Singleton Class

q) P95

```
package login;
class Gr
{
  // Singleton class
private static Gr g1 = null;
  private Gr()
  {
    Sop("running Gr() constructor");
  }
  public static Gr getInstance()
  {
    if(g1==null)
    {
      Sop("Creating an instance of Gr");
      g1 = new Gr();
    }
    return g1;
  }
  void sample1()
  {
    Sop("running sample1() of Gr Class");
  }
}
class Run3
{
  psvm(String[] args)
  {
    Sop("Program starts...");
    // Get Instance of class Gr
    Gr g2 = Gr.getInstance();
    g2.sample1();
    Gr g3 = Gr.getInstance();
    g3.sample1();
    Sop("Program ends...");
  }
}
```

30*

1* Any java class which allows to create only one instance is known as Singleton Class.

2* A java class can be qualified as a singleton class, if it meets following criterias.

2.1* The constructor should be private

2.2* The class should have a static public method which returns an instance of the class. The method should return only one instance whenever it is invoked.

3* A class should have referrence variable of same type which has to be private.

O/p: Program starts...
creating an instance of Gr
running Gr() constructor
running sample1 of class Gr
running Sample of class Gr
Program ends...

OBJECT CLASS

13-02-2013 | WEDNESDAY

30*
```
/*
    Gr g2 = Gr.getInstance();
    Gr g3 = Gr.getInstance();
    Gr g4 = Gr.getInstance();
    Gr g5 = Gr.getInstance();
*/
```

System class

1*) System is a class available in java.lang package.

2*) java.lang package is imported to every java class by default, hence no need to import explicitly

pg6
10)
```
package com.jspiders.demo;
class Department;
{
 static Student st = new Student(); // st is type of Student
}
class Student
{
  int stID = 1209;
  double stMarks = 81.22;
  psvm (String[] args)
  {
     System.out.println("Student ID: "+stID);
     Sop("Student Marks: "+stMarks);
  }
}
public class Demo1
  {  psvm (String[] args)
     {
       Sop("Program starts...");
       Department.st.dispStDetails();
       System.out.println("program ends...");
     }
  }
```

3*) System class contains two static reference variables which are final

3.1 * in is a type of InputStream.

3.2 * out is a type of PrintStream

4*) The PrintStream Object contains several methods used to print characters on the Standard Output device /Monitors

5*) InputStream contains members or methods used read from Standard Input Device

System.out.println means System is class, out is a static reference variable of type PrintStream and println is a member of PrintStream Object."

toString()

p97 -

ii)
```
package com.jspiders.objectclassdemo;
class A
{

}
public class Demo1
{
  psvm(String[] args)
  {
    Sop("Program starts...");
    A a1 = new A();
    // String s1 = a1.toString();
    Sop(a1.toString()); // (explicitly call to toString)
    Sop("----------");
    A a2 = new A();
    Sop(a2.toString());
    Sop("program ends...");
  }
}
```

O/P:-

```
Program starts...
com.jspiders.objectclass demo.A@1a16869
----------
com.jspiders.object class demo.A@1e278b9
Program ends...
```

1* Object class is the supermost class in java where every class has to inherit member of object class.

2* This class is available in package java.lang.

3* The toString() of object class is public method of return type String.

Whenever this method is invoked, the method returns String representation of object.

4* The string representation is in format of

fullyqualified classname@hexadecimal address.

Whenever a reference variable is printed the toString() method is implicitly invoked, hence return value of toString() method is displayed.

P98

12) package com.jspiders.objectclassdemo;

```
public class Demo2
{
    psvm(String[] args)
    {
        Sop("program starts...");
        A a1 = new A();
        Sop("a1") // implicit call to toString()
        Sop("-- ---------");
        A a2 = new A();
        Sop(a2); // implicit call to toString()
        Sop("Program ends...");
    }
}
```

```
Program starts...
com.jspiders.objectclassdemo.A@1de1279
----------
com.jspiders.objectclassdemo.A@1e27869
Program ends...
```

5.* A class can override toString() method of Object class In such case the object will have overrided implementation, the original implementation will be lost.

