The books on core java currently existing in market are conventional in their approach in putting the core concepts and the reader has hard time in absorbing the core concepts from the conventional approach employed in the book. Further, in most of the occasions reader looks for quick solutions to the frequent questions that arise in his mind on core concepts. The current book uses experiential learning approach where reader plays an active role. The book is thought provoking and stimulates the reader to think beyond basics. Authors through their experience of teaching java for more than 15 years found that there is no book in market which has a pin point focus only on practical concepts with needed theory. There is a need for book which can give hands on approach to students for their core java concepts and handy reference to the candidates to prepare for their technical aptitudes as well as interview. The book will play a key role for the students during their academics and even during their job hunting. Authors have focused on two distinct approaches while writing the book, first one is to suspect and provide answers to the questions which arise in students mind while studying java and second one is what an expert might expect from a new comer in the IT industry through their interactions with students and alumni working in IT industries. . It's a collection of simple and tricky questions with practically tested answers. Every program is associated with the output produced by it. Language used is very simple with necessary comments wherever required.

ISBN 978-93-85882-87-6

Publisher & Owner
Archana Rajendra Ghodke
Harshwardhan Publication Pvt.Ltd.
At.Post.Limbaganesh, Tq.Dist.Beed
Pin-431126 (Maharashtra)
Cell : 9850203295

A Comprehensive Guide to Crack Java Aptitude

A Comprehensive Guide to Crack Java Aptitude

Dr. Poornima G. Naik
Dr. Kavita S. Oza

A Comprehensive Guide to Crack Java Aptitude

By

Dr. Poornima G. Naik

Dr. Kavita S. Oza

Reg.No.U74120 MH2013 PTC 251205

Harshwardhan Publication Pvt.Ltd.

At.Post.Limbaganesh,Tq.Dist.Beed

Pin-431126 (Maharashtra) Cell:07588057695,09850203295

harshwardhanpubli@gmail.com, vaidyawarta@gmail.com

All Types Educational & Reference Book Publisher & Distributors

A Comprehensive Guide to Crack Java Aptitude

© Dr. Poornima G. Naik

Dr. Kavita S. Oza

❖ **Publisher :**
Harshwardhan Publication Pvt.Ltd.
Limbaganesh, Dist. Beed (Maharashtra)
Pin-431126, vidyawarta@gmail.com

❖ **Printed by :**
Harshwardhan Publication Pvt.Ltd.
Limbaganesh, Dist. Beed, Pin-431126

❖ **Page design & Cover :**
Shaikh Jahurodden

❖ **Edition: July 2017**

ISBN 978-93-85882-87-6

❖ **Price : 399/ -**

Acknowledgements

Many individuals share credit for this book's preparation. We extend our sincere thanks to Late Prof. A.D.Shinde, the Founder Director and Managing Trustee who has been a constant source of inspiration for us throughout our career. His support is really a driving force for us. Also, we would like to thank Dr.R.A.Shinde, Hon'ble Secretary, CSIBER for his whole hearted support and continuous encouragement. We are grateful to Dr. M.M.Ali, Director CSIBER for his invaluable guidance. We take this opportunity to thank Dr.V.M.Hilage, Trustee Member, CSIBER, Dr. R.V.Kulkarni, H.O.D., Department of Computer Studies, Dr. R.R.Mudholkar, H.O.D., Department of Computer Science, Shivaji University for showing a keen interest in the matter of this book and extending all support facilities for the in-timely completion of this book. The material covered in this book is a systematic effort taken towards solving the various queries which we received from our students time to time, during our 15+ years tenure of teaching Java at PG level. Last but not the least we thank all faculty members and non-teaching staff of department of computer studies, CSIBER, Kolhapur and department of Computer Science, Shivaji University, Kolhapur who have made contribution to this book either directly or indirectly.

Dr. Poornima.G.Naik
Dr. Kavita.S.Oza

Preface

It gives us an immense pleasure to bring out a book on Java concepts entitled '**A Comprehensive Guide to Crack Java Aptitude**'. This book is intended to serve those who have just started their tour of Core Java. In this book, we have tried to stick to how, why and what of Java rather than what a typical book of Java offers. It serves as a reference material for those who have just started learning Java and want to learn more of Java internals. This book will help you to fill in the "gaps" by answering several 'why' and 'how to' questions in Java. To understand the contents of this book a working knowledge of Java and some basic concepts of Java are required. This is not a complete reference to Java, so some familiarity with Java is essential.

The books on core java currently existing in market are conventional in their approach in putting the core concepts and the reader has hard time in absorbing the core concepts from the conventional approach employed in the book. Further, in most of the occasions reader looks for quick solutions to the frequent questions that arise in his mind on core concepts. The current book uses experiential learning approach where reader plays an active role. The book is thought provoking and stimulates the reader to think beyond basics. Authors through their experience of teaching java for more than 15 years found that there is no book in market which has a pin point focus only on practical concepts with needed theory. There is a need for book which can give hands on approach to students for their core java concepts and handy reference to the candidates to prepare for their technical aptitudes as well as interview. The book will play a key role for the students during their academics and even during their job hunting. Authors have focused on two distinct approaches while writing the book, first one is to suspect and provide answers to the questions which arise in students mind while studying java and second one is what an expert might expect from a new comer in the IT industry through their interactions with students and alumni working in IT industries. . It's a collection of simple and tricky questions with practically tested answers. Every program is associated with the output produced by it. Language used is very simple with necessary comments wherever required.

Java is evolving like anything beyond leaps and bounds and all these advanced Java technologies demand the thorough understanding of core Java concepts. In short Java is become heart of computer science curricula and basic need for employability. In view of this, the book

aims at providing the ins and outs of java programming language. Java has already seen as era of computing but still remains as one of the key language inspite of C,C++, python, C# etc. Its popularity is due to its wide applicability. Most of the Employers check student's logic for employability using core concepts of Java. The book aims at providing the same.

Moving from C++ to Java requires learning about the new development environment rich in API and application scope which is a challenging task from the programmer's perspective. When you move away from the familiar tools and a familiar environment to the new one, the new tools often seem awkward as compared to the old favourites and you soon become frustrated loosing your way. To put you on the right track at the beginning what is needed is either a proper guidance or a good book clarifying the basic concepts and creating interest in the subject. From the end user's point of view learning the new environment is a matter of a day or two. But from the programmer's point of view developing application for the new environment is a new ball game.

Java is one big complex world and it demands a systematic effort to reach to its roots. In this volume we have started with the most basic concepts of Java and have adopted a simple one-idea-at-a-time approach to the Java package hierarchy. Instead of mixing several concepts and ideas in a single program and making it complex and difficult to comprehend we have started with the most fundamental concept that the beginners need to know and showed him/her how to master that concept only and then we have introduced the next concept. We have observed that when exposed to a steady programming of clear ideas and exercises many students have been able to truly master the fundamental concepts of Java.

Note to a Reader

The book is the result of the questions collected from the various resources available on the internet whose references are provided at the end of the book. The authors have tried to provide more conceptual insight in to the c constructs unleashing the Java internals both textually and visually on many occasions. Few concepts described in the book are borrowed from one of the references included at the end of this book which are supplemented with suitable working examples in order to make the concepts easy to comprehend.

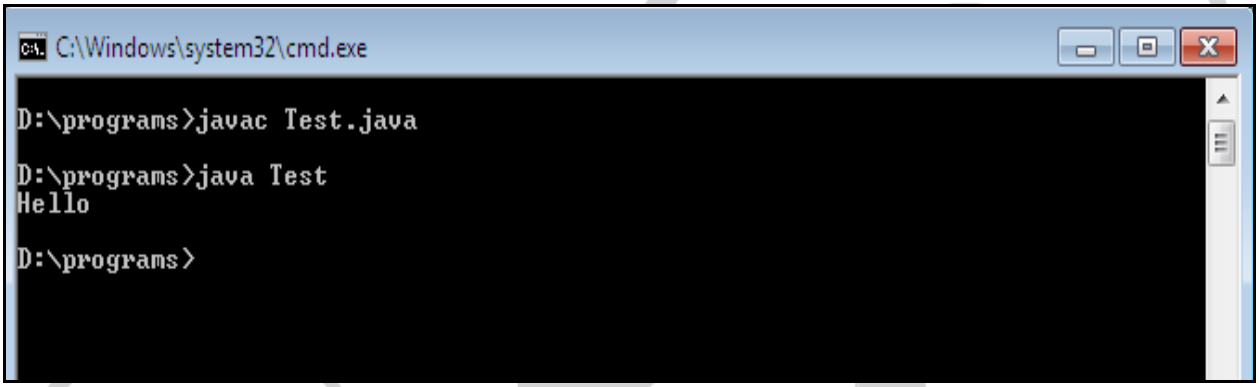
Dr. Poornima. G. Naik
Dr. Kavita .S. Oza

Q. No. 1 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
abstract class Test
 {
   public static void main(String args[])
   {
      System.out.println("Hello");
   }
}
```

Output :



Explanation :

In the given program, Test class is declared as abstract. Also, the main method is declared as static. Hence the main() method is invoked by JVM without creating the object of Test class employing the class name as shown below:

Test.main();

Since we can't create object of abstract classes in Java, it is guaranteed that object of class with main() is not created by JVM. Hence the given program does not generate any compilation time error and generates the output Hello.

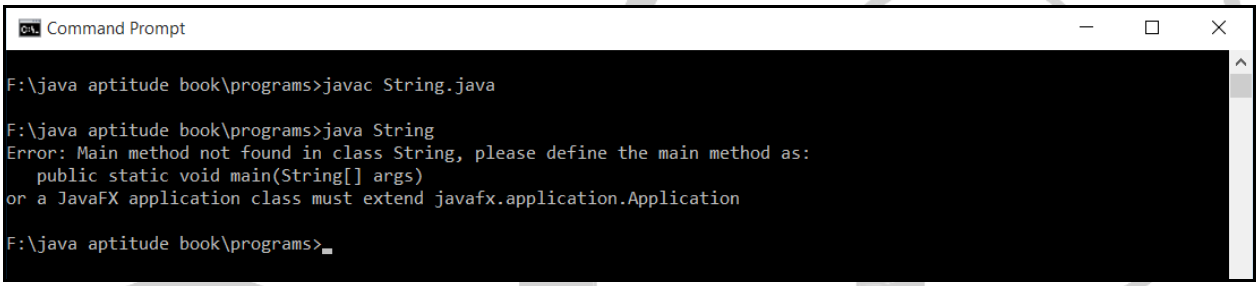
Q. No. 2 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
public class String
{
   public static void main (String[] args)
   {
         System.out.println("Hello!");
   }
}
```

Output :

The program compiles successfully but generates the following runtime error.



Explanation :

String is a predefined class name residing in java.lang package and java.lang package is included in every java program by default. In the given program String class is defined in default package. Hence now there are two String classes available, one in java.lang package and another in default package.

During the execution of the above program, the Main thread is looking for *main* method() with predefined String class array argument. But here, it found *main* method() with user defined String class. Whenever Main thread sees a class name, it tries to search that class scope by scope.

- First it looks for the class in the program,
- Second it searches the package.
- If not found, then JVM follows delegation hierarchy principle to load that class.

  If all the above searches fail, then ultimately a run-time error is generated.

In the given program, the String class in the default package contains a main method with the following prototype.

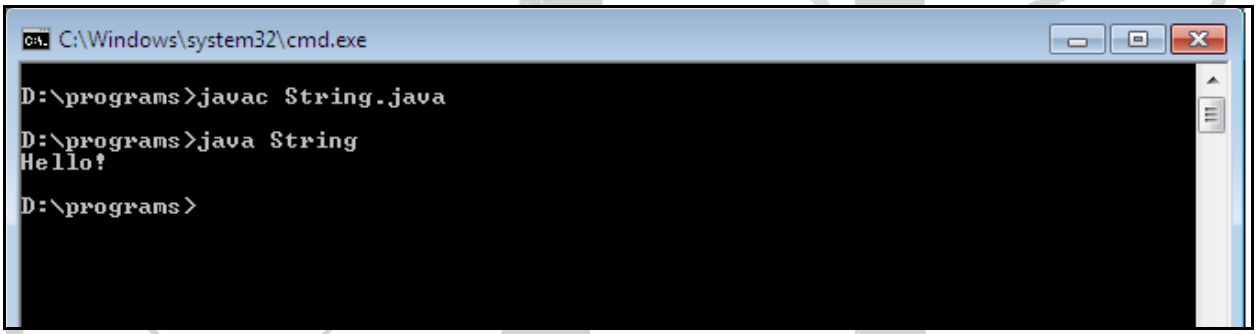
public static void main(String[])

where the argument to the main() method is a String class defined in default package and not the one existing in java.lang package. Hence the generated error.

To run above program successfully fixing the error, provide full path of String class, i.e. *String*. Modify the program as shown below and re-execute it.

```
public class String
{
   public static void main (String[] args)
   {
         System.out.println("Hello!");
   }
}
```

On execution of the program now, the following output is generated.



Q. No. 3 *Category : Data Types*

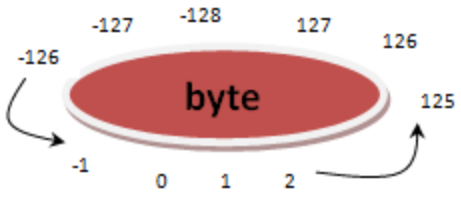
What is the output generated on execution of the following Java Program?

```
public class Test2
{
   public static void main(String args[])
   {
      byte a = 126;
      System.out.println(a);
      a+=10;
      System.out.println(a);
   }
}
```

Output :

Explanation :

The range of the char data type is 127 to -128. The byte data type shows cyclic properties, i.e. if the byte variable is increased beyond its maximum value, then it attains the minimum value of -128. Similarly, if the char variable is decreased beyond its minimum value, then it attains a maximum value of 127, in a cyclic order as shown below:



Hence the variable c is incremented as shown in the following table:

126 +1=		127
126 + 2	=	128
126 + 3	=	-127
126 + 4	=	-126
126 + 5	=	-125
126 + 6	=	-124
126 + 7	=	-123
126 + 8	=	-122
126 + 9	=	-121
126 + 10	=	-120

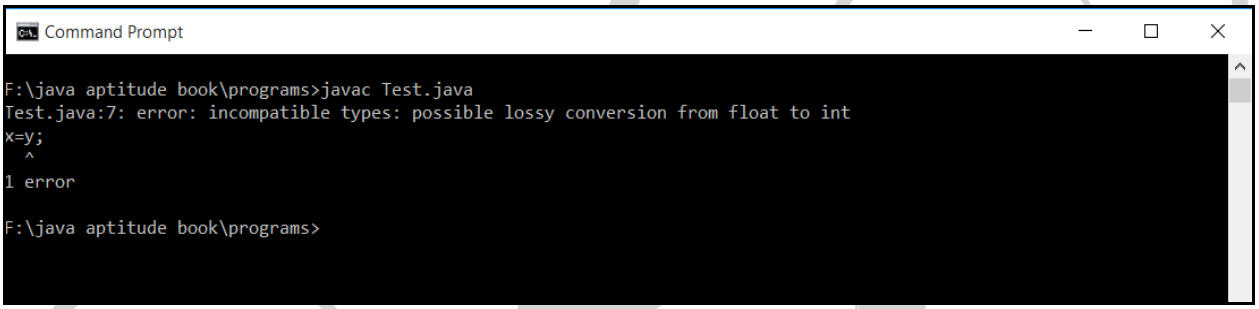
Hence the given program generates the output -120.

Q. No. 4 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test
{
public static void main(String[] args)
{
        int x;
        float y=10;
        x=y; ← Statement 7
        System.out.println("x : " + x);
}
}
```

Output :



Explanation :

In the given program, the variable x is of type int whereas y is a floating point variable. Since java is a strongly typed language, it does not permit assigning a wider data type (y) to a narrower data type (x). Hence a compiler error is generated.

However, the error can be removed by replacing the statement 7) in the above program with the following statement :

x=(int)y;

Modify the program as shown below and re-execute it. (Modified statement is shown in bold).

```
public class Test
{
    public static void main(String[] args)
   {
          int x;
          float y=10;
          x=(int)y;
          System.out.println("x : " + x);
   }
}
```

On re-execution of the program, the following output is generated.

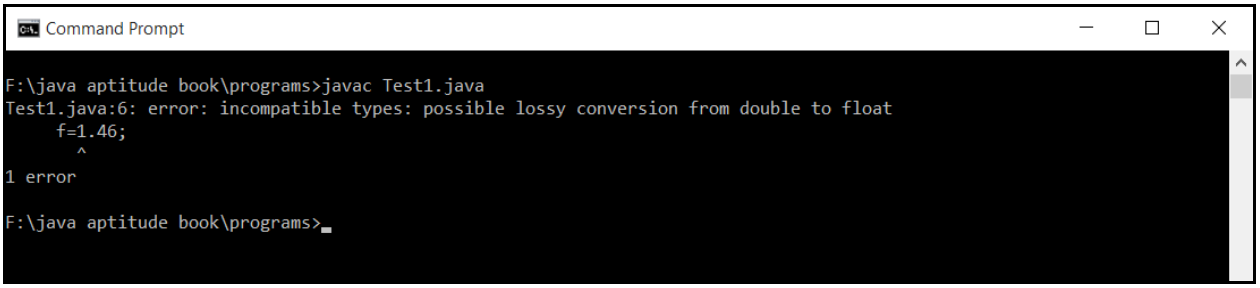
Q. No. 5 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test1
{
  public static void main(String args[])
  {
    float f;
    f=1.46;
    System.out.println("f : " + f);
  }
}
```

Statement 6

Output :



Explanation :

In Java, floating point literals are of type double by default and since Java is a strongly typed language, floating point literal cannot be assigned to a variable of type float. However, you can change the literal to make it a float by appending f or F at the end of the floating point literal. Hence the above error can be removed by replacing the statement (6) in the above program with the statement shown below:

x=1.46f;

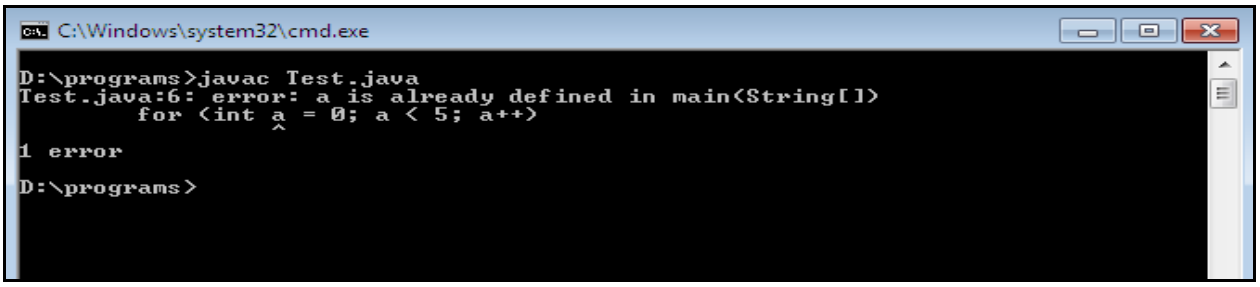
On re-execution of the program, the following output is generated.

Q. No. 6 *Category : Scope and Life Time of Variables*

What is the output generated on execution of the following Java Program?

```
class Test
{
   public static void main(String args[])
   {
      int a = 5;
      for (int a = 0; a < 5; a++)
      {
         System.out.println(a);
      }
   }
}
```

Output :



Explanation :

In the given program, variable ‘a’ is re-declared with the same block scope which is not allowed. The scope of the variable declared in the for loop is considered to be same as the scope of a block in which the loop exists. Re-declaration of a variable is not allowed in any language.

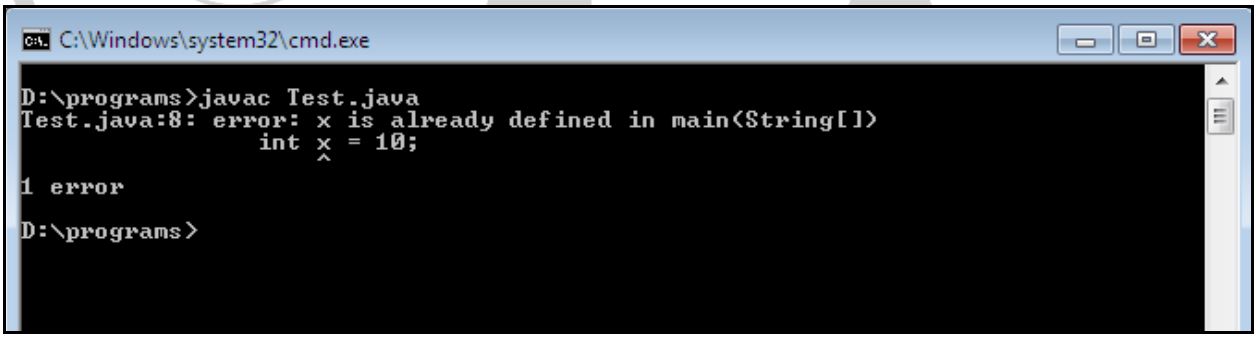
A similar program in C++ works since the scope of a variable declared in for loop and the contained block are considered to be different.

Consider the following java program:

```
class Test
{
   public static void main(String args[])
   {
      {
         int x = 5;
         {
            int x = 10;←Statement 8
            System.out.println(x);
         }
      }
   }
}
```

In the above program, variable x is re-declared again in statement 8).

On execution of the program, the following output is generated for the similar reason.

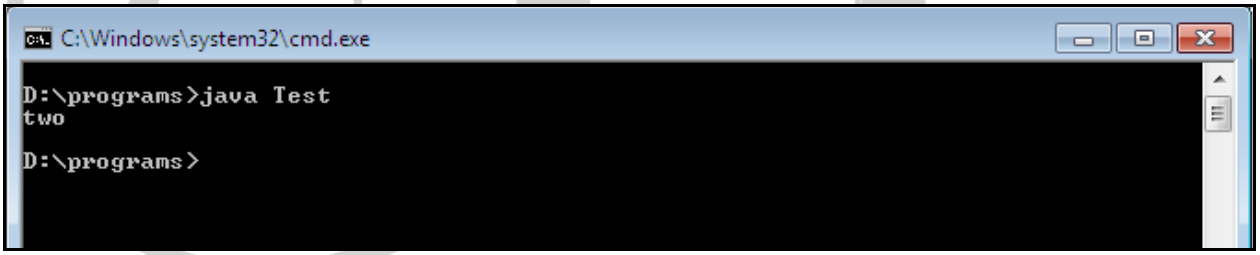


Q. No. 7 *Category : Control Statements*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main(String[] args)
   {
      String str = "two";
      switch(str)
      {
         case "one":
            System.out.println("one");
            break;
         case "two":
            System.out.println("two");
            break;
         case "three":
            System.out.println("three");
            break;
         default:
            System.out.println("no match");
      }
   }
}
```

Output :



Explanation :

The given program, compiles and executes successfully in JDK 1.8.

Beginning with JDK 7, we can use a string literal/constant to control a switch statement, which is not possible in C/C++.

Q. No. 8 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
public class Test2
{
   public static void main(String[] args)
{
        Test1 t1 = new Test1();
        t1.fun(5);
 }
}

class Test1 {
   void fun(int x)
 {
       System.out.println("fun() of Test1 called: x = " + x);
   }
}
```

Output :

Explanation :

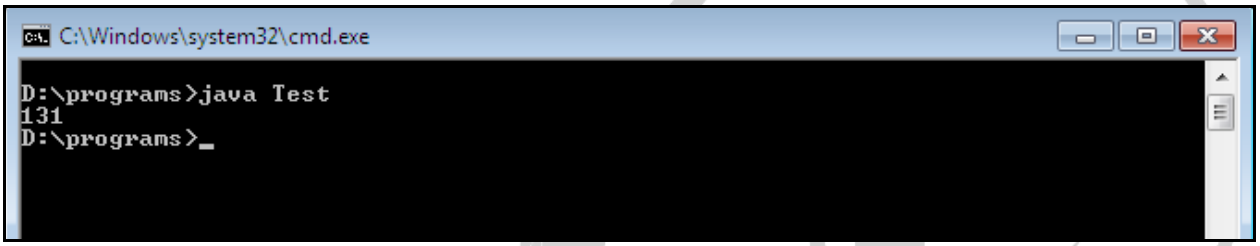
The Java program compiles and runs fine. Note that *Test1* and *fun()* are not declared before their use. Unlike C++, we don't need forward declarations in Java. Identifiers (class and method names) are recognized automatically from source files. Similarly, library methods are directly read from the libraries, and there is no need to create header files with declarations. Java uses naming scheme where package and public class names must follow directory and file names respectively. This naming scheme allows Java compiler to locate library files without any hitch.

