

Def : The main objective of Generics is to provide Type-Safety and to resolve Type-Casting problems.

Case 1: Type-Safety

Arrays are always type safe that is we can give the guarantee for the type of elements present inside array.

For example if our programming requirement is to hold String type of objects it is recommended to use String array.
In the case of string array we can add only string type of objects by mistake if we are trying to add any other type we will get compile time error.

eg:

```
String name[] = new String[500];
name[0] = "Navin Reddy";
name[1] = "Haider";
name[2] = new Integer(100); //CE: incompatible types found: java.lang.Integer
                           required: java.lang.String
```



That is we can always provide guarantee for the type of elements present inside array and hence arrays are safe to use with respect to type that is arrays are type safe.

But collections are not type safe that is we can't provide any guarantee for the type of elements present inside collection.



4

```
Editor - (D:\TestJava)\ - 0 X
File Edit View Search Document Project Tools Browser Emmet Window Help
D:\ New Volume Directory Object Functions
1 D: RECYCLE.BIN 2 C:\ CorbaApache 3 Enterprise 4 Frontend 5 Front end videos 6 GitHub 7 Gitment 8 GitHub 9 IntelliJ
1 public class Test { 2 public static void main(String[] args){ 3 // I need to hold only String type of data 4 // which one to choose Arrays or Generics 5 //Arrays =====> directly deals the data at memory level 6 //Generics =====> deals the data through API code 7 8 9 10 11 12 13 } 14
```

```
File Edit View  
16-12-2022_garment_Cast_Codes - Notepad  
  
String name1 = (String)al.get(0);  
String name2 = (String)al.get(1);  
String name3 = (String)al.get(2);//Exception in thread "main" :: java.lang.ClassCastException  
java.lang.Integer cannot be cast to java.lang.String  
;;;  
  
Hence we can't provide guarantee for the type of elements present inside collections that is collections are not safe to use with respect to type.  
  
Case 2: Type-Casting  
In the case of array at the time of retrieval it is not required to perform any type casting.  
eg::  
String name[] = new String[500];  
name[0] = "Navin Reddy";  
name[1] = "Haider";  
;;;  
;;;  
String data = name[0];//here type casting is not required.  
  
But in the case of collection at the time of retrieval compulsory we should perform type casting otherwise we will get compile time error.  
  
eg::
```

```
16.12.2022.generics-class-notes - Notepad  
File Edit View  
;;;;;  
String data =name[0];//here type casting is not required.
```

But in the case of collection at the time of retrieval compulsory we should perform type casting otherwise we will get compile time error.

```
eg:  
ArrayList al =new ArrayList();  
al.add("NavinReddy");  
al.add("Haider");  
String name1= al.get(0);//CE: incompatible types : found : java.lang.Object  
required: java.lang.String
```

6

String name1=(String) al.get(0);//At the time of retrieval type casting is mandatory

That is in collections type casting is bigger headache.

To overcome the above problems of collections(type-safety, type casting)sun people introduced generics concept in 1.5v hence the main objectives of generics are:

1. To provide type safety to the collections.
2. To resolve type casting problems.

Ln 77 Col 1
22°C
Mostly cloudy



100% Windows (C:\EF7) UTF-8

EN

IN

16.12.2022 20:30

```
public class Test {  
    public static void main(String[] args){  
        //TypeSafety becoz of non-generic version code  
        ArrayList<String> al =new ArrayList<String>();  
        al.add("sachin");  
        al.add("dhoni");  
  
        //TypeCasting is required  
        String name1 = al.get(0);  
        String name2 = al.get(0);  
    }  
}
```

File Edit View

To hold only string type of objects we can create a generic version of `ArrayList` as follows.

```
ArrayList<String> al =new ArrayList<String>();
al.add("NavinReddy");
al.add(10);//CE: can't find symbol
          symbol: method add(int)
location : class java.util.ArrayList<java.lang.String>
al.add(10)
```

For this `ArrayList` we can add only `String` type of objects by mistake if we are trying to add any other type we will get compile time error that is through Generics we are getting type safety.

At the time of retrieval it is not required to perform any type casting we can assign elements directly to `String` type variables.

