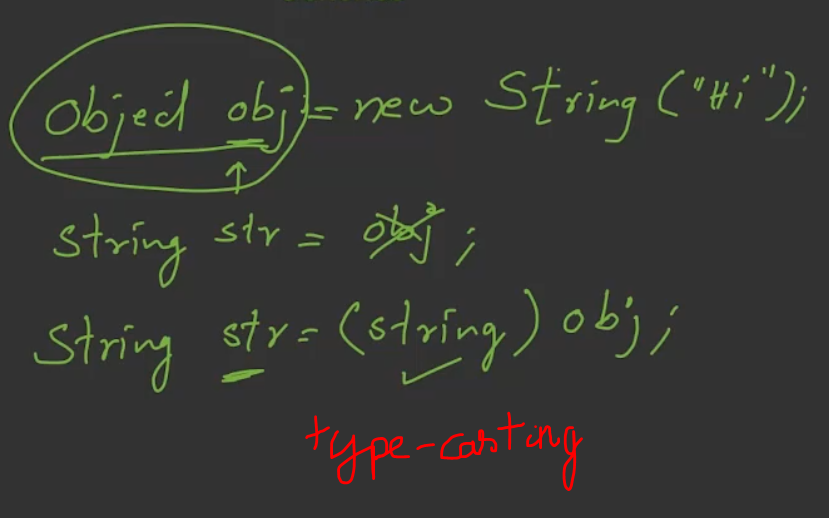
# **Before Generics**

Class Object 🡪 this the built-in-class / mother classes for all the JAVA classes.  
Every class is directly/indirectly from object class.

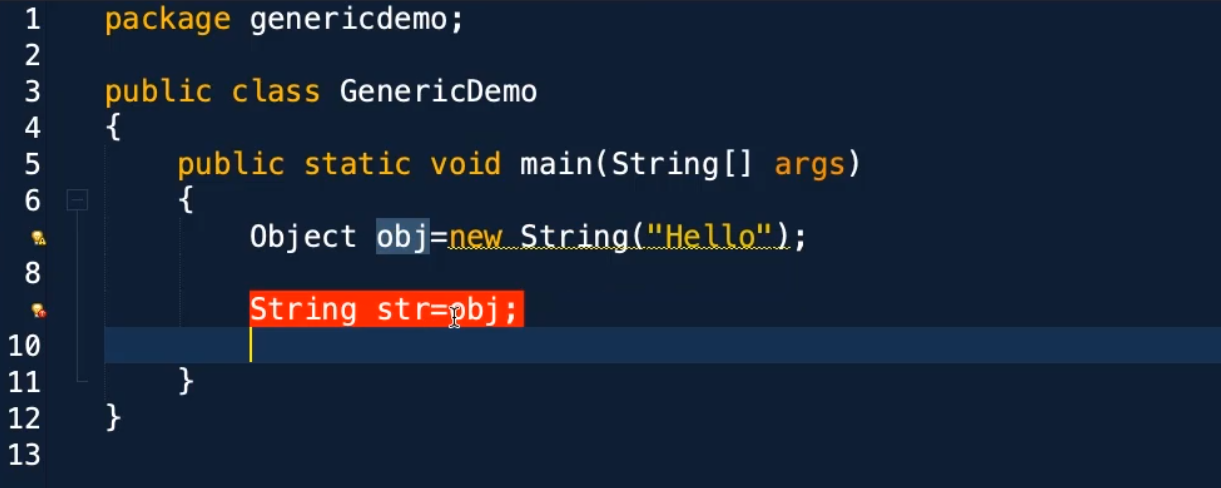
## **Class object**

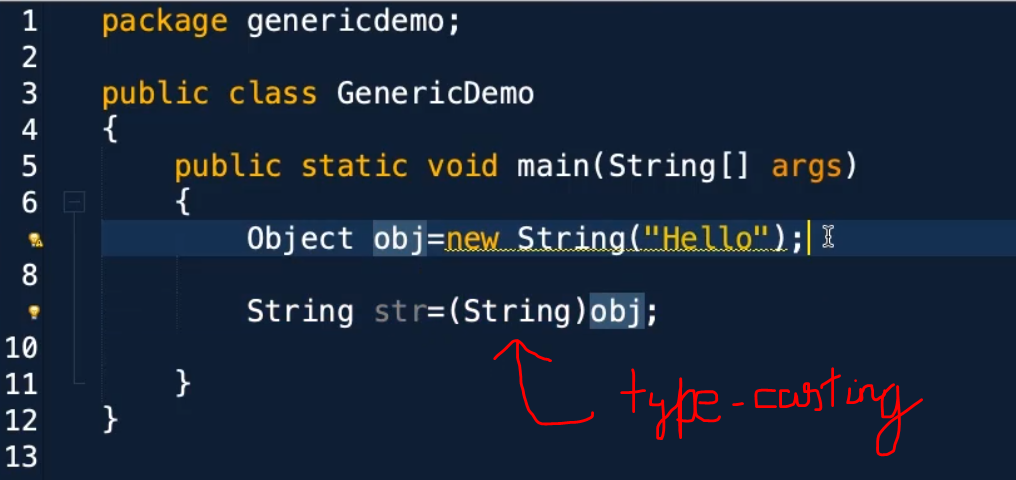
Achieving generalization.  
 ***Object obj1 = new String(“Hi”); [ Generalization ]***With the reference of an object (obj1) of the Object [parent class] class.  
We can assign the object of any child class (String).

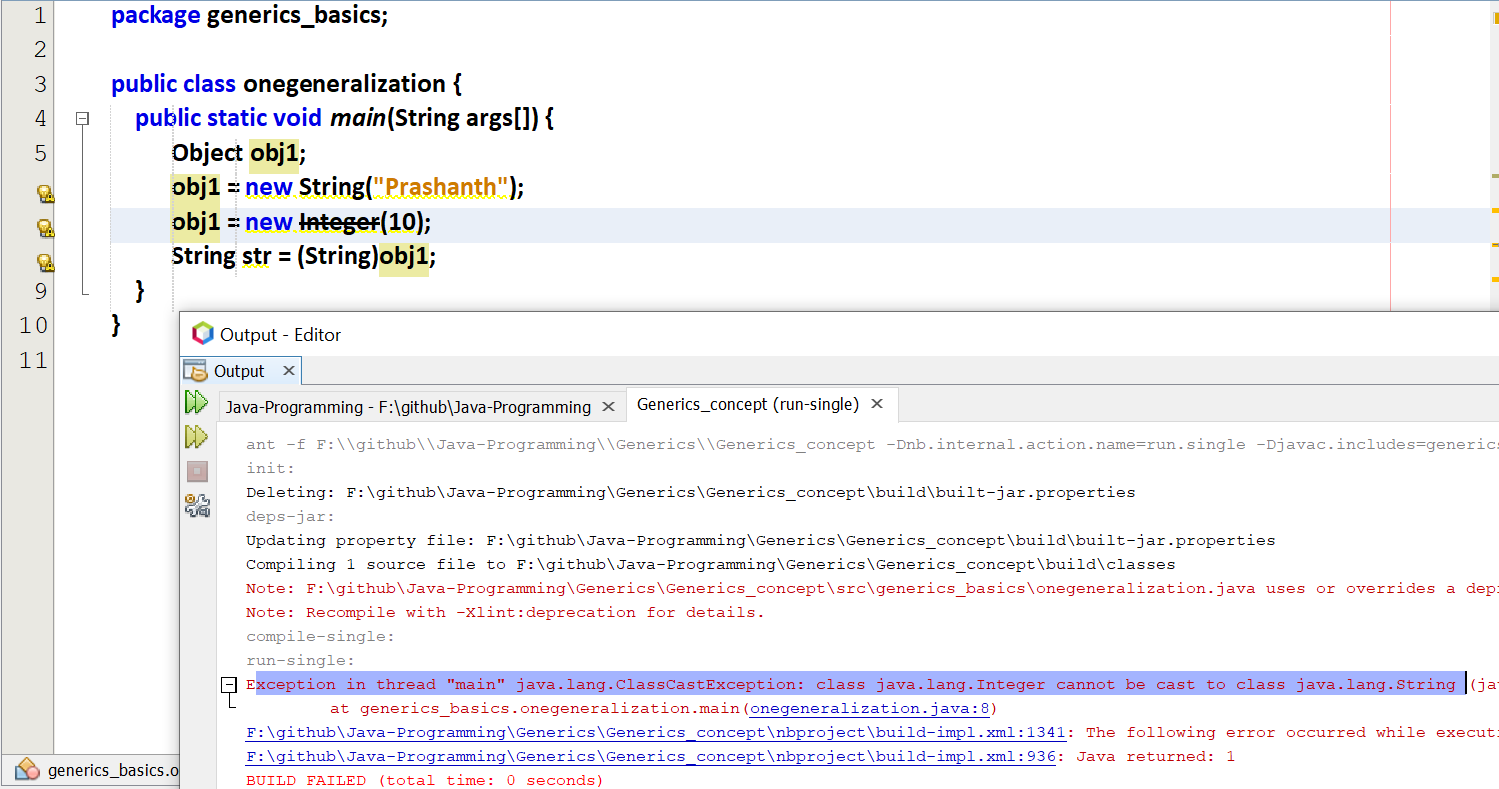
With Only one reference (obj1), I can hold the objects of various classes (String, Int, Float etc.)



Before generics were introduced, java uses Object class for generalization.

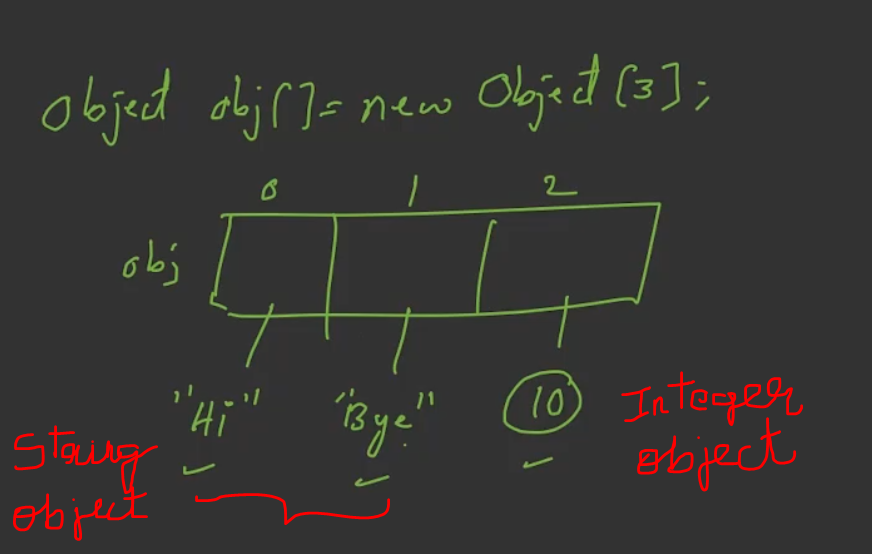
  
Objects cannot be converted into strings.





