Theory: Generic methods

19 minutes 0 / 5 problems solved

In previous topics we have discussed generic classes and how one can use them. But Java also has generic methods that can be very useful. Type parameter in them can be an argument, that is passed to the method and is used in its logic. It can be a return type in it as well.

All methods can declare their own type parameters, regardless of the class they belong to. Moreover, the class may not even be generic, but its methods can be.

> Static methods cannot use type parameters of their class! Type parameters of the class these methods belong to can only be used in instance methods. If you want to use type parameters in a static method, declare this method's own type parameters.

Let's take a look at generic static and instance methods and find out how they are used.

§1. Generic static methods

The following static method is declared as generic. We remind you that it can belong to a generic or a non-generic class because it does not matter for generic methods.

```
1 public static <T> T doSomething(T t) {
2     return t;
3 }
```

The type parameter `T` can be used to declare the return type and the type of method's arguments. A generic method can take or return values of non-generic types as well. For instance, the following method takes a generic array and returns its size as `int`.

```
1 public static <E> int length(E[] array) {
2     return array.length;
3 }
```

A generic method's body is declared like that of any other method.

We can pass an array of integers to the method we defined earlier and find its length now:

```
1 Integer[] array = { 1, 2, 3, 4 };
2 int len = length(array); // pass an array of Integer's
```

Or pass a string to some other generic method:

```
1 String string = doSomething("str"); // pass a string
```

> Recall that the type parameter can represent only reference types, not primitive types.

As an example of another generic method, take a look at the following one that prints elements of a generic array.

Current topic:

Generic methods

Topic depends on:

× Generic programming

Topic is required for:

Type Bounds

Wildcards

Trees in Java

Hash table in Java

Table of contents:

```
public static <E> void print(E[] array) {
    for (int i = 0; i < array.length; i++) {
        System.out.print(array[i] + " ");
    }
    System.out.println();
}
```

Let's create an array and print it using this method.

```
Character[] characters = { 'a', 'b', 'c' };
print(characters);
```

As a result, you can see the following line:

```
a b c
```

In this example we used the `void` keyword for a declaration of the method because it does not return any variable.

Note, the type parameter section can (as a usual generic class or field) contain more than one type parameter separated by commas.

The following method, for example, declares two type parameters

```
public static <T, U> void method(T t, U u) {
    // do something
}
```

Static generic methods are often used to write generic algorithms that do not depend on the type they operate on. This can be convenient when the method has to be used independently from the class it belongs to. Generic static methods are widely used in different operations with arrays and collections, for example, sorting an array, searching a value in a collection, reversing an array and so on.

§2. Generic instance methods

Instance methods can have their own type parameters as well as static methods. There is no difference in their declaration compared to static methods, excluding the absence of the `static` keyword.

```
class SimpleClass {

    public <T> T getParameterizedObject(T t) {
        return t;
    }
}
```

The class above does not provide a type parameter, so we have to specify the type parameter in its declaration to make the method `getParameterizedObject` generic.

Note, that in this example we cannot use `T` as the type of class's field, because it belongs to the method rather than the class itself.

Now we can create an instance of the class and invoke the method passing a value. Then we can get the result of the same type without any castings.

```
SimpleClass instance = new SimpleClass();
Integer value = instance.getParameterizedObject(601);
```

Obviously, value will be equal to `601` here.

Note, generic instance methods declared outside a generic class is a rare case.

Normally, you will write and find generic static methods or instance methods that belong to a generic class.

In the next case a generic class has an instance method that adds its own type parameter:

```
class SimpleClass<T> {

    public <U> T getParameterizedObject(T t, U u) {
        return t;
    }
}
```

The method gets arguments of the class's type and method's own type. The method returns the variable of type `T` which is the class's type.

§3. Conclusion

Generic methods are a powerful tool in Java in general and generic programming in particular. They are used in some specific situations when spreading of a type parameter on the whole class is useless or when some variables have to be processed within a block. Such methods can have one or more type parameters that can be used either as passed arguments or as return types. Generic methods can be static or instance, and depending on this they have different usage in real programming.

Report a typo

144 users liked this theory. **4** didn't like it. **What about you?**

😍 🙂 😐 ☹️ 😡

Start practicing

Comments (4) Hints (0) Useful links (0) Show discussion