13)

```
package com.jspiders.objectclassdemo;
class B
{
  int i;
  B(int i)
  {
    this.i = i;
  }
  public String toString()
  {
    return ""+this.i; // String format of i
  }

public class Demo3
{
  psvm(String[] args)
  {
   Sop("Program starts...");
   B b1 = new B(23);
   Sop(b1);
   Sop("----------");
   B b2 = new B(289);
   Sop(b2);
   Sop("Program ends...");
  }
}
```

O/p:

```
Program starts...
23
----------
289
Program ends...
```

equals method

14) P100

```
package com.jspiders.object class demo;

class C
{
    int i;
    C(int i)
    {
        this.i = i;
    }
}

public class Demo4
{
    psvm(string[] args)
    {
        Sop("Program starts...");
        C c1 = new C(28);
        Sop("c1 = "+c1);
        Sop("-----------");
18.     C c2 = new C(372);
19.     Sop("c2 = "+c2); 20. Sop("compare c1 and c2");
21.     Sop(c1==c2); // false
22.     Sop("-----------");
        Sop(c1.equals(c2)); // false
        Sop("Program ends...");
    }
}
```

O/P:
Program starts
c1 = com.jspiders.objectclassdemo.C@12fe278
---
c2 = com.jspiders.objectclassdemo.C@12bf18
compare c1 and c2
false

Syntax:

```
public boolean equals(Object obj)
{

}
```

1) Equals method in object class is a public method which returns boolean type. This method takes an argument type 'Object'

2) Whenever this method is invoked on an instance, the method compares the current object with passed object based on the address of object and returns either ~~false~~ true or false.

If we have to compare Object based on object fields then Equals() method has to be overrided. While overriding the current object field can be referred by using 'this' keyword, the recieved object member should be first downcasted, then its member should be referred.

P101

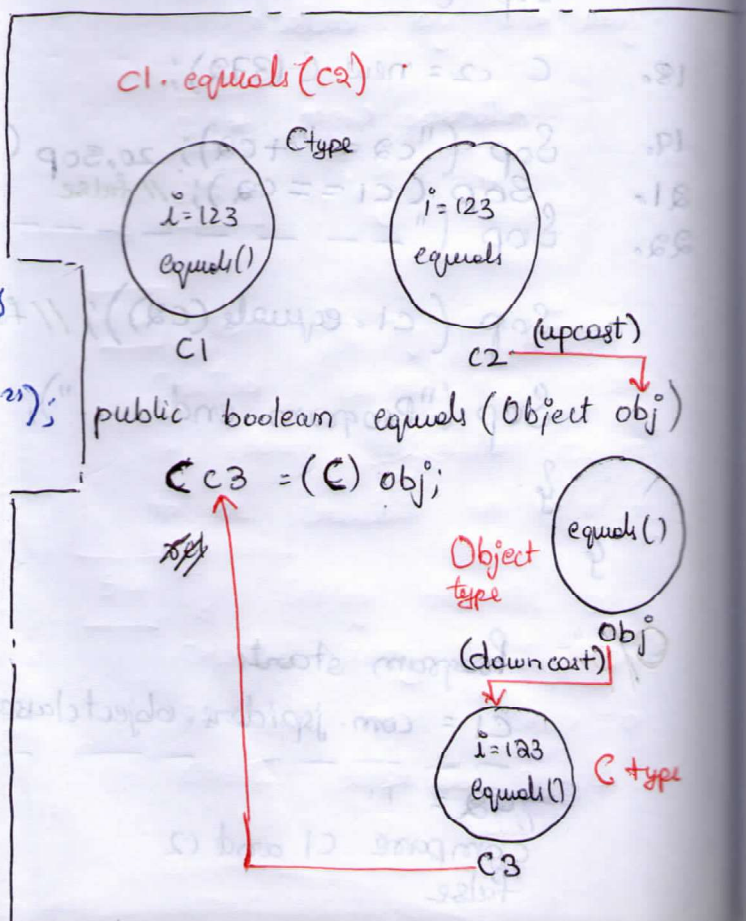
15)

```
package com.jspiders.objectclass.demo;
class C
{
    int i;
    C(int i)
    {
        this.i = i;
    }
    public boolean equals(Object obj)
    {
        C c3 = (C) obj; //downcast
        return this.i == c3.i;
    }
}
public class Demo4
{
    psvm(String[] args)
    {
        Sop("Program starts...");
        C c1 = new C(123);
        Sop("c1=" + c1);
        C c2 = new C(123);
        Sop("c2=" + c2);
        Sop("compare c1 and c2");
        Sop(c1==c2)
        Sop("-----------");
        Sop(c1.equals(c2));
        Sop("Program ends...")
    }
}
```

```
/*
    int j;

return (this.i == c3.i && this.j ==
        c3.j);
*/
```