I am trying to type-cast integer to a string.  
At compile time, it is accepting without any prompt.   
But the error is raised only at the run-time.  
[integer object cannot be converted into string]

## **Array of objects**





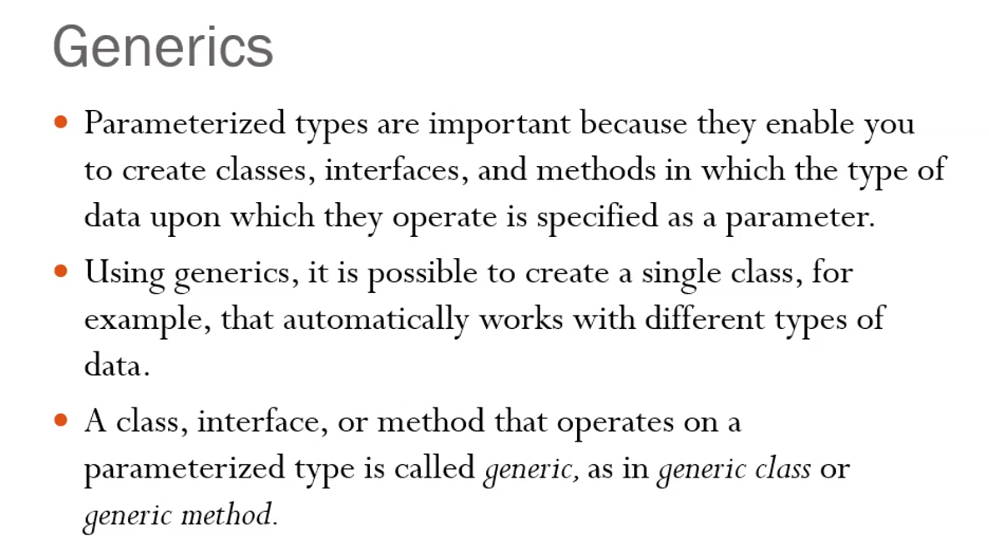
Here the 0th and 1st index 🡪 string  
2nd index 🡪 int  
string and int are also objects, so the array of type objects stores it.  
But the main draw-back is, the error is raised only at the run-time.

Therefore we want an array to store only integers. If the 0th index is integer, then the complete array must be filled with integers.  
Therefore we want an array to store only string. If the 0th index is string , then the complete array must be filled with string.

But the array must be capable of storing any data-type. Before using that array we should mention that data-type.

