Java Generics

In this post we will see what is Java Generics and why were they introduced in Java 5.

Before we begin discussing about Generics let us first take a dive in understanding what java code looks like without Generics.

Let us write a code that add all the numbers in List (Java 5 or greater version)

List<Integer> numbers = **new** ArrayList<Integer>();

**for** (**int** i = 1; i <= 5; i++) {

numbers.add(i);

}

int sum = 0;

**for** (Integer e : numbers) {

sum = sum + e;

}

System.***out***.println(sum);

What does the above code do? Well, its fills the lists with elements of “int” datatype. Then an enhanced for loop iterates the list and adds every element in list. Then they are printed. Yes it does that. But there is one more thing.

When the elements are added to the List, it seems **numbers.add(i);** adds primitive type data to the list. But actually it does not. Underneath it does something like this **numbers.add(new Integer(i));** Concept of automatically wrapping the primitive type(int) to wrapper type(Integer) is called Autoboxing and that is exactly what is happening behind the scenes.

The enhanced for loop uses Iterator<E> beneath it. Now when we iterate it adds the primitive type “sum” to the Integer object. The addition of numbers is done something like this by compiler

**sum = sum + e.intValue();**

As we see the Autoboxing is done automatically by compiler. But this was not the situation prior to Java 5.

Now let us write this same code when Generics were not introduced in Java.

List list = **new** ArrayList();

list.add(**new** Integer(1));

list.add(**new** Integer(2));

list.add(**new** Integer(3));

list.add(**new** Integer(4));

list.add(**new** Integer(5));

**int** sum = 0;

**for** (Iterator it = list.iterator(); it.hasNext();) {

**int** temp = ((Integer) it.next()).intValue();

sum = sum + temp;

}

System.***out***.println(sum);

So looking at the code you now know that previous code was much better as it was easily understood and not as verbose as this piece of code.

Also look at the iteration.

**int** temp = ((Integer) it.next()).intValue();

it.next() – Get the next value.

(Integer) it.next() – Cast it to Integer.

((Integer) it.next()).intValue() – gives the “int” type value i.e. primitive.

That been said Autoboxing was not provided prior to Java 5. So these kinds of statements are to be written every time in iteration and there can be problem of *ClassCastException*.

List list = **new** ArrayList();

list.add(**new** Integer(1));

list.add(**new** Integer(2));

list.add(**new** Integer(3));

list.add(**new** Integer(4));

list.add("String comes in.");//String in list.

**int** sum = 0;

**for** (Iterator it = list.iterator(); it.hasNext();) {

**int** temp = ((Integer) it.next()).intValue();

sum = sum + temp;

}

System.***out***.println(sum);

You might argue that what String is doing in Integer list. Well it is not exactly Integer list. It is List of Object and hence the adding String or any other type in this List is perfectly valid. But this code gives ClassCastException when you run this piece of code. The reason is that you cannot add Integer to String. That is not valid. So it can be full of surprises at the run time.

So Generics are used to make sure that compile will guarantee that List will not accept anything else than what is specified in <> angular brackets.

So Generics improves readability of code and robustness.