**Generics**

JDK 5.0 introduces several new extensions to the Java programming language. One of these is the introduction of generics.

Generics was added in Java 5 to provide **compile-time type checking** and removing risk of ClassCastException that was common while working with collection classes.

The whole collection framework was re-written to use generics for type-safety.

Example:

Before java 5:

|  |  |
| --- | --- |
| List list = new ArrayList();  list.add("abc");  list.add(new Integer(5)); //OK  for(Object obj : list){  //type casting leading to ClassCastException at runtime  String str=(String) obj;  } | This code compiles fine but throws ClassCastException at runtime because we are trying to cast Object in the list to String whereas one of the element is of type Integer.  **Hint:: instanceof** |

Java 5

|  |  |
| --- | --- |
| List<String> list1 = new ArrayList<String>();  // java 7 ? List<String> list1 = new ArrayList<>();  list1.add("abc");  //list1.add(new Integer(5)); //compiler error  for(String str : list1){  //no type casting needed, avoids ClassCastException  } | In this case at the time of declaration of collection/List we are specify the **Type**  not allow other type to add  //So While using not required type casting. |

List<Integer> list = new ArrayList<Integer>();

list.add(1000);     //works fine

list.add("lokesh"); //compile time error;

When you write above code and compile it, you will get below error: “*The method add(Integer) in the type*List<Integer>*is not applicable for the arguments (String)*“. Compiler warned you. This exactly is generics sole purpose i.e. Type Safety.

[**type erasure**](https://en.wikipedia.org/wiki/Type_erasure):

It essentially means that all the extra information added using generics into sourcecode will be removed from byte code generated from it. Inside bytecode, it will be old java syntax which you will get if you don’t use generics at all. This necessarily helps in generating and executing code written prior java 5 when generics were not added in language.

The code

List<Integer> list = new ArrayList<Integer>();

 list.add(1000);     //works fine

If you compare the bytecode of above example with/without generics, then there will not be any difference. Clearly compiler removed all generics information. So, above code is very much similar to below code without generics.

List list = new ArrayList();

 list.add(1000);

**What does the following code fragment print?**

List <String> l1 = new ArrayList<String>();

List<Integer> l2 = new ArrayList<Integer>();

System.out.println(l1.getClass() == l2.getClass());

You might be tempted to say false, but you'd be wrong. It prints true, because all instances of a generic class have the same run-time class, regardless of their actual type parameters.