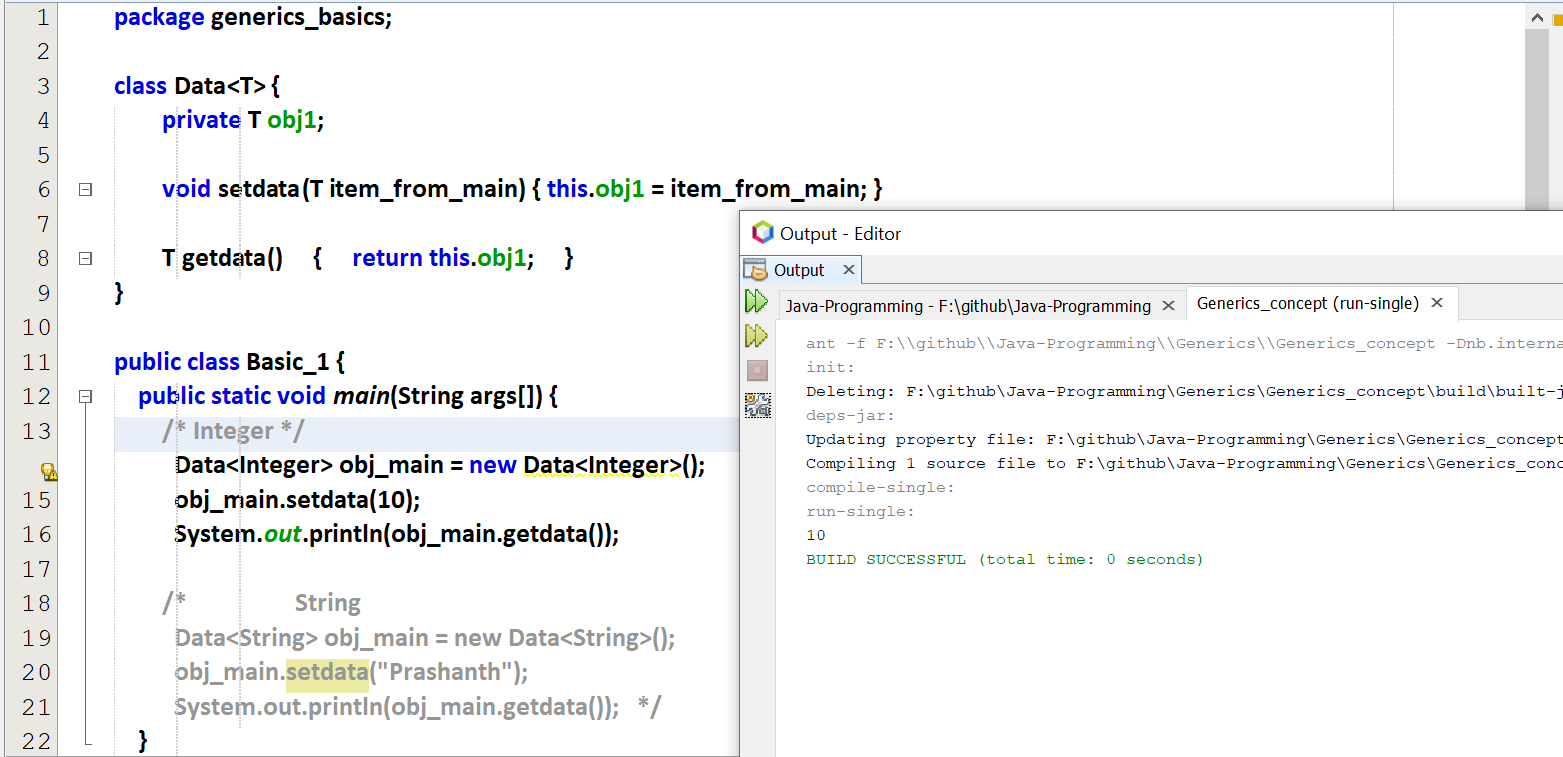
This can be done by using generics

# **Generics**

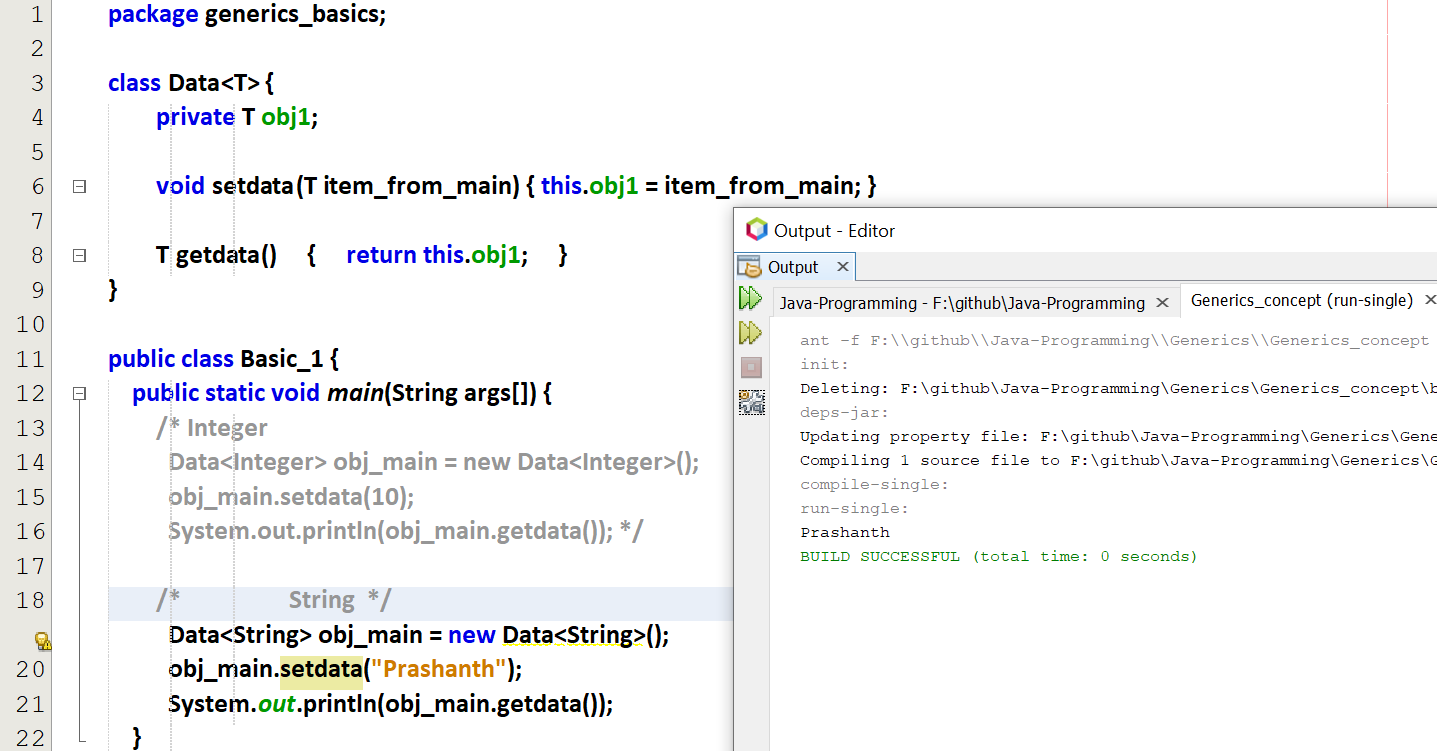


Templates in C++ 🡪 primitive data-types  
Generics in JAVA 🡪 objects

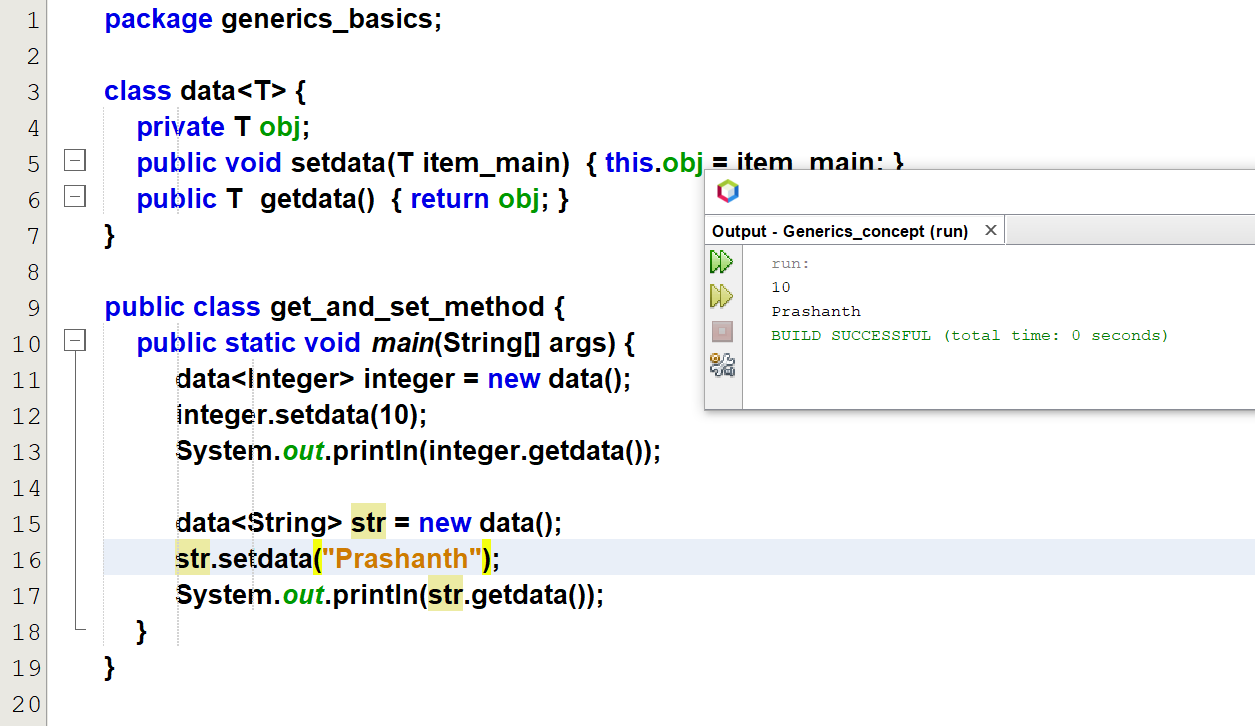
## **Refer Basic\_1.java**



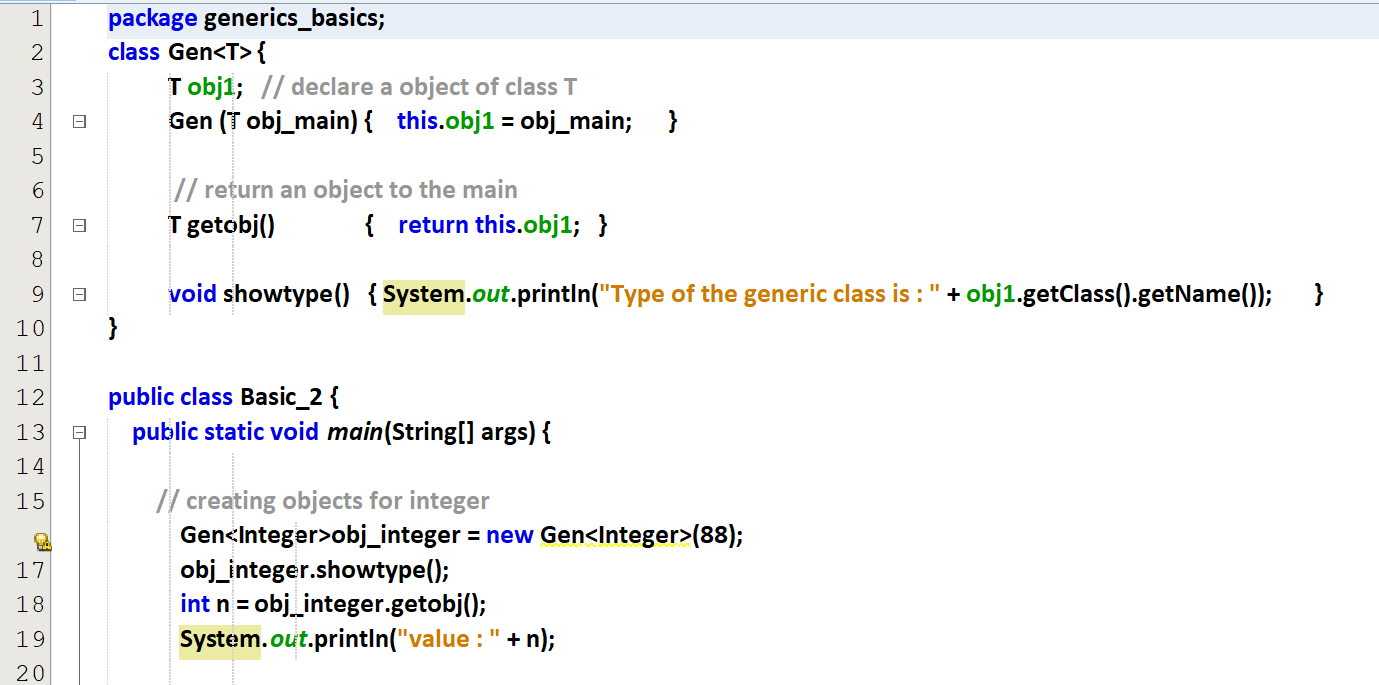
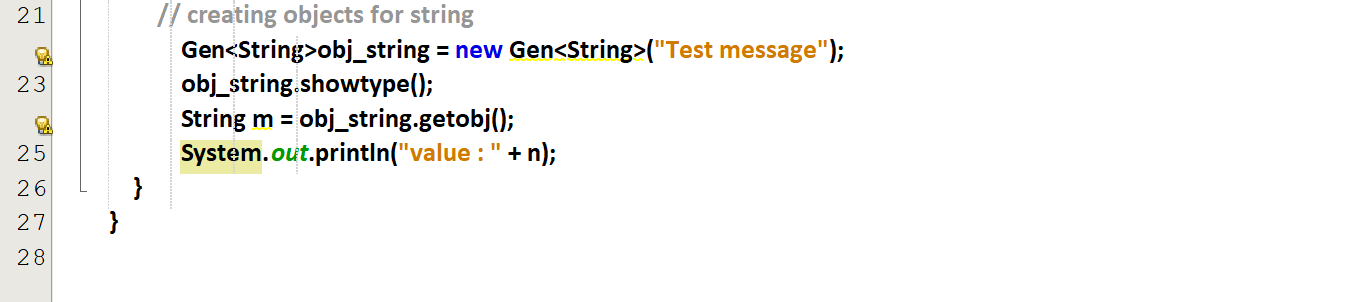
While getting the data, there is no need of type casting unlike objects.