Q. No. 9 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main(String[] args)
   {
      System.out.print('A' + 'B');
   }
}
```

Output :



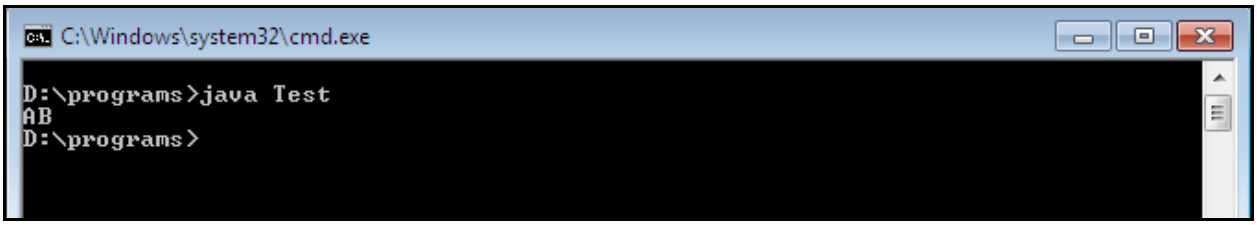
Explanation :

When we use single quotes, the characters 'A' and 'B' are converted to int. This is called widening primitive conversion. After conversion to integer, the numbers are added ('A' is 65 and 'B' is 66) and 131 is printed.

Modify the program as shown below and re-execute it.

```
public class Test
{
   public static void main(String[] args)
   {
      System.out.print('A');
      System.out.print('B');
   }
}
```

On execution of the program, the following output is generated.



The widening primitive conversion happens only when '+' operator is present.

Widening primitive conversion is applied to convert either or both operands as specified by the following rules. The result of adding Java chars, shorts or bytes is an int:

- If either operand is of type double, the other is converted to double.
- Otherwise, if either operand is of type float, the other is converted to float.
- Otherwise, if either operand is of type long, the other is converted to long.
- Otherwise, both operands are converted to type int

Q. No. 10 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main(String[] args)
   {
      int i = 257;
      System.out.println("Conversion of int to byte.");
      byte b = (byte) i;
      System.out.println("i : " + i + " b : " + b);
   }
}
```

Output :

Explanation :

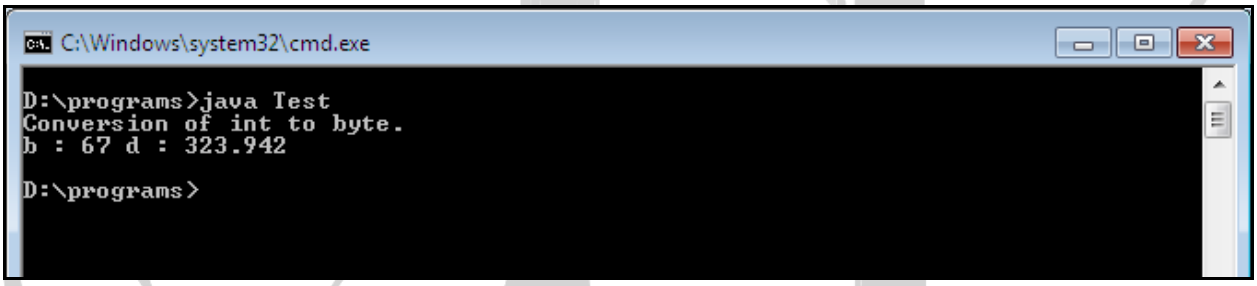
The range of byte data type is 0 to 256.

While assigning integer value to byte type, the number is reduced to modulo 256(range of byte).

While assigning double value to byte type, the fractional part is lost and then the number is reduced to modulo 256(range of byte) as shown in the following java program.

```
public class Test
{
   public static void main(String[] args)
   {
      double d = 323.942;
      System.out.println("Conversion of int to byte.");
      byte b = (byte) d;
      System.out.println("b : " + b + " d : " + d);
   }
}
```

On execution of the above java program, the following output is generated.



Q. No. 11 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
class Test2
{
        public static void main(String args[])
        {
                if (args == null)
                        System.out.println("No arguments!");
        }
}
```

Output :

Explanation :

If no command-line arguments are passed to the program, then args will be empty and not null. Hence the given java program does not generate any output. Now, replace if statement in the above program with the statement given below:

if (args.length == 0)

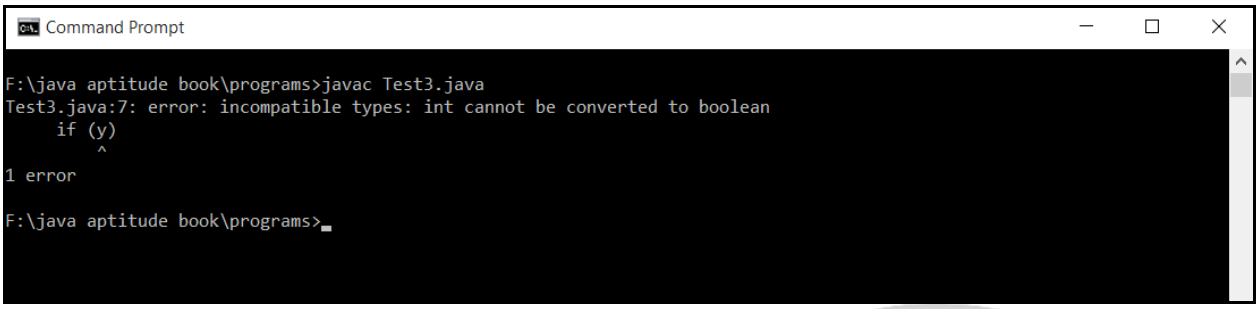
Compile and re-execute the program, the following output is generated.

Q. No. 12 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test3
{
  public static void main(String args[])
  {
    int y=1;
    if (y)                                  Statement 6
        System.out.println("true");
    else
        System.out.println("false");
  }
}
```

Output :



Explanation :

Boolean values can only be true or false and cannot be 0 or 1 or other numeric values as in other programming languages like C. This is part of Java's strong data typing. Hence the above error can be removed by replacing statement (6) in the above program with the following statement :

if (y==1)

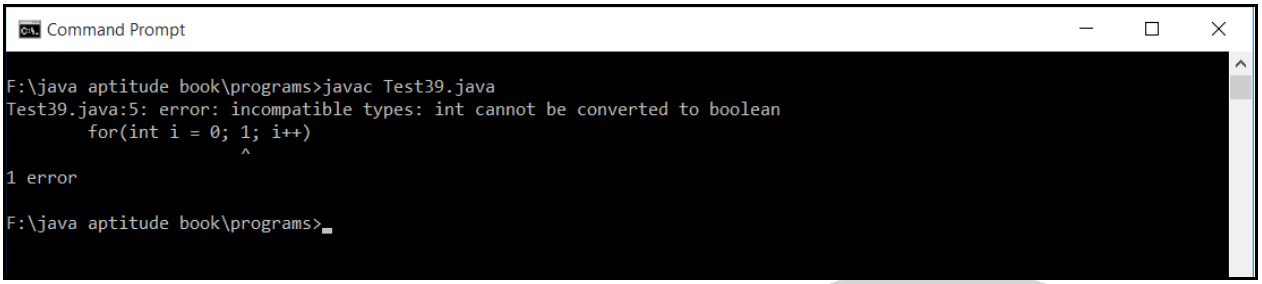
On re-execution of the above program with the modification suggested above, the following output is generated.

Q. No. 13 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
class Test39
 {
   public static void main(String[] args)
   {
      for(int i = 0; 1; i++)
      {
         System.out.println("Hello");
         break;
      }
   }
}
```

Output :



Explanation :

There is an error in condition check expression of for loop. Java differs from C++(or C) here. C++ considers all non-zero values as true and 0 as false. Unlike C++, an integer value expression cannot be placed where a boolean expression is expected in Java. The correct version of the above program is

```
class Test39
 {
   public static void main(String[] args)
   {
      for(int i = 0; true; i++)
      {
         System.out.println("Hello");
         break;
      }
   }
}
```

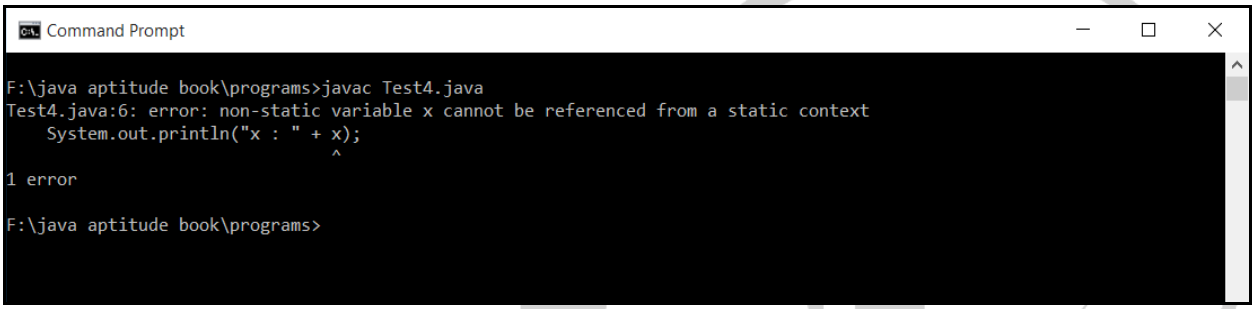
On execution of the above program, the following output is generated.

Q. No. 14 *Category : Variable Declaration Issues*

What is the output generated on execution of the following Java Program?

```
public class Test4
{
  int x=10; ← Statement 3
  public static void main(String args[])
  {
    System.out.println("x : " + x);
  }
}
```

Output :



Explanation :

In the given program, x is a non-static variable which cannot be referenced in a static method, main(). Hence the error.

The error can be removed in two ways.

Method 1 : By declaring the variable x as static. Replace the statement (3) in the above program with the statement given below :

```
static int x=10;
```

Method 2 : By creating the object of class Test and then, accessing the variable x as shown below:

```
public class Test
{
  int x=10;
  public static void main(String args[])
  {
    Test obj = new Test();
    System.out.println("x : " + obj.x);
  }
}
```

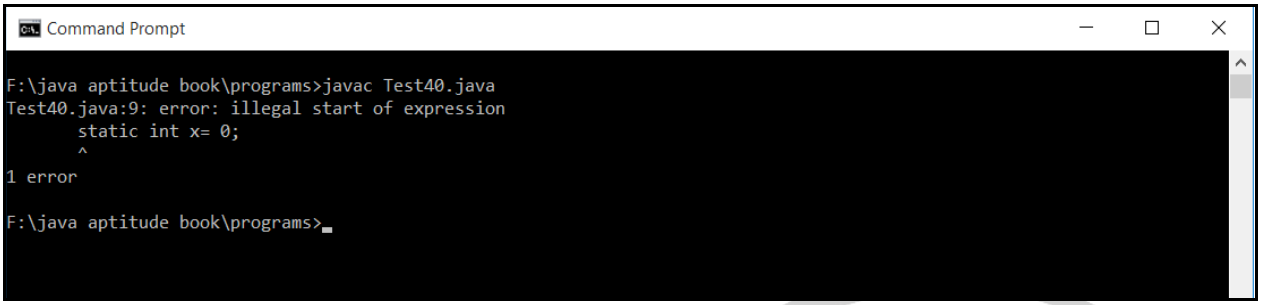
On incorporating either of the two changes and re-executing the program, the following output is generated.

Q. No. 15 *Category : Variable Declaration Issues*

What is the output generated on execution of the following Java Program?

```
class Test40
 {
   public static void main(String args[])
   {
      System.out.println(fun());
   }
   static int fun()
   {
      static int x= 0;
      return ++x;
   }
}
```

Output :



Explanation :

Unlike C++, static local variables are not allowed in Java. We can have static members with class scope to count number of function calls and other purposes that C++ local static variables serve but the local static variables generate error. The correct version of the program is

```
class Test40
 {
  public static void main(String args[])
  {
     System.out.println(fun());
  }
  static int fun()
  {
     int x= 0;
     return ++x;
  }
}
```

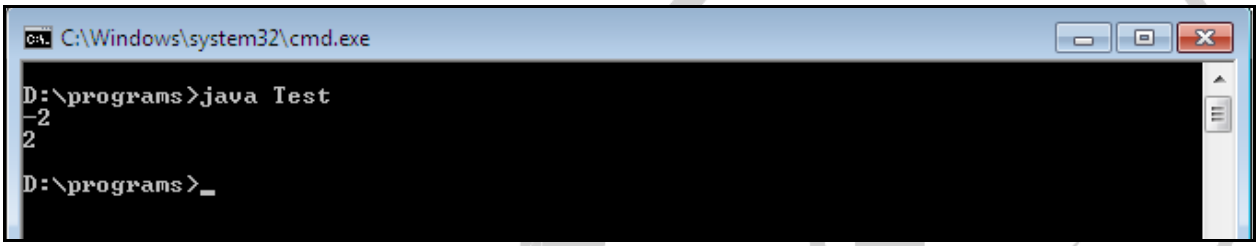
On execution of the above program, the following output is generated.

Q. No. 16 *Category : Operators*

What is the output generated on execution of the following Java Program?

```
class Test
{
   public static void main(String args[])
   {
      int x = -4;
      System.out.println(x>>1);
      int y = 4;
      System.out.println(y>>1);
   }
}
```

Output :



Explanation :

In C/C++ there is only one right shift operator '>>' which should be used only for positive integers or unsigned integers. Unlike C++, Java supports following two right shift operators.

- >> (Signed right shift operator) and
- >>> (Unsigned right shift operator)

The operator '>>' uses the sign bit (left most bit) to fill the trailing positions after shift. If the number is negative, then 1 is used as a filler and if the number is positive, then 0 is used as a filler.

In Java, the operator '>>>' is unsigned right shift operator. It always fills 0 irrespective of the sign of the number.

The binary representation of -4 is (32 bits)

1	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*
0	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*0*	*1*	*0*	*0*

After performing the operation,

-4 >> 1

the bits are shifted to the right except the sign bit and we get,

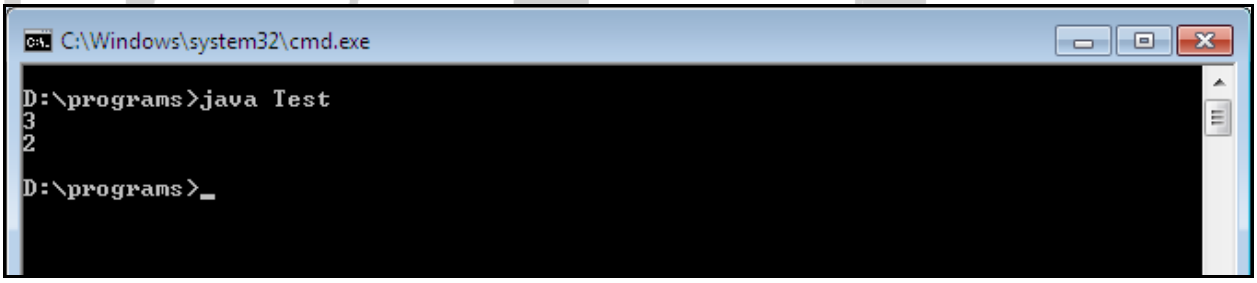
1	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***
0	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***

0 is used as a filler. The same argument applies to shifting the bits representing the decimal no. 4 where 0 is used for the sign bit since the number is positive.

Consider the following Java program:

```
class Test
{
   public static void main(String args[])
   {
     int x = -4;
     System.out.println(x>>>30);
     int y = 4;
     System.out.println(y>>>1);
   }
}
```

On execution of the above program, the following output is generated.



The binary representation of -4 is (32 bits)

1	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***
0	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***0***	***1***	***0***	***0***

x is stored using 32 bit 2's complement form.

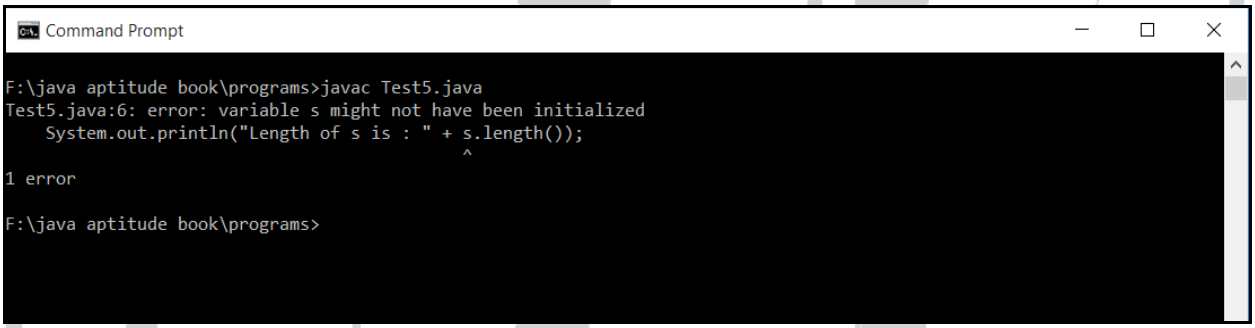
Since the sign bit used for representing the positive integer is 0, using >>> operator is equivalent to dividing the number by 2 as is revealed in the second output generated in the above program.

Q. No. 17 *Category : Variable Initialization Issues*

What is the output generated on execution of the following Java Program?

```
public class Test5
{
  public static void main(String args[])
  {
    String s;← Statement 5
    System.out.println("Length of s is : " + s.length());
  }
}
```

Output :

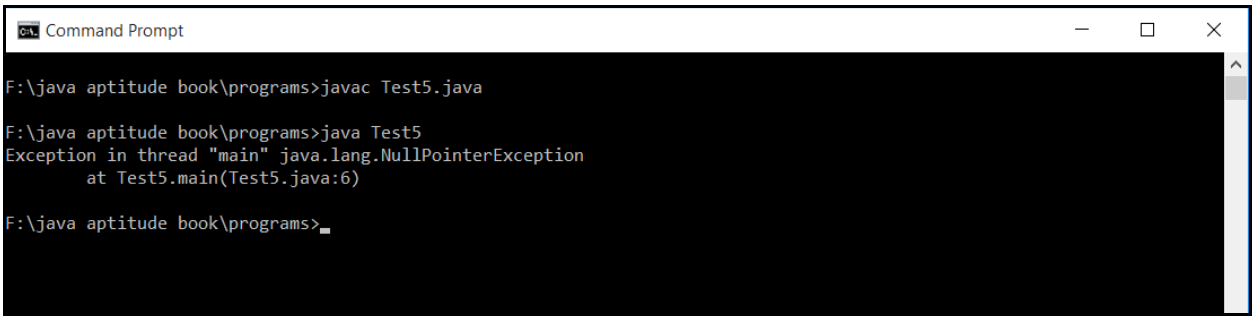


Explanation :

If the statement (5) in the above program is replaced with the one given below,

String s=null;

and executed, the program will successfully compile but the following runtime error message is displayed.



To remove the runtime error, replace the statement (5) in the above program with the one given below:

String s="";

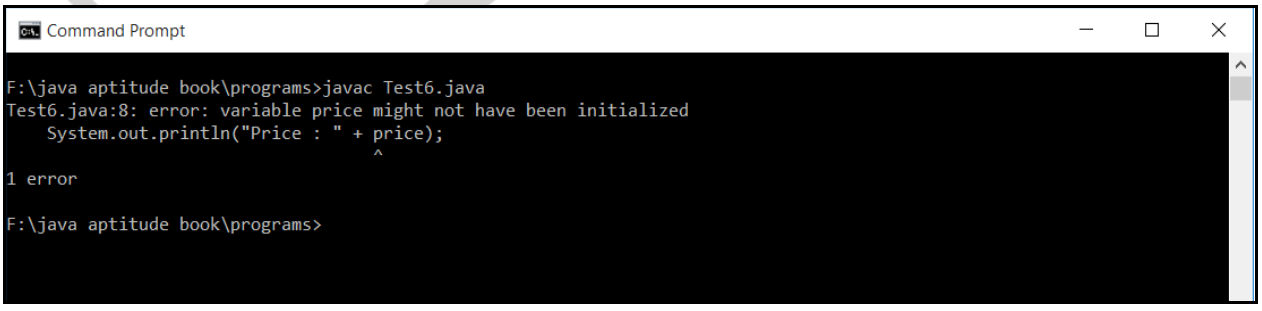
Now, the variable s is initialized and references a valid string of length zero. Hence the generated output.

Q. No. 18 *Category : Variable Declaration Issues*

What is the output generated on execution of the following Java Program?

```
public class Test5
{
  public static void main(String args[])
  {
    int weight=10, price;← Statement 5
    if (weight <= 10)
      price=100;
    System.out.println("Price : " + price);
  }
}
```

Output :



Explanation :

In Java, it is mandatory to initialize all local variables. The compiler will consider all possible paths of execution and check whether all local variables are initialized across all paths.

The above error can be removed either by initializing the price by replacing the statement (5) in the above program with the one given below:

int weight=10, price=0;

OR

Adding else part of if statement as shown below:

else

price=500;

Modifying the variable declaration statement as shown below and re-executing the program, the following output is generated.

int weight=10, price=0;

OR

Modify the program as shown below and re-execute the program.

```
public class Test6
{
 public static void main(String args[])
 {
  int weight=10, price;
  if (weight <= 10)
    price=100;
  else
    price=200;
  System.out.println("Price : " + price);
 }
}
```

On execution of the above program, the following output is generated.

Q. No. 19 *Category : String Operations*

What is the output generated on execution of the following Java Program?

```
public class Test26
{
    public static void main(String args[])
    {
            String s="Java";
            s.concat(" Programming");
           System.out.println(s);
    }
}
```

The reason for this is that in java, string objects are immutable. Immutable simply means unmodifiable or unchangeable. Once string object is created its data or state can't be changed without creating a new string object. In the above example two String objects are created. “Java” and “Java Programming”, but the reference variable s still references the first String object “Java”. This scenario can be changed as shown in the following example.

```
public class Test
{
    public static void main(String args[])
    {
      String s="Java";
      s=s.concat(" Programming");
      System.out.println(s);
    }
}
```

In the above example, we explicitly assign the new String object "Java Programming" to a reference variable s. Hence, s will deference the String object "Java" and will now reference a new String object "Java Programming". Hence on execution of the above program the following output is generated.

Q. No. 20 *Category : String Operations*

What is the output generated on execution of the following Java Program?

