Generics in Java

The **Java Generics** programming is introduced in J2SE 5 to deal with type-safe objects.

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

it provides [compile time](http://javarevisited.blogspot.com/2012/03/what-is-static-and-dynamic-binding-in.html) type-safety and ensures that you only insert correct Type in collection and avoids ClassCastException in runtime.

Read more: <http://javarevisited.blogspot.com/2012/06/10-interview-questions-on-java-generics.html#ixzz4YDH4d3vU>

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.

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

Before Generics, we need to type cast.

1. List list = new ArrayList();
2. list.add("hello");
3. String s = (String) list.get(0);//typecasting

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

1. List<String> list = new ArrayList<String>();
2. list.add("hello");
3. 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.

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

**Syntax** to use generic collection

1. ClassOrInterface<Type>

**Example** to use Generics in java

1. ArrayList<String>

Full Example of Generics in Java

Here, we are using the ArrayList class, but you can use any collection class such as ArrayList, LinkedList, HashSet, TreeSet, HashMap, Comparator etc.

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
9. String s=list.get(1);//type casting is not required
10. System.out.println("element is: "+s);
12. Iterator<String> itr=list.iterator();
13. while(itr.hasNext()){
14. System.out.println(itr.next());
15. }
16. }
17. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestGenerics1)

Output:element is: jai  
 rahul  
 jai

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");
9. //Now use Map.Entry for Set and Iterator
10. Set<Map.Entry<Integer,String>> set=map.entrySet();
12. Iterator<Map.Entry<Integer,String>> itr=set.iterator();
13. while(itr.hasNext()){
14. Map.Entry e=itr.next();//no need to typecast
15. System.out.println(e.getKey()+" "+e.getValue());
16. }
18. }}

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestGenerics2)

Output:1 vijay  
 2 ankit   
 4 umesh

Generic class

A class that can refer to any type is known as generic class. Here, we are using **T** type parameter to create the generic class of specific type.

Let’s see the simple example to create and use the generic class.

**Creating generic class:**

1. class MyGen<T>{
2. T obj;
3. void add(T obj){this.obj=obj;}
4. T get(){return obj;}
5. }

The T type indicates that it can refer to any type (like String, Integer, Employee etc.). The type you specify for the class, will be used to store and retrieve the data.

**Using generic class:**

Let’s see the code to use the generic class.

1. class TestGenerics3{
2. public static void main(String args[]){
3. MyGen<Integer> m=new MyGen<Integer>();
4. m.add(2);
5. //m.add("vivek");//Compile time error
6. System.out.println(m.get());
7. }}

Output:2

Type Parameters

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

1. T - Type
2. E - Element
3. K - Key
4. N - Number
5. V - Value

Generic Method

Like generic class, we can create generic method that can accept any type of argument.

Let’s see a simple example of java generic method to print array elements. We are using here **E** to denote the element.

1. public class TestGenerics4{
3. public static < E > void printArray(E[] elements) {
4. for ( E element : elements){
5. System.out.println(element );
6. }
7. System.out.println();
8. }
9. public static void main( String args[] ) {
10. Integer[] intArray = { 10, 20, 30, 40, 50 };
11. Character[] charArray = { 'J', 'A', 'V', 'A', 'T','P','O','I','N','T' };
13. System.out.println( "Printing Integer Array" );
14. printArray( intArray );
16. System.out.println( "Printing Character Array" );
17. printArray( charArray );
18. }
19. }

[Test it Now](http://www.javatpoint.com/opr/test.jsp?filename=TestGenerics4)

Output:Printing Integer Array  
 10  
 20  
 30  
 40  
 50  
 Printing Character Array  
 J  
 A  
 V  
 A  
 T  
 P  
 O  
 I  
 N  
 T

Wildcard in Java Generics

The ? (question mark) symbol represents wildcard element. It means any type. If we write <? extends Number>, it means any child class of Number e.g. Integer, Float, double etc. Now we can call the method of Number class through any child class object.

Let's understand it by the example given below:

1. import java.util.\*;
2. abstract class Shape{
3. abstract void draw();
4. }
5. class Rectangle extends Shape{
6. void draw(){System.out.println("drawing rectangle");}
7. }
8. class Circle extends Shape{
9. void draw(){System.out.println("drawing circle");}
10. }

13. class GenericTest{
14. //creating a method that accepts only child class of Shape
15. public static void drawShapes(List<? extends Shape> lists){
16. for(Shape s:lists){
17. s.draw();//calling method of Shape class by child class instance
18. }
19. }
20. public static void main(String args[]){
21. List<Rectangle> list1=new ArrayList<Rectangle>();
22. list1.add(new Rectangle());
24. List<Circle> list2=new ArrayList<Circle>();
25. list2.add(new Circle());
26. list2.add(new Circle());
28. drawShapes(list1);
29. drawShapes(list2);
30. }}

drawing rectangle  
drawing circle  
drawing circle

**Can we use Generics with Array?**

This was probably most simple generics interview question in Java, if you know the fact that Array doesn't support Generics and that's why Joshua Bloch suggested in Effective Java to prefer List over Array because *List can provide compile time type-safety* over Array.

Read more: <http://javarevisited.blogspot.com/2012/06/10-interview-questions-on-java-generics.html#ixzz4YDKNYFMH>

Code that uses generics has many benefits over non-generic code:

**1) Stronger type checks at compile time:** A Java compiler applies strong type checking to generic code and issues errors if the code violates type safety. Fixing compile-time errors is easier than fixing runtime errors, which can be difficult to find.

**2) Elimination of casts:** If you use generics, then explicit type casting is not required.

**3) Enabling programmers to implement generic algorithms:** By using generics, programmers can implement generic algorithms that work on collections of different types, can be customized, and are type safe and easier to read.

- See more at: <http://www.java2novice.com/java_interview_questions/why-generics/#sthash.8NcanNt8.dpuf>