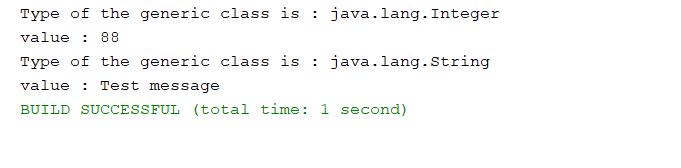


## **Refer get\_and\_set\_method**

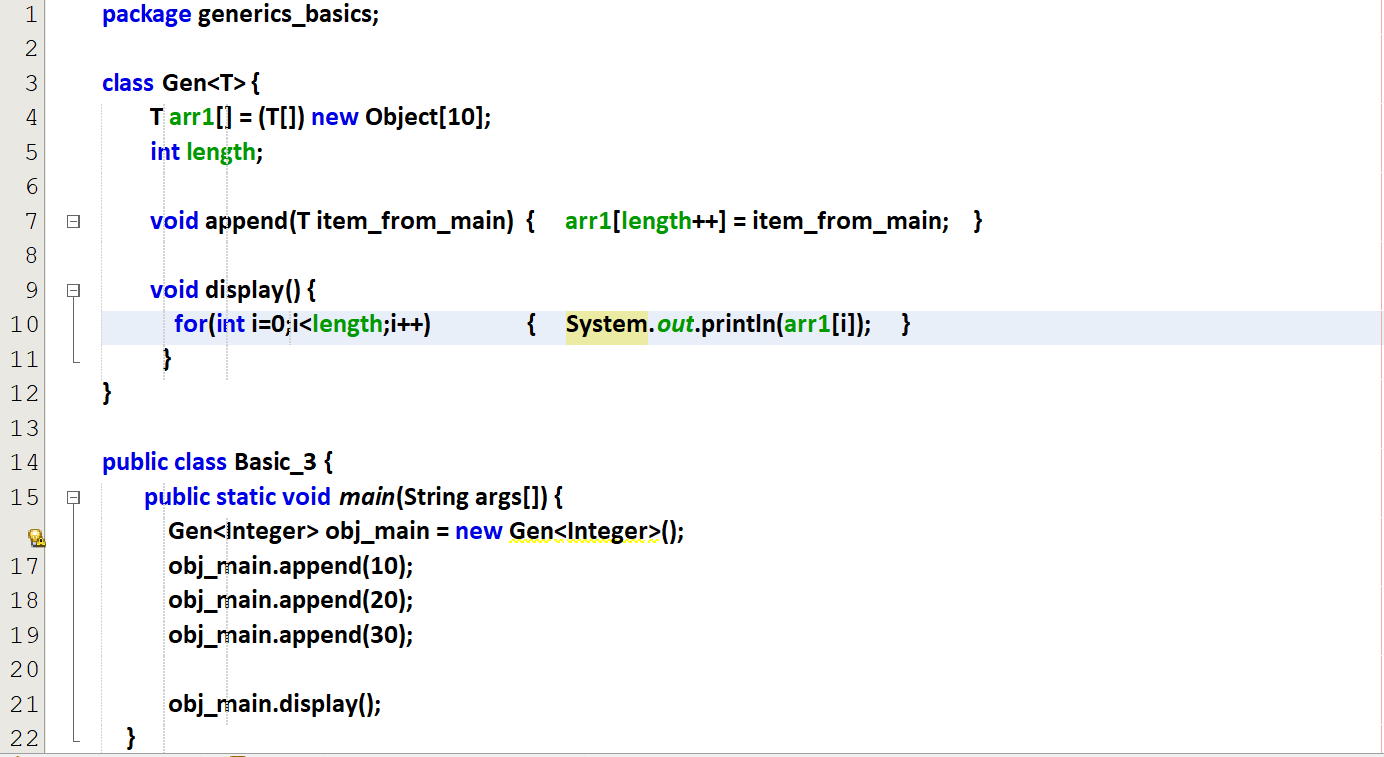


## **Refer basic\_2\_assignvalue\_via\_constructor**

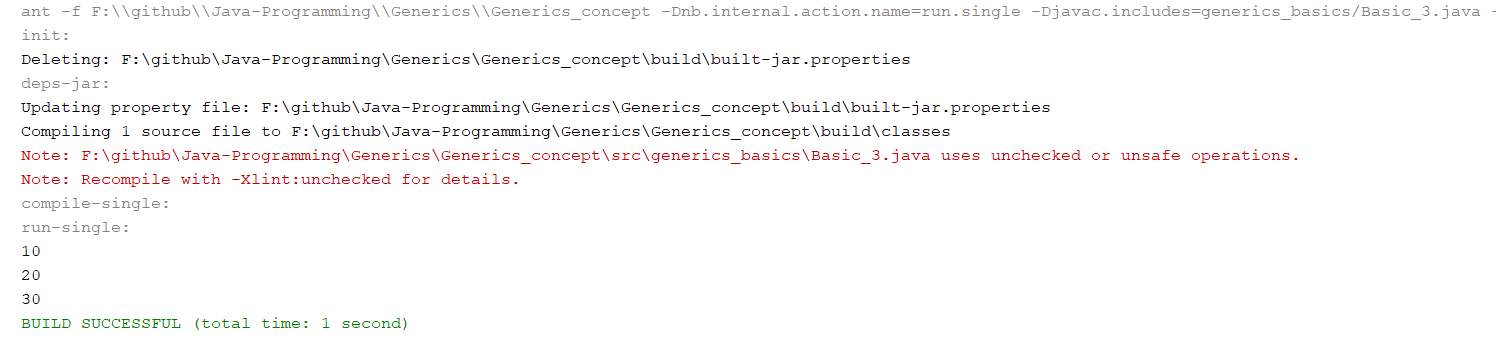
  


Output

## **Refer basic\_3\_array\_operations\_.java**



Output

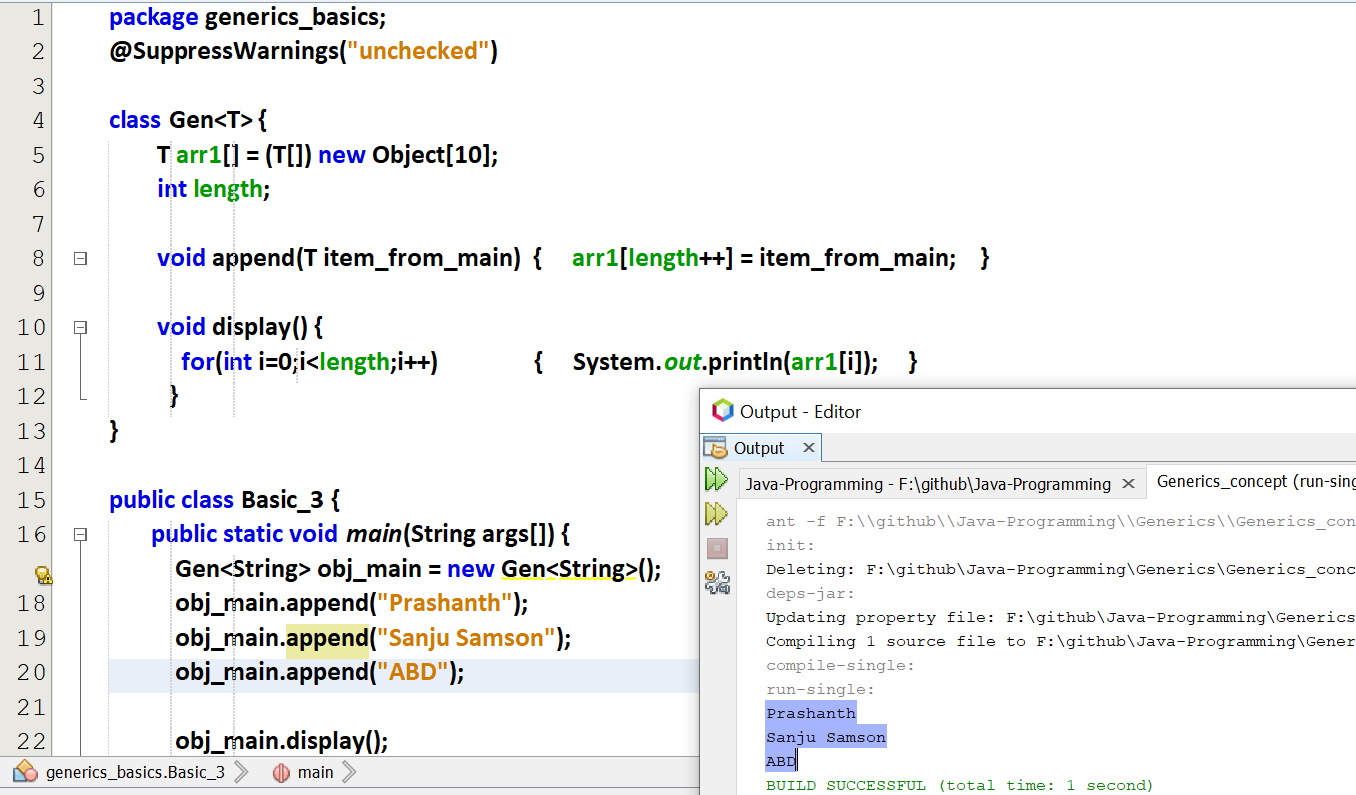


Reason for warning 🡪 Here the object is converted into Generic, we can avoid this by using   
@SuppressWarnings(“unchecked”)