```
public class Test7
{
        public static void main(String args[])
        {
                String s1=new String("Java"); ← Statement 5
                String s2=new String("Java"); ← Statement 6
                if (s1==s2) ←Statement 7
                     System.out.println("s1 and s2 are equal");
                else
                    System.out.println("s1 and s2 are not equal");
}
}
```

Output :

Explanation :

When the new operator is used for creating the String object, the objects are created on heap which are referenced by s1 and s2 in statements 5) and 6), respectively in the given program. Hence s1 and s2 refer to the addresses of the two String objects and not their contents. Hence == operator used in statement 7) compares the addresses of two String objects and not their contents. For comparing the content of two String objects use equals() method of String class instead of == operator.

Perform the following changes in the above program.

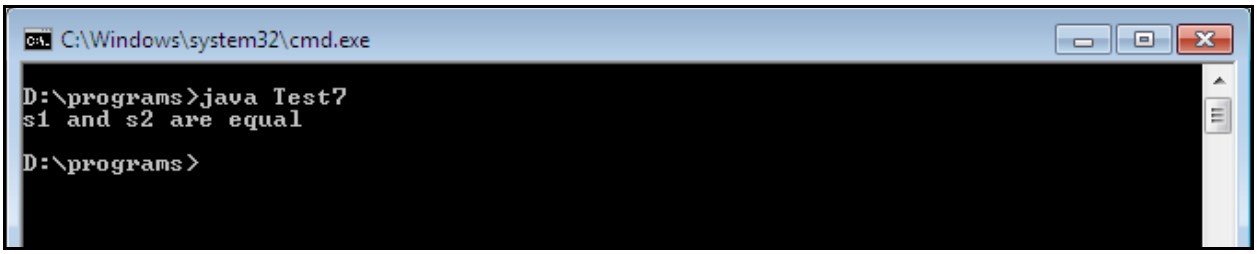
Change the statement (7) in the above program with the statement given below:

if (s1.equals(s2))

Re-write the above program using equals() method for comparing the two strings as shown below and re-execute:

```
public class Test7
	{
		public static void main(String args[])
		{
			String s1=new String("Java");
			String s2=new String("Java");
			if (s1.equals(s2))
			  System.out.println("s1 and s2 are equal");
			else
			  System.out.println("s1 and s2 are not equal");
		}
	}
```

On execution of the program now, the following output is generated.



The memory allocation for String objects and corresponding references is depicted in the following Figure.

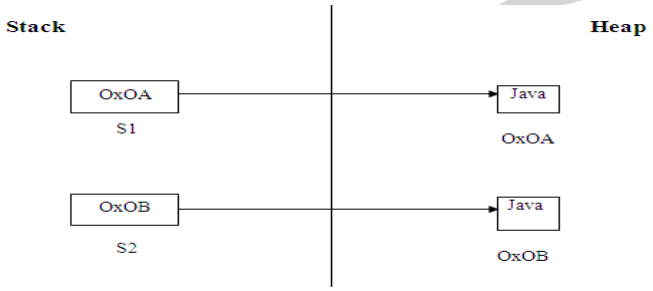


Figure. Memory Allocation for String Objects

S1 and S2 variables are created on stack which reference two distinct String objects created on heap. Hence statement 7 in the above program, compares addresses of two String objects and not their contents. For comparing the content of two String objects use equals() method inherited from the Object class and implemented in the String class.

Change the statements (5) and (6) with the statements given below:

String s1="Java";

String s2="Java";

and re-execute the program. The following output is generated.

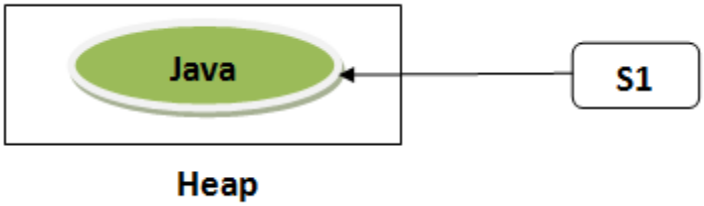
When the String objects are created without using new operator, the String objects are created in a string pool on the heap. There is only one String object with the unique content in the string

pool. There can be multiple references referencing a single String object in a string pool.

When the statement

String s1="Java";

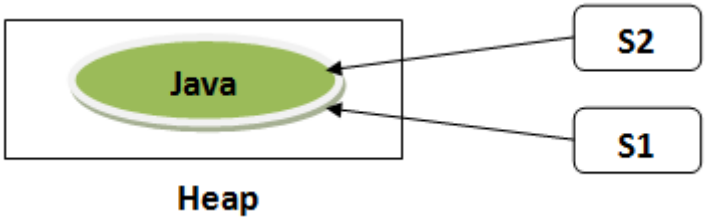
is executed, the String object "Java" is created in a string pool as shown below:



When the statement

String s2="Java";

is executed, the string pool is searched for a String object "Java". Since the string pool already contains the searched object, no new String object is created and s2 will reference the same String object as referenced by s1 as shown below:



Q. No. 21 *Category :Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test56
{
   public static void main(String[] args)
   {
      Integer a = 128, b = 128; ← Statement 5
      System.out.println(a == b);

      Integer c = 100, d = 100; ← Statement 7
      System.out.println(c == d);
   }
}
```

Output :

Explanation :

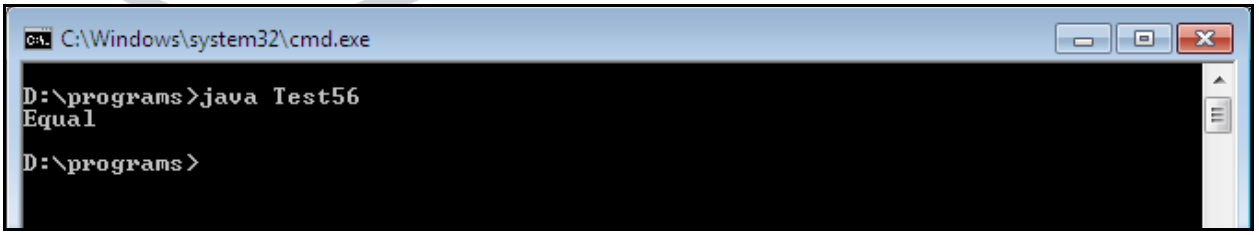
Java caches the numbers between -128 and 127. In statement (5) the values of a and b are boxed. Hence comparison between a and b amounts to comparing two java objects which returns false since the addresses of the objects are compared and not their contents.

In contrast to this in statement (7), the values of c and d are not boxed, hence comparing c and d is same as comparing the primitive values.

For comparing two Integer objects with values greater than 127, use equals() method of Integer class as shown below:

```
public class Test56
{
   public static void main(String[] args)
   {
      Integer a = 128, b = 128;
      if(a.equals(b))
        System.out.println("Equal");
   }
}
```

On execution of the above program, the following output is generated.



Q. No. 22 *Category : Operators*

What is the output generated on execution of the following Java Program?

```
public class Test8
{
    public static void main(String args[])
    {
        int x=0, y=1, z=1;
        if (x != 0 & (y=2)==1) ← Statement 6
        {
        }
        System.out.println("y : " + y);
        if (x != 0 && (z=2)==1) ← Statement 10
        {
        }
        System.out.println("z : " + z);
}
    }
```

Output :

Explanation :

In Java, bitwise operators do not employ short-circuit evaluation of complex conditions whereas logical operators do employ them. Hence in the following complex condition evaluation

Condition 1 & Condition 2

Condition 2 is guaranteed to be evaluated under all circumstances irrespective of the outcome of Condition 1.

In contrast to this, in the evaluation of the following expression.

Condition 1 && Condition 2

Condition 2 is evaluated if and only if Condition 1 evaluates to true, since in that case the result of the whole condition is dependent on outcome of Condition 2. For the same reason, in the

evaluation of the following expression

Condition 1 || Condition 2

Condition 2 is evaluated if and only if Condition 1 evaluates to false.

Hence, in the above program, in statement 6 using bitwise AND operator, the first condition evaluates to false, still condition 2 is executed and the value 2 is assigned to y as seen in the output generated. On the contrary, in statement 10 using logical AND operator, since the first logical condition evaluates to false, the second condition is not evaluated and the variable z retains the value of 1 as seen in the generated output.

Q. No. 23 *Category : Functions*

What is the output generated on execution of the following Java Program?

```
public class Test9
{
  public static void main(String args[])
  {
     int[] a={10,20};
     int value=25;
    System.out.println("Before method call : ");
    System.out.println("a[0] : " + a[0] + " a[1] : " + a[1] +" value : " + value);
    change(a,value);
    System.out.println();
    System.out.println("After method call : ");
    System.out.println("a[0] : " + a[0] + " a[1] : " + a[1] +" value : " + value);
  }

  public static void change(int[] a, int value)
  {
    a[0]=100;
    a[1]=200;
    value=50;
  }
}
```

Output :

Explanation :

In a method call in java, all primitive data types are passed by value and all the objects are passed by reference.

In Java, arrays are implemented as objects and hence are passed by reference. As a result all the changes carried out to array members in the called function are reflected in the calling program which is not the case with the primitive data type such as value.

Q. No. 24 *Category : Functions*

What is the output generated on execution of the following Java Program?

```
class Data
{
        public String data_string;
        Data (String d)
        {
                data_string=d;
        }
}
class Printer
{
        public void print(Data d)
        {
                System.out.println(d.data_string);
        }
        public void modify(Data d)
        {
                d.data_string="Modified....";
        }
}
```

```
public class Test10
{
        public static void main(String args[])
        {
                Data d = new Data("Hello from Java");
                int x=10;
                Printer p = new Printer();
                p.print(d); ← Statement 29
                System.out.println("x : " + x);
                p.modify(d);
                modifyX(10);
                p.print(d);
                System.out.println("x : " + x);
        }
        public static void modifyX(int x)
        {
                x=1000;
        }
}
```

Output :

Explanation :

In the above program, in statement 29, modify() method on Printer object is invoked by passing reference of Data class object which modifies data_string member of Data class to 'Modified' which is reflected in the calling program. On the contrary, modifyX() method accepts a variable of type int and changes its value to 1000 inside the function. As seen in the output generated, the change carried out in the method modifyX() is not visible after the function returns and the variable x retains its original value of 10 before the function call.

Q. No. 25 *Category : Functions*

What is the output generated on execution of the following Java Program?

```
public class Test11
{
  public static void main(String args[])
  {
    String str=new String("MCA");
    int value=25;
    System.out.println("Before method call : ");
    System.out.println("str : " + str + " value : " + value);
    change(str,value);
    System.out.println();
    System.out.println("After method call : ");
    System.out.println("str : " + str + " value : " + value);
  }

  public static void change(String str, int value)
  {
    str="MBA";
    value=50;
  }
}
```

Output :

Explanation :

String is a built-in java class. Hence, even if String objects are passed by reference, the changes performed in the method are not reflected in the calling program. The reason for this is String is immutable in Java.

In the given program replace the String class with either StringBuffer or StringBuilder classes as

shown below and re-execute the program.

```
public class Test11
{
  public static void main(String args[])
  {
    StringBuffer str=new StringBuffer("MCA");
    int value=25;
    System.out.println("Before method call : ");
    System.out.println("str : " + str + " value : " + value);
    change(str,value);
    System.out.println();
    System.out.println("After method call : ");
    System.out.println("str : " + str + " value : " + value);
  }

  public static void change(StringBuffer str, int value)
  {
    str.replace(0,3,"MBA");
    value=50;
  }
}
```

On execution of the program, the following output is generated.

Q. No. 26 *Category : Arrays*

Which one of the following will declare an array and initialize it with five numbers?

1. ***Array a = new Array(5);***
2. ***int [] a = {23,22,21,20,19};***
3. ***int a [] = new int[5];***
4. ***int [5] array;***

Output :

Option 2 is the legal way to declare and initialize an array with five elements.

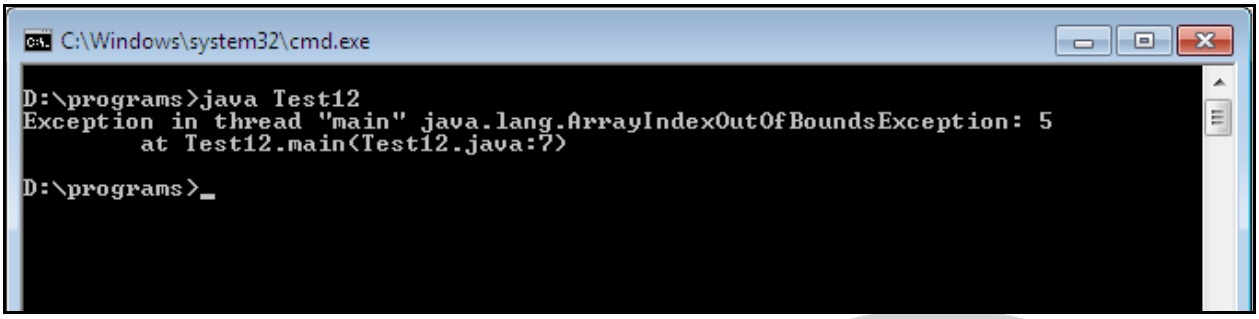
Explanation :

The following rules apply to the declaration of arrays in Java.

1. Arrays must never be given a size when declared. The size is only needed when the array is actually instantiated
2. Array class is declared in java.lang.reflect package which is not instantiable. Array object is not the same as an array object
3. Any of the following syntax can be used for declaring an array.
   i) int[] a;
   ii) int a[];
4. Arrays can be initialized at the time of declaration using the following syntax.
   int[] a={1,2,3,4,5};
5. Arrays can be declared and initialized in separate statements as shown below:
   int[] a = new int[5];
   for(int i=0;i<5;i++)
   a[i]=i+1;
6. ArrayIndexOutOfBoundsException is thrown if the array is referenced with illegal index, beyond its size as shown in the following program.

```
import java.lang.reflect.Array;
public class Test
    {
        public static void main(String[] args)
        {
        int[] a = new int[5];
            for(int i=0;i<10;i++)
                a[i]=i+1;
          }
        }
```

On execution of the above program, the following exception is thrown.



Q. No. 27 *Category : Arrays*

Which of the following are valid array declarations?

1. **public int a [ ]**
2. **static int [ ] a**
3. **public [ ] int a**
4. **private int a [3]**
5. **private int [3] a [ ]**
6. **public final int [ ] a**

Output :

Options (1) and (2) are valid array declarations.

Explanation :

Option	Description
1	Valid.
2	Valid.
3	The brackets are in the wrong place. Such a declaration results in the following compiler error. illegal start of type public [] int a; ^ <identifier> expected public [] int a; ^ 2 errors

4	In Java, 1-D arrays cannot be declared with a fixed size. A closing bracket is expected in place of the 3. Such a declaration results in the following compiler error. Test.java:3: ']' expected private int a[3]; ^ 1 error
5	Correct method of declaring an array is private int[]a[]; or private int[][] a; The declaration as in option (5) would result in the following compiler error. ']' expected private int [3] a []; ^ <identifier> expected private int [3] a []; ^ 2 errors

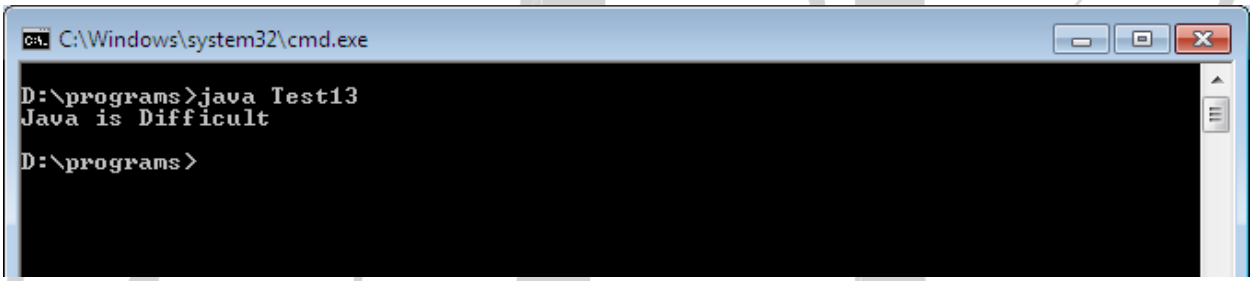
6	Final variable must be assigned values at the time of declaration and blank final variables must be initialized in a constructor. The declaration as in option (6) would result in the following compiler error.
	
	variable a might not have been initialized
	public class Test
	^
	1 error
	The correct way of declaring a blank final array is
	public class Test
	{
	public final int []a=new int[1];
	
	public Test()
	{
	a[0]=10;
	}
	public static void main(String args[])
	{
	Test t=new Test();
	System.out.println(t.a[0]);
	}
	}
	On execution of the above Java program, the following output is generated.
	10

Q. No. 28 *Category : String Operations*

What is the output generated on execution of the following Java Program?

```
public class Test
{
  public static void main(String args[])
  {
   String str="Java is Difficult";
   str.replace("Difficult","Easy"); ← Statement 6
   System.out.println(str); ← Statement 7
  }
}
```

Output :



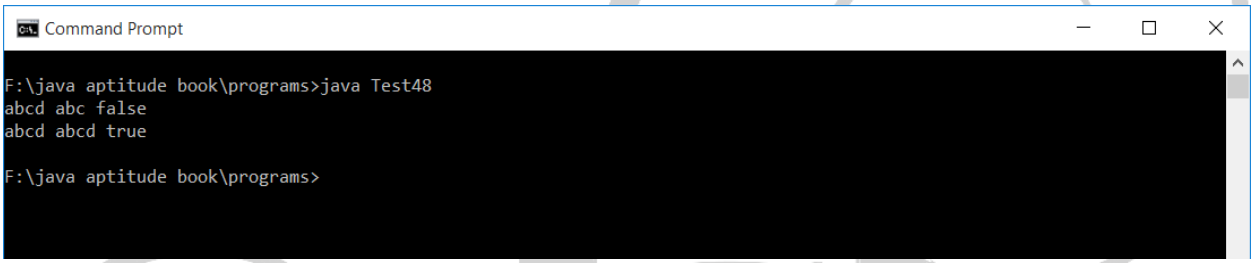
Explanation :

Since String class objects are immutable, the contents of the object cannot be modified without creating a new one. In the given program, in statement 6) a new String object with the statement scope is created and is immediately deleted in the next statement. Hence when statement 7) is reached, str will continue to point to the String object, “Java is Difficult”. Hence the generated output.

Q. No. 29 *Category : String Operations*

What is the output generated on execution of the following Java Program?

```
public class Test48
{
   public static void main(String args[])
   {
      String s1 = "abc";
      String s2 = s1;
      s1 += "d";
      System.out.println(s1 + " " + s2 + " " + (s1 == s2));

      StringBuffer sb1 = new StringBuffer("abc");
      StringBuffer sb2 = sb1;
      sb1.append("d");
      System.out.println(sb1 + " " + sb2 + " " + (sb1 == sb2));
   }
}
```

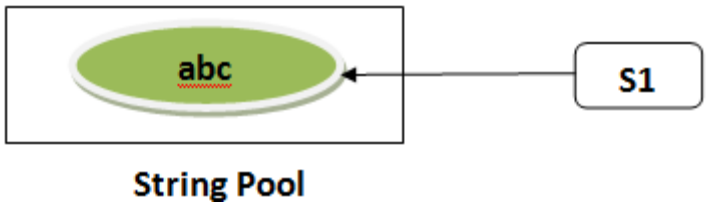


In Java, String is immutable and string buffer is mutable. So string s2 and s1 both pointing to the same string abc. And, after making the changes the string s1 points to abcd and s2 points to abc, hence false. While in string buffer, both sb1 and sb2 both point to the same object. Since string buffer are mutable, making changes in one string also make changes to the other string. So both string still pointing to the same object after making the changes to the object (here sb2).

After the execution of the statement,

String s1 = "abc";

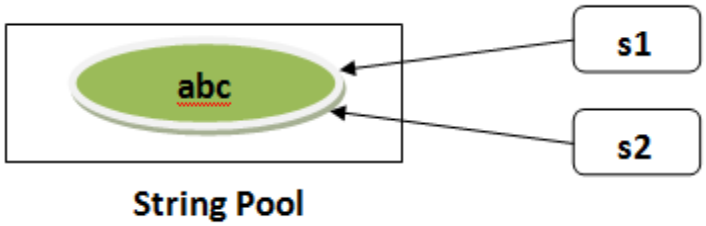
a string object “abc” is created in a string pool and s1 points to the String object as shown below:



After the execution of the statement

String s2 = s1;

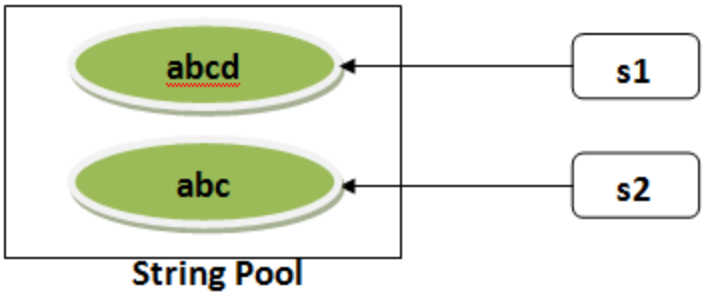
both s1 and s2 point to the same String object "abc" as shown below:



After the execution of the statement

s1 += "d";

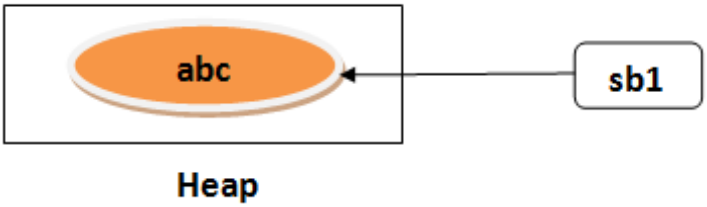
the a new String object "abcd" is created which is referenced by s1 whereas s2 continues to reference to the old String object "abc" as shown below:



In contrast to this, the StringBuffer object is created on heap. After the execution of the statement

StringBuffer sb1 = new StringBuffer("abc");

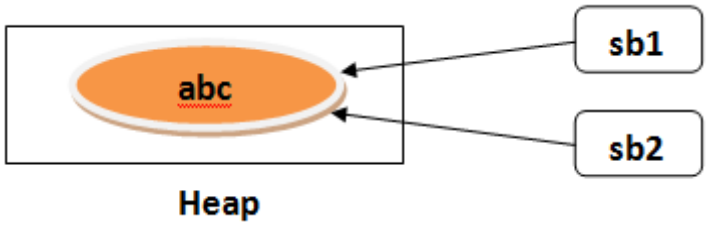
a StringBuffer object is created on heap which is pointed by sb1 as shown below:



After the execution of the statement

StringBuffer sb2 = sb1;

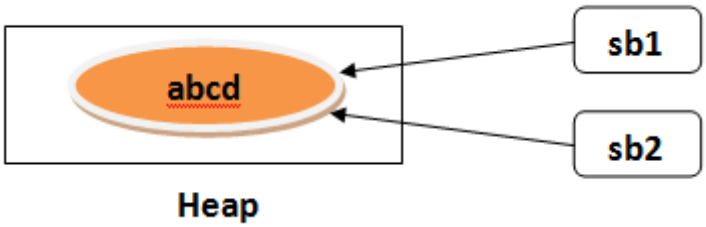
a new reference sb2 is created on stack which will point to the same StringBuffer object "abc" on heap as shown below.



After the execution of the statement

sb1.append("d");

the existing StringBuffer object "abc" is modified to "abcd" as shown below and now both sb1 and sb2 point to the modified StringBuffer object "abcd".



Q. No. 30 *Category : Operators*

```
class Test
{
   public static void main(String [] args)
   {
      int x = 0x40000000;
      x = x >>> 30;
      System.out.println(x);
   }
}
```

Output :

Explanation :

The >>> operator moves all bits to the right, zero filling the left bits.

Consider the statement

int x = 0x40000000;

x is represented in memory using 32 bits as shown below:

0100 0000 0000 0000 0000 0000 0000 0000

Hence the decimal value of x now is 2^{30}.

After performing the operation

x = x >>> 30;

all the bits are shifted to the right by 30, zero filling the left bits.

Hence after the operation x is represented in memory as

0000 0000 0000 0000 0000 0000 0001

whose decimal equivalent is 1.

Thus, the program generates the output 1.

Consider the following program:

```
class Test
{
   public static void main(String [] args)
   {
      int x = 0x80000000;
     System.out.println(Integer.toBinaryString(x));
      System.out.println(x);
   }
}
```

x is represented in memory using 32 bits as shown below:

1000 0000 0000 0000 0000 0000 0000 0000

The left most bit is a sign bit which is 1 for negative numbers and 0 for positive numbers. Hence the decimal representation of the above number is

$$-2^{31} = -2147483648$$

On execution of the above program, the following output is generated.

Consider the following program:

```
class Test
{
   public static void main(String [] args)
   {
      int x = 0x80000000;
      x=x>>>31;
      System.out.println(x);
   }
}
```

Shifting bits to the right 31 times and replacing left bits with 0 in

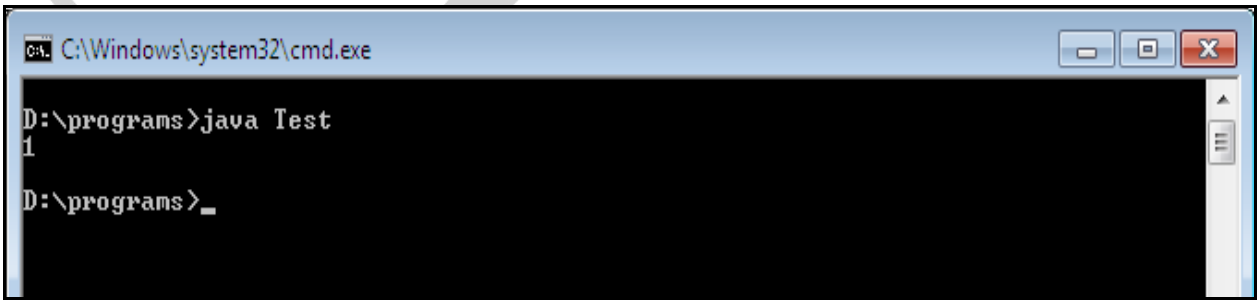
1000 0000 0000 0000 0000 0000 0000 0000

we get,

0000 0000 0000 0000 0000 0000 0000 0001

whose decimal equivalent is 1.

On execution of the program, the following output is generated.



Consider the following program:

```
class Test
{
   public static void main(String [] args)
   {
      int x = 0x80000000;
      x=x>>31;
      System.out.println(x);
   }
}
```

The >> operator moves all bits to the right specified time, keeping the sign bit intact.

x is represented in memory using 32 bits as shown below:

1000 0000 0000 0000 0000 0000 0000 0000

Hence the operation

x=x>>31;

shifts all the bits to the right 31 times and we get

1111 1111 1111 1111 1111 1111 1111

which is representation of -1.

On execution of the program, the following output is generated.