// c2 upcasted to Object type i.e obj

O/p:- Program starts...
compare c1 and c2
false
true
Program ends...



hashcode()

15-02-2013 | Friday

16)

```
package com.jspiders.objectclassdemo;
class D
{
}
public class Demo5
{
    psvm (String[] args)
    {
        Sop ("Program starts...");
        D d1 = new D();
        Sop (d1.hashcode());
        Sop("----------");
        D d2 = new D();
        Sop (d2.hashcode());
        Sop ("Program ends...");
    }
}
```

O/p:-

```
Program starts...
30269696
----------
24052850
Program ends...
```

1*) hashcode() method of an Object class is a public method of return type integer.

Whenever this method is invoked, the method returns a hashcode value which is an integer value based on the object address.

2*) A class override an hashcode() method in such case original implementation is lost and will have overrided implementation.

String Class

17) package com.jspiders.stringclassdemo;

```
public class Demo1
{
  psvm(String[] args)
  {
   Sop("Program starts...");
   String str1 = new String("test");
   Sop("str1 = "+ str1);
   Sop("-----------");
   String str2 = new String("test");
   Sop("str2 = "+ str2);
   Sop("-----------");
   Sop("Compare str1 and str2");
   Sop(str1==str2);
   Sop("-----------");
   Sop(str1.equals(str2));
   Sop("Program ends...");
  }
}
```

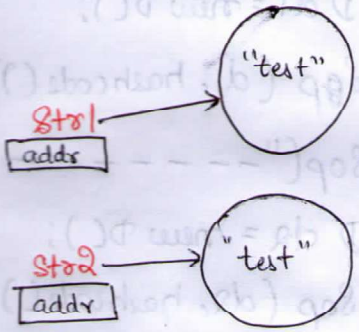
O/p:-

```
Program starts...
str1 = test
-------
str2 = test
-------
Compare str1 and str2
false
-------
true
Program ends...
```

com.jspiders.stringclass

1* String class is used handle the string values in a program this class is available in java.lang package.

2* String class is final class it cannot be inherited to any subclass



3* String objects can be created by two ways

3.1* new operator

3.2* using " "

4* When String object is created using new operator it allows duplicate object to be created in memory.

5* If String objects are created using " " then it will not allow to create duplicate objects.