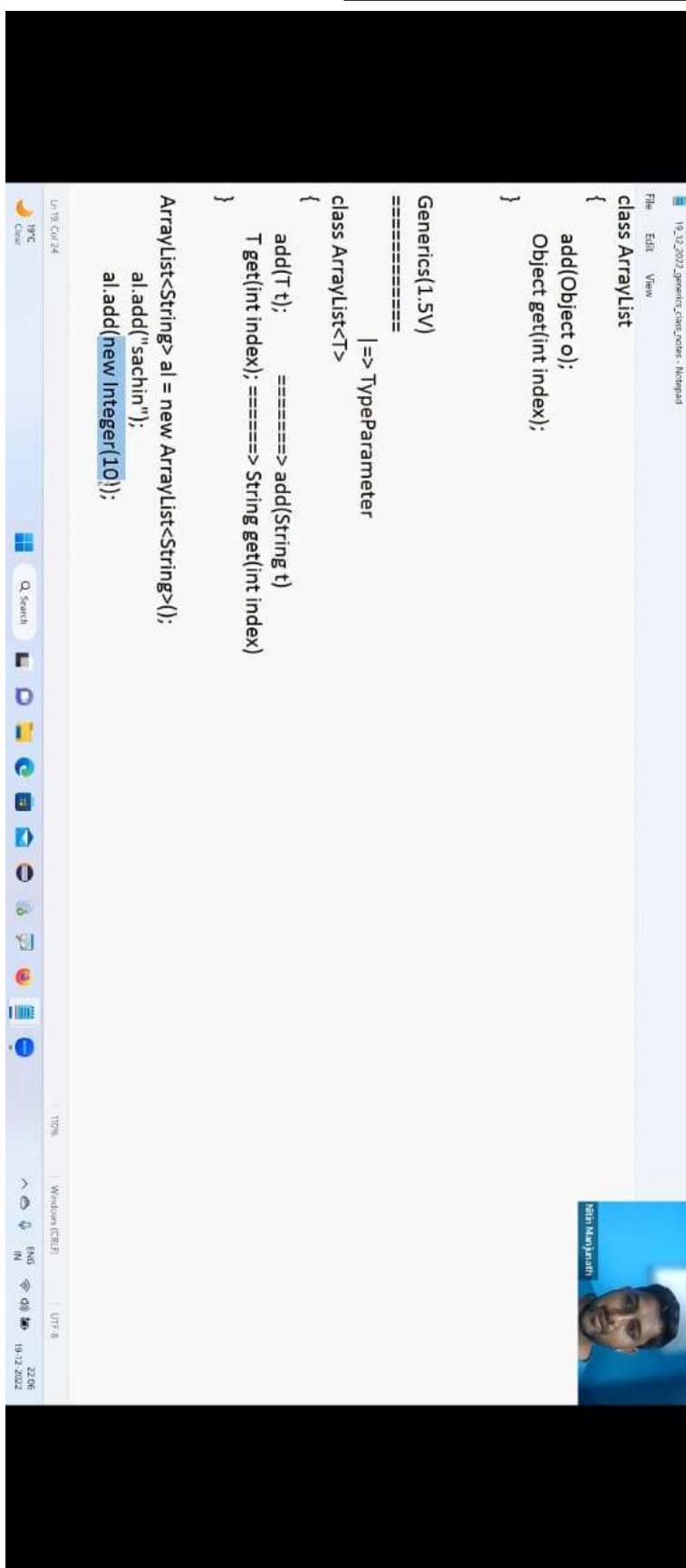
```
ArrayList<String> al = new ArrayList<String>();
al.add("NavinReddy");
.....
....;
String name = al.get(0); // type casting is not required as it is an TypeSafe
That is through generic syntax we can resolve type casting problems.
```



That is through generic syntax we can resolve type casting problems.

```
import java.util.*;  
  
public class Test {  
    public static void main(String[] args){  
        //<String> =====> TypeParameter  
        //List =====> BaseParameter  
  
        List<String> l1 =new ArrayList<String>();//valid  
        Collection<String> l2 = new ArrayList<String>();//valid  
        ArrayList<Object> l3 = new ArrayList<String>();//invalid  
    }  
}
```

Generics part1



```
19-12-2022 Generics Class Notes - Nitin Manjrekar
File Edit View
class ArrayList
{
    add(Object o);
    Object get(int index);
}

Generics(1.5V)
=====
|=> TypeParameter
class ArrayList<T>
{
    add(T t);      =====> add(String t)
    T get(int index); =====> String get(int index)
}

ArrayList<String> al = new ArrayList<String>();
al.add("sachin");
al.add(new Integer(10));
```

The screenshot shows a Java code editor window. The code defines a class `ArrayList` with a type parameter `T`. It contains two methods: `add` which takes an `Object` and returns nothing, and `get` which takes an `int` index and returns an `Object`. Below this, an `ArrayList` of `String` is created and populated with the values "sachin" and a new `Integer(10)`.

The status bar at the bottom of the editor shows the following information:

- Line 19, Col 24
- 19°C
- Q Search
- W D E S A P C
- 110%
- Windows (CRLF)
- UTF-8
- ENG
- IN
- 22:06
- 19-12-2022