Consider the following program:

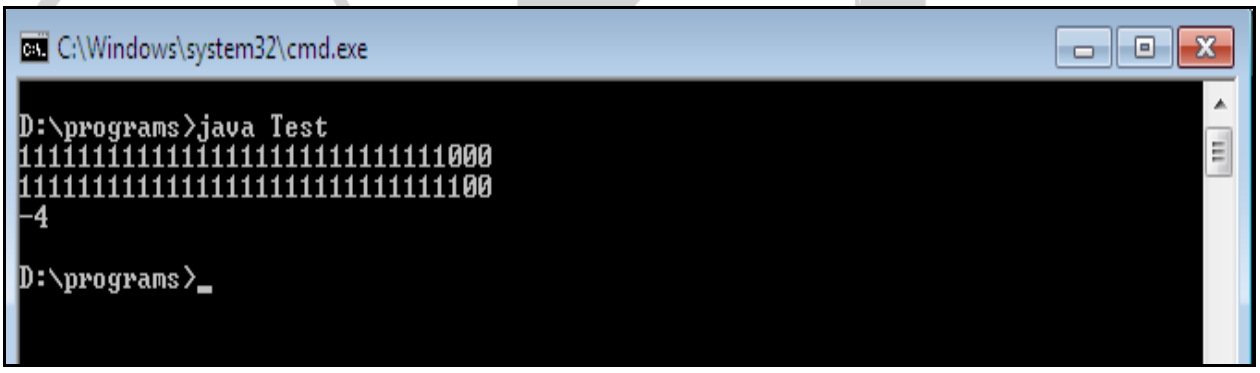
```
class Test
{
   public static void main(String [] args)
   {
      int x = 8;
      System.out.println(Integer.toBinaryString(x));
      x=x>>1;
      System.out.println(Integer.toBinaryString(x));
      System.out.println(x);
   }
}
```

On execution of the program, the following output is generated:

Consider the following program:

```
class Test
{
   public static void main(String [] args)
   {
      int x = -8;
      System.out.println(Integer.toBinaryString(x));
      x=x>>1;
      System.out.println(Integer.toBinaryString(x));
      System.out.println(x);
   }
}
```

On execution of the program, the following output is generated:

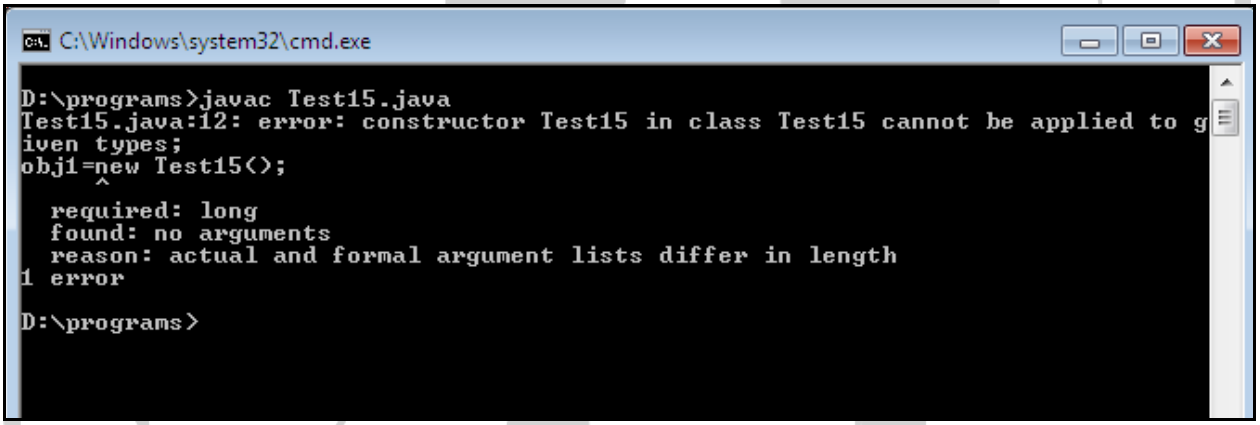


Q. No. 31 *Category : Functions*

What is the output generated on execution of the following Java Program?

```
public class Test15
{
        long var;
        public Test(long param)
        {
                var=param;
        }

        public static void main(String args[])
        {
                Test obj1, obj2;
                obj1=new Test();
                obj2=new Test(5);
        }
}
```

Output :



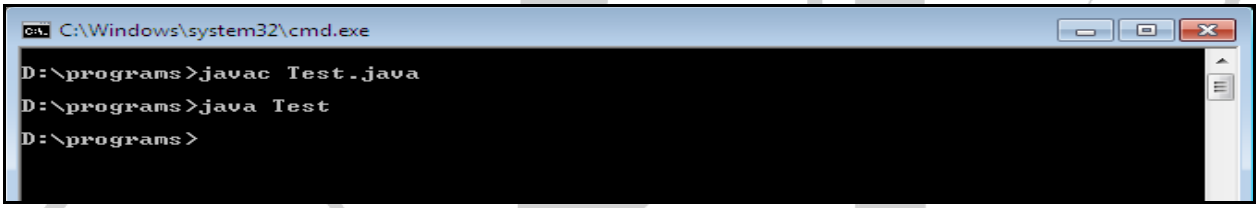
Explanation :

If the class does not contain any constructors, then the default constructor is inserted by Java runtime Environment. On the other hand, if the class contains parameterized constructors, then the default constructor must be explicitly added to the class for the successful execution of the statements invoking default constructor, otherwise a compiler error is generated.

For the proper execution of the program add a default constructor to the class as shown below:

```
public class Test
{
        long var;
        public Test()
        {
        }
        public Test(long param)
        {
                var=param;
        }
        public static void main(String args[])
        {
                Test obj1, obj2;
                obj1=new Test();
                obj2=new Test(5);
        }
}
```

On execution of the above program, the following output is generated.



Q. No. 32 *Category : Variable Declaration Issues*

What is the output generated on execution of the following Java Program?

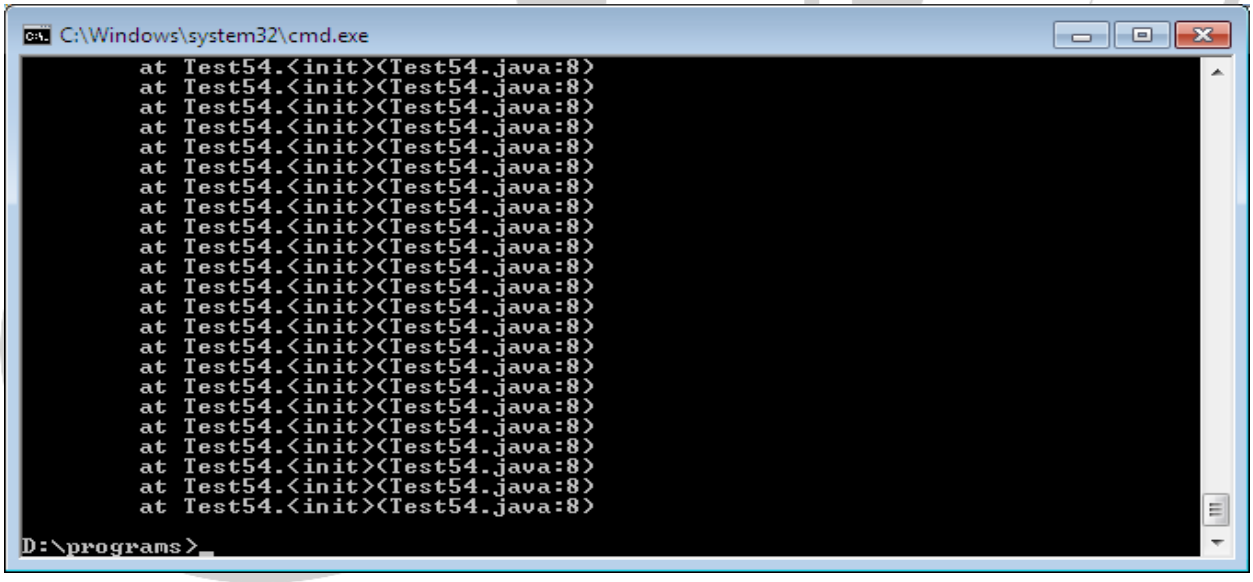
```
class Test54
{
   Test54()
   {
      System.out.println("SIBER");
   }

   static Test54 a = new Test54();  ← Statement  8

   public static void main(String args[])
   {
      Test54 b;
      b = new Test54(); ← Statement 13
   }
}
```

Output :

Explanation :

We know that static variables are initialized when a class loads and are called only once. Now statement 13 results in creation of object which in turn calls the constructor and "SIBER" is printed second time.

If in statement 8 static variable is not used the object would have been called recursively infinitively leading to StackOverFlow error as shown below:



Q. No. 33 *Category : Variable Declaration Issues*

What is the output generated on execution of the following Java Program?

```
class Test55
{
   static int num;
   static String mystr;

   // constructor
   Test55()
   {
      num = 100;
      mystr = "Constructor";
   }

   // First Static block
   static
   {
      System.out.println("Static Block 1");
      num = 10;
      mystr = "Block1";
   }

   // Second static block
   static
   {
      System.out.println("Static Block 2");
      num = 20;
      mystr = "Block2";
   }

   public static void main(String args[])
   {
      Test55 a = new Test55();
      System.out.println("Value of num = " + a.num);
      System.out.println("Value of mystr = " + a.mystr);
   }
}
```

Output :

Explanation :

Static block gets executed when the class is loaded in the memory. A class can have multiple Static blocks, which are executed in the same sequence in which they have been written into the program.

Note: Static Methods can access class variables without using object of the class. Since constructor is called when a new instance is created so firstly the static blocks are called and after that the constructor is called. If we would have run the same program without using object, the constructor would not have been called.

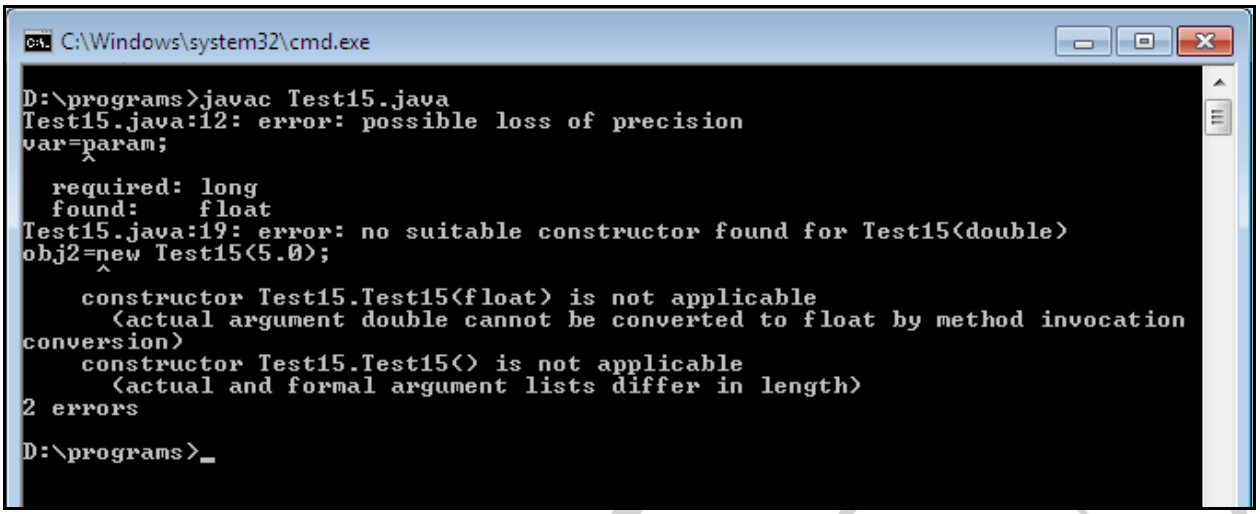
Remove the constructor from the class and re-execute the program. The following output is generated.

Q. No. 34 *Category : Functions*

What is the output generated on execution of the following Java Program?

```
Public class Test15
{
        long var;
        public Test15()
        { }

        public Test15(float param)
        {
        var=param;
        }

        public static void main(String args[])
        {
        Test15 obj1, obj2;
        obj1=new Test15();
        obj2=new Test15(5.0);
        }
}
```

Output :



Explanation :

In Java, a floating point constant is treated as a double constant and a variable of larger size cannot be assigned to the variable of narrower size without explicit type casting. To specify the floating point constant as of type float use the suffix ‘f’ or ‘F’.

Modify the program as shown below in line with the explanation given above (Modified statements are shown in bold) and re-execute it.

```
public class Test15
{
   long var;
   public Test15()
   {

   }

   public Test15(float param)
   {
        var=(long)param;
   }
```

```
    public static void main(String args[])
   {
     Test15 obj1, obj2;
     obj1=new Test15();
     obj2=new Test15(5.0f);
   }
}
```

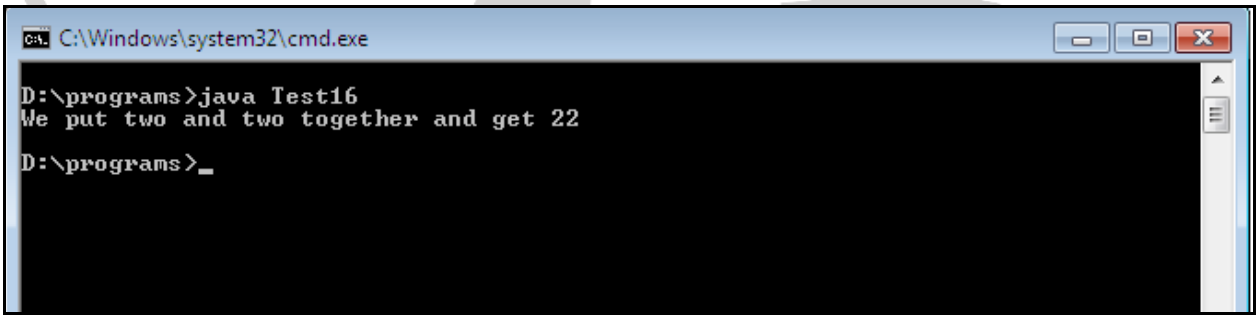
On execution of the above program, the following output is generated.

Q. No. 35 *Category : Operators*

What is the output generated on execution of the following Java Program?

```
public class Test16
{
  public static void main(String args[])
  {
    System.out.println("We put two and two together and get " + 2 + 2);
  }
}
```

Output :



Explanation :

When the expression in System.out.println() statement is executed, the first integer literal 2 is

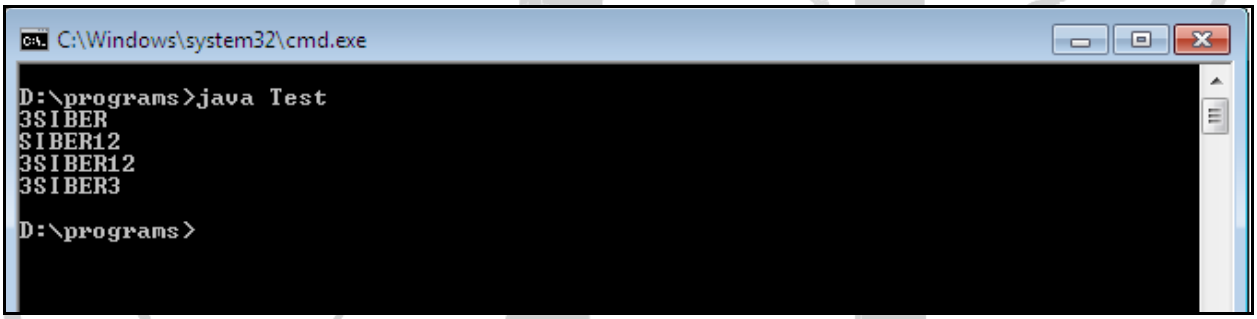
promoted to a string literal 2 for the first concatenation resulting in the string literal. The result is then concatenated with the string literal 2 resulting in 22.

Q. No. 36 *Category : Operators*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main(String[] args)
   {
      System.out.println(1+2+"SIBER");
      System.out.println("SIBER"+1+2);
      System.out.println(1+2+"SIBER"+1+2);
      System.out.println(1+2+"SIBER"+(1+2));
   }
}
```

Output :



Explanation :

This unpredictable output is due the fact that the compiler evaluates the given expression from left to right given that the operators have same precedence. Once it encounters the String, it considers the rest of the expression as of a String (again based on the precedence order of the expression).

- **System.out.println(1 + 2 + "SIBER");** // It prints the addition of 1 and 2 which is equal to 3 and concatenates it with the string "SIBER" resulting in the output ***3SIBER***.
- **System.out.println("SIBER" + 1 + 2);** //It prints the concatenation of 1 and 2 which is 12 since it encounters the string initially. Basically, Strings take precedence because they

have a higher casting priority than integers do. “SIBER” is concatenated with “1” resulting in a string “SIBER1” which is then concatenated with “2” resulting in the string “SIBER12”.

- **System.out.println(1 + 2 + “SIBER” + 1 + 2);** //It prints the addition of 1 and 2 and concatenates it with “SIBER”, 1 and 2 in that order resulting in output ***3SIBER12***.
- **System.out.println(1 + 2 “SIBER” + (1 + 2));** //It prints the addition of both 1 and 2 and 1 and 2 based due the precedence of ( ) over +. Hence expression in ( ) is calculated first and then the further evaluation takes place resulting in the output ***3SIBER3***.

Q. No. 37 *Category : Operators*

What is the output generated on execution of the following Java Program?

```
public class Test1
{
   public static void main(String[] args)
   {
      int a = 20, b = 10, c = 30;
      a = b+++c;
      System.out.printf("a : %d b : %d ",a,b);
   }
}
```

Output :

Explanation :

The expression a=b+++c is compiled as

(b++) +c

Hence the entire expression is equivalent to the atomic operations

a=b+c

b=b+1

Hence a = 40 and b becomes equal to11 on incrementing. Hence the given program generates the output

a = 40 and b = 11

Q. No. 38 *Category : Operators*

What is the output generated on execution of the following Java Program?

```
public class Test1
{
   public static void main(String[] args)
   {
      int x = 5, y = 8;
      System.out.println("Concatenation (x+y)= " + x + y);
      System.out.println("Addition (x+y) = " + (x + y));
   }
}
```

Output :

Explanation :

The expression

"Concatenation (x+y)= " + x + y;

is evaluated as follows:

First x is added to "concatenation (x+y) = " producing "concatenation (x+y) = 5" and then 8 is further concatenated producing a string "concatenation (x+y) = 58"

In contrast to this, in the second expression**,**

"Addition (x+y) = " + (x + y);

is evaluated as follows:

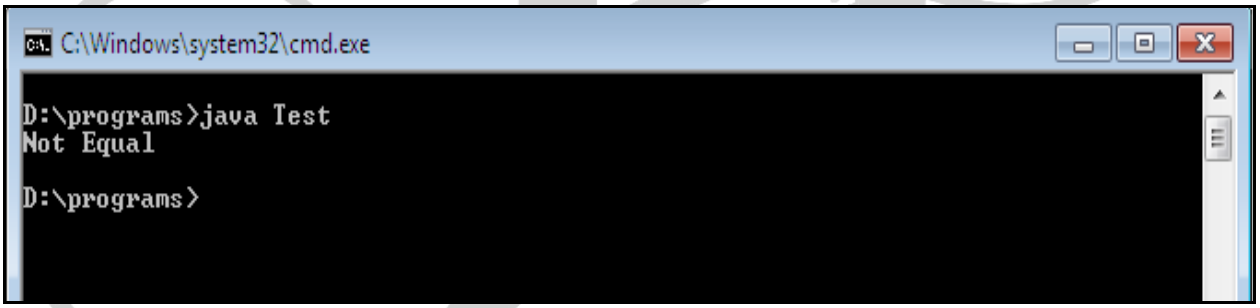
Since x+y is placed in a bracket, it has higher precedence and has is computed first(=13) and the result obtained is concatenated with the string "Addition (x+y) =" resulting in the string Addition (x+y) = 13.

Q. No. 39 *Category : Operators*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main(String[] args)
   {
       Integer x = 10, y = 400;
       if (x == y)
         System.out.println("Equal");
       else
         System.out.println("Not Equal");

   }
}
```

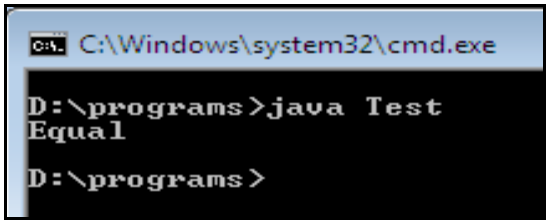
Output :

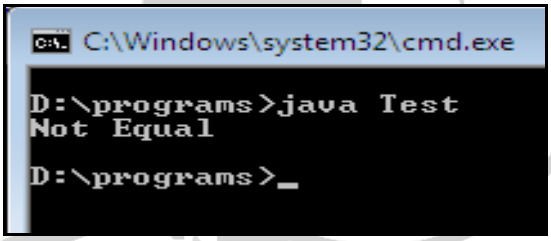


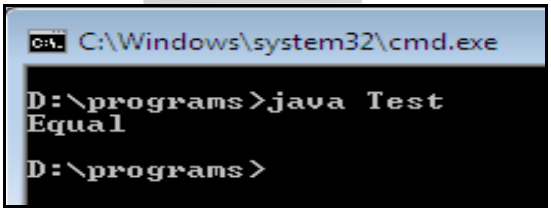
Explanation :

Since x and y refer to different objects, we get the output as "Not Same". In order to compare their contents use the equals() method inherited by java.lang.Object class and overridden by integer class as shown below.

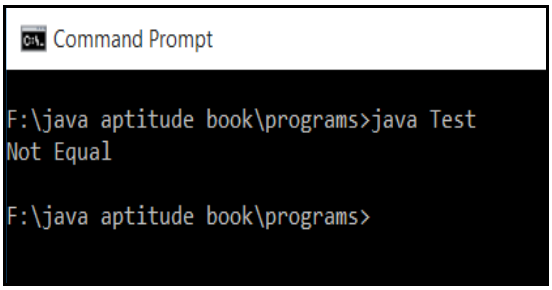
Program	*Output*

public class Test
{
public static void main(String[] args)
{
Integer x = 100;
Integer y=100;
if (x==y)
System.out.println("Equal");
else
System.out.println("Not Equal");
 	
}
}
	
	Explanation
	Since both x and y are less than 128, they are cached into memory and primitive data types and not objects are compared which prints ”Equal”.

public class Test
{
public static void main(String[] args)
{
Integer x = 1000;
Integer y=1000;
if (x==y)
System.out.println("Equal");
else
System.out.println("Not Equal");
 	
}
}
	
	Since both x and y are greater than 128, they are boxed to create the objects and the addresses of the objects are compared which prints “Not Equal”.

public class Test
{
public static void main(String[] args)
{
Integer x = 1000;
Integer y=1000;
if (x.equals(y))
System.out.println("Equal");
else
System.out.println("Not Equal");
 	
}
}
	
	Since both x and y are greater than 128, they are boxed to create the objects and equals() method of Integer wrapper class is used to compare their contents which prints “Equal”.

```
public class Test
{
   public static void main(String[] args)
   {
       Integer x = 100;
       Integer y=1000;
       if (x.equals(y))
         System.out.println("Equal");
       else
         System.out.println("Not Equal");

   }
}
```



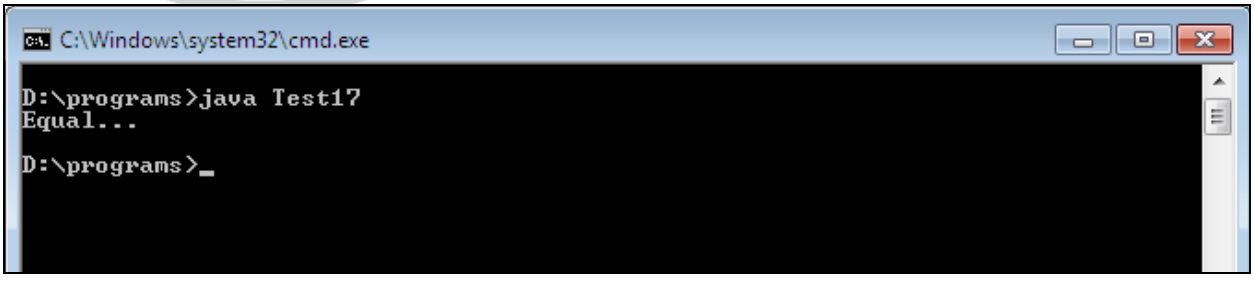
Even if x < 128 equals() method can be used for comparing x and y which generates the output "Not Equal".

