**GENERICS**

**Why Generics?**

“In layman’s term, generics force type safety in java language.”

Without this type safety, your code could have infected by various bugs which get revealed only in runtime. Using generics, makes them highlighted in compile time itself and make you code robust even before you get the bytecode of your java source code files.

“Generics add stability to your code by making more of your bugs detectable at compile time.”

So now we have a fair idea why generics are present in java in the first place. Next step is get some knowledge about how they work in java. What actually happen when you use generics in your source code.

**How Generics works in java?**

* **Type Safety:-** What exactly is type safety? It’s just a guarantee by compiler that if correct Types are used in correct places then there should not be any ClassCastException in runtime. A usecase can be list of Integer i.e. List<Integer>. If you declare a list in java like List<Integer>, then java guarantees that it will detect and report you any attempt to insert any non-integer type into above list.
* Another important term in java generics is “[**type erasure**](https://en.wikipedia.org/wiki/Type_erasure)“. It essentially means that **all the extra information added using generics into source code will be removed from bytecode 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.
* Eg of type erasure.

class Geeks<T extends String> {

 // Here, T will be replaced by String i.e. java.lang.String

    T str;

    Geeks(T o)

    {

        str = o;

    }

    T getob()

    {

        return str;

    }

}

Above code will be converted to below by removing extra code

class Geeks

{

//Here, T will be replaced by String i.e. java.lang.String

String str;

Geeks(String o)

{

str=o;

}

String getob()

{

return str;

}

}

**Advantages of Generics:**

**Programs that uses Generics has got many benefits over non-generic code:**

1. **Code Reuse:** We can write a method/class/interface once and use for any type we want.

.

1. Type Safety : Generics make errors to appear compile time than at run time (It’s always better to know problems in your code at compile time rather than making your code fail at run time). Suppose you want to create an ArrayList that store name of students and if by mistake programmer adds an integer object instead of string, compiler allows it. But, when we retrieve this data from ArrayList, it causes problems at runtime.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // A Simple Java program to demonstrate that NOT using  // generics can cause run time exceptions  import java.util.\*;    class Test  {      public static void main(String[] args)      {          // Creatinga an ArrayList without any type specified          ArrayList al = new ArrayList();            al.add("Sachin");          al.add("Rahul");          al.add(10); // Compiler allows this            String s1 = (String)al.get(0);          String s2 = (String)al.get(1);            // Causes Runtime Exception          String s3 = (String)al.get(2);      }  } |

Output :

Exception in thread "main" java.lang.ClassCastException:

java.lang.Integer cannot be cast to java.lang.String

at Test.main(Test.java:19)

**How generics solve this problem?**  
At the time of defining ArrayList, we can specify that this list can take only String objects.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // Using generics converts run time exceptions into  // compile time exception.  import java.util.\*;    class Test  {      public static void main(String[] args)      {          // Creating a an ArrayList with String specified          ArrayList <String> al = new ArrayList<String> ();            al.add("Sachin");          al.add("Rahul");            // Now Compiler doesn't allow this          al.add(10);            String s1 = (String)al.get(0);          String s2 = (String)al.get(1);          String s3 = (String)al.get(2);      }  } |

Output:

15: error: no suitable method found for add(int)

al.add(10);

^

.

1. Individual Type Casting is not needed: If we do not use generics, then, in the above example every-time we retrieve data from ArrayList, we have to typecast it. Typecasting at every retrieval operation is a big headache. If we already know that our list only holds string data then we need not to typecast it every time.

filter\_none

edit

play\_arrow

brightness\_4

|  |
| --- |
| // We don't need to typecast individual members of ArrayList  import java.util.\*;    class Test  {      public static void main(String[] args)      {          // Creating a an ArrayList with String specified          ArrayList <String> al = new ArrayList<String> ();            al.add("Sachin");          al.add("Rahul");            // Typecasting is not needed          String s1 = al.get(0);          String s2 = al.get(1);      }  } |

.

1. Implementing generic algorithms: By using generics, we can implement algorithms that work on different types of objects and at the same they are type safe too.

**Generics with wildcards:**