P164

18) package com.jspiders.stringclassdemo;

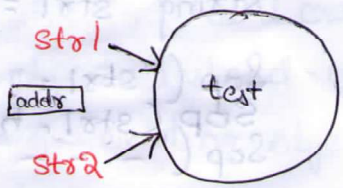
```
public class Demo1
{ psvm(String[] args)
  { Sop("Program starts...");
    String str1 = "test";
    Sop("str1 = "+ str1);
    Sop("----------");

    String str2 = "test";
    Sop("str2 = "+ str2);
    Sop("----------");
    Sop("compare str1 and str2");
    Sop(str1 == str2);
    Sop("----------");
    Sop(str1.equals(str2));
    Sop("Program ends...");
  }
}
```

O/P :-

```
Program starts...
str1 = test
str2 = test
----------
compare str1 and str2
true
----------
true
Program ends...
```

* If already a string object exists it uses it otherwise creates new object.

6* Whether an identical object exists or not is verified using equals() method.



7* In String Class toString() method is overrided to return the value of object.

8* The equals() method is also overrided to compare ~~two~~ two String objects based on value.

9* hashcode() method is also overrided to return hashcode value based on String value.

P105

19)
```
package

public class Demo3
{
  psvm (String [ ] args)
  {
   Sop(" Program starts ... ");

   String str1 = new String ("test");

   Sop ("str1 = " + str1);
   Sop (str1.hashcode());
   Sop ("- - - - - - - - - - - -");

   String str2 = new String("test");

   Sop ("str2 = "+ str2);
   Sop (str2.hashcode());
   Sop ("- - - - - - - - - - - - - -");
   Sop ("Program ends ...");
  }
}
```

O/p :-

```
Program starts ...
str1 = test
3556498
- - - - - - - - - - - - -
str2 = test
3556498
Program ends ...
```

String Class :- immutable

20) P106

```
package com.jspiders.stringclassdemo;

public class Demo4
{
    psvm (String[] args)
    {
        Sop ("Program starts...");
        String s1 = "developer";
        String s2 = "developer";
        String s3 = "developer";
        String s4 = "testing";
        Sop ("s1=" + s1);
        Sop ("s2=" + s2);
        Sop ("s3=" + s3);
        Sop ("s4=" + s4);
        Sop ("---------");
        s2 = "tester";
        s4 = "tested";
        Sop ("s1=" + s1);
        Sop ("s2=" + s2);
        Sop ("s3=" + s3);
        Sop ("s4=" + s4);
        Sop ("Program ends...");
    }
}
```

O/P:-

```
Program starts...
s1 = developer
s2 = developer
s3 = developer
s4 = testing
---------
s1 = developer
s2 = tester
s3 = developer
s4 = tested
Program ends...
```

10 * String is a immutable type because once a string object is created the object value cannot be changed.

If you change value of object, it will create a new object instead of changing existing object

This is known as immutable.

P107

j	a	v	a	d	e	v	e	l	o	p	e	r
0	1	2	3	4	5	6	7	8	9	10	11	12

21)

```
package com.jspiders.stringclassdemo;
pulic class Demo15
{
  psvm (String[] args)
  {
    Sop ("Program starts...");
    String s1 = "javadeveloper";
    Sop ("s1 = "+s1);
    Sop ("String length: "+s1.length());
    Sop ("char @ 4: " + s1.charAt(4));
    Sop ("index of char: "+s1.indexOf('e'));
    Sop ("index of char: "+s1.lastIndexOf('e'));
    Sop ("contains dev: "+s1.contains("dev"));
    Sop ("starts with java: "+s1.startsWith("java"));
    Sop ("ends with per: "+s1.endsWith("per"));
    Sop ("is empty? "+s1.isEmpty());
    Sop ("substring: "+s1.subString(8));
    Sop ("substring: "+s1.subString(4,11));
    Sop (s1.toUpperCase());
    Sop (s1.toLowerCase());
    Sop ("Program ends...");
  }
}
```

4 = d
10 = p

/* (i, i-1)
(4, 11-1)
(4, 10) */

O/P:

Program starts...
S1 = javadeveloper
String length : 13
char @ 4 : d
Index of char: 5
index of char: 11
contains dev : true
starts with java: true
ends with per: true
is empty? false
Substring : loper
substring : develop
Program ends...

