1. **Usage of Java Generics**
2. Implement Generic Interface to get the minimum and maximum number in the given array.

**Input:**

3, 6, 2, 8, 6

**Output:**

Minimum value: 2

Maximum value: 8

1. Write a generic method to count the number of elements in a collection that have a specific property (for example, odd integers, prime numbers, palindromes)

**Input:**

1, 2, 3, 4, 11, 17

**Output:**

Number of odd integers = 4

1. Write a generic Java program to find the first integer in a list that is relatively prime to a list of specified integers

**Input:**

3, 4, 6, 8, 11, 15, 28, 32

**Output**:

11 is relatively prime to 7 18 19 25

1. Write a Java generic program to fine palindrome of a number

**Input:**

32468

**Output:**

86423

**Reference links:**

**2.Generics**

1. https://www.geeksforgeeks.org/finding-minimum-and-maximum-element-of-a-collection-in-java/

2. <http://www-inf.it-sudparis.eu/cours/java/javatutorial/java/generics/QandE/generics-answers.html>

3. <http://www-inf.it-sudparis.eu/cours/java/javatutorial/java/generics/QandE/generics-answers.html>

4. <https://github.com/sayannath/Collection-and-Generics/blob/master/src/CodeChef/Palindrome.java>

# Generics in Java

**Generics** means **parameterized types**. The idea is to allow type (Integer, String, … etc., and user-defined types) to be a parameter to methods, classes, and interfaces. Using Generics, it is possible to create classes that work with different data types. An entity such as class, interface, or method that operates on a parameterized type is a generic entity.

### Why Generics?

The **Object** is the superclass of all other classes, and Object reference can refer to any object. These features lack type safety. Generics add that type of safety feature.

### Types of Java Generics

**Generic Method:** Generic Java method takes a parameter and returns some value after performing a task. It is exactly like a normal function, however, a generic method has type parameters that are cited by actual type. This allows the generic method to be used in a more general way. The compiler takes care of the type of safety which enables programmers to code easily since they do not have to perform long, individual type castings.

|  |
| --- |
| // Java program to show working of user defined  // Generic functions    class Test {  // A Generic method example  static <T> void genericDisplay(T element)  {  System.out.println(element.getClass().getName()  + " = " + element);  }    // Driver method  public static void main(String[] args)  {  // Calling generic method with Integer argument  genericDisplay(11);    // Calling generic method with String argument  genericDisplay("GeeksForGeeks");    // Calling generic method with double argument  genericDisplay(1.0);  }  } |

**Generic Classes:** A generic class is implemented exactly like a non-generic class. The only difference is that it contains a type parameter section. There can be more than one type of parameter, separated by a comma. The classes, which accept one or more parameters, are known as parameterized classes or parameterized types.

// Java program to show working of user defined

// Generic classes

// We use < > to specify Parameter type

class Test<T> {

// An object of type T is declared

T obj;

Test(T obj) { this.obj = obj; } // constructor

public T getObject() { return this.obj; }

}

// Driver class to test above

class Main {

public static void main(String[] args)

{

// instance of Integer type

Test<Integer> iObj = new Test<Integer>(15);

System.out.println(iObj.getObject());

// instance of String type

Test<String> sObj

= new Test<String>("GeeksForGeeks");

System.out.println(sObj.getObject());

}

}

# Generics in Java

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects. It makes the code stable by detecting the bugs at compile time.

Before generics, we can store any type of objects in the collection, i.e., non-generic. Now generics force the java programmer to store a specific type of objects.

## Advantage of Java Generics

There are mainly 3 advantages of generics. They are as follows:

**1) Type-safety:** We can hold only a single type of objects in generics. It doesn?t allow to store other objects.

1. List list = new ArrayList();

2. list.add(10);

1. list.add("10");
2. With Generics, it is required to specify the type of object we need to store.
3. List<Integer> list = new ArrayList<Integer>();
4. list.add(10);
5. list.add("10");// compile-time error

**2) Type casting is not required:** There is no need to typecast the object.

1. List list = new ArrayList();

2. list.add("hello");

3. String s = (String) list.get(0);//typecasting

4. After Generics, we don't need to typecast the object.

5. List<String> list = new ArrayList<String>();

6. list.add("hello");

7. String s = list.get(0);

**3) Compile-Time Checking:** It is checked at compile time so problem will not occur at runtime. The good programming strategy says it is far better to handle the problem at compile time than runtime.

