# How to make a Java Generic method static?

Asked 11 years, 6 months ago Modified 11 months ago Viewed 197k times

202

The following is a snippet on how to make a java generic class to append a single item to an array. How can I make appendToArray a static method. Adding static to the method signature results in compile errors.

```
public class ArrayUtils<E> {

        public E[] appendToArray(E[] array, E item) {
            E[] result = (E[])new Object[array.length+1];
            result[array.length] = item;
            return result;
        }
}
```

51

java generics

Share Edit Follow

asked Dec 10, 2010 at 13:05

Chris Johnson
**2,391** 2 17 17

What compile errors do you get? Also, why not just use one of the standard library containers? – Karl Knechtel Dec 10, 2010 at 13:08

1 Compile Error: I was actually adding the static modifier incorrect.. Using Collections: Yes using a collection would be ideal yet the question is not about collections vs array's, my use case requires an array. – Chris Johnson Dec 10, 2010 at 13:30

Note you'll need to use (EVIL) reflection to stop client code throwing an exception in some but not all circumstances (nice). It's best to avoid reference arrays. – Tom Hawtin - tackline Dec 10, 2010 at 14:26

## 5 Answers

Sorted by:
Reset to default

Date modified (newest first)

-1

From javadoc

Generic Methods

Generic methods are methods that introduce their own type parameters. This is similar to declaring a generic type, but the type parameter's scope is limited to the method where it is declared. Static and non-static generic methods are allowed, as well as generic class constructors.

The syntax for a generic method includes a list of type parameters, inside angle brackets, which appears before the method's return type. For static generic methods, the type parameter section must appear before the method's return type.

The Util class includes a generic method, compare, which compares two Pair objects:

```
public class Util {
    public static <K, V> boolean compare(Pair<K, V> p1, Pair<K, V> p2) {
        return p1.getKey().equals(p2.getKey()) &&
               p1.getValue().equals(p2.getValue());
    }
}


public class Pair<K, V> {

private K key;
private V value;

public Pair(K key, V value) {
    this.key = key;
    this.value = value;
}

public void setKey(K key) { this.key = key; }
public void setValue(V value) { this.value = value; }
public K getKey()   { return key; }
public V getValue() { return value; }
}
```

The complete syntax for invoking this method would be:

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.<Integer, String>compare(p1, p2);
```

The type has been explicitly provided, as shown in bold. Generally, this can be left out and the compiler will infer the type that is needed:

```
Pair<Integer, String> p1 = new Pair<>(1, "apple");
Pair<Integer, String> p2 = new Pair<>(2, "pear");
boolean same = Util.compare(p1, p2);
```

This feature, known as type inference, allows you to invoke a generic method as an ordinary method, without specifying a type between angle brackets.

After understanding this doc , for your question answer is that :

```
public static <I> I[] appendToArray(I[] array, I item)
```

Share Edit Follow

answered Jul 5, 2021 at 18:25

Doktor

77 1 5

I'll explain it in a simple way.

16

Generics defined at Class level are completely separate from the generics defined at the (static) method level.

```
class Greet<T> {

    public static <T> void sayHello(T obj) {
        System.out.println("Hello " + obj);
    }
}
```

When you see the above code anywhere, please note that the T defined at the class level has nothing to do with the T defined in the static method. The following code is also completely valid and equivalent to the above code.

```
class Greet<T> {

    public static <E> void sayHello(E obj) {
        System.out.println("Hello " + obj);
    }
}
```

**Why the static method needs to have its own generics separate from those of the Class?**

> This is because, the static method can be called without even instantiating the Class. So if the Class is not yet instantiated, we do not yet know what is T. This is the reason why the static methods needs to have its own generics.

So, whenever you are calling the static method,

```
Greet.sayHello("Bob");
Greet.sayHello(123);
```

JVM interprets it as the following.

```
Greet.<String>sayHello("Bob");
Greet.<Integer>sayHello(123);
```

Both giving the same outputs.

```
Hello Bob
Hello 123
```

Share Edit Follow

edited May 24, 2020 at 14:21

answered May 23, 2020 at 9:59

Vishnu Vivek


**1,359** 1 16 27

340

the only thing you can do is to change your signature to

```
public static <E> E[] appendToArray(E[] array, E item)
```

**Important details:**

Generic expressions preceding the return value always introduce (declare) a new generic type variable.

Additionally, type variables between types ( `ArrayUtils` ) and static methods ( `appendToArray` ) never interfere with each other.

So, what does this mean: In my answer `<E>` would hide the `E` from `ArrayUtils<E>` if the method wouldn't be `static` . AND `<E>` has nothing to do with the `E` from `ArrayUtils<E>` .

To reflect this fact better, a more correct answer would be:

```
public static <I> I[] appendToArray(I[] array, I item)
```

Share Edit Follow

edited Jul 10, 2015 at 20:58

Uyghur Lives Matter
**17.3k** 40 105 135

answered Dec 10, 2010 at 13:09

scheffield
**6,350** 2 28 31

31 Please also be aware that there is absolutely no relationship between the class-level type variable `E` and the static method type variable `E` . I consider it much better practice to use a different variable name when declaring generic methods, static or otherwise, inside generic classes. – Judge Mental Oct 4, 2013 at 3:34

but in this case I can pass one objects of different types into the params. Like I can pass Integer[] array as first param and Double item. – pinkpanther May 20, 2015 at 8:50

pinkpanther: True, but it doesn't do any harm, because the static method only ever operates on an array object that is passed to it via a parameter, so its elements are sure to have the correct type. – Dabbler Aug 22, 2016 at 19:19

86

```
public static <E> E[] appendToArray(E[] array, E item) { ...
```

Note the `<E>` .

Static generic methods need their own generic declaration ( `public static <E>` ) separate from the class's generic declaration ( `public class ArrayUtils<E>` ).

If the compiler complains about a type ambiguity in invoking a static generic method (again not likely in your case, but, generally speaking, just in case), here's how to explicitly invoke

a static generic method using a specific type ( `_class_.<_genericTypeParams_>_methodName_` ):

```
String[] newStrings = ArrayUtils.<String>appendToArray(strings, "another
string");
```

This would only happen if the compiler can't determine the generic type because, e.g. the generic type isn't related to the method arguments.

Share Edit Follow

edited Apr 15, 2013 at 15:30

answered Dec 10, 2010 at 13:16

Bert F

**82k** 11 103 123

How can I import that static method? – withoutOne Apr 22 at 10:00

11

You need to move type parameter to the method level to indicate that you have a generic method rather than generic class:

```
public class ArrayUtils {
    public static <T> E[] appendToArray(E[] array, E item) {
        E[] result = (E[])new Object[array.length+1];
        result[array.length] = item;
        return result;
    }
}
```

Share Edit Follow

answered Dec 10, 2010 at 13:09

axtavt

**234k** 40 499 475

1 This will not work because you have not defined the generic type E. In this case you still have to have generic type <E> in the class definition. – George Xavier Jun 20, 2019 at 20:44