With integer

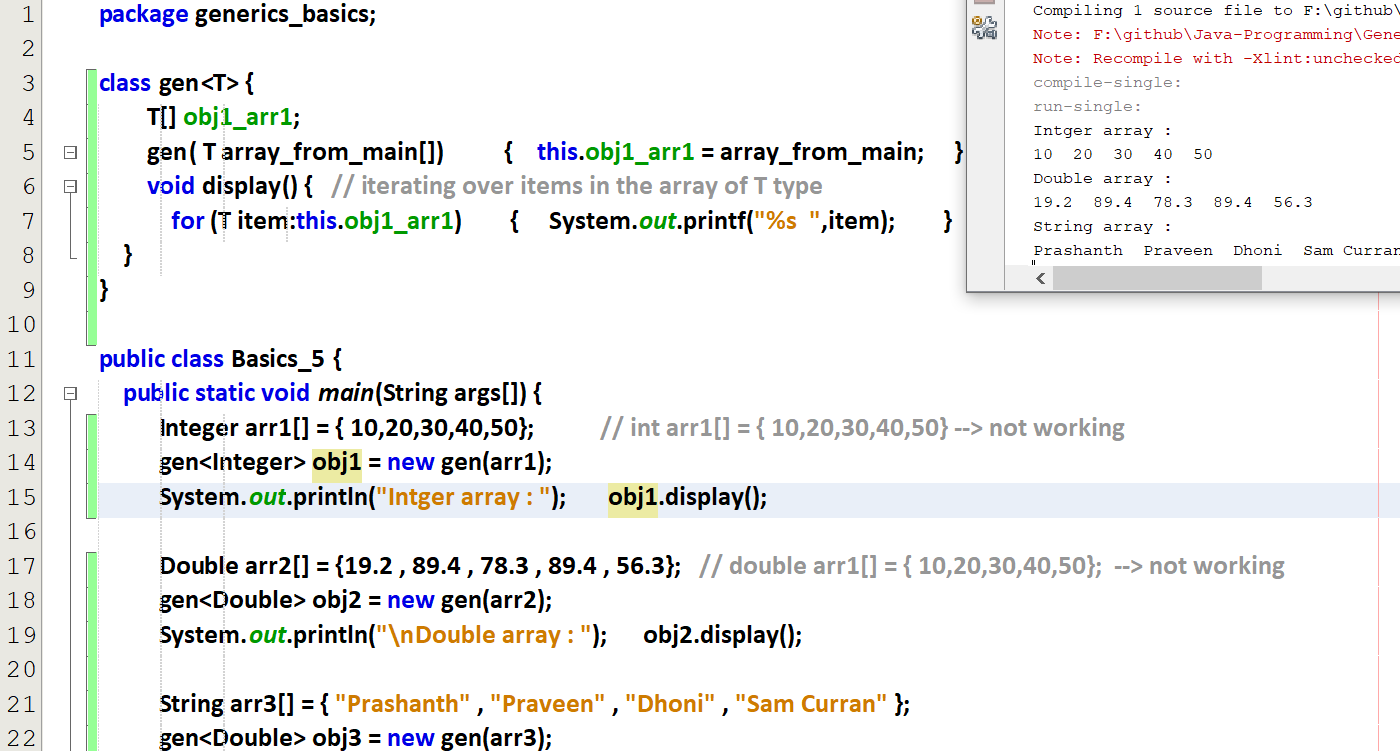


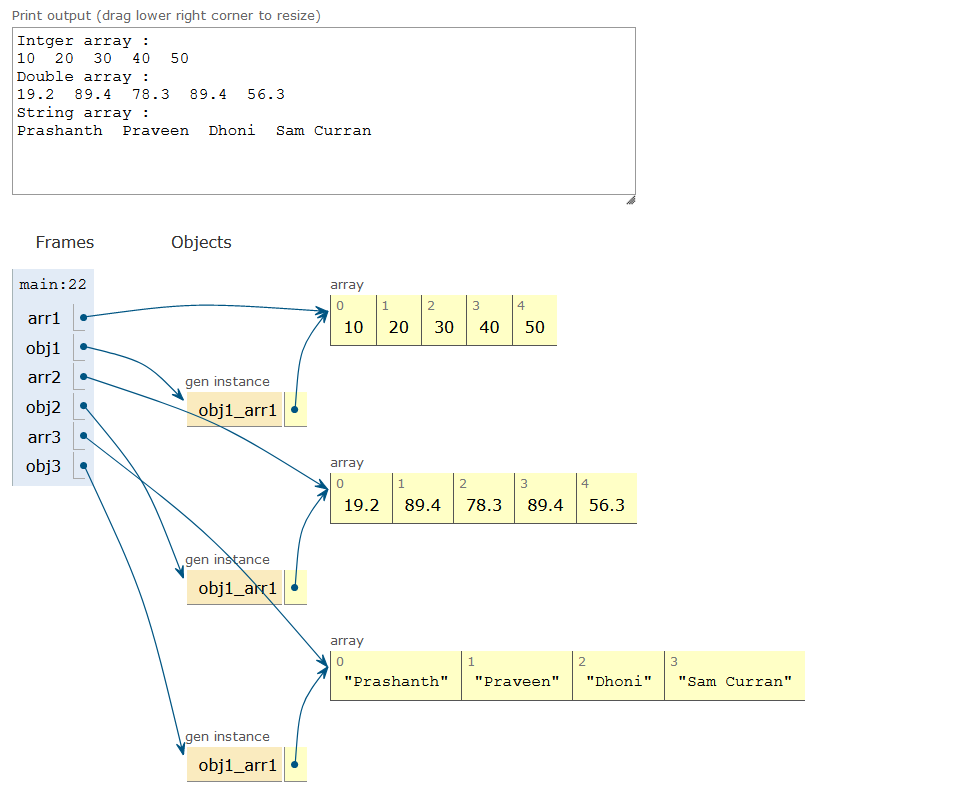
With strings

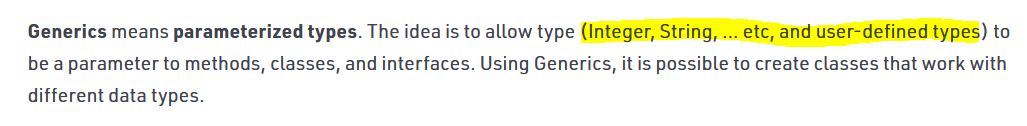


## **Refer basic\_4\_array\_as\_a\_parameter.java**

## **Refer basic\_5.java Array**



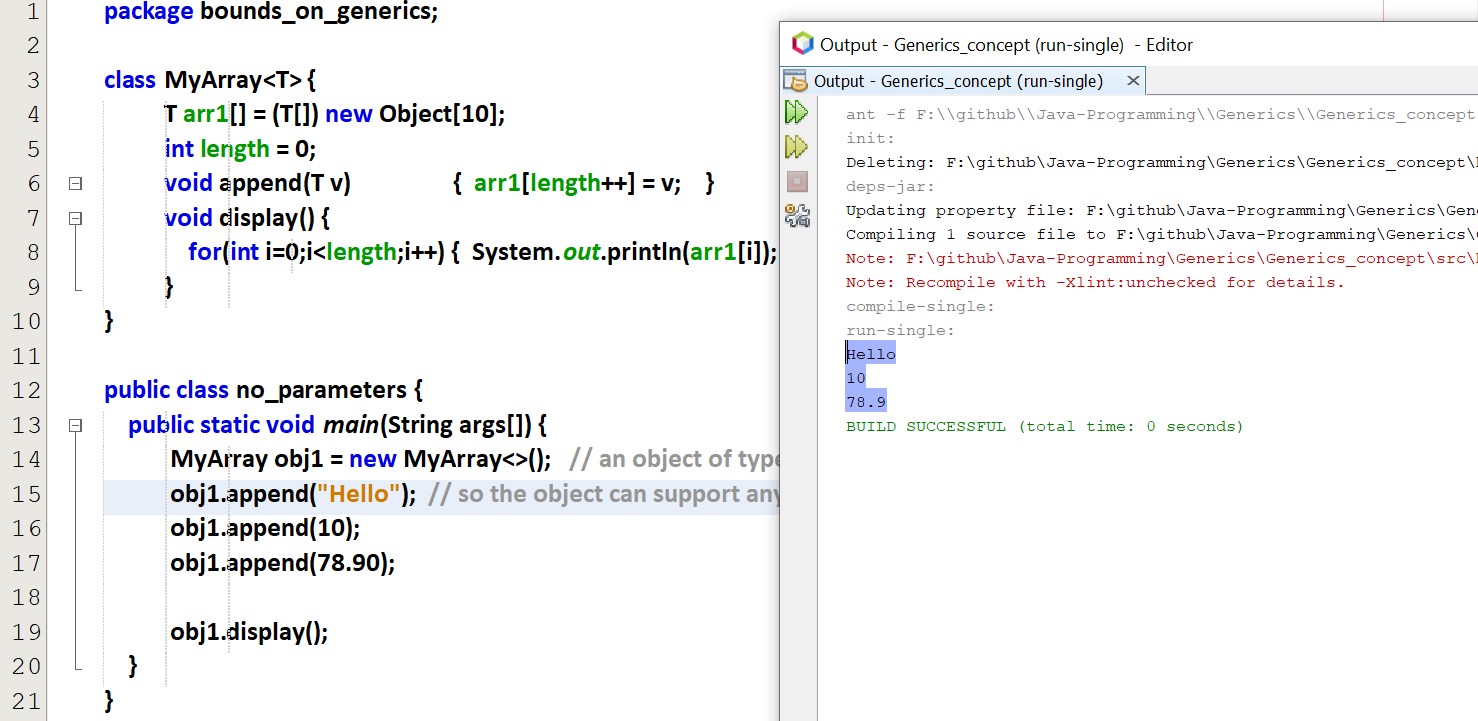




# **Bounds on Generics**



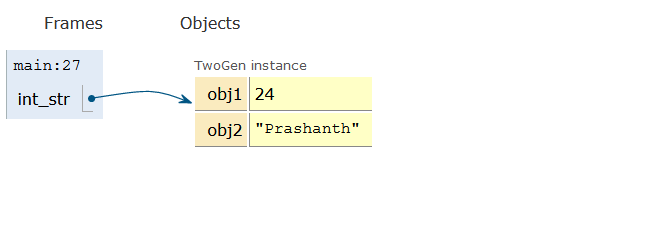
## **Refer no\_parameters .java**

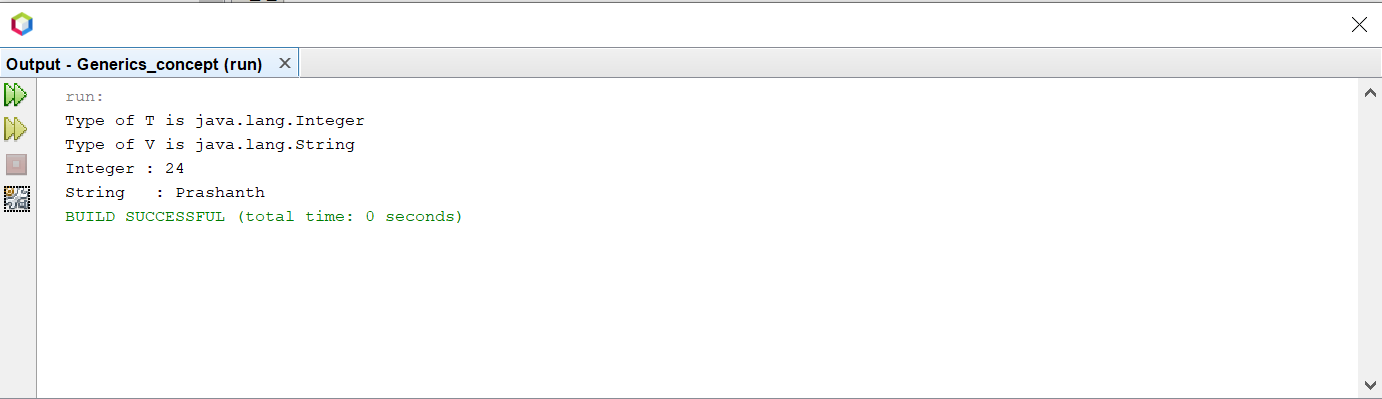


obj1 🡪 an object of type generic, since parameters are not given.  
Since the class is also Generic, it supports any type of object.

## **Refer multiple\_parameters.java**

Output:

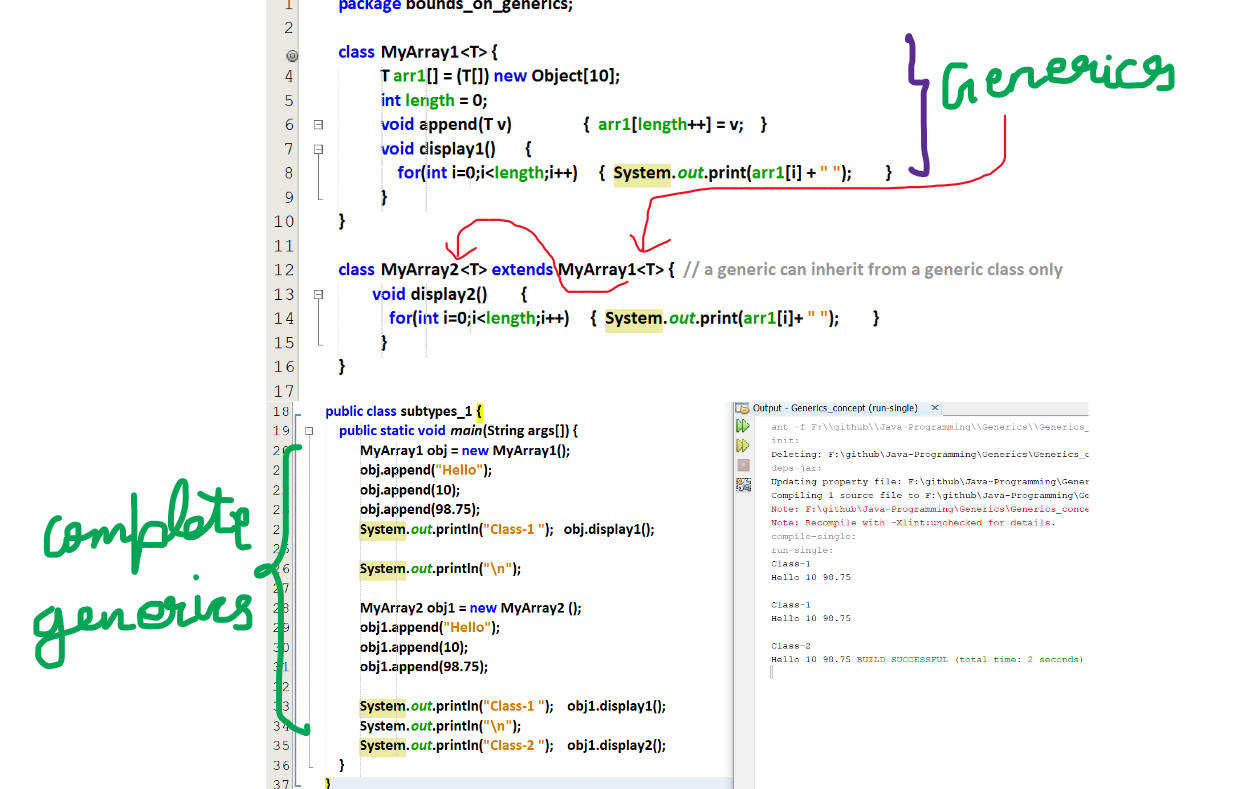


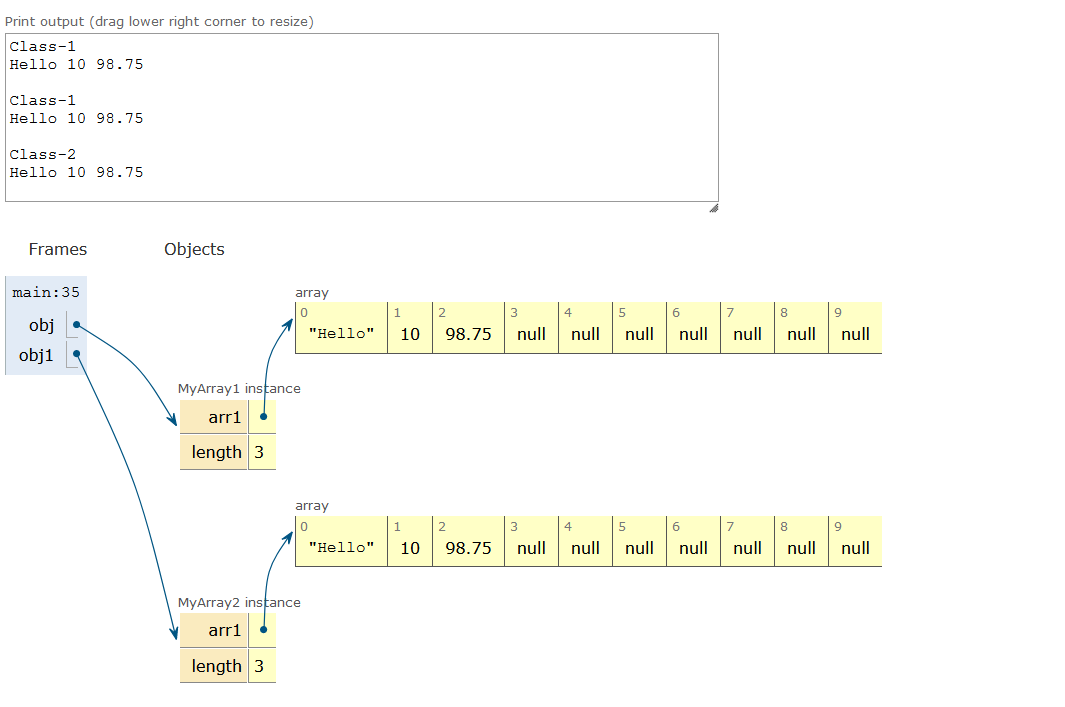


## **Subtypes**

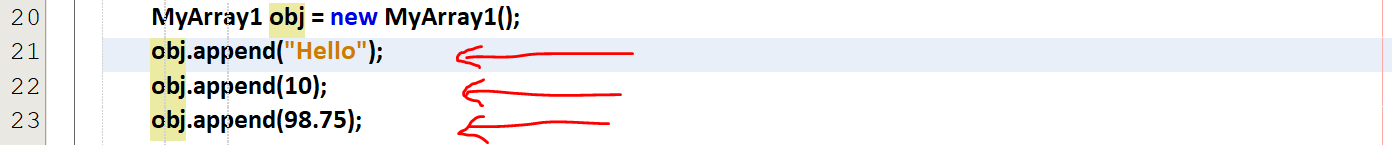
A normal class cannot able to inherit Generic classes.  
A generic class only can able to inherit Generic classes.

### **Refer subtypes\_1.java**



Both the classes are generic

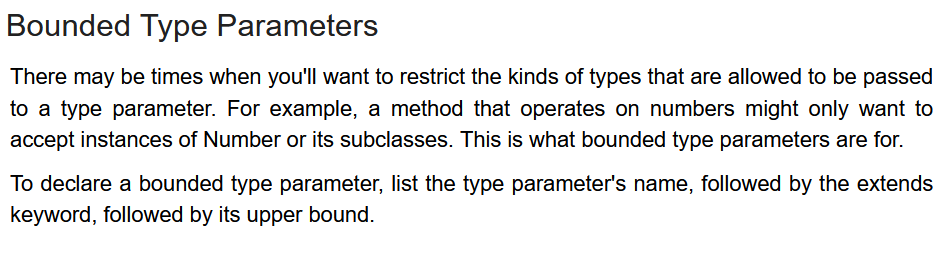
### **Refer subtypes\_2.java**



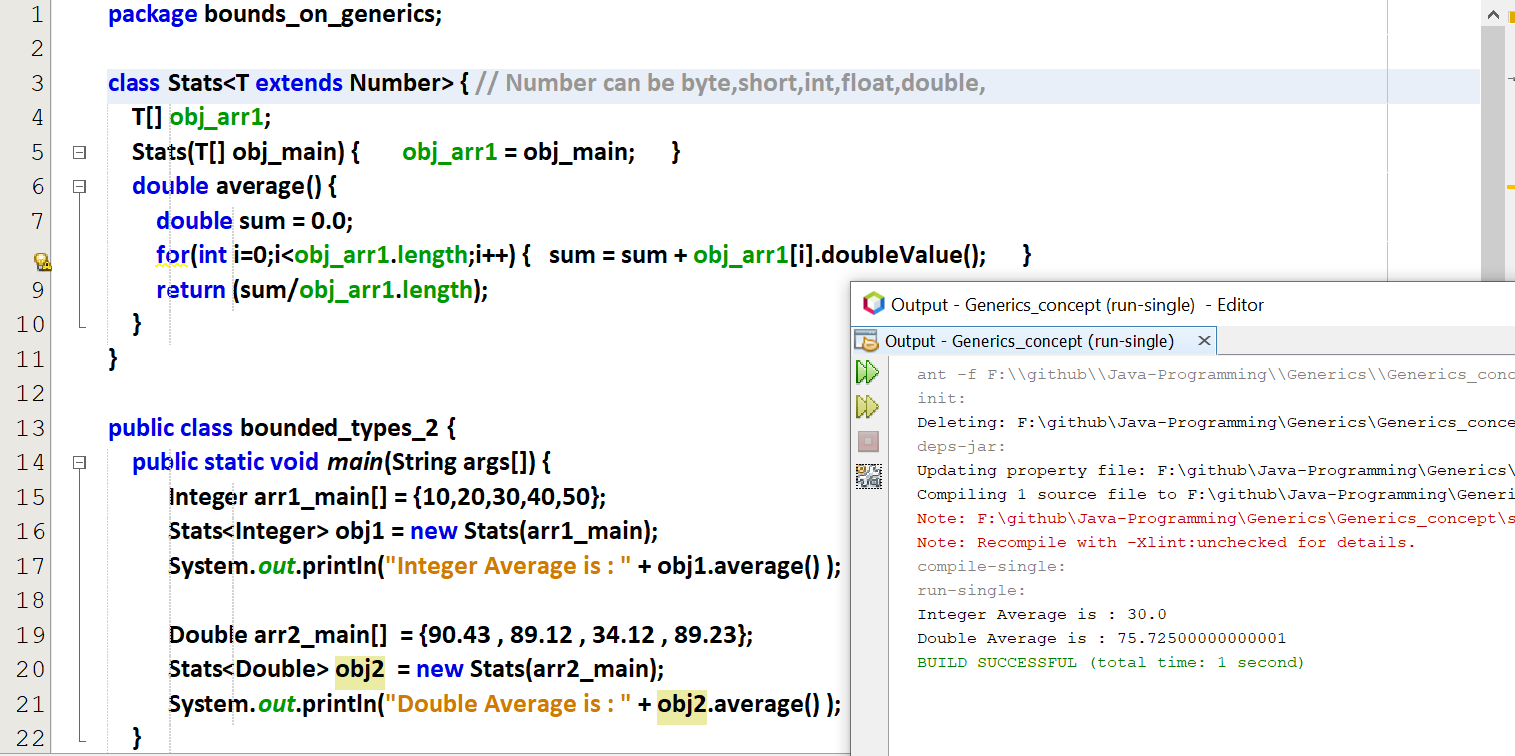
Myarray1 🡪 Generic   
object for Myarray1 🡪 Generic

Myarray2 extends Myarray1<String>  
Myarray2 🡪 String  
object for Myarray2 🡪 String

