Arrays ::TypeSafety

```
String[] names = new String[500];
names[0] = "sachin";
names[1] = "dhoni";
names[2] = new Integer(10);
```
Incompatible types
CE

STL -> Standard Template Library (C++)

Collections :No TypeSafety

```
ArrayList al = new ArrayList();
    al.add("sachin");
    al.add("dhoni");
    al.add( new Integer(10));
    ;;;;;;
    ;;;;;;
    ;;;;;;
    ;;;;;;
    String name1 =(String) al.get(0);
    String name2 =(String) al.get(1);
    String name3 =(String) al.get(2);
```
ClassCastException
Runtime

Generics

Collection
- |-> Provide Type-Safety
- |-> Resolve problem of typecasting

```
ArrayList<String> al = new ArrayList<String>();
        al.add("sachin");
        al.add("dhoni");
        al.add(new Integer(10));
```
CE: incompatible types

Arrays : Homogenous,no problem while retrieving

```
String[] names = new String[500];
names[0] = "sachin";
names[1] = "dhoni";

String data = names[0];
```
Typecasting not requried

Collection :Typecasting is required

```
ArrayList al = new ArrayList();
    al.add("sachin");
    al.add("dhoni");

String name = al.get(0);
```
CE: incompatible types

Typecasting is required

```
ArrayList<String> al = new ArrayList<String>();
        al.add("sachin");
        al.add("dhoni");
        al.add("kohli");

String name = al.get(0);
```
"Homogenous"

"Typecasting is not required"

ParameterType

```
ArrayList <String>  al = new ArrayList<String>();
```
BaseType

Polymorphism

```
List<String> al      = new ArrayList<String>();
Collection<String> c = new ArrayList<String>();

ArrayList<Object>  al1 = new ArrayList<String>();
```
Polymorphism not applicable at parameterype

```
List<Integer> al = new ArrayList<Integer>();
    List<int> al = new ArrayList<int>();
```
CE: unexpected type

Collection before jdk1.4v

```
class ArrayList
{
    add(Object o);
    Object get(int index);
}
```
⇩
```
al.add("sachin");
String name = (String) al.get(0);
```

Generics(1.5V)

TypeParameter

```
class ArrayList<T>
{
    add(T t);
    T get(int index);
}
```
<T> -> String

```
add(String data)
String get(int index)
```

```
ArrayList<String> al =new ArrayList<String>();
        al.add("sachin");
        al.add(new Integer(10));
```
CE:suitable method not found

```
ArrayList<String> al =new ArrayList<String>();
        al.add("sachin");
        al.add("kohli");
        al.add("dhoni");

    String name = al.get(0);
```
Typecasting is not required

```
class Test<T>
{

}
```
```
Test<Integer> t1 = new Test<Integer>();
Test<String>  t2 = new Test<String>();
```
"Unbounded Types"

BoundedTypes

```
class Test<T extends X  >
{

}
```
1. if X is a class type, then type parameter can be of 'X' type or its 'child class'.
2. if X is a interface type, then type parameter can be of 'X' type or its 'implementation class'.

```
class Test<T  implements  X  >
{

}
```
we can't defined bounded types using implements and super keyword.

```
class Test<T  super  X  >
{

}
```

Bounded types with combination

```
class Test< T extends Number & Runnable>
{

}
```
T can be any type which extends Number class and implements Runnable interface

```
class Test< T extends Number & Runnable & Comparable>
{

}
```
T can be any type which extends Number class and implements Runnable and Comparable interface

```
class Test< T extends Number & String>  //Invalid(Multiple inheritance through classes is not supported in java )
{

}
```

```
class Test< T extends Runnable & Comparable >
{

}
```
T can be any type which implements Runnable & Comparable interface

```
class Test< T extends Runnable & Number > //Invalid
{
```
rule in java : first extends then implements
```
}
```

```
class Test<T>
{

}
```
TypeParameter

```
class Test<Nitin>
{
    "convention to name Type-Parameter is :: T"
}
```
TypeParameter (valid identifier)

```
class HashMap<K,V>
{

}
```
Type-Parameter (Multiple Type parameter)

```
class Test<T,U>
{

}
```
Type-Parameter (Multiple Type parameter)

```
ArrayList<String> l=new ArrayList<String>();
    m1(l);
    ;;;;;
    ;;;;;

ArrayList<Integer> l1=new ArrayList<Integer>();
    m1(l1);
    ;;;;;;
    ;;;;;;

ArrayList<Double> l2=new ArrayList<Double>();
    m1(l2);
    ;;;;;;
    ;;;;;;

ArrayList<Student> l3=new ArrayList<Student>();
    m1(l3);
```

```
public static void m1(ArrayList<String> al){
    ;;;;
    ;;;;
}
public static void m1(ArrayList<Integer> al){
    ;;;;
    ;;;;
}
public static void m1(ArrayList<Double> al){
    ;;;;
    ;;;;
}
public static void m1(ArrayList<Student> al){
    ;;;;
    ;;;;
}
```

Solution ::Generic method and Wild-Card character(?)

```
public static void m1(ArrayList<?> al){
    ;;;;;
    ;;;;;
}
```

Runtime objects

```
ArrayList<?> al =  new  ArrayList<String>
                   new  ArrayList<Integer>
                   new  ArrayList<Double>
                   new  ArrayList<Student>
```
Into this ArrayList, what type of data can we add?

al.add(null);

Runtime object

```
ArrayList<String>  al = new ArrayList<String>();
```
Into this ArrayList, what type of data can we add?

```
al.add("sachin");
al.add(null);
```

```
ArrayList<? extends X>  al
```
1. X is class, then X type or its child class
2. X is interface, then X type or its implementation class

Into this ArrayList, what type of data can we add?

al.add(null)

Runtime object

```
ArrayList<? extends Number> al  = new ArrayList<Integer>();
                                = new ArrayList<Number>();
                                = new ArrayList<String>();
```