jdk 1.5 →

	StringBuilder	StringBuffer	String
1*	new operator	new operator	new operator or double quote " "
2*	toString() is overrided	toString() is overrided	toString() is overrided
3*	equals() is not overrided	equals() is not Overrided	equals is overrided
4*	mutable - append() exists - insert() exists	mutable - append() exists - insert() exists	immutable
5*	reverse() exists	reverse() exists	reverse() does not exists
6*	final Class	final Class	final Class
7*	not thread Safe	thread safe (synchronized)	not threadsafe

* thread safe means all threads are synchronized i.e threads wait till other threads are executed.

P108

22)
```
package com.jspiders.stringclassdemo;
public class Demo6
{
  psvm(String[] args)
  {
    Sop("program starts...");
    StringBuffer sb1 = new StringBuffer("java");
    StringBuffer sb2 = new StringBuffer("java");
    Sop ("sb1 =" + sb1);
    Sop ("sb2 =" + sb2);
    Sop (sb1 == sb2);
    Sop (sb1.equals(sb2));
    Sop ("-----------");
    sb1.append("developer");
    Sop ("sb1 =" + sb1);
    Sop (sb1.reverse());
    Sop ("Program ends...");
  }
}
```

```
Program starts
sb1 = java
sb2 = java
false
false
-----------
sb1 = javadeveloper
repolevedavaj
```

Arrays

18-02-2013 | MONDAY

P109

1)
```
package com.jspiders.arrays;

public class Demo1
{
  psvm (String[] args)
  {
    Sop("Program starts...");
    int[] intArray = new int[5];
    Sop("array name : "+intArray);
    Sop("Total elements: "+intArray.length);
    Sop("2nd element : "+intArray[1]);
    Sop("Array elements are");
  // normal for loop
    for(int i=0; i<intArray.length; i++)
    {
      Sop(i+" element is "+intArray[i]);
    }
    Sop("-----------");
  // for each loop
    for(int k:intArray)
    {
      Sop("element "+k);
    }
    Sop("Program ends...");
  }
}
```

O/P:
```
Program starts...
array name : [I@3479404a
Total elements : 5
2nd element : 0
Array elements are
```

1*

Declaration:

arraytype [ ] arrayname ;

or

arraytype arrayname[ ];

2*

Initialize:

arrayname = new arraytype[size]

always int type

O/p continued...

```
0 element is 0
1 element is 0
2 element is 0
3 element is 0
4 element is 0
-----------
element 0
element 0
element 0
element 0
element 0
Program ends
```

2) package com.jspiders.[redacted]array;

```
import java.util.Arrays;

public class Demo2
{
  psvm (String[] args)
  {
    Sop ("Program starts...");
    int[] intArray = new int[5];
    intArray[0] = 67;
    intArray[1] = 12;
    intArray[2] = 35
    intArray[3] = (int) 28.98;
    intArray[4] = 41;
    Sop ("array elements before sort:");
    for (int k: intArray)
    {
      Sop (k);
    }

    Arrays.sort (intArray);
    Sop ("array elements after sort:");
    for (int k: intArray)
    {
      Sop (k);
    }

    Sop ("Program ends...");
  }
}
```

before sort

0	67	int type
1	12	-"-
2	35	-"-
3	28	-"-
4	41	-"-

after sort

12	0
28	1
35	2
41	3
67	4

O/P: Program starts:
array elements before sort:
67
12
35
28
41

Continued....

array elements after sort
12
28
35
41
67
Program ends...