Q. No. 40 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test17
{
  public static void main(String args[])
  {
    Integer iObj=new Integer(10);
   int i=10;
    if (iObj==i)
          System.out.println("Equal...");
    else
          System.out.println("Not Equal...");
  }
}
```

Output :



Explanation :

Boxing conversion converts expressions of primitive type to corresponding expressions of reference type. For example, from int primitive type to Integer wrapper class object. While comparing Integer class object with int, Integer class object is unboxed and two primitive data types are compared.

If both the operands are of Integer object type with value greater than 128, then references are compared as illustrated in the following example.'

```
public class Test17
{
  public static void main(String args[])
  {
    Integer iObj=new Integer(1000);
    Integer i=new Integer(1000);
    if (iObj==i)
          System.out.println("Equal...");
    else
          System.out.println("Not Equal...");
  }
}
```

On execution of the above program, the following output is generated.

Since both I and iObj are Integer wrapper class objects their addresses are compared. For comparing their contents use the expression

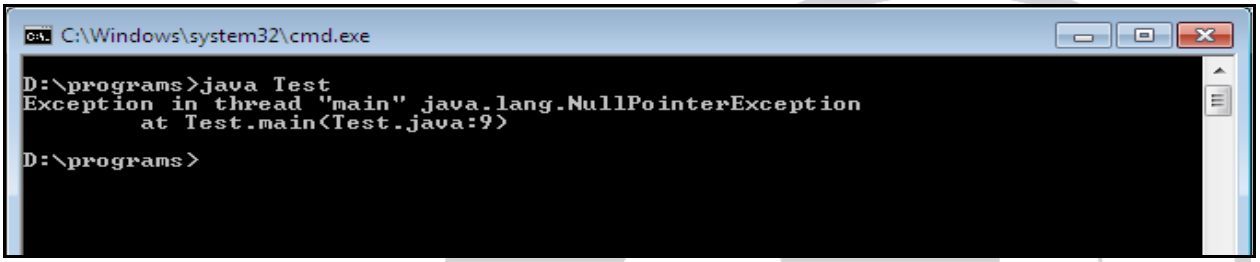
iObj.equals(i)

Q. No. 41 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main (String[] args) throws java.lang.Exception{

        Integer i = null;
        int a = i;
   }
}
```

Output :



Explanation :

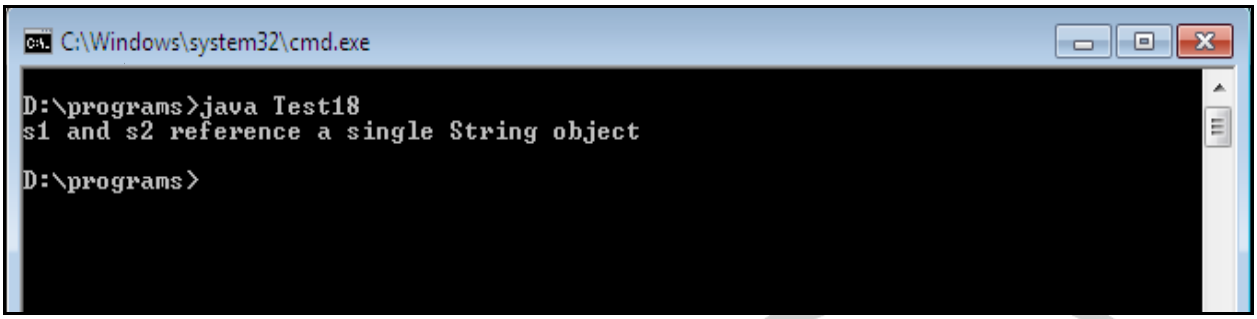
An Integer can be null. However, unboxing null Integer throws NullPointerException. During auto-boxing and unboxing operations, compiler simply throws NullPointerException error if a null value is assigned to primitive boxed data type.

Q. No. 42 *Category : String Operations*

What is the output generated on execution of the following Java Program?

```
public class Test18
{
    public static void main(String args[]){
      String s1="Welcome";
      String s2="Welcome";
      if (s1==s2)
      {
          System.out.println("s1 and s2 reference a single String object");
      }
      else
      {
          System.out.println("s1 and s2 reference different String objects");
      }
    }
}
```

Output :



Explanation :

Consider the difference between the following statements :

String s1="Welcome";

String s2="Welcome

and

String s1=new String("Welcome");

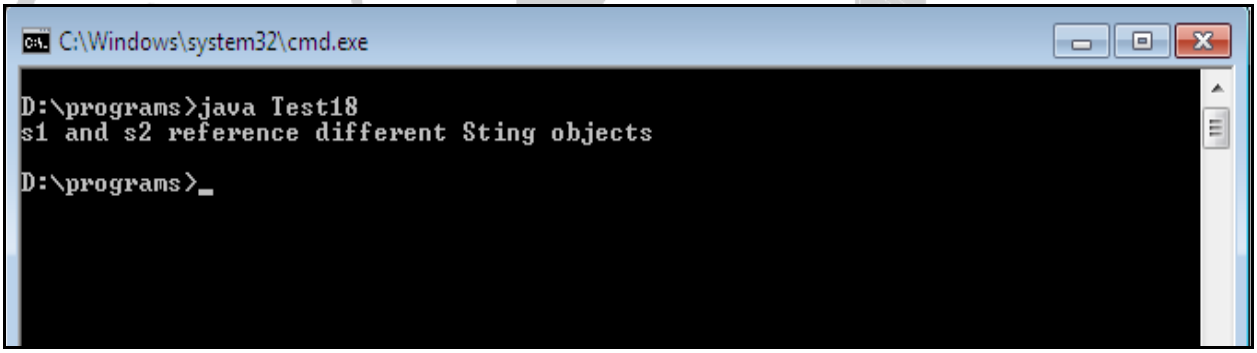
String s2=new String("Welcome);

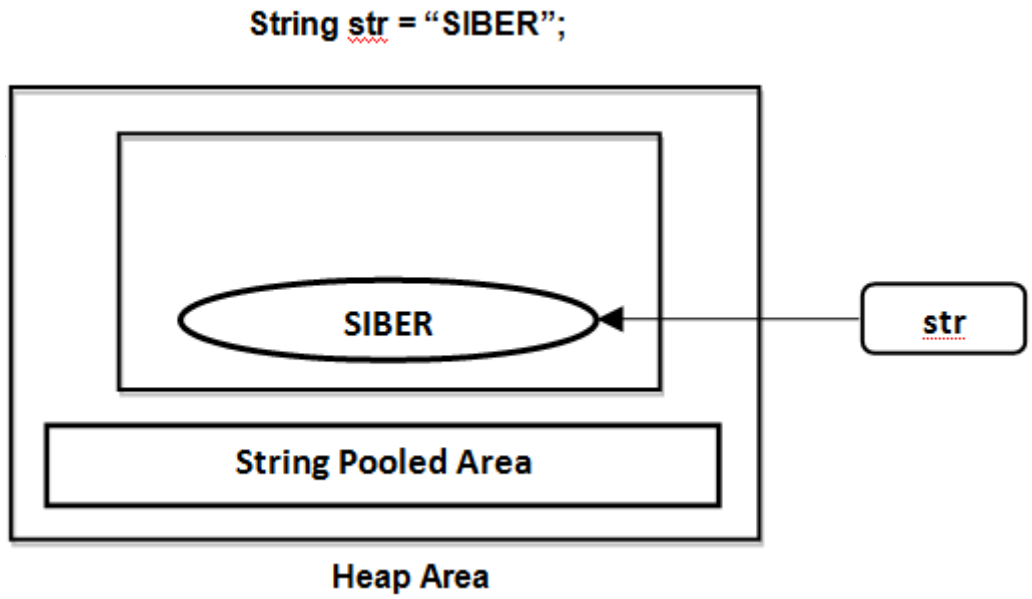
Each time you create a string literal, the JVM checks the string constant pool first. If the string already exists in the pool, a reference to the pooled instance is returned. If string doesn't exist in the pool, a new string instance is created and placed in the pool. Hence in the first block of statements given above, only a single String object is created which is referenced by two String references s1 and s2.

On the contrary, in the second block of statements given above two different String objects are created on heap which are referenced by two different references s1 and s2 as illustrated in the following example.

```
public class Test
{
    public static void main(String args[])
    {
      String s1=new String("Welcome");
      String s2=new String("Welcome");
      if (s1==s2)
      {
       System.out.println("s1 and s2 reference a single Sting object");
      }
      else
      {
       System.out.println("s1 and s2 reference different Sting objects");
      }
    }
}
```

On execution of the above program the following output is generated.





Direct Initialization(String Constant) :

In this case, a String constant object will be created in String pooled area which is inside heap area in memory. As it is a constant, we can't modify it, i.e. String class is immutable.

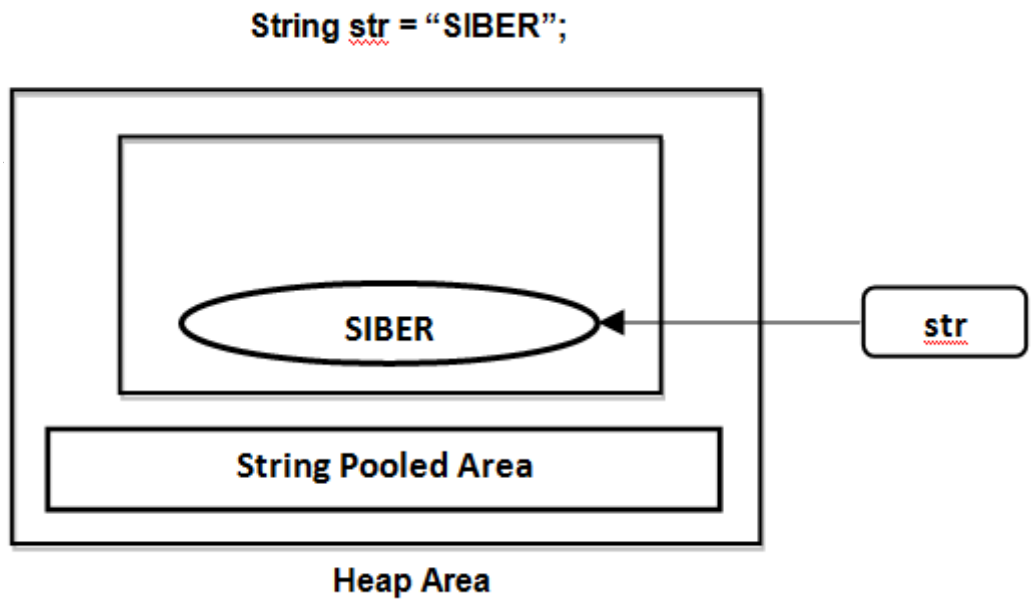
Examples:

Consider the following example:

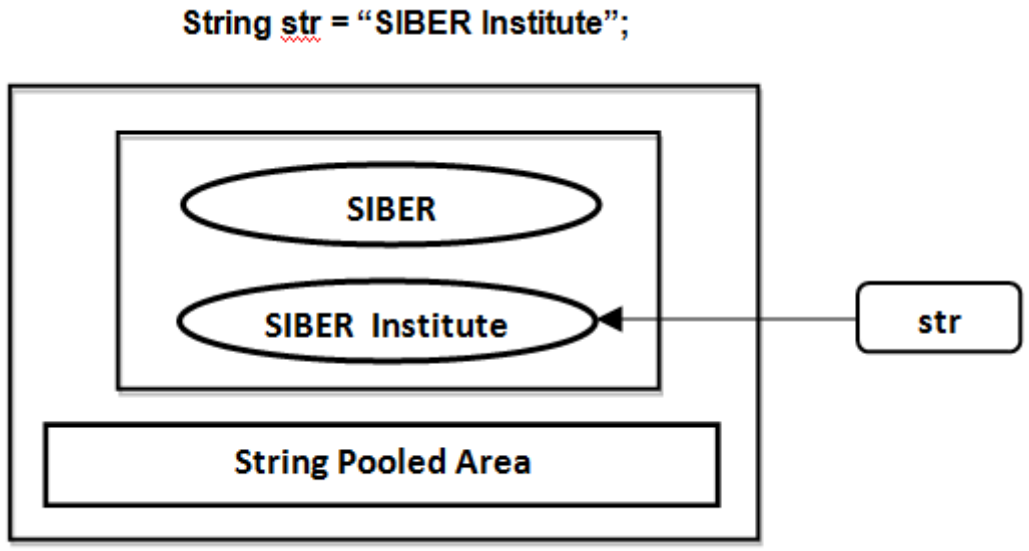
String str="SIBER";
str= "SIBER Institute";

The above statement will make str point to a new String constant "SIBER Institute" rather than modifying the previous String constant. At this point, the string pool contains two String constants "SIBER" and "SIBER Institute" pointed by str as illustrated below:

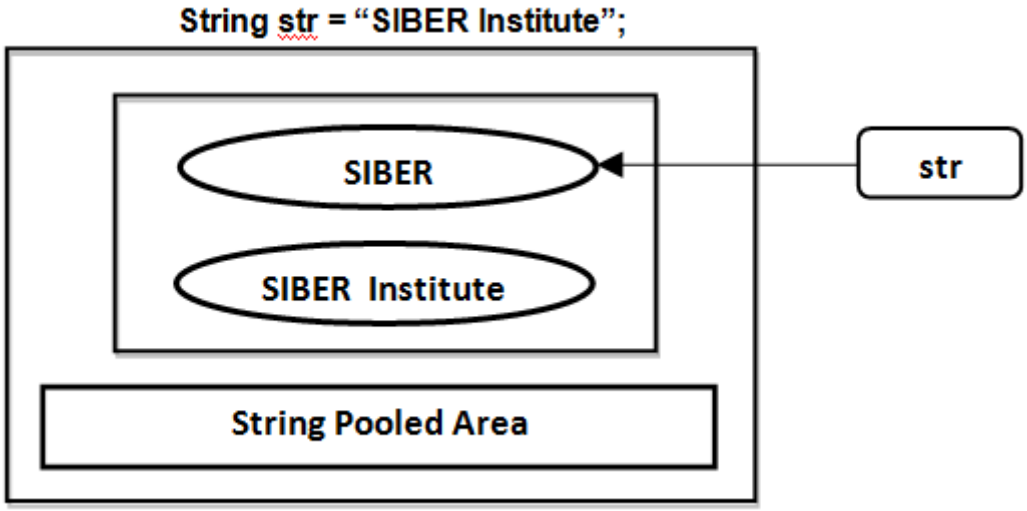
After the execution of the statement, **String str = "SIBER";** the status of the string pool is as shown below:



After the execution of the statement **String str = "SIBER Institute";** , the status of the string pool now is as shown below:

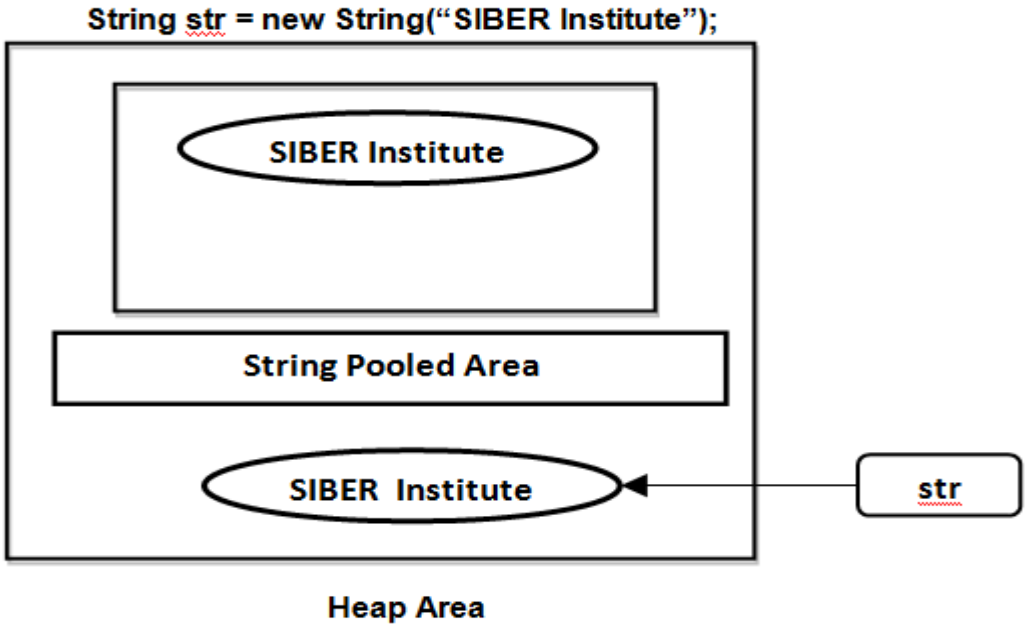


If we execute the statement str = "SIBER" in the next line, then java runtime will first check if the given String constant is present in String pooled area. If it is present then str will point to it, otherwise a new String constant is created as shown below.



Object Initialization (Dynamic):

In this method, a String object will be created in heap area (not inside String pooled area as in upper case). We can modify it. Also with same value, a String constant is also created in String pooled area, but the variable will point to String object in heap area only.

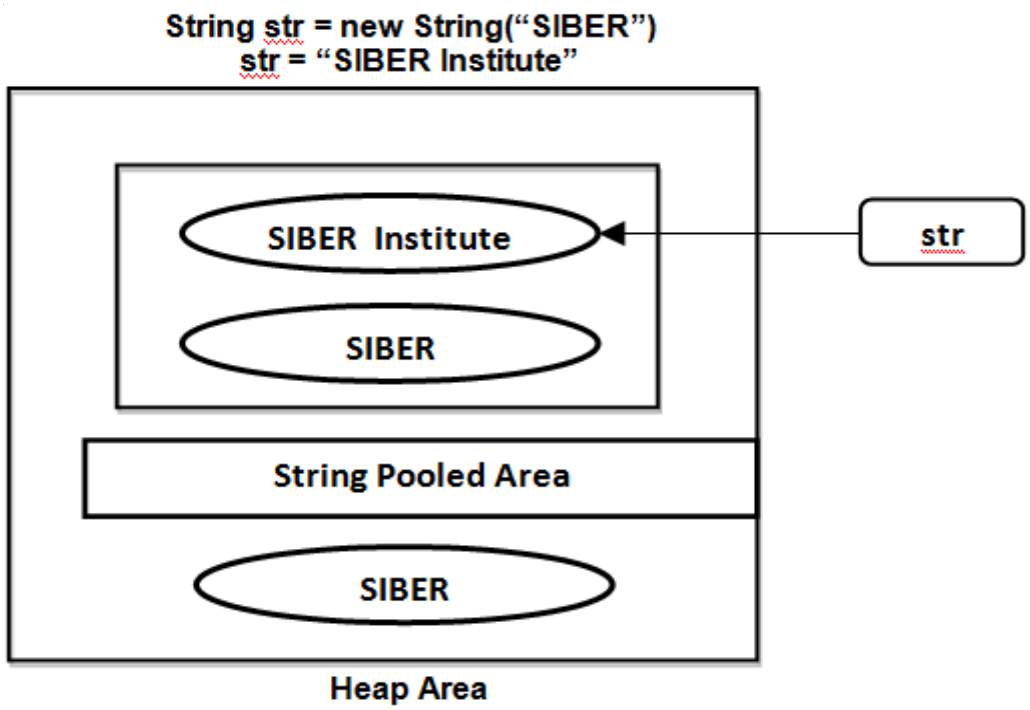


Examples:

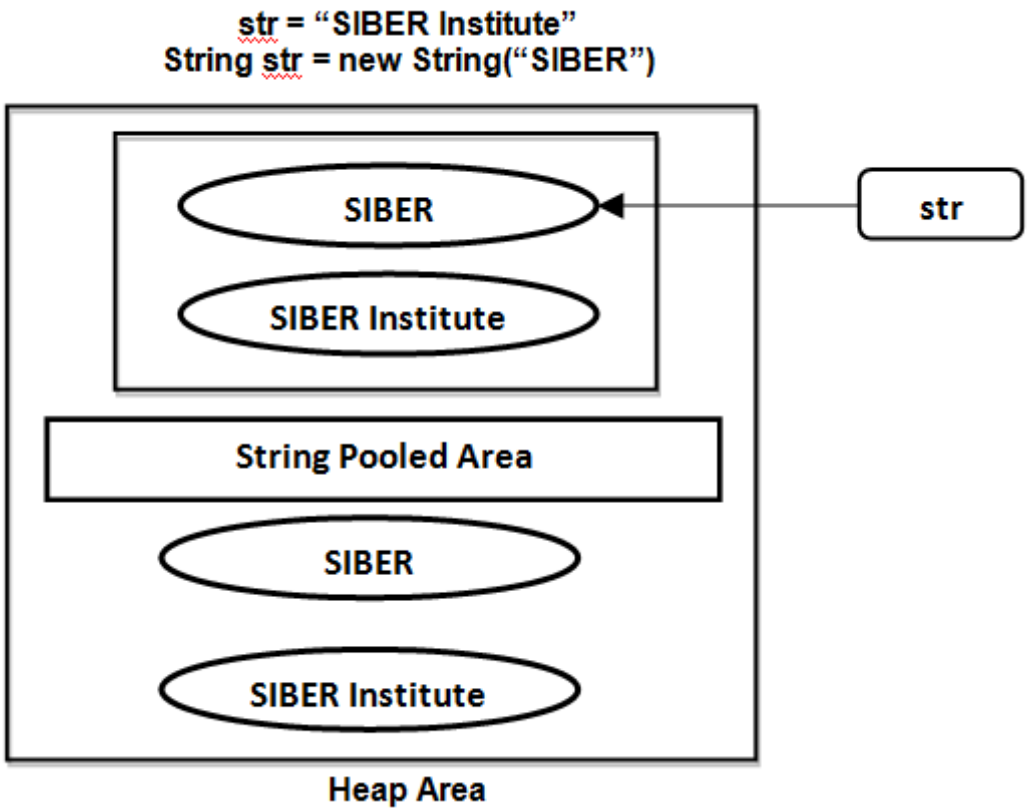
Consider the following example:

String str=new String(“SIBER”);

str=“SIBER Institute”;



The string “SIBER” is created in the heap area as well as string pool whereas the string “SIBER Institute” is created in the string pool as shown in the above Figure.



Now this is a direct assignment, so String constant with value "SIBER Institute" is created in String pooled area and str will point to that.

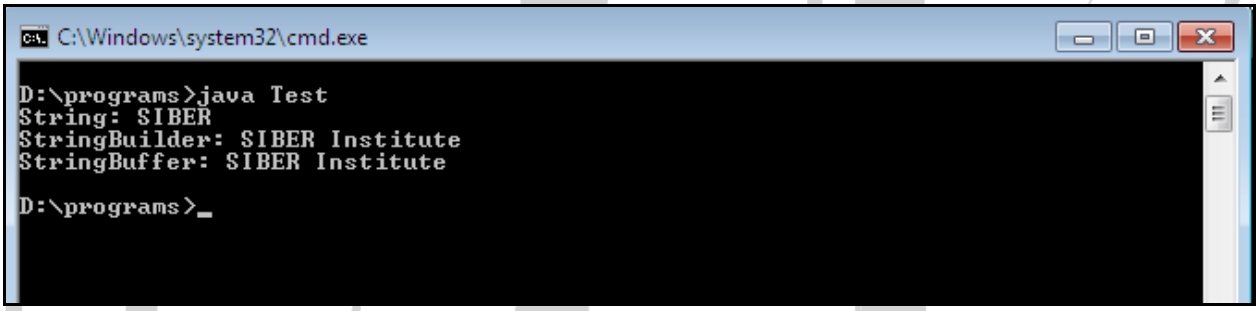
Note: If we again write str = new String("SIBER"), then it will create a new object with value "SIBER", rather than pointing to the available objects in heap area with same value. But if we write str = "SIBER", then it will point to String constant object with value "SIBER", present in String pooled area.

