

# What is Generics in Java?



## Generic Programming in Core JAVA



## Overview

Word 'GENERICs' was introduced in Java5 first time. Now in present time, it is one of the most powerful feature of programming languages. Generic programming can be enabled by the programmer to create various classes, interfaces and methods in which type of stored data is specified like a parameter. Generic feature provides facility to write an algorithm independent of any specific data type. Generics also gives type-safety. Meaning of type safety is to ensure that any operation will be performed on the right type of data before executing that operation.

Using the generic feature in programming, it is possible to create a single class or interface or method that can automatically works with all types of data-types (like: Integer, String, Float etc). It expands the ability to reuse the code safely and easily.

Before the Generics was not introduced, generalized classes, interfaces or methods were created using the references of type object because object is the super class of all classes in JAVA programming, but this way of programming could not ensure the type-safety.

In this article, we will learn about JAVA Generics, how to create own generics class and methods and its advantages with the help of simple examples. In Java, Generics enhanced the reusability of code.

## Syntax:


(to create an object of generic type)

```
<class_name> <data type> <reference> = new <class_name> <data type> ();  
OR  
<class_name> <data type> <reference> = new <class_name> <>();
```

## Basic need of Generics

Let's take a scenario where we create a list in Java for storing Integer values; here we have to write:

```
List my_list = new LinkedList();  
my_list.add(new Integer(1));  
Integer i = my_list.iterator().next();
```



Wait!, our compiler is complaining about the last line. It is not knowing that what type of data is returned. The compiler requires explicit casting here:

```
Integer i = (Integer) my_list.iterator.next();
```

There is not any contract that give guarantee that the return type of this list will be an integer. This defined list can hold any type of object. We can only know that we retrieve a list by inspecting the context. When we look at data types, it will be only guarantee that it is an object, therefore we requires an explicit cast to ensure that our type is safe.

This cast can annoy, we are knowing that the type in the list is integer. The cast will also clutter our code. It can be caused type related runtime-errors if a programmer makes a mistake with the time of explicit casting.

It will be very much easier that if any programmers can express their intention of using various specific types and the compiler can be ensured the correctness of these type. That is the basic or we can say core idea behind the generics in programming.

Here I modify the first line of the code snippet by below line:

```
List<Integer> my_list = new LinkedList<>();
```


When we add the diamond operator <> containing the type than we narrow the specialization of this list only to integer data type, we can specify the type that can be held inside the list. The compiler enforces the data type at the time of compilation.

In small programs (code-segments), this will seem as trivial addition, but in larger programs, this will add significant robustness and make the program easier to read.

## Generics Method in JAVA

**Note:** *The ArrayList class (this is a predefined class in JAVA under java.util package) can be used for storing data of any kind of data-type.*

**Generic methods** are the methods that can be written with only a single method declaration and user/programmer can be called this method with various different types of arguments. Here, the compiler ensures the correctness of type that is used.



Here, we have some following properties of generic methods:

- Generic methods can contain a type parameter (i.e. diamond operator by enclosing the type) before the return type of the any method declaration
- Type parameters can bound
- Generic methods can contain different type of parameters separated by commas (,) in the method signature
- Method body for a generic method will be just like a normal method

Here, we have example below to define a generic method for converting an array to list:

```
public <TT> List<TT> fromArrayToList(TT[] a)
{
    return Arrays.stream(a).collect(Collectors.toList());
}
```

In above example, the < TT > in the method signature tells that the method deals with generic type TT. This is needed always even if the method will return void.

As mentioned, the generic method can deal any number of generic type. If we require to modify the method to deal with 2 data types: type TT and type GG, it will be written as:

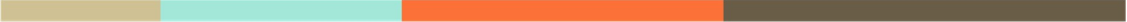
```
public static <TT , GG> List<GG> fromArrayToList(TT[] a, Function<TT,
                                                    GG> mapperFunction)
{
    return Arrays.stream(a).map(mapperFunction).collect(Collectors.toList());
}
```

## Example of Generics Method in JAVA

### Source Code

```
import java.util.ArrayList;

class GenericMethodEx
{
    public static void main(String[] args) {
```



```
// Here we are creating an array list for Integer data
ArrayList<Integer> listI = new ArrayList<>();
listI.add(4);
listI.add(5);
System.out.println("ArrayList for Integers: " + listI);

// Here we are creating an array list for data
ArrayList<String> listS = new ArrayList<>();
listS.add("Four");
listS.add("Five");
System.out.println("ArrayList for Strings: " + listS);

// Here we are creating an array list for Double data
ArrayList<Double> listD = new ArrayList<>();
listD.add(4.5);
listD.add(6.5);
System.out.println("ArrayList for Doubles: " + listD);
}
```

## Output

```
ArrayList for Integers: [4, 5]
ArrayList for Strings: [Four, Five]
ArrayList for Doubles: [4.5, 6.5]
```

## Explanation

In this example, we use the ArrayList predefined class for storing elements of type Integer, String, and Double. This will be possible because of Java Generics only.

Here, please notice that below line,

```
ArrayList<Integer> listI = new ArrayList<>();
```

We use 'Integer' in the angle brackets (<>). These angle brackets are known as the type parameter in JAVA-Generics. The type parameter can be used for specifying the type of data or we can say objects.



## Example of Generics Class in JAVA

Till here, we know how generic method can work in Java. Let us see how can be created own generics class in JAVA.

### Source Code

```
class GenericsClsTest<TT>
{
    // declare a variable of type 'TT'
    private TT data;

    public GenericsClass(TT data)
    {
        this.data = data;
    }

    // this method will return a variable type 'T'
    public TT getData()
    {
        return this.data;
    }
}

class GenericsClassEx
{
    public static void main(String[] args)
    {
        // Here I am initializing generic class for Integer data
        GenericsClass<Integer> ObjI = new GenericsClass<>(5);
        System.out.println("Generic Class returns: " + ObjI.getData());

        // Here I am initializing generic class for String data
        GenericsClass<String> ObjS = new GenericsClass<>("Java Programming");
        System.out.println("Generic Class returns: " + ObjS.getData());
    }
}
```



## Output

```
Generic Class returns: 5  
Generic Class returns: Java Programing
```

## Explanation

In this example, we create a Generic Class named GenericsClsTest. This class can use to work with various type of data.

```
class GenericsClsTest<TT>  
{  
    ...  
}
```

So, TT indicates the type parameter. In the Main class, we create objects of GenericsClsTest named ObjI and ObjS.

When we create ObjI, the type parameter TT is replaced by Integer data type. This means ObjI uses the GenericsClsTest to work with integer type of data.

And when we create ObjS, the type parameter TT is replaced by String data type. This means ObjS uses the GenericsClsTest to work with string type of data.