3) P111

```
package com.jspiders.arrays;
import java.util.Arrays;
public class Demo3
{
    psvm (String[] args)
    {
        Sop ("Program starts...");
        String[] strArray = new String[5]; // array of String type
        strArray[0] = "demo";
        strArray[1] = "zen";
        strArray[2] = "abc";
        strArray[3] = "Sample";
        strArray[4] = "132";
        Sop ("array elements");
        for (String s1 : strArray)
        {   Sop (s1);
        }
        Arrays.sort (strArray);
        Sop ("sorted elements");
        for (String s1 : strArray)
        {   Sop (s1);
        }
        Sop ("Program ends...");
    }
}
```

O/p :-

```
Program starts...
array elements
demo
zen
abc
Sample
132
```

continued...

```
sorted elements
132
abc
demo
Sample
zen
Program ends...
```

4)
```
package com.jspiders.arrays;
class A
{
}
class B extends A
{
}
class C extends B
{
}
public class Demo4
{
  psvm(String[] args)
  {
    Sop("Program starts...");
    A[] array1 = new A[5]; // array of A type
    array1[0] = new A();
    array1[1] = new C(); // C casted to A type
    array1[2] = new B(); // B casted to A type
    array1[3] = new A();
    array1[4] = new C(); // C casted to A type
    Sop("Array elements");
    for(A a1: array1)
    {
      Sop(a1);
    }
    Sop("Program ends...");
  }
}
```

19-02-2013 | Tuesday

Arrays

1*) Array is a Collection of homosenins
In an array we can store similar

2*) Derived array are declared using derived Class.

3*) Array can be intialized using
* Dimension
* Array intializer

4*) Ex:- Dimension:
int[] array1 = new int[5]

While using dimension the size of the array should be specified. Whenever an array is intialized using dimension the default value will be stored in each element.

5*) Ex:- Array Initialization:
int[] array1 = {10,20,30,40,50};

Using array initializer the elements are specified with comma separation.

6*) We can refer each element of an array using index.

7*) Java provides class named Array which contains several methods like sort and search...

8*) If we declare the array of Object type, we can store array type of data.

9*) In this type of array each element is of Object type.

P113

3)
```
import java.util.Scanner
class A
{
}
class B
{
}
~~public class Demo~~
```

```
public class Demo1
{
  psvm (String[] args)
  {
    Object[] objArray = new Object[5];
    ObjArray[0] = "test";
    ObjArray[1] = new class A;
    ObjArray[2] = new class B;
    ObjArray[3] = new StringBuilder("demo");
    ObjArray[4] = new Scanner(System.in);
    Sop("obj array elements");
    for (Object obj : objArray)
    {
      Sop(Obj);
    }
    Sop("Program ends...");
  }
}
```

O/p:-
```
Program starts...
Object array element
test
Com.jspiders.arraydemo.A@3eca90
Com.js
demo
java.util.Scanner[delimiters
```

Wrapper Class

1) Java provides Wrapper class to convert primitive type to an Object type.

2) For every primitive type, Wrapper class type is existing

3) All wrapper classes are available in java.lang package.

4) The numeric inherits an abstract class by Name Number.

5) All Wrapper Classes are final Class.

6) In each Wrapper Class the toString() method, equals() and hashcode() methods are Overrided.

Data type	Wrapper Class	
byte	Byte	byte Obj
short	Short	short obj
int	Integer	int obj
long	Long	long obj
float	Float	float obj
double	Double	double obj
char	Character	char obj
boolean	Boolean	boolean obj

P114

4) package com.jspiders.arraydemo;

```
public class Demo2
{ psvm (String[] args)
  {
    Sop ("Program starts...");
    Integer intObj = new Integer(56); // boxing Operation
/*  Integer intObj = 56;  auto boxing [from jdk 1.5 onwards]*/
    Sop ("intObj:" + intObj);
    int k = intObj.intValue(); // unboxing operation
    Sop ("k = " + k);
    Sop ("Program ends...");
  }
}
```

O/P:

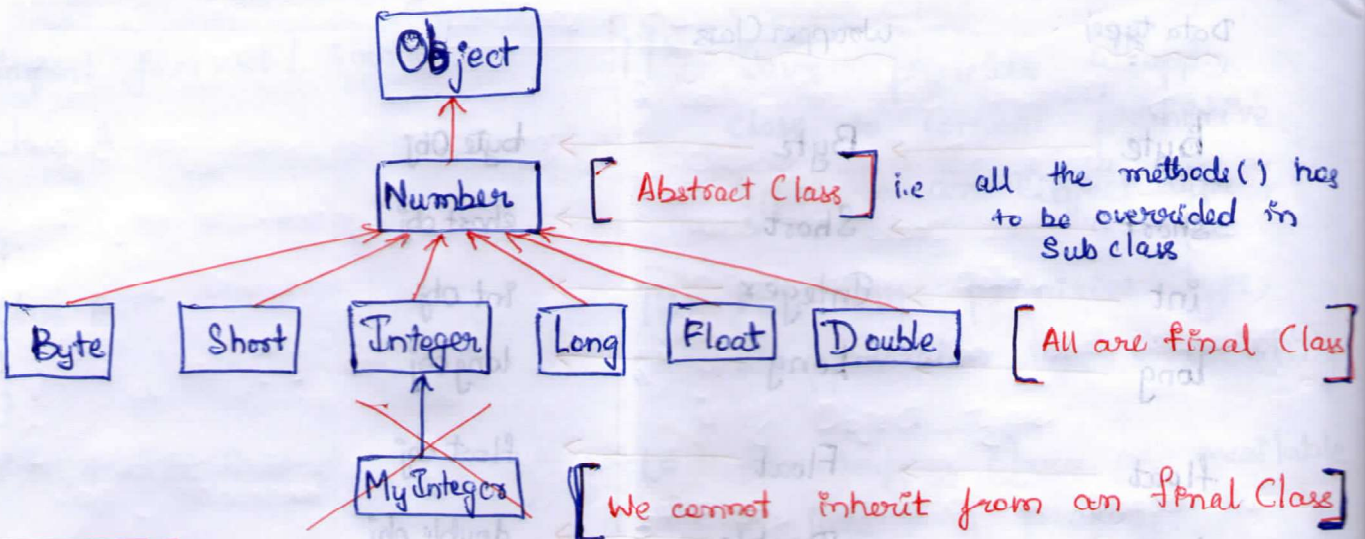
```
Program starts
intObj: 56
k: 56
Program ends
```

17) Converting primitive type to Object type is known as boxing operation. The boxing operation is done by using Wrapper Class.

18) Converting a boxed Object back to a primitive type is known as unboxing operation

19) If compiler does boxing on its own its known as auto boxing.

20) Each wrapper class provides a method to get the primitive value [intValue()].



P115

5)
```
package com.jspiders.arraydemo3;
import java.util.Scanner;
class A
{
}
class B
{
}
public class Demo3
{
  psvm (String[] args)
  {
  Sop ("Program starts...");
  Object[] objArray = new Object[5];
  objArray[0] = "test"; //auto upcast
  objArray[1] = new A(); //auto upcast
  objArray[2] = new B(); //auto upcast
// auto boxing and auto upcast
  objArray[3] = 12;
//auto boxing and auto upcast
  objArray[4] = true;
  Sop ("Object array elements");
  for (Object obj : objArray);
  {  Sop(obj);
  }
  A a1 = (A) objArray[1]; // explicit downcast
```

* In an array of Object type whenever any element is added, all the data are casted to Object type.

* If we had a primitive type data to Object type Array, first the primitive type is converted to Wrapper type object, next the Wrapper type is casted to Object type. Both auto boxing & auto upcasting happens.

Int Q: How does an data is stored in Object type array?

* When Object array element is read, the data will be in a Object format, that data has to be downcasted to the respective Sub class type.

```
int i = (int) objArray[3]; // type casted to int type
Sop(a1);
Sop(i);
Sop("Program ends...");
}
}
```

O/p:- Program starts...

Object array element

---

* * * * * * *