## **Bounded Types**



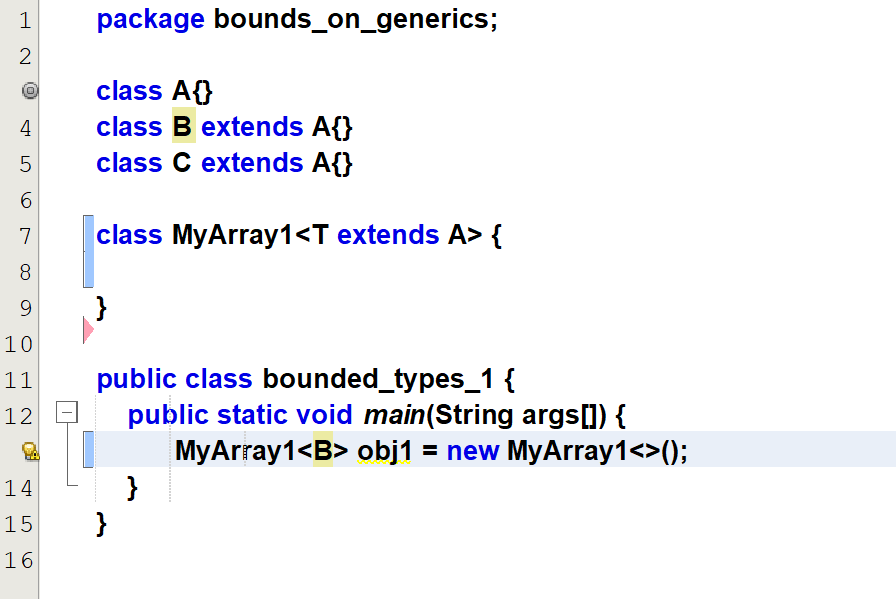
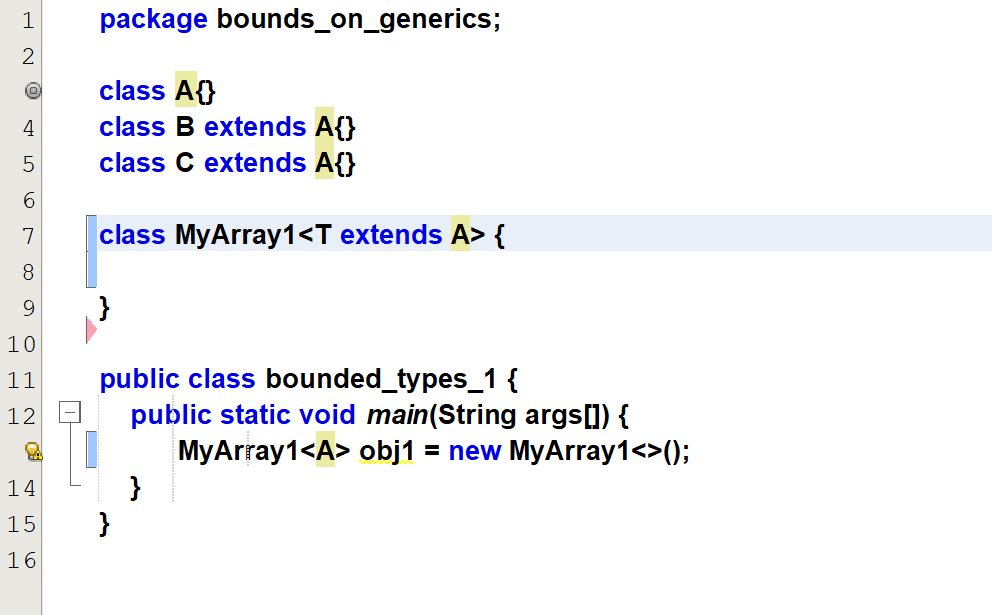
### **Refer bound\_types\_1.java**



  
If we are using Number, we can use all other classes (Float, Int, Double etc) inheriting from Number.

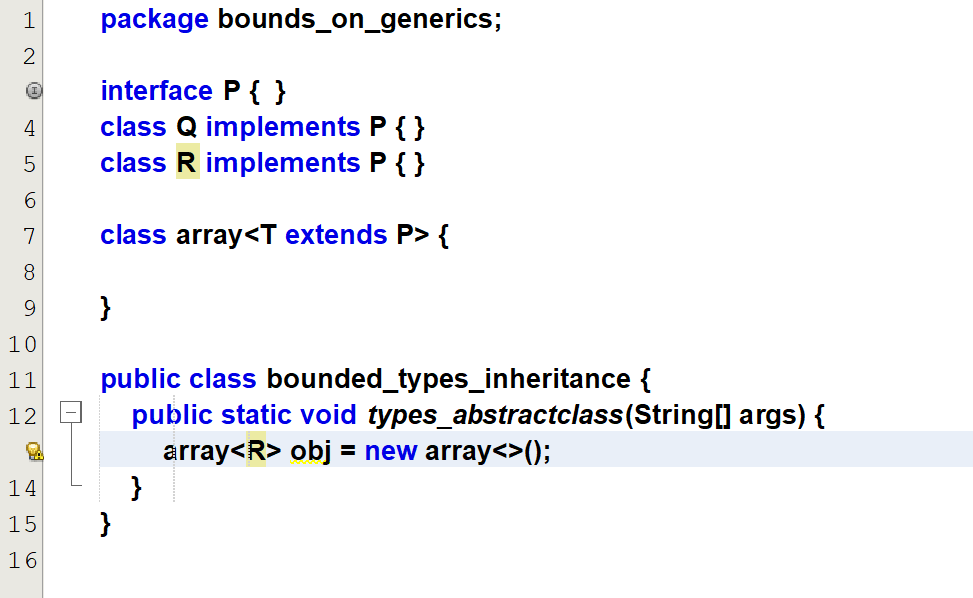
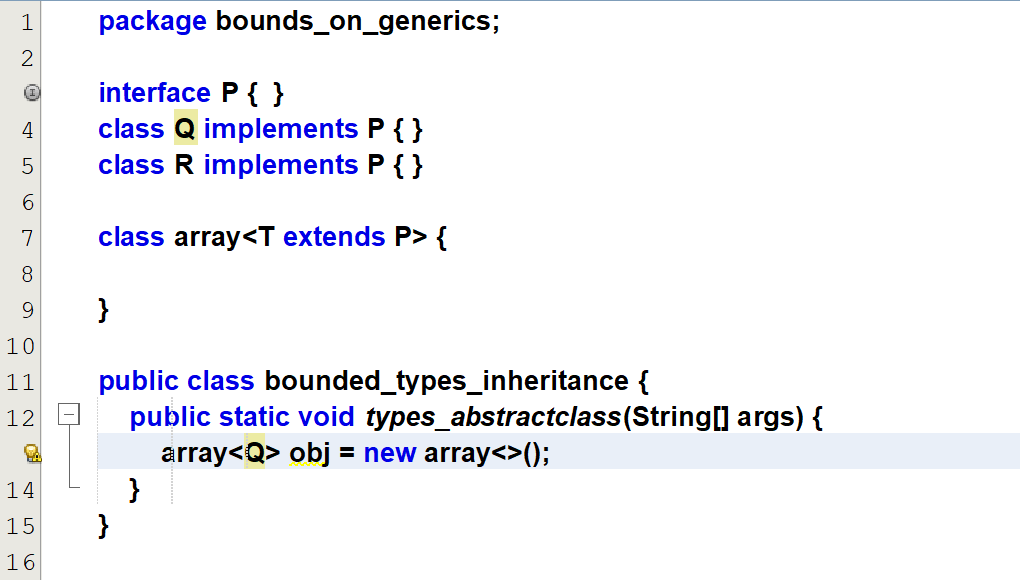
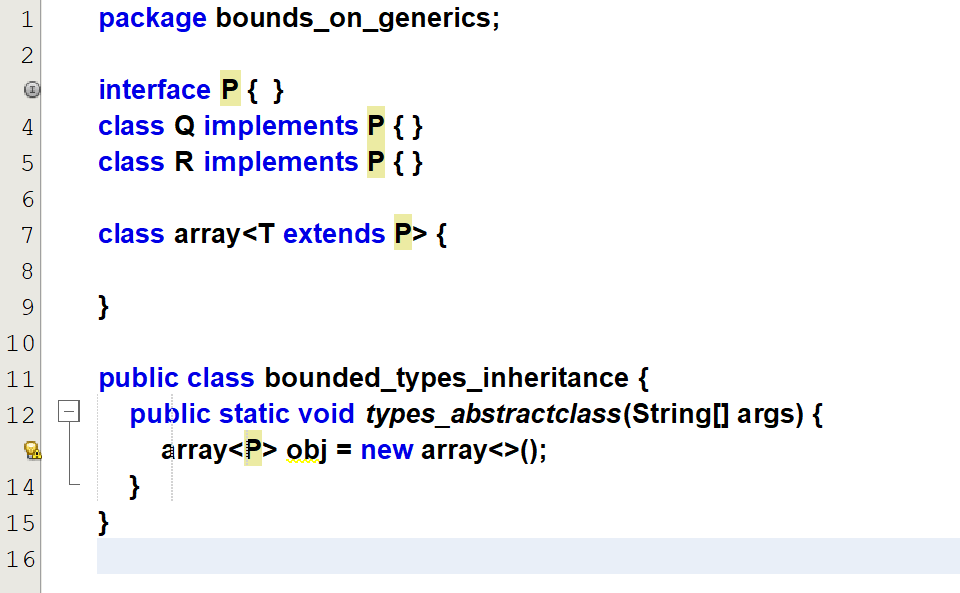
Forcefully we are making the generic class to be of Number

### **Refer bound\_types\_inheritance.java**



We can use class-A or   
We can use any other class inheriting from class-A.

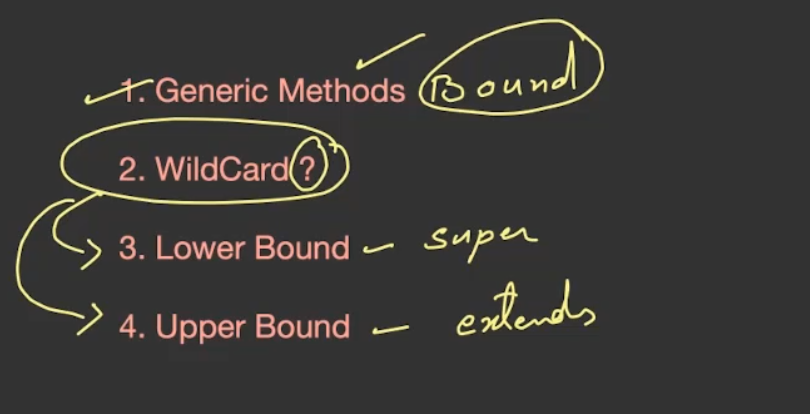
### **Refer bound\_types\_abstractclass.java**



For interface, we can use extends also.  
We can use class-P or any other class inheriting from class-P.

# **Generic Methods**

Till now we look into generic classes, from now we will look upon the generic methods.



