Generics Methods and Varargs

This is third post in Java Generics tutorials. In previous post we discussed Introduction to Java Generics and cast-iron guarantee provided in Generics.

In this post we will see how to write Generic Methods and Varargs.

Generic Method

Let us write a generic method that will accept the array of any type and return the list for that type.

**public** **static** <T> List<T> toList(T[] arr) {

List<T> list = **new** ArrayList<T>();

**for** (T t : arr) {

list.add(t);

}

**return** list;

}

<T> means a type parameter. It is passed as the method argument and appears in return type too. Also we use as type resolving.

You can call this method by using different types.

String[] str = { "Monday", "Tuesday", "Wednesday", "Thursday" };

List<String> list = ArrayToListDemo.*toList*(str);

Integer[] ints = { 1, 2, 3, 4 };

List<Integer> intsList = ArrayToListDemo.*toList*(ints);

Now if you see the parameter for the method toList(T[] arr) it is an array of type T. Now if we want to pass one or two parameters then we need to write like this

List<Integer> intsList = ArrayToListDemo.*toList*(**new** Integer[] { 1, 2 });

List<String> strlist = ArrayToListDemo.*toList*(**new** String[] { "Monday" });

This is a clumsy process.

There is another way to pass variable number of arguments to the method. It is called Varargs. It has a special syntax which is more convenient to use. It must be used only as last parameter to any method.

**public** **static** <T> List<T> toList(**T...** arr) {

List<T> list = **new** ArrayList<T>();

**for** (T t : arr) {

list.add(t);

}

**return** list;

}

See the method parameter toList(**T...** arr). We can call this method like this

List<Integer> intsList = ArrayToListDemo.*toList*(1, 2);

List<String> strlist = ArrayToListDemo.*toList*("Monday", "Tuesday");

At runtime, the Varargs are filled in array and then passed to method. So using Varargs can be used to reduce some clutter in code.