Q. No. 43 *Category : String Operations*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main(String[] args)
   {
      String s1 = "SIBER";
      s1.concat(" Institute");
      System.out.println("String: " + s1);

      StringBuilder s2 = new StringBuilder("SIBER");
      s2.append(" Institute");
      System.out.println("StringBuilder: " + s2);

      StringBuffer s3 = new StringBuffer("SIBER");
      s3.append(" Institute");
      System.out.println("StringBuffer: " + s3);
   }
}
```

Output :



Explanation :

Change the statement

```
s1.concat(" Institute");
```

to

```
s1=s1.concat(" Institute");
```

and re-execute the program.

On execution the following output is generated.

1. **Consider the statements**
   - ***i. s1 = "SIBER";***
   - ***ii. s1.concat(" Institute");***

Since the string objects are immutable, i.e. a String object cannot be modified without creating a new object, on execution of statement (ii) a new statement scoped string object "SIBER Institute" is created which is immediately set ready for garbage collection and s1 continues to point to the old string object "SIBER".

2. **Consider the statements**
   - ***i. s1 = "SIBER";***
   - ***ii. s1=s1.concat(" Institute");***

In the above example, on execution of statement (ii), a new String object "SIBER Institute" is created and s1 starts pointing to the new String object and old object "String" is ready for garbage collection.

In contrast to the String class both StringBuilder and StringBuffer classes are mutable, hence their content can be modified. Hence the generated output.

When to use which one :

- If a string is going to remain constant throughout the program, then use String class object because a String object is immutable.
- If a string can change (example: lots of logic and operations in the construction of the string) and will only be accessed from a single thread, using a StringBuilder is good enough.

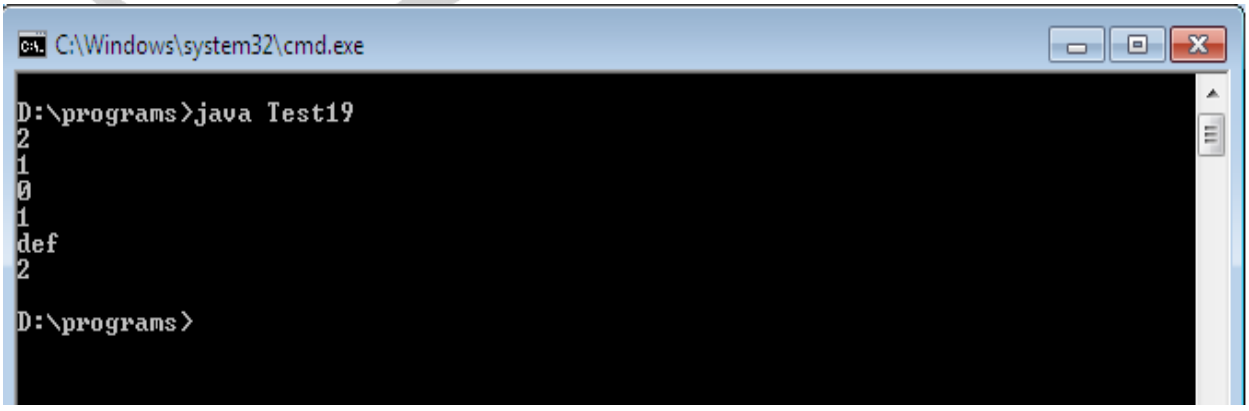
If a string can change, and will be accessed from multiple threads, use a StringBuffer because StringBuffer is synchronous so you have thread-safety.

Q. No. 44 *Category : Control Statements*

What is the output generated on execution of the following Java Program?

```
public class Switch2
{
  final static short x=2;
  final static int y=0;
  public static void main(String[] args)
  {
    for(int z=0;z<4;z++)
    {
      switch(z)
      {
        case x:
            System.out.println("0");
        case x-1:
            System.out.println("1");
           break;
        default:
            System.out.println("def");
        case x-2:
            System.out.println("2");
      }
    }
  }
}
```

Output :



Explanation :

Unlike C and C++ variables are allowed as case expressions in java which are load time entities so that their values are available at runtime. The rules employed in executed in switch statement are as follows:

1. Matching case statement is searched. If found, is executed till break statement is encountered.
2. If the matching case statement is not found, default block is executed till the break statement is encountered.
3. If the default statement is not encountered, the control skips to the statement next to the switch statement.

After substitution of value of x, the switch-case statement is transformed as shown below:

```
switch(z)
    {
     case 2:
         System.out.println("0");
     case 1:
         System.out.println("1");
        break;
     default:
         System.out.println("def");
     case 0:
         System.out.println("2");
    }
```

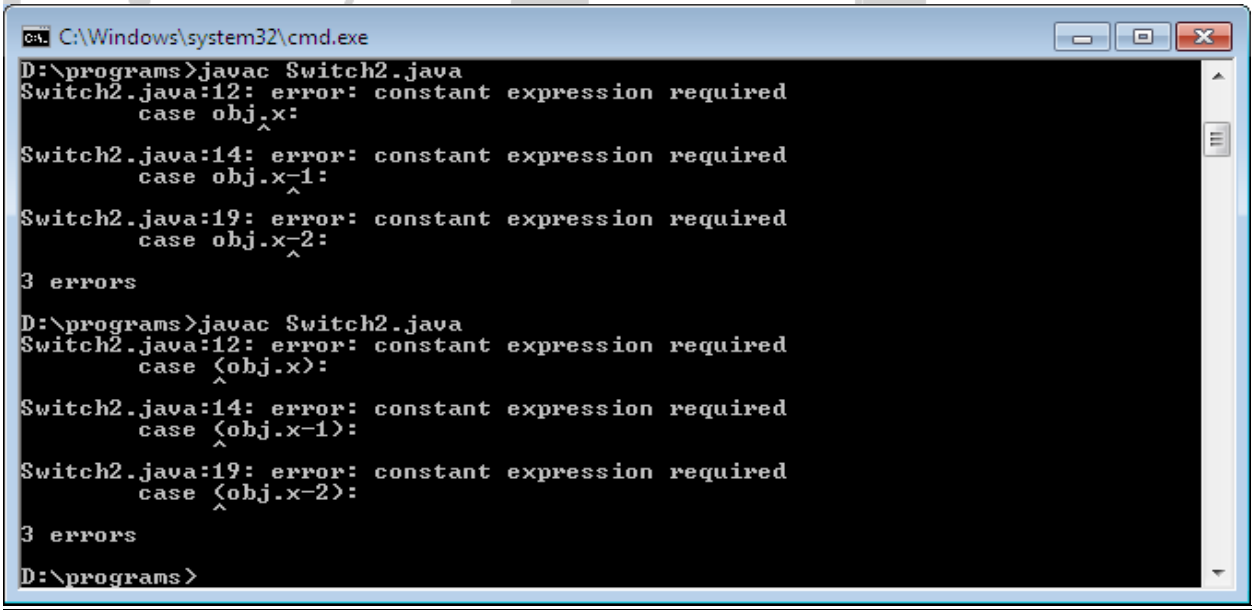
Different blocks executed for different values of z are depicted in the following table:

Value of z	*Blocks Executed*	*Output*
0	case 0	2
1	case 1 (break exists)	1
2	case 2 (break does not exist) and case 1	0 1
3	default (break does not exist)and case 0	def 2

Re-write the program as shown below and re-execute it.

```
public class Switch2
{
  final short x=2;
  final static int y=0;
  public static void main(String[] args)
  {
    Switch2 obj=new Switch2();
    for(int z=0;z<4;z++)
    {
      switch(z)
      {
        case obj.x:
            System.out.println("0");
        case obj.x-1:
            System.out.println("1");
            break;
        default:
            System.out.println("def");
        case obj.x-2:
            System.out.println("2");
      }
    }
  }
}
```

On execution of the program, the following output is generated.



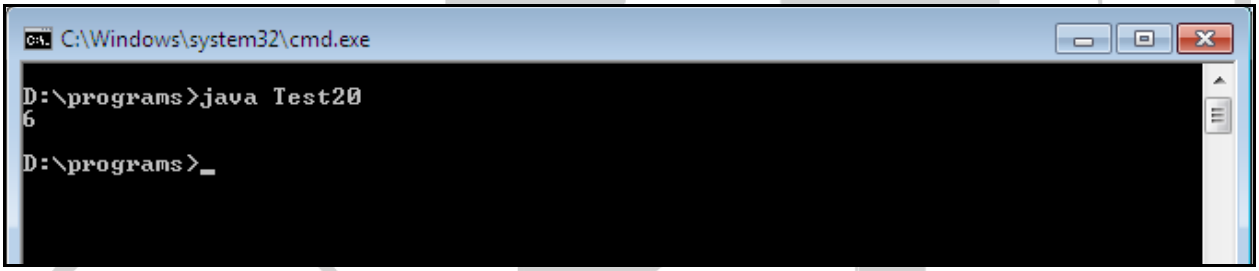
Since x is now a run-time variable is not available at load time.

Q. No. 45 *Category : operators*

What is the output generated on execution of the following Java Program?

```
public class Test20
{
        public static void main(String args[])
        {
        int x=12;
        System.out.println( x >> 1); ← Statement 6
        }
}
```

Output :



Explanation :

If the statement 6 in the above program, is replaced with

System.out.println(x >>> 1);

the same output is generated.

Similarly, x >> 2 and x >>> 2 will shift each bit to the right by 2 positions and the output generated is 3, which is equivalent to dividing the number by 2.

Both x >> n and x >>> n are equivalent to dividing the number x by 2, n times, is x is a positive number. Hence

$$x >> n \equiv x >>> n \equiv x/2^n$$

Unsigned shifting, on the other hand, ignores the sign of the number, and unsigned right shifting

a negative number will result in a positive number, the sign bit 1 is replaced with 0 -- which is why you generally shouldn't use the unsigned shift unless you are considering your number as unsigned.

In the above program, replace statement 5 with

int x=-12;

and re-execute the application.

The following output is generated.

-6

Now, replace the statement 6 with

System.out.println(x >>> 1);

and re-execute the application. The following output is generated.

2147483642 (= 2^{31} - 6)

An integer in Java is presented using 4 bytes with the most significant bit reserved for storing sign and >>> operator shifts the sign bit resulting in the designed number being output.

The following table summarizes the output generated in various cases.

Statement	***Result***
12 >> 1	6
12 >>> 1	6
12 >> 2	3
12 >>> 2	3
-12 >> 1	-6
-12 >> 2	-3
-12 >>> 1	2147483642 (=2^{31} - 6)
-12 >>> 2	1073741821 (=2^{30} - 3)

In brief,

>>operator moves the bits towards right. It doesn't move the sign bit.

>>>operator moves the bits towards right, the sign bit is also moved.

Q. No. 46 *Category : String Operations*

What is the output generated on execution of the following Java Program?

```
public class Test21
{
    public static void main(String args[])
{
      String s=30+30+"Java"+40+40;
      System.out.println(s);
    }
}
```

Output :

Explanation :

In java, String concatenation is implemented through the StringBuilder (or StringBuffer) class and its append method. String concatenation operator produces a new string by appending the second operand onto the end of the first operand. The string concatenation operator can concatenate not only string but primitive values also.

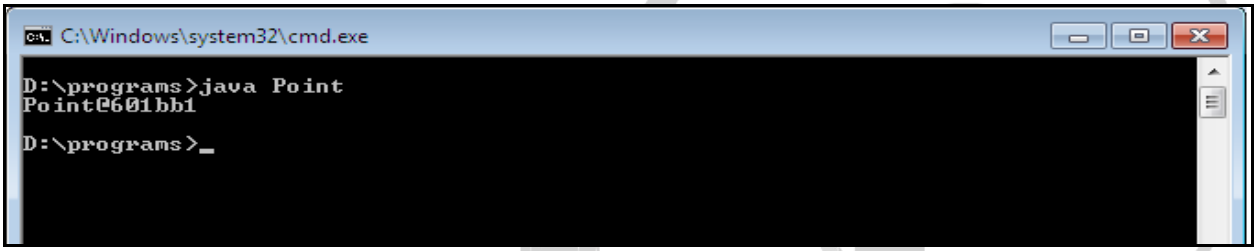
Q. No. 47 *Category : Abstract Data Types*

What is the output generated on execution of the following Java Program?

```
public class Point
{
  public int x,y;
  public Point()
  {
    x=y=0;
  }
```

```
 public Point(int x1,int y1)
 {
   x=x1;
   y=y1;
 }

 public static void main(String args[])
 {
   Point pt=new Point(10,20);
   System.out.println(pt);
 }
}
```

Output :



Explanation :

When an object is passed to System.out.println() method java runtime invokes toString() method of that class, which can be overridden to generate a formatted output. If the toString() method is not overridden, then the inherited Object class's toString() method is invoked which displays

<ClassName>@HashCode

where HashCode is a unique code used to identify an object and is different for different instances of the class.

Replace the main() method in the above program with the code given below:

```
public static void main(String args[])
 {
   Point pt=new Point(10,20);
   System.out.println(pt);
   Point pt1=new Point(100,200);
   System.out.println(pt1);
 }
```

and re-execute the application. The following output is generated.

Hence from the generated output it is evident that, the HashCode is unique to each object.

Q. No. 48 *Category : Abstract Data Types*

What is the output generated on execution of the following Java Program?

```
public interface MyInterface
{
int k = 4;   /* Line 3 */
}
```

Which three piece of codes are equivalent to line 3?

1. ***final int k = 4;***
2. ***public int k = 4;***
3. ***static int k = 4;***
4. ***abstract int k = 4;***
5. ***volatile int k = 4;***
6. ***protected int k = 4;***

Output :

(1), (2) and (3) are correct.

Explanation :

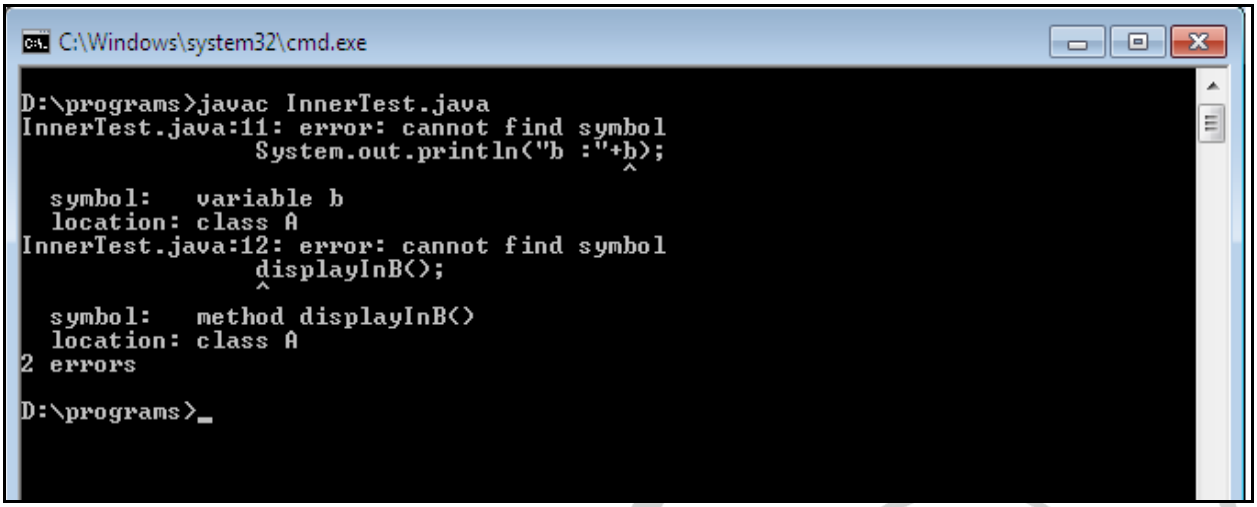
Interfaces can have constants, which are always implicitly public, static, and final. Interface constant declarations of public, static, and final are optional in any combination.

Q. No. 49 *Category : Inner Classes*

What is the output generated on execution of the following Java Program?

```
class A
{
        private int a=10;
        public A()
        {

        }
        public void displayInA()
        {
           System.out.println("a :"+a);
           System.out.println("b :"+b);
           displayInB();
        }
        class B
        {
          private int b=20;
          public void displayInB()
          {
             System.out.println("a :"+a);
             System.out.println("b :"+b);
          }

        }
}

public class InnerTest
{
  public static void main(String args[])
  {
    A obj=new A();
    obj.displayInA();
  }
}
```

Output :



Explanation :

A static class within another class is called ***Nested class*** or static Inner class.

A non-static class within another class is called ***Inner class***.

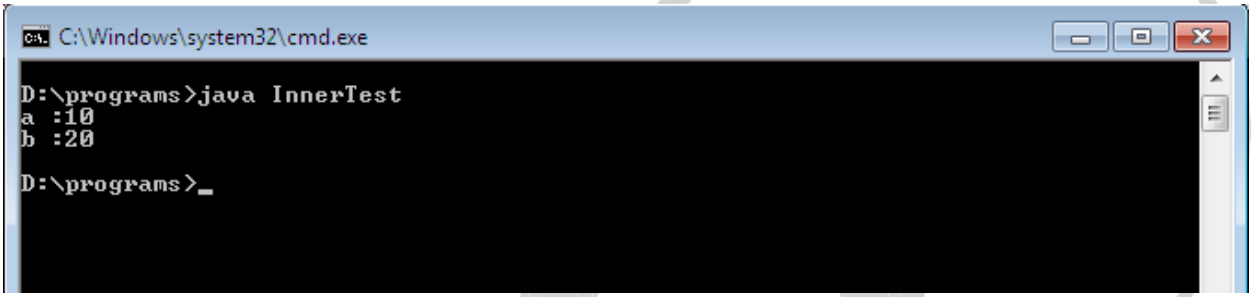
Nested class has direct access to the data members of outer class where as outer class can access data members of the inner class only by creating the object of inner class.

Rewrite the class A as shown below.

```
class A
{
        private int a=10;
        B obj;

        public A()
        {
          obj=new B();
        }

        public void displayInA()
        {
          obj.displayInB();
        }
```

```
        class B
        {
          private int b=20;


          public void displayInB()
          {
                System.out.println("a :"+a);
                System.out.println("b :"+b);
          }
        }
}
```

Re-execute the application, the following output is generated.

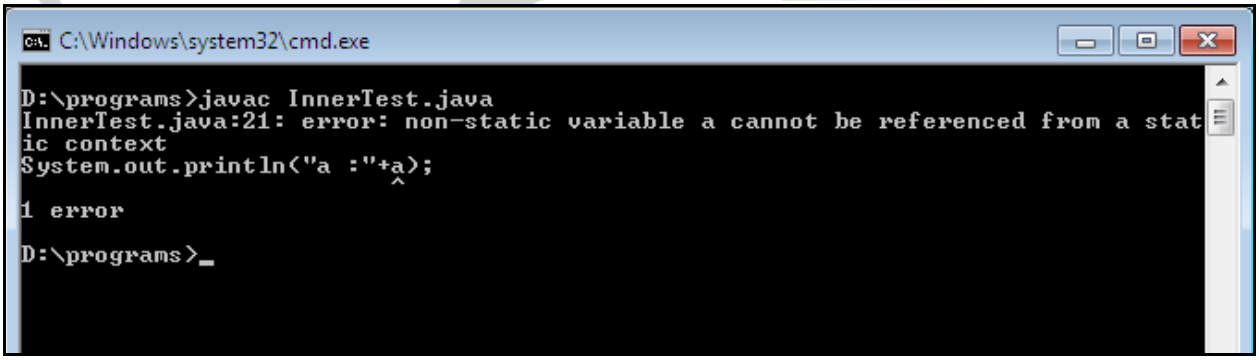


Hence we arrive at the following data access rule in the case of non-static nested classes.

Rule 1 : Nested inner class can access the data members of outer class directly.

Rule 2 : Outer class can access the members of the nested inner class only through the objects of that class.

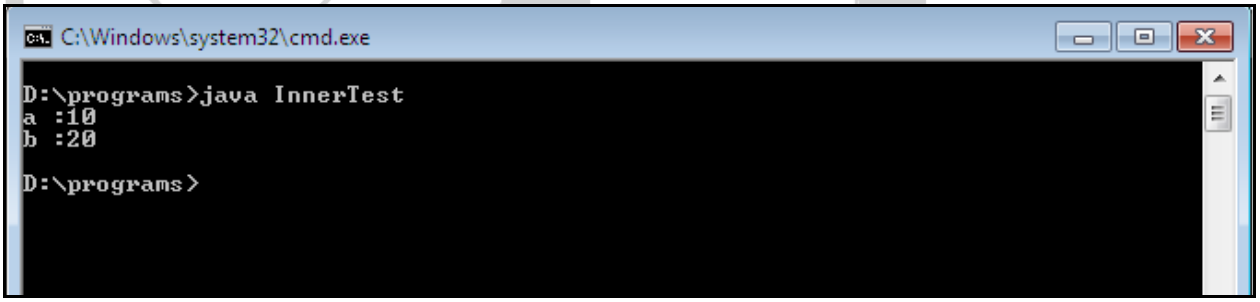
In the above example, if the nested class is declared as static and the program is re-executed, the following error message is generated.



Rewrite the class A as shown below.

```
class A
{
        private int a=10;
        B obj;
        public A()
        {
           obj=new B();
        }

        public void displayInA()
        {
          obj.displayInB();
        }

        static class B
        {
         private int b=20;
         A obj;

         public void displayInB()
         {
                obj=new A();
                System.out.println("a :"+obj.a);
                System.out.println("b :"+b);
         }
        }
}
```

Re-execute the application, the following output is generated.



Hence we arrive at Rule 3.

Rule 3 : Static nested class can access the members of the outer class only through the objects of that class.

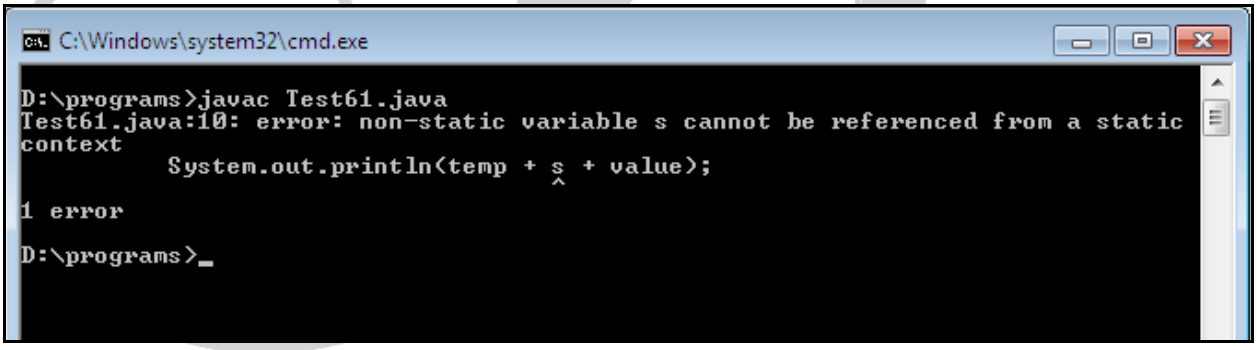
Non-Static nested classes are referred to as Inner classes.

Q. No. 50 *Category : Inner Classes*

What is the output generated on execution of the following Java Program?

```
public class Test61
{
   private static int value = 20;
   public int s = 15;
   public static int temp = 10;
   public static class Nested
   {
     private void display()
     {
        System.out.println(temp + s + value);
     }
   }

   public static void main(String args[])
   {
     Test61.Nested inner = new Test61.Nested();
     inner.display();
   }
}
```

Output :

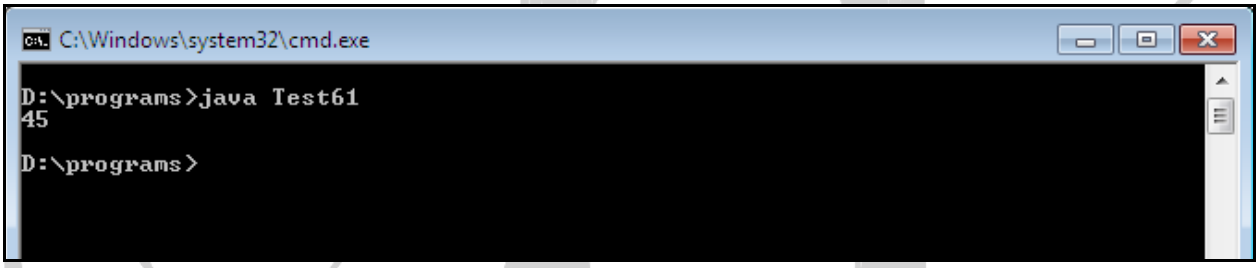


Explanation :

The static inner class, Nested, can only access the static members of the outer class directly. For accessing the non-static members an object of outer class must be created. Hence the correct version of the given program is : (Modified statements are shown in bold).

```
public class Test61
{
   private static int value = 20;
   public int s = 15;
   public static int temp = 10;
   public static class Nested
   {
    Test61 obj=new Test61();
    private void display()
    {
        System.out.println(temp + obj.s + value);
    }
   }

   public static void main(String args[])
   {
    Test61.Nested inner = new Test61.Nested();
    inner.display();
   }
}
```

On execution of the program, the following output is generated:



The following table depicts the accessibility issues in nested classes:

Data Members	***Type***	***Outer Class***	***Inner Class***
Outer Class	Non-Static	-	Direct Access
Outer Class	Static	-	Through instance of Outer Class
Inner Class	Non-Static	Through instance of Inner Class	-
Inner Class	Static	Through instance of Inner Class	-

Q. No. 51 *Category : Inner Classes*

What is the output generated on execution of the following Java Program?

