**Generics java**

To provide type safety and to resolve type casting problems (runitme error)

Array is type safe. Collections not type safe

int[] arr= new int[5];

arr[0]= 3443; //compile error

In array, type casting not required

ArrayList l= new ArrayList();

l.add("sdd");

l.add(10); // no compile error but runtime .

String s= (String)l.get(1); //runtime error, Incompatible types (classCastException)

In collections, type casting required.

polymorphism: usage of parent ref to hold child object (arrayList, LinkedList, Vector -> Stack)

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

List l1= new Vector(); | Stack

Collection<String> l= new ArrayList<String>();

//Polymorphism concept applicable for base type

ArrayList: base type

String: parameter type

ArrayList<Object> l= new ArrayList<String>();//error, incompatible types

//Polymorphism concept not applicable for parameterized type.

Bounded types valid only for extends, not for super or implements keyword. Can define bounded

types in combination also

class Three\_Demo<T extends Number & Runnable>{}

But first should be class and then the interface

class Three\_Demo<T extends Runnable & Number>{} //Compile error.

class Three\_Demo<T extends Thread & Number>{} //Compile error. Cannot extend more than 1 class

class Three\_Demo<T implements Runnable>(){} // Compile error

class A<X>{} //Can use any valid java identifier instead of T, using T to denote Type

parameter.

Can declare any number of type parameters as per our reqt.

Class Test<A,B>{}

Class Test<A,B,C>{}

Class Test<K,V>{}

Wildcard character:

public void m1(ArrayList<?> list){}

public void m1(ArrayList<? extends X> list){}

public void m1(ArrayList<? super X> list){}

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

ArrayList<? extends Number> list= new ArrayList<Integer>();

ArrayList<? super String> list= new ArrayList<Object>();

ArrayList<?> list= new ArrayList<?>(); //Invalid

ArrayList<?> list= new ArrayList<? extends Number>(); //Invalid

Generic method:

class Test<T>

{

//we can use T within this class based on our reqt

}

//If we want type parameter at method level before return type:

class Test{

public <T> void m1(T ob){

//We can use T anywhere within this method based on our reqt.

}}

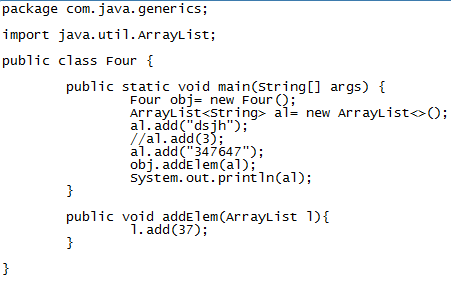
//We can define bounded type at method level too

public <T extends > void m1(T ob){

If we send generic object to non-generic area, starts behaving like non-generic object

Non generic to generic area: behaves like generic object i.e. location in which object is

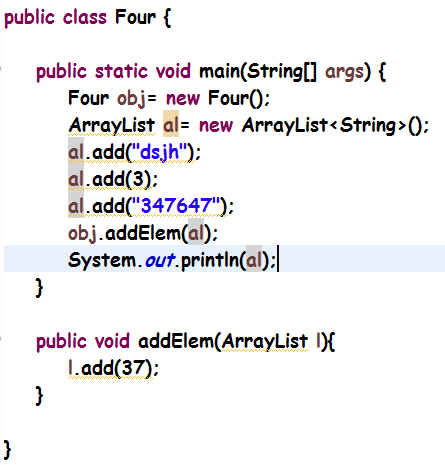
present based on that behavior will be defined.

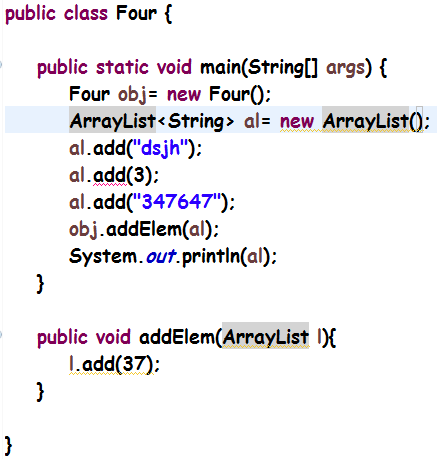


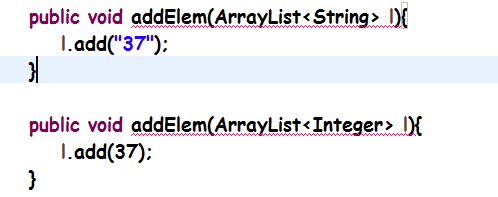
o/p: dsjh, 347647, 37

At compile time it is applicable, not applicable at runtime.

For non-generic reference, can assign generic object.







The most commonly used type parameter names are:

* E - Element (used extensively by the Java Collections Framework)
* K - Key
* N - Number
* T - Type
* V - Value
* S,U,V etc. - 2nd, 3rd, 4th types