Agenda:
1. Introduction
2. Type-Safety
3. Type-Casting
4. Generic Classes
5. Bounded Types
6. Generic methods and wild card character(?)
7. Communication with non generic code
8. Conclusions

Deff : 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: incompatbile 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.

For example if our programming requirement is to hold only string type of objects it is never recommended to go for ArrayList.
By mistake if we are trying to add any other type we won't get any compile time error but the program may fail at runtime.

eg:
```
   ArrayList al =new ArrayList();
      al.add("NavinReddy");
      al.add("Haider");
      al.add(new Integer(10));
            ;;;
            ;;;
            ;;;
      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::
```
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

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

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.

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.

eg:
```
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.

Conclusions
=========
1. Polymorphism concept is applicable only for the base type but not for parameter type
[usage of parent reference to hold child object is called polymorphism].

```
eg: ArrayList<String> al =new ArrayList<String>();
       List<String> al =new ArrayList<String>();
       Collection<String> al =new ArrayList<String>();
       Collection<Object> al =new ArrayList<String>();//CE: incompatible types
```

2.
Collections concept applicable only for objects , Hence for the parameter type we can use any class or interface name but not primitive value(type).Otherwise we will get compile time error.

```
            eg: ArrayList<int> al =new ArrayList<int>();//CE: unexcpected type

      found:primitive
                                                                        required:
reference
```