```
public class Outer
{
public void someOuterMethod()
{
  //Line 5
}
public class Inner { }
public static void main(String[] argv)
 {
  Outer ot = new Outer();
  //Line 10
 }
}
```

Which of the following code fragments inserted, will allow to compile?

A. new Inner(); //At line 5

B. new Inner(); //At line 10

C. new ot.Inner(); //At line 10

D. new Outer.Inner(); //At line 10

Output :

Option A compiles without problem.

Explanation :

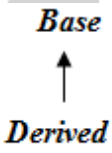
When the application is executed with remaining options, the following compiler errors are generated.

new Inner(); //At line 10	Outer.java:11: non-static variable this cannot be referenced from a static context new Inner(); ^ 1 error
new ot.Inner(); //At line 10	Outer.java:11: package ot does not exist new ot.Inner(); ^ 1 error
new Outer.Inner(); //At line 10	Outer.java:11: non-static variable this cannot be referenced from a static context new Outer.Inner(); ^ 1 error

Q. No. 52 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

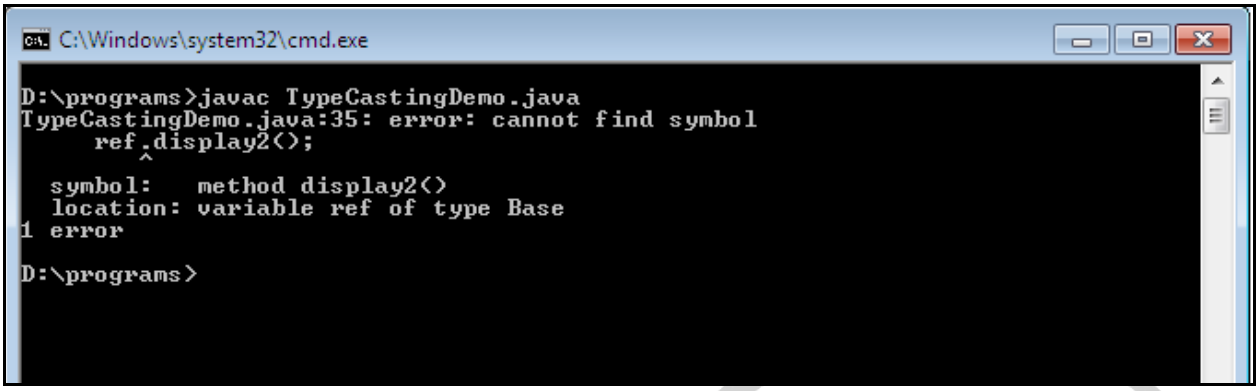
The following Java program implements the following class hierarchy.



The Base class contains two methods display and display1. The Derived class extends Base class and overrides display1() method and adds a new method display2().

```
class Base
{
  public void display()
  {
        System.out.println("Inside display of Base class...");
  }

   public void display1()
  {
        System.out.println("Inside display1 of Base class...");
  }
}

class Derived extends Base
{
  public void display1()
  {
        System.out.println("Inside display1 of Derived class...");
  }

  public void display2()
  {
        System.out.println("Inside display2 of Derived class...");
  }
}

public class TypecastingDemo
{
  public static void main(String args[])
  {
        Base ref;
        ref=new Derived();
        ref.display();
        ref.display1();
        ref.display2();
  }
}
```

Output :



Explanation :

In the given program, Base class has two member functions display() and display1(). The Derived class

- Inherits the member display() from the Base class
- Overrides the member display1() of Derived class and
- Add a new member function display2()

Consider the following statement

Base ref=new Base();

In the above statement, a Base class reference points to the object of Base class.

Hence ref can only be used for accessing the Base class functionality. Using ref, the Base class methods display() and display1() can be accessed.

Now, consider the statement,

Base ref=new Derived();

In this case, the base class reference is initialized with the object of Derived class.

Hence ref can be used to access the base class functionality inherited and overridden by the Derived class. Since there is no display() method in Derived class, the inherited version of display() method is invoked using ref whereas display1() method is overridden in the Derived class. Hence ref invokes the overridden method of Derived class. Since display2() method is newly added by Derived class. It can only be accessed by proper typecasting of ref to Derived class type as shown below:

((Derived)ref).display2().

Invoking the display2() method directly using ref generates the compiler error.

In the given program, replace the statement

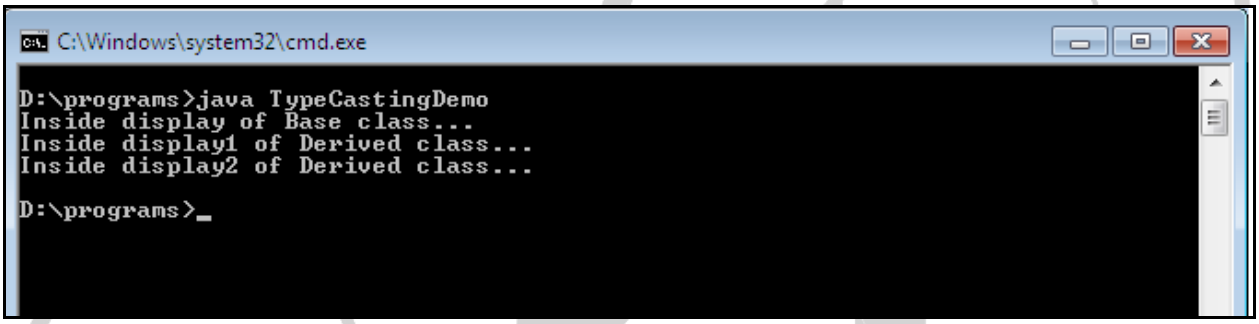
ref.display2()

with the one given below:

((Derived)ref).display2();

and re-execute the program.

On the re-execution of the program with the given modification, the following output is generated.



The following rules apply for the member function accessibility issues:

Rule 1 : Derived class has access to all the inherited members of base class (public, protected and default if both base and derived classes are in the same package).

Rule 2 :When derived class overrides the members of base class, the overridden members of derived class can be accessed through the derived class object stored in base class reference and base class can be accessed through the base class object stored in base class reference.

Rule 3 : The new methods added by the derived class can be accessed in any one of the following ways:

- through the derived class object stored in derived class reference directly.
- through the derived class object stored in base class reference through type casting.

Rule 4 : The new methods added by the derived class cannot be accessed by base class object stored in base class reference or derived class object stored in base class reference directly.

The following table depicts the method invocations in different cases:

Base ref = new Base();	
Statement	***Method Invoked***
ref.display()	display() method of Base class
ref.display1()	display1() method of Base class
ref.display2()	Compiler Error
Base ref = new Derived();	
ref.display()	Inherited display() method of Base class
ref.display1()	display1() method of Derived class
ref.display2()	Compiler Error
((Derived)ref).display2()	Display2() method of Derived class

Q. No. 53 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Base
{
   public void Print()
   {
      System.out.println("Base");
   }
}

class Derived extends Base
 {
   public void Print()
   {
      System.out.println("Derived");
   }
}
```

```
public class Test41
{
   public static void DoPrint( Base o )
   {
      o.Print();
   }
   public static void main(String[] args)
  {
      Base x = new Base();
      Base y = new Derived();
      Derived z = new Derived();
      DoPrint(x);
      DoPrint(y);
      DoPrint(z);
   }
}
```

Output :

Explanation :

Predicting the first line of output is easy. We create an object of type Base and call DoPrint(). In this case, an object of type Base is assigned to Base class reference. DoPrint calls the print function and we get the output ***Base***.

DoPrint(y) causes second line of output. Like C++, assigning a derived class reference to a base class reference is allowed in Java. Therefore, the expression Base y = new Derived() is a valid statement in Java. In DoPrint(), o starts referring to the same object as referred by y. Also, unlike C++, functions are virtual by default in Java. So, when we call o.print(), the print() method of Derived class is called due to run time polymorphism present by default in Java. Using the base class reference, the overridden methods of derived class can be invoked.

DoPrint(z) causes third line of output, we pass a reference of Derived type and the print() method of Derived class is called again. The point to note here is: unlike C++, object slicing doesn't happen in Java. Because non-primitive types are always assigned by reference

The following table depicts the method invocations in various cases:

	Method Invoked Print() of	
Statement	***Base Class Version***	***Derived Class Version***
Base x = new Base();	√	
Base y = new Derived();		√
Derived z = new Derived();		√

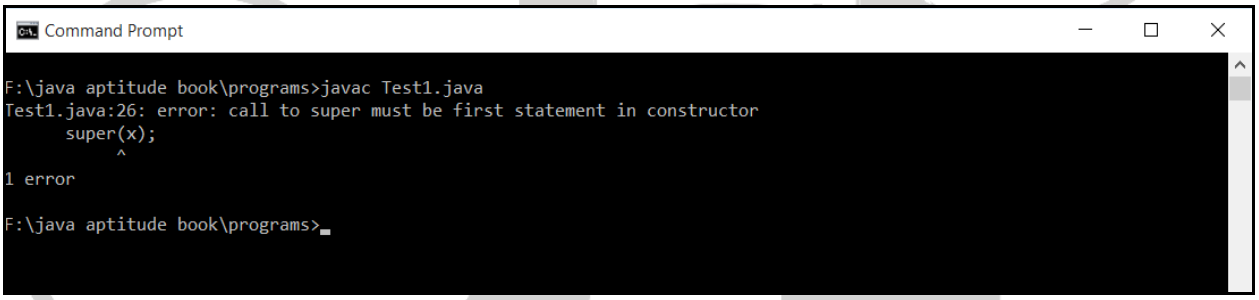
Q. No. 54 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Base
{
   int x;
   Base()
   {
   }
   Base (int x)
   {
     this.x=x;
   }
   public void getDetails()
   {
      System.out.printf("Base class ");
   }
}

class Derived extends Base
{
   int y;
   Derived()
   {
   }
```

```
    Derived (int x, int y)
    {
      this.y=y;
      super(x);
    }
    public void getDetails()
    {
        System.out.printf("Derived class\n");
        super.getDetails();
    }
}

public class Test1
{

    public static void main(String[] args)
    {
        Base obj = new Derived(10,20);
        obj.getDetails();
    }
}
```

Output :



Explanation :

Re-write the two argument constructor in the Derived class as shown below and re-execute the application.

```
Derived (int x, int y)
    {
      super(x);
      this.y=y;
    }
```

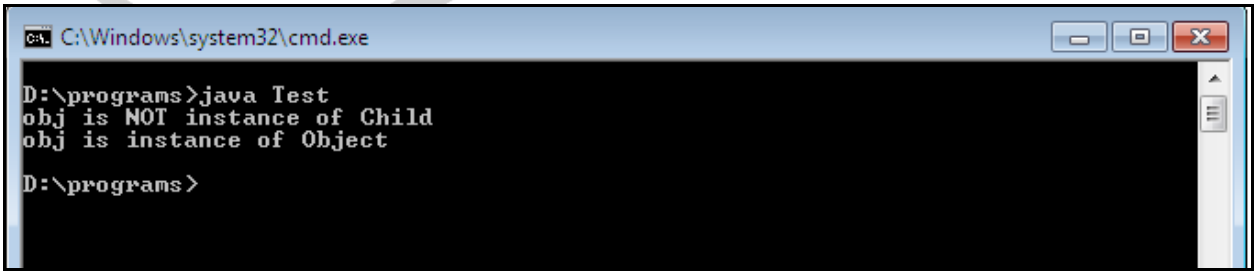
On execution of the program, the following output is generated.

Q. No. 55 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Parent { }
class Child extends Parent { }

class Test
{
   public static void main(String[] args)
   {
      Parent obj = new Parent();
      if (obj instanceof Child)
        System.out.println("obj is instance of Child");
      else
        System.out.println("obj is NOT instance of Child");

      if (obj instanceof Object)
        System.out.println("obj is instance of Object");
      else
        System.out.println("obj is NOT instance of Object");
   }
}
```

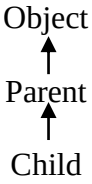
Output :



Explanation :

The java instanceof operator is employed in Java to test whether the object is an instance of the

specified type (class or subclass or interface). In the given program, the following class hierarchy exists.



An object is an instanceof a particular class is by default is also an instance of every other class above it in the class hierarchy but not an instance of the classes below it. Hence, in the above class hierarchy, an instance of Child class is also an instance of Parent and Object classes, but an object of Parent class is an instance of Parent and Object class but not an instance of Child class.

Q. No. 56 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

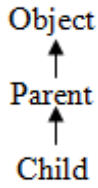
```
class Parent {  }
class Child extends Parent { }
class Test
{
   public static void main(String[] args)
   {
      Child obj = new Child();

      // A simple case
      if (obj instanceof Child)
        System.out.println("obj is instance of Child");
      else
        System.out.println("obj is NOT instance of Child");

      // instanceof returns true for Parent class also
      if (obj instanceof Parent)
        System.out.println("obj is instance of Parent");
      else
        System.out.println("obj is NOT instance of Parent");

      // instanceof returns true for all ancestors (Note : Object
      // is ancestor of all classes in Java)
```

```
        if (obj instanceof Object)
          System.out.println("obj is instance of Object");
        else
          System.out.println("obj is NOT instance of Object");
    }
}
```

Output :

Explanation :

The following class hierarchy exists in the given program.



obj is an instance of Child class. Hence it is also an instance of Parent class and Object class.

Q. No. 57 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

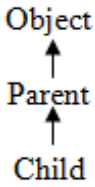
```
class Parent
{

}

class Child extends Parent
 {

}
```

```
class Test
{
   public static void main(String[] args)
   {
      // Reference is Parent type but object is
      // of child type.
      Parent obj = new Child();
      if (obj instanceof Child)
        System.out.println("obj is instance of Child");
      else
        System.out.println("obj is NOT instance of Child");
      if (obj instanceof Object)
        System.out.println("obj is instance of Object");
      else
        System.out.println("obj is NOT instance of Object");
   }
}
```

Output :

Explanation :

The following class hierarchy exists in the given program.



In the given program, obj is a Parent class reference initialized with Child class object. Hence obj is an instance of Child class. Hence it is also an instance of Parent class and Object class.

Q. No. 58 *Category : Inheritance*

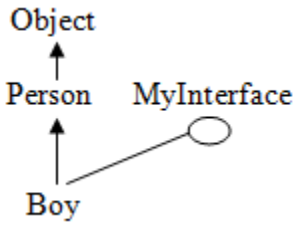
What is the output generated on execution of the following Java Program?

```
class Person
{

}

class Boy extends Person implements MyInterface
{
}

interface MyInterface
{
}

public class Test1
{
   public static void main(String[] args)
   {

      Person obj1 = new Person();
      Person obj2 = new Boy();

      // As obj is of type person, it is not an
      // instance of Boy or interface

      System.out.println("obj1 instanceof Person: " +
                  (obj1 instanceof Person));
      System.out.println("obj1 instanceof Boy: " +
                  (obj1 instanceof Boy));
      System.out.println("obj1 instanceof MyInterface: " +
                  (obj1 instanceof MyInterface));

      // Since obj2 is of type boy, whose parent class is
      // person and it implements the interface MyInterface
      // it is instance of all of these classes

      System.out.println("obj2 instanceof Person: " +
                  (obj2 instanceof Person));
      System.out.println("obj2 instanceof Boy: " +
                  (obj2 instanceof Boy));
      System.out.println("obj2 instanceof MyInterface: " +
                  (obj2 instanceof MyInterface));
   }
}
```

Output :

Explanation :

The following class hierarchy exists in the given program.



Consider the statement

Person obj1 = new Person();

In the above statement, obj1 is a Person class reference initialized with Person class object. Hence obj1 is an instance of Object and Person and not an instance of MyInterface and Boy.

Now, consider the statement

Person obj2 = new Boy();

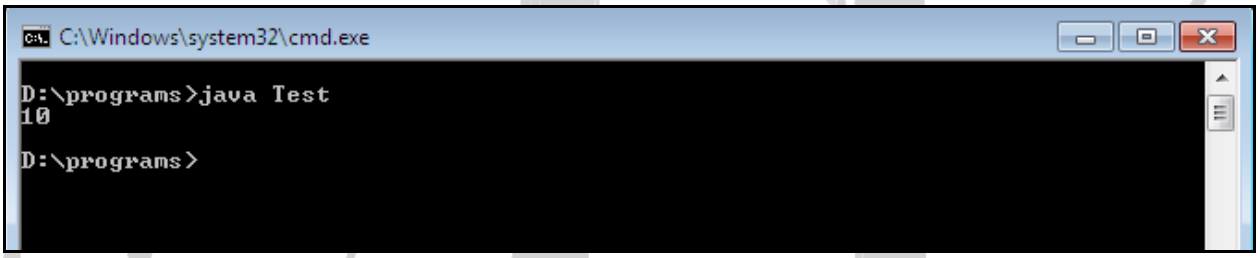
In the above statement, obj2 is a Person class reference initialized with Boy class object. Hence obj2is an instance of Boy class and also an instance of Person, an instance of MyInterface and an instance of Object.

Q. No. 59 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class A
{
   int x = 10;
}

class B extends A
{
   int x = 20;
}

public class Test
{
   public static void main(String args[])
   {
      A a = new B();
      System.out.println(a.x);
   }
}
```

Output :



Explanation :

The given java program illustrates the fact that runtime polymorphism cannot be achieved by data members

In above program, both class A(super class) and B(sub class) have a common variable 'x'. Now we create object of class B, referred by 'a' which is of type of class A. Since variables are not overridden, so the statement "a.x" will always refer to data member of super class.

In order to access the member x of class B using super class reference a, modify the program as shown below:

```
class A
{
   int x = 10;
}

class B extends A
{
   int x = 20;
}

public class Test
{
   public static void main(String args[])
   {
      A a = new B();
      System.out.println(((B)a).x);
   }
}
```

On execution of the above program, the following output is generated.

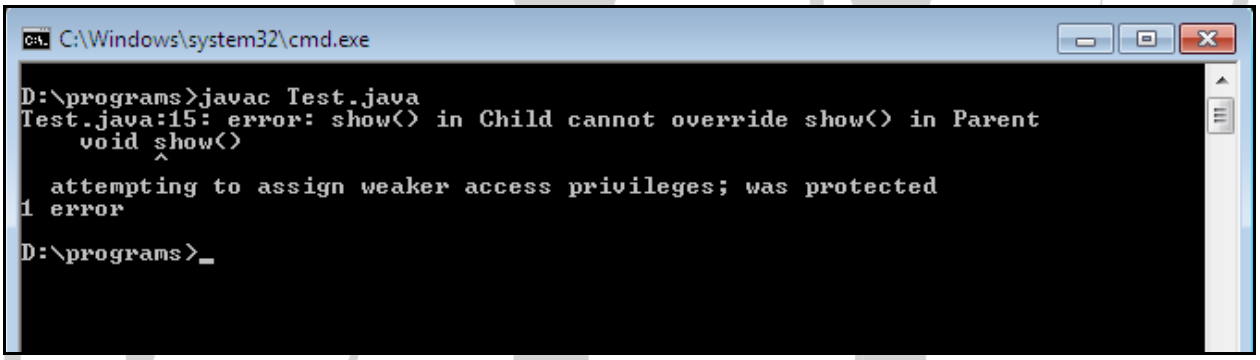
Q. No. 60 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Parent
{
   protected void show()
   {
      System.out.println("Inside show method of Parent");
   }
}
```

```
class Child extends Parent
{
   void show()
   {
     System.out.println("Inside show method of Child");
   }
}

public class Test
{
  public static void main(String[] args)
  {
    Parent p=new Child();
    p.show();
  }
}
```

Output :



Explanation :

In overridden method, we cannot give weaker access, but reverse is possible. For example, a protected method in base class cannot be overridden as private in derived class. In the given program, the show() method in the base class is protected whereas the overridden method in the derived class has default access which is weaker, hence the compiler error. This program won't compile as we have given weaker access in derived class. Interchange the access privileges of the two classes as shown in the following example and re-execute the program.

```
class Parent
{
   void show()
   {
      System.out.println("Inside show method of Parent");
   }
}

class Child extends Parent
{
   protected void show()
   {
     System.out.println("Inside show method of Child");
   }
}

public class Test
{
  public static void main(String[] args)
  {
     Parent p=new Child();
     p.show();
  }
}
```

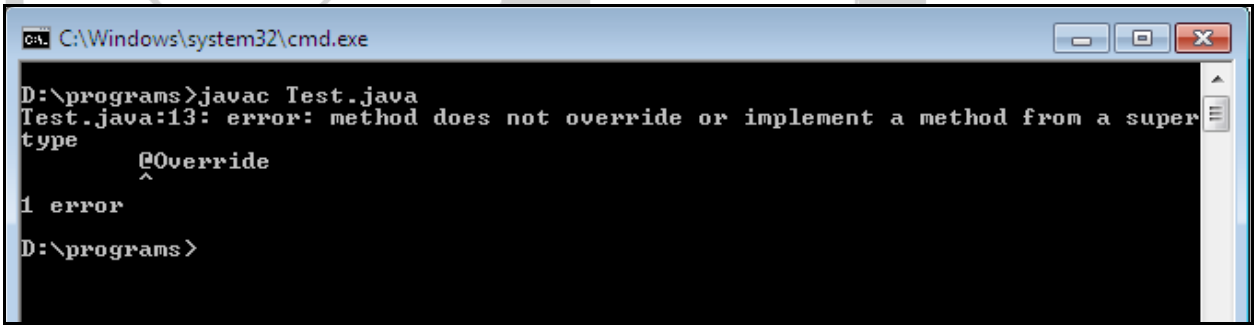
On execution of the program now, the following output is generated.

Q. No. 61 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static class superclass
   {
      static void print()
      {
         System.out.println("print in superclass.");
      }
   }

   public static class subclass extends superclass
   {
      @Override
      static void print()
      {
         System.out.println("print in subclass.");
      }
   }

   public static void main(String[] args)
   {
      superclass A = new superclass();
      superclass B = new subclass();
      A.print();
      B.print();
   }
}
```

Output :

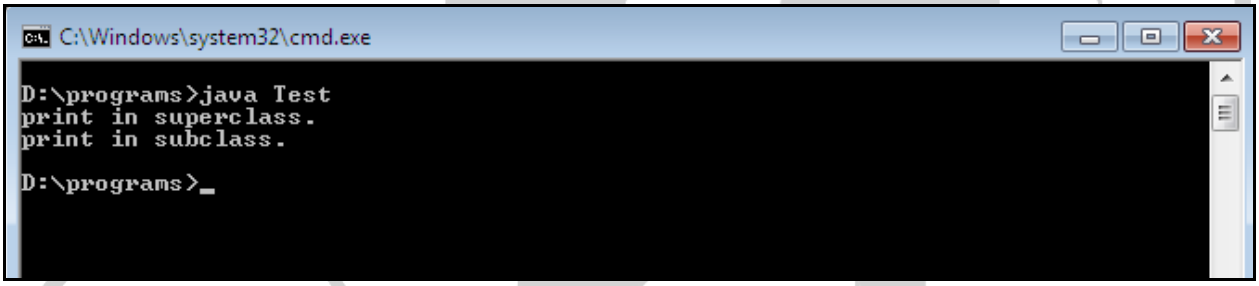


Explanation :

Static methods cannot be overridden.

In the given program, remove the annotation @Override and re-execute the application. The following output is generated.

Remove the static keywords in print() method definitions in both superclass and subclass and re-execute the application. The following output is generated.



- Methods are not static in this code.
- During compilation, the compiler has no idea as to which print has to be called since compiler goes only by referencing variable not by type of object and therefore the binding would be delayed to runtime and therefore the corresponding version of print will be called based on type on object.

Important Points

- private, final and static members (methods and variables) use static binding while for virtual methods (In Java methods are virtual by default) binding is done during run time based upon run time object.
- Static binding uses Type information for binding while Dynamic binding uses Objects to resolve binding.

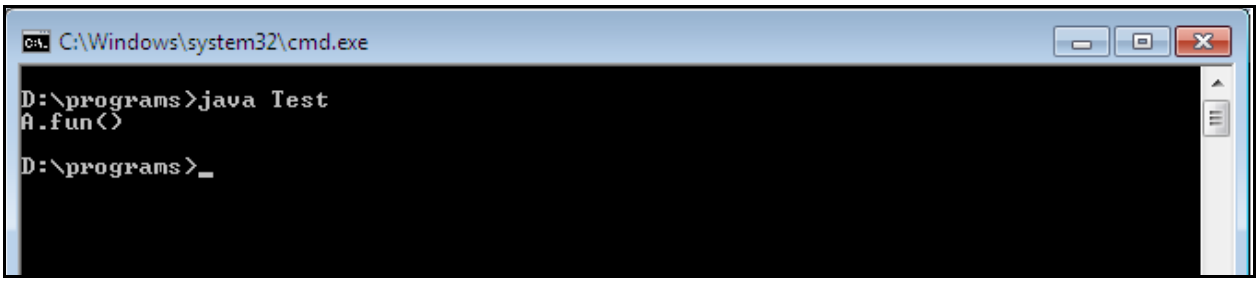
- Overloaded methods are resolved (deciding which method to be called when there are multiple methods with same name) using static binding while overridden methods using dynamic binding, i.e, at run time.

Q. No. 62 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class A
{
  static void fun()
 {
    System.out.println("A.fun()");
  }
}

class B extends A
{
  static void fun()
 {
    System.out.println("B.fun()");
  }
}

public class Test
 {
  public static void main(String args[])
{
    A a = new B();
    a.fun();
  }
}
```

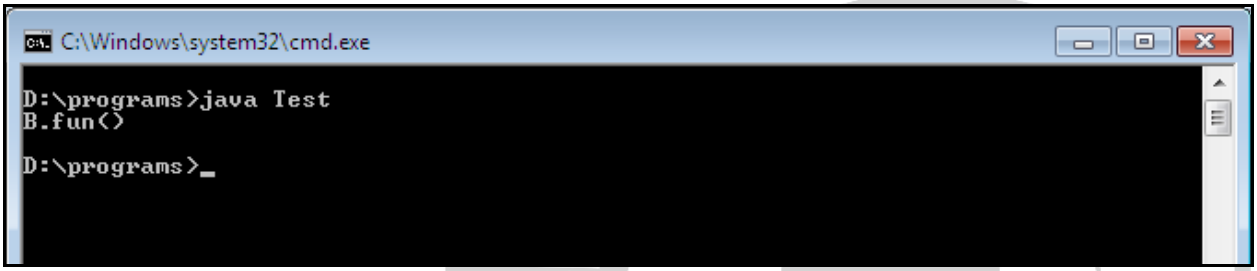
Output :



Explanation :

In Java, if name of a derived class static function is same as base class static function then the derived class static function shadows (or conceals) the base class static function. It is the case of static binding where function name is bound to the object at compilation time.

