el casteo a Integer en la línea 3, tenemos a Integer como parámetro en la línea 1. Sin embargo, hay una gran diferencia aquí. El compilador ahora puede checar el tipo correcto del programa en tiempo de compilación. Cuando decimos que myIntList es declarada de tipo List<Integer>, esto nos dice algo acerca del tipo de la variable, que será cierto siempre que sea usado. En contraste, el casteo nos dice que puede ser cierto en un solo punto del programa.

Con esto se logra facilitar la lectura del código y aseguramos que sea un programa robusto.

# Definir Genéricos Simples

Aquí hay un pequeño extracto de las definiciones de las interfaces java.util.List y java.util.Iterator:

public interface List <E> {

void add(E x);

Iterator<E> iterator();

}

public interface Iterator<E> {

E next();

boolean hasNext();

}

Este código puede ser familiar, excepto por la cosa esa en los símbolos “<>”. Estas son las declaraciones de *Parámetros de Tipo Formal* de las interfaces java.util.List y java.util.Iterator.

Los parámetros de tipo se pueden utilizar a lo largo de la declaración de los genéricos, mejor que donde declaras tipos ordinarios, aunque hay importantes restricciones que veremos más adelante.

En la introducción vimos invocaciones a las declaraciones de tipo genérico, List<Integer>. En la invocación (usualmente llamada de tipo parametrizado), todas las ocurrencias del parámetro de tipo formal (la letra “E” en este caso) son reemplazadas por el tipo del argumento actual (en este caso Integer).

Puedes imaginar que List<Integer> es una versión de List, donde “E” es reemplazada de manera uniforme por Integer, como el siguiente ejemplo lo muestra:

List<Integer> myIntList = new LinkedList<Integer>(); // 1

myIntList.add(new Integer(0)); // 2

Integer x = myIntList.iterator().next(); // 3

Esta intuición es servicial, pero también es engañosa.

Es de ayuda, porque el tipo parametrizado List<Integer> de hecho tiene métodos como los de esta expresión.

Es engañosa porque la declaración de un genérico nunca se amplía en este camino. No se crean diferentes copias del código para cada uno de los tipos, ni en memoria, ni en el código, ni en el binario, ni en el disco. Una clase genérica solo se compila una vez y en un único archivo class, ya sea una clase o una interface.

Los parámetros de tipo son análogos a los parámetros comunes usados en los métodos o constructores. Al igual que un método tiene parámetros de valores formales que describen el tipo de valores que operan en el, una declaración genérica tiene parámetros de tipo formal. Cuando un método es invocado, los argumentos actuales son sustituidos por los parámetros formales, y el cuerpo del método es evaluado. Cuando una declaración genérica es invocada, los tipos actuales de argumentos son sustituidos por los parámetros de tipo formal.