## **Refer method\_1.java**

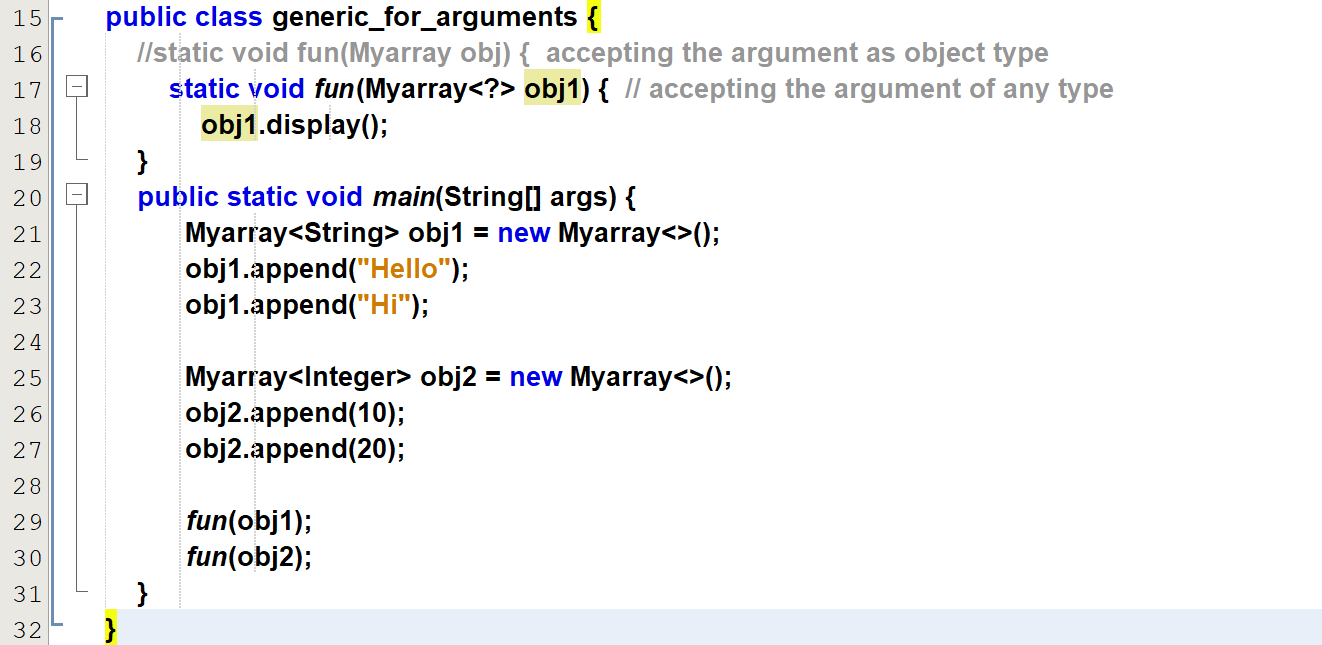
## **Refer method\_2.java**

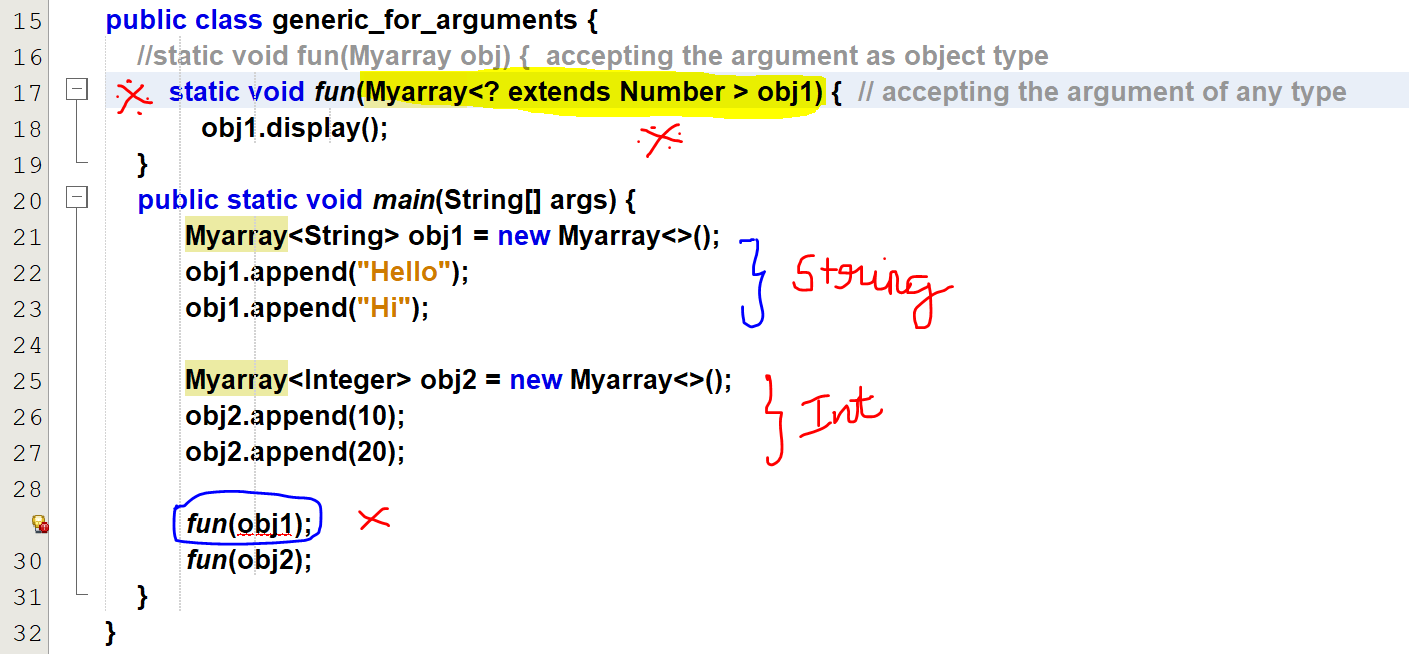
Now the class E can inherit only from Number.  
So trying to enter the string 🡪 error

## **Refer generic\_for\_arguments.java**

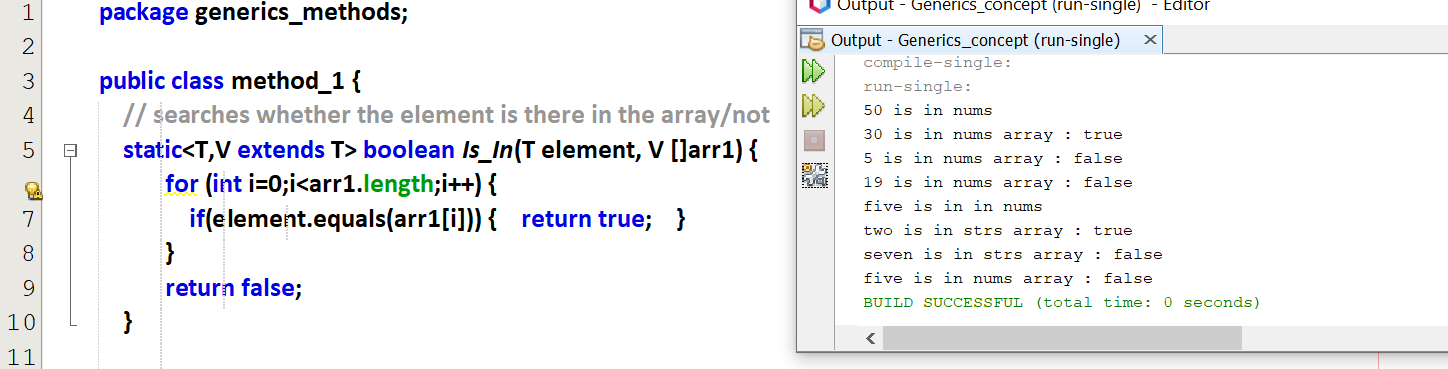
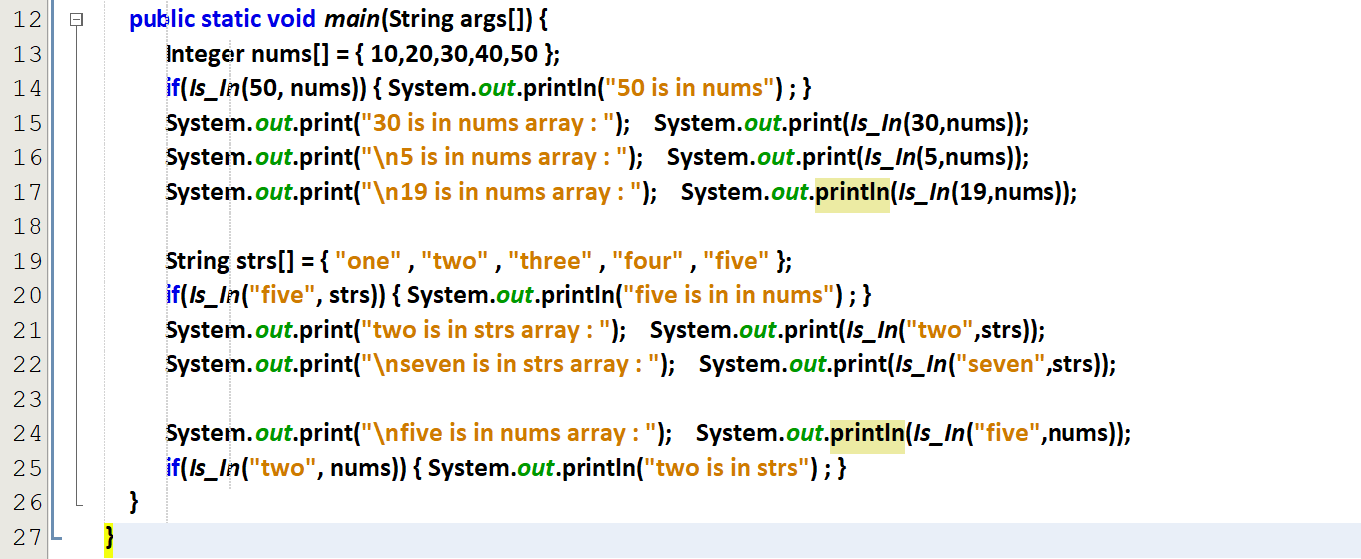
If I use ***Myarray<T> obj1***, then I must change the function type also as generic like static <T> void fun().

But my aim is to have the arguments as generic without making the functions as generic .So I can use ***Myarray<?> obj1***



Making my arguments to be only integer and not string.  


## **Refer challenge [element present or not]**

# **Need for wild card**



After putting ?