* List<String> list = new ArrayList<String>();
* list.add("hello");
* list.add(32);//Compile Time Error

## Full Example of Generics in Java

1. **import** java.util.\*;
2. **class** TestGenerics1{
3. **public** **static** **void** main(String args[]){
4. ArrayList<String> list=**new** ArrayList<String>();
5. list.add("rahul");
6. list.add("jai");
7. //list.add(32);//compile time error
8. String s=list.get(1);//type casting is not required
9. System.out.println("element is: "+s);
11. Iterator<String> itr=list.iterator();
12. **while**(itr.hasNext()){
13. System.out.println(itr.next());
14. }
15. }
16. }
17. import java.util.\*;
18. class TestGenerics1{
19. public static void main(String args[]){
20. ArrayList<String> list=new ArrayList<String>();
21. list.add("rahul");
22. list.add("jai");
23. //list.add(32);//compile time error
24. String s=list.get(1);//type casting is not required
25. System.out.println("element is: "+s);
26. Iterator<String> itr=list.iterator();
27. while(itr.hasNext()){
28. System.out.println(itr.next());
29. }
30. }
31. }

## Example of Java Generics using Map

Now we are going to use map elements using generics. Here, we need to pass key and value. Let us understand it by a simple example:

1 import java.util.\*;

2 class TestGenerics2{

3 public static void main(String args[]){

4 Map<Integer,String> map=new HashMap<Integer,String>();

5 map.put(1,"vijay");

6 map.put(4,"umesh");

7 map.put(2,"ankit");

8 //Now use Map.Entry for Set and Iterator

9 Set<Map.Entry<Integer,String>> set=map.entrySet();

10 Iterator<Map.Entry<Integer,String>> itr=set.iterator();

11 while(itr.hasNext()){

12 Map.Entry e=itr.next();//no need to typecast

13 System.out.println(e.getKey()+" "+e.getValue());

14 }

15

16 }}

### Type Parameters in Java Generics

The type parameters naming conventions are important to learn generics thoroughly. The common type parameters are as follows:

* T – Type
* E – Element
* K – Key
* N – Number
* V – Value

[Generics](https://www.geeksforgeeks.org/generics-in-java/) make a class, interface and, method, consider all (reference) types that are given dynamically as parameters. This ensures type safety. Generic class parameters are specified in angle brackets “<>” after the class name as of the instance variable.

[Generic constructors](https://www.geeksforgeeks.org/constructor-getgenericparametertypes-method-in-java-with-examples/) are the same as generic methods. For generic constructors after the public keyword and before the class name the type parameter must be placed. Constructors can be invoked with any type of a parameter after defining a generic constructor. A constructor is a block of code that initializes the newly created object. It is an instance method with no return type. The name of the constructor is same as the class name. Constructors can be Generic, despite its class is not Generic.

[Generic Interfaces](https://www.geeksforgeeks.org/bounded-types-generics-java/)in Java are the interfaces that deal with abstract data types. Interface help in the independent manipulation of java collections from representation details. They are used to achieving multiple inheritance in java forming hierarchies. They differ from the java class. These include all abstract methods only, have static and final variables only. The only reference can be created to interface[,](https://www.geeksforgeeks.org/generics-in-java/) not objects, Unlike class, these don’t contain any constructors, instance variables. This involves the “[implements](https://www.geeksforgeeks.org/extends-vs-implements-in-java/)” keyword. These are similar to generic classes.

The benefits of Generic Interface are as follows:

1. This is implemented for different data types.
2. It allows putting constraints i.e. bounds on data types for which interface is implemented.

A [Collection](https://www.geeksforgeeks.org/collections-in-java-2/) is a group of individual objects represented as a single unit. Java provides [Collection Framework](https://www.geeksforgeeks.org/java-collection-tutorial/) which defines several classes and interfaces to represent a group of objects as a single unit. Finding minimum and maximum element of a Collection can be easily done using the Collections.min() and Collections.max() method. These are static method of Collections Class in Java.