Remove the keyword static from the function definitions in both class A and class B and re-execute the application. The following output is generated.



The function call of instance methods is resolved at runtime, dynamic polymorphism. Now it resolves to the case of dynamic binding.

Hence, we arrive at the following rules.

Rule : 1. Static function calls are resolved at compilation time (static binding).

2. Instance method calls are resolved at runtime (dynamic binding).

Consider the following class hierarchy:

```
class A
{
  static void fun()
 {
    System.out.println("A.fun()");
  }
}

class B extends A
{
  static void fun()
 {
    System.out.println("B.fun()");
  }
}
```

On execution of the statements

```
A a = new A();
a.fun();
```

OR

```
A a = new B();
a.fun();
```

fun() method of class A is invoked.

On execution of the statements

```
B b = new B();
b.fun();
```

fun() method of class B is invoked.

Now, consider the following class hierarchy.

```
class A
{
void fun()
 {
    System.out.println("A.fun()");
  }
}

class B extends A
{
void fun()
 {
    System.out.println("B.fun()");
  }
}
```

On execution of the statements

```
A a = new A();
a.fun();
```

fun() method of class A is invoked.

On execution of the statements

```
A a = new B();
a.fun();
```

fun() method of class B is invoked.

On execution of the statements

```
B b = new B();
b.fun();
```

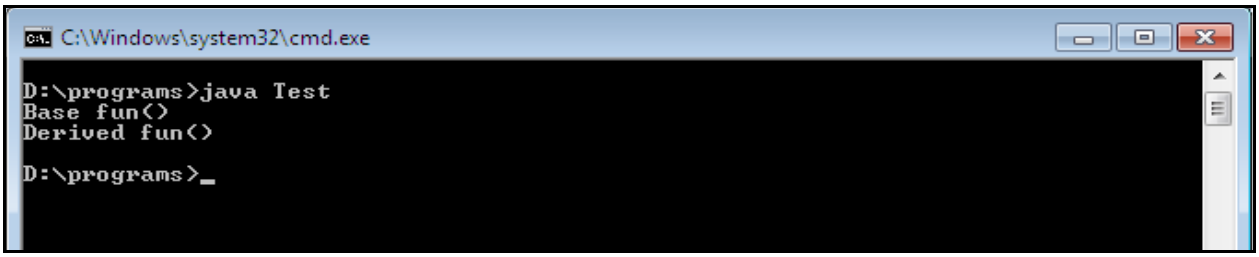
fun() method of class B is invoked.

Q. No. 63 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class A
{ }

class B extends A
{ }

class Base
{
   A fun()
   {
      System.out.println("Base fun()");
      return new A();
   }
}

class Derived extends Base
{
   B fun()
   {
      System.out.println("Derived fun()");
      return new B();
   }
}

public class Test
{
   public static void main(String args[])
   {
      Base base = new Base();
      base.fun();
      Derived derived = new Derived();
      derived.fun();
   }
}
```

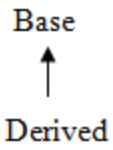
Output :



Explanation :

It is perfectly alright for overridden derived class method to return the derived class object of the super class object returned by the corresponding super class version. Such return types are called covariant return types. In the given program, the following class hierarchies exist.

and



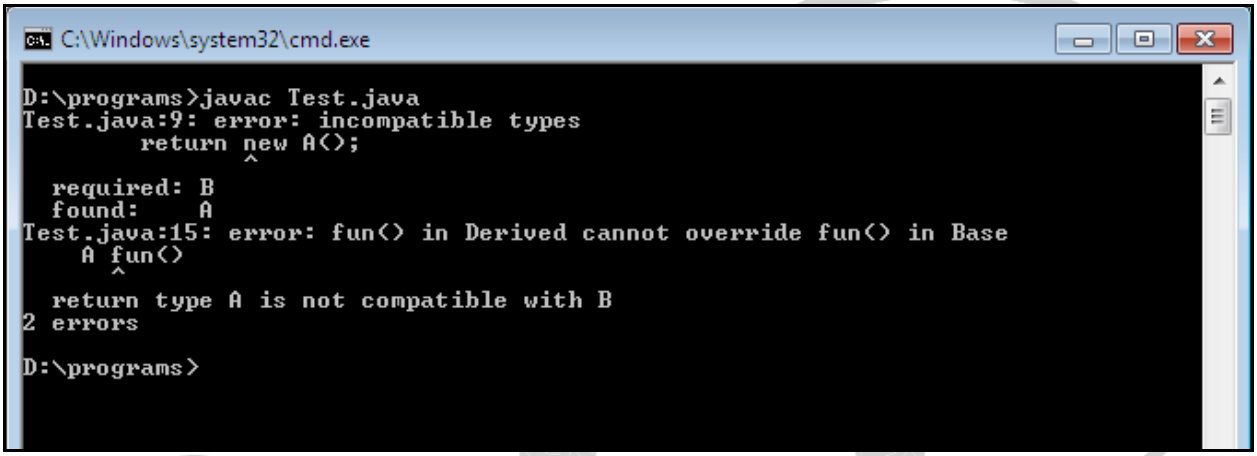
The method fun() in the base class returns a reference of class A. The overridden method fun() in the Derived class can return a reference of either class A or class B. Since A and B are related by the class hierarchy shown above.

Interchange the return types of function fun() in classes A and B as shown below and re-execute the application.

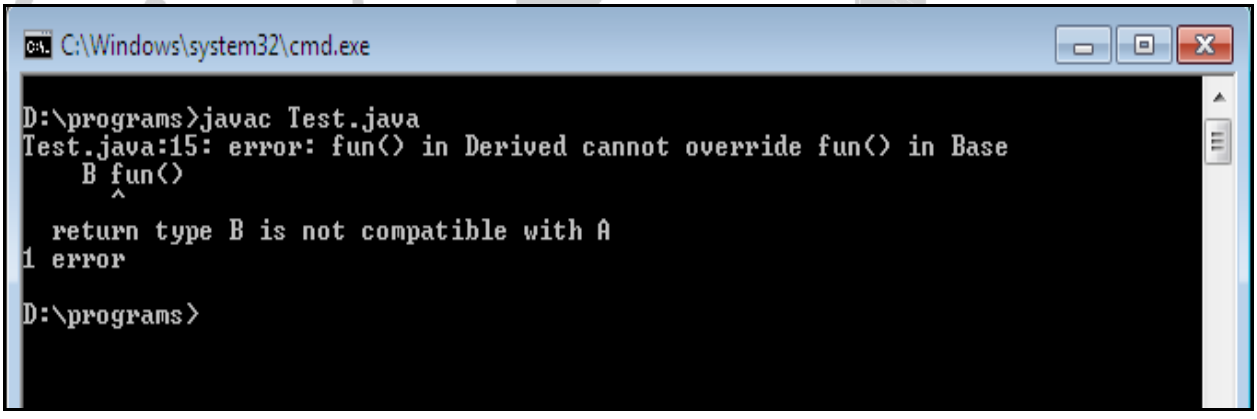
```
class Base
{
   B fun()
   {
      System.out.println("Base fun()");
      return new A();
   }
}
```

```
class Derived extends Base
{
   A fun()
   {
      System.out.println("Derived fun()");
      return new B();
   }
}
```

On execution of the application, the following output is generated.



Re-define class B as shown below and re-execute the application. The following output is generated.

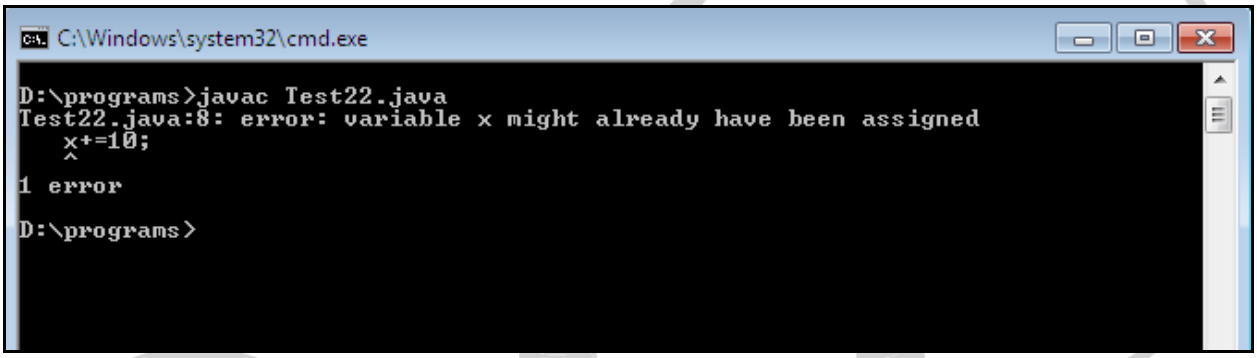


Q. No. 64 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
public class Test22
{
  public static void main(String args[])
 {
   final int x;
   x=10;
   System.out.println("x : " + x);
   x+=10; ← Statement 8
  }
 }
```

Output :



Explanation :

The uninitialized final variable is referred to as a blank final variable. But once the blank final variable is initialized in a code, its value cannot be changed later in the program.

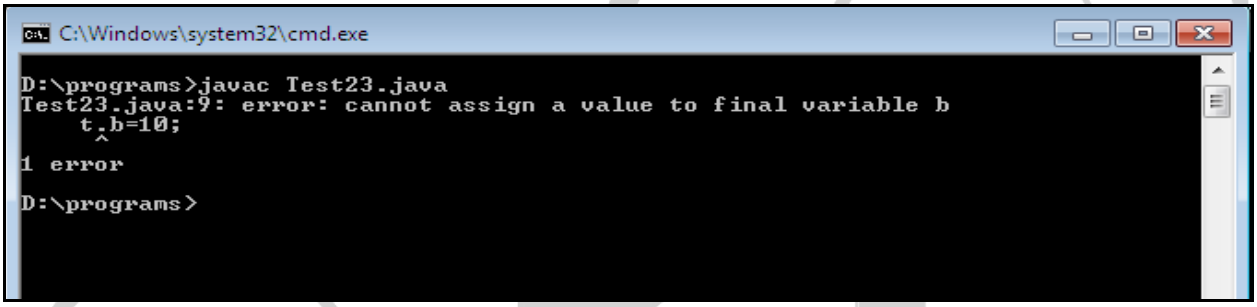
Comment out statement 8 in the above program and re-execute, the following output is generated.

Q. No. 65 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
public class Test23
{
  public final int b;
  public static void main(String args[])
  {
    Test23 t=new Test23();
    t.b=10;
    System.out.println(t.b);
  }
}
```

Output :



Explanation :

Class scoped blank final variables can only be initialized in a constructor.

Modify the above program as shown below and re-execute.

```
public class Test
{
  public final int b;
  public Test()
  {
    b=100;
  }

  public static void main(String args[])
  {
    Test t=new Test();
    System.out.println(t.b);
  }
}
```

On execution of the modified program, the following output is generated.

Q. No. 66 Category : Type Casting

Consider the following program

```
class Fruit
{
  boolean seedless=false;
  void display()
  {
     System.out.println("Inside fruit...");
  }
}

public class Grapes extends Fruit
{
boolean seedless=true;
  void display()
  {
System.out.println("Inside Grapes...");
  }

boolean isSeedless()
  {
     return true;
  }

  public static void main(String args[])
  {
    Grapes g=new Grapes();
    Fruit f=g;
    System.out.println(((Fruit)g).seedless);
  }
}
```

Which methods are invoked in the following method invocations?

f.display()
((Fruit)f).display();
f.isSeedless()
(Grapes)f).isSeedless()
f.seedless
g.seddless
((Fruit)g).seedless

Output :

f.display()	Grape's display() method
((Fruit)f).display();	Grape's display() method
f.isSeedless()	Compiler Error Grapes.java:26: cannot find symbol symbol : method isSeedless() location: class Fruit System.out.println(f.isSeedless());
(Grapes)f).isSeedless()	Grape's isSeedless() is invoked
f.seedless	Fruit's seedless member is accessed
g.seddless	Grape's seedless member is accessed
((Fruit)g).seedless	Fruit's seedless member is accessed

Q. No. 67 *Category : Java Basics*

How many class files are generated on compilation of the following Java program?

```
class A
{
  B b;
  A()
  {
        b=new B();
  }
```

```
  public void printAB()
 {
        System.out.println("Inside A...");
        b.printB();
 }
 class B
 {
  public void printB()
  {
        System.out.println("Inside B...");
  }
 }
}
public class Test
{
 public static void main(String[] args)
 {
  A obj=new A();
  obj.printAB();
 }
}
```

Output :

Three .class files are generated as shown below:

```
C:\Program Files\Java\jdk1.5.0\bin>dir *.class
 Volume in drive C is BOOTDISK
 Volume Serial Number is 04D8-29BF
 Directory of C:\Program Files\Java\jdk1.5.0\bin
07/24/2015  05:14 PM               478 A$B.class
07/24/2015  05:14 PM               529 A.class
07/24/2015  05:14 PM               303 Test.class
               3 File(s)          1,310 bytes
```

A$B.class corresponds to class B nested inside class A.

On execution of the above program the following output is generated.

```
Inside A...
Inside B...
```

Explanation :

On compilation of java source code, one file is generated for each class present in the code whose name is same as class name. In the case of nested classes, the name convention used is Consider third level of nesting as shown in the following Java program.

```
class A
{
  B b;
  A()
  {
          b=new B();
  }
  public void printABC()
  {
          System.out.println("Inside A...");
          b.printBC();
  }
  class B
  {
    C c;
    B()
    {
     c=new C();
    }
    public void printBC()
    {
          System.out.println("Inside B...");
          c.printC();
    }

    class C
    {
     public void printC()
     {
          System.out.println("Inside C...");
     }
    }
  }
}
```

```
public class Test1
{
  public static void main(String[] args)
  {
    A obj=new A();
    obj.printABC();
  }
}
```

On compilation of the above Java program the following .class files are generated.

```
C:\Program Files\Java\jdk1.5.0\bin>dir *.class
 Volume in drive C is BOOTDISK
 Volume Serial Number is 04D8-29BF

 Directory of C:\Program Files\Java\jdk1.5.0\bin
07/24/2015  05:21 PM               506 A$B$C.class
07/24/2015  05:21 PM               606 A$B.class
07/24/2015  05:21 PM               534 A.class
07/24/2015  05:21 PM               308 Test1.class
               4 File(s)          1,954 bytes
```

ABC refers to class C nested inside class B which in turn is nested inside class A.

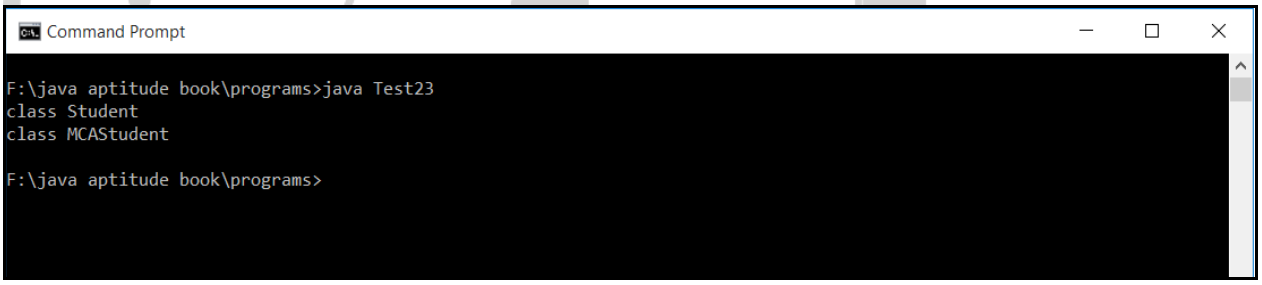
Q. No. 68 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Student
{
  Student()
  {
  }
  public void display()
  {
        System.out.println("In generic student class...");
  }
}
```

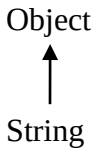
```
class MCAStudent extends Student
{
MCAStudent()
  {
  }
  public void display()
  {
        System.out.println("In MCA student class...");
  }
}

public class Test
{
 public static void main(String[] args)
 {
   Student sRef=new Student();
   MCAStudentmsRef=new MCAStudent();
   Object objRef=sRef;
   System.out.println(objRef.getClass());
   objRef=msRef;
   System.out.println(objRef.getClass());
 }
}
```

Output :



Explanation :

In Java, a subclass reference can be assigned to a super class reference because a subclass object can be used wherever a super class object can be used. This is called upcasting as references are assigned up the inheritance hierarchy. Consider the following example.



In Java every class is-a Object. Hence String is Object.

String strRef= new String("MCA");

Hence using upcasting we have,

Object objRef = strRef;

Both references denote the same String object after the assignment.

One might be tempted to invoke methods exclusive to the String subclass via the super class reference objRef, for example,

objRef.length()

This, however, will not work, as the compiler has no way of knowing what object the reference objRef is denoting. It is resolved only at runtime. It only knows the class of the reference. As the definition of the Object class does not have a method called length(), the statement objRef.length() would be flagged as a compile time error.

Hence we arrive at a general rule:

Rule : In upcasting, only the methods inherited from base class or overridden in the subclass can be invoked using the base class reference.

The following example invokes the getClass() method inherited from the Object class. Dynamic method lookup determines which method definition binds to the method signature getClass().

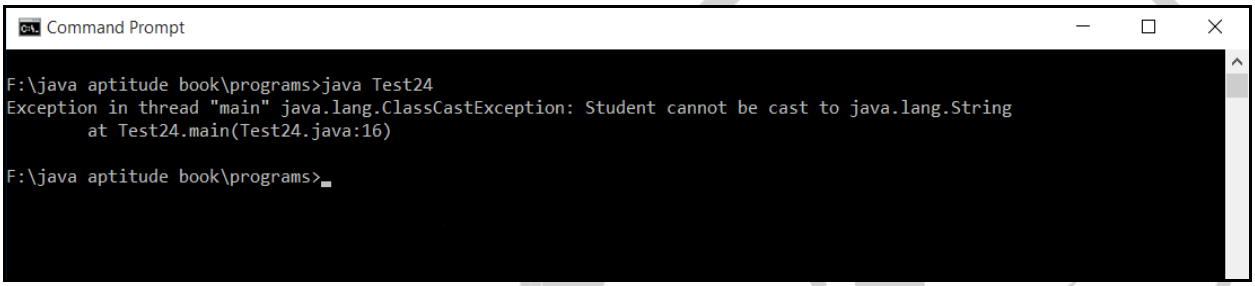
Q. No. 69 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
class Student
{
  Student()
  {
  }
  public void sayHello()
  {
        System.out.println("Hello from Student...");
  }
}
```

```
public class Test
{
  public static void main(String[] args)
  {
      Object objRef=new Student();
      String strRef=(String)objRef;
      System.out.println(strRef.getClass());
  }
}
```

Output :



Explanation :

Down-casting ensures the correct inheritance relationship between source and destination reference types at compile time. Hence the application successfully compiles. However, the cast can be invalid at runtime. In Q., if at runtime, the reference objRef denotes an object of class Object or some unrelated subclass of class Object, then obviously casting the value of such a reference to that of String would be illegal. In such a case ClassCastException would be thrown as runtime as shown in the example given above.

Replace the main() method in the above Java program with the code given below and re-execute the program.

```
Student studRef = (Student)objRef;
studRef.sayHello();
```

On execution of the program, the following output is generated.

Q. No. 70 *Category : Constructors*

What is the output generated on execution of the following Java Program?

```
class Test1
{
   Test1(int x)
   {
      System.out.println("Constructor called " + x);
   }
}

// This class contains an instance of Test1
public class Test43
{
   Test1 t1 = new Test1(10);

   Test43(int i)
   {
     t1 = new Test1(i);
   }

   public static void main(String[] args)
   {
       Test43 t2 = new Test43(5);
   }
}
```

Output :

Explanation :

In the given program, main() method instantiates an object of class Test43 invoking a one argument constructor. The initialization with class declaration in Java is like initialization using Initializer List in C++. An object of class Test1 is created invoking a one argument constructor passing it a value 10. The constructor generates the output.

Constructor called 10

Then, the one argument constructor of class Test43 is invoked with a value 5, which in turn creates an object of class Test1 invoking a one argument constructor passing it a value of 5. The constructor generates the output

Constructor called 5.

Hence the program generates the output

Constructor called 10

Constructor called 5.

The thumb rule is that in the object creation, the members are initialized first and then the constructor is invoked which may overwrite the values of data members.

Consider the following program

```
public class Test
{
   int x=10;

   Test(int i)
   {
     x=i;
   }

   public void display()
   {
      System.out.println("x : "+x);
   }

   public static void main(String[] args)
   {
       Test t = new Test(5);
       t.display();
   }
}
```

On execution of the above program, the following output is generated.

In the execution of the above program, on creating the object of the class Test, the data member x is first initialized to 10 and then one argument constructor is invoked with the value 5 which overwrites the value of x to 5 as is revealed in the output.

Q. No. 71 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class First
{
   public First() {  System.out.println("a"); }
}

class Second extends First
{
   public Second()  {  System.out.println("b"); }
}

class Third extends Second
{
   public Third()   {  System.out.println("c"); }
}

public class Test50
{
   public static void main(String[] args)
   {
      Third c = new Third();
   }
}
```

Output :

Explanation :

In the case of class inheritance, the constructors are invoked in the reverse order of their inheritance. Hence, while creating a new object of 'Third' type, before calling the default constructor of Third class, the default constructor of super class i.e., Second class, is called and again before the default constructor of super class, default constructor of its super class, i.e. First class is called.

Q. No. 72 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class First
{
   int i = 10;

   public First(int j)
   {
      System.out.println(i);
      this.i = j * 10;
   }
}

class Second extends First
{
   public Second(int j)
   {
      super(j);
      System.out.println(i);
      this.i = j * 20;
   }
}
```

```
public class Test51
{
   public static void main(String[] args)
   {
      Second n = new Second(20);
      System.out.println(n.i);
   }
}
```

Output :

Explanation :

Since the 'Second' class doesn't declare a variable 'i', the variable is inherited from the super class. Also, the constructor of parent is called when we create an object of Second. During the execution of the program, the main() method creates the object of Second class and invokes a parameterized constructor with a value 20. The Second class constructor in turn invokes the parameterized constructor of First class passing it the same value. The First class constructor first prints the value of i (=10) and then modifies its value to 200. The control is then returned to the Second class constructor. It prints the value of i (=200) and then modifies its value to 400. Finally, main() method prints the same (=400). Hence the given program generates the output

10

200

400

Q. No. 73 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Alpha
{
   public String type = "a ";
   public Alpha()
 {
      System.out.print("alpha ");
}
}

public class Beta extends Alpha
{
   public Beta()
{
System.out.print("beta ");
 }

   void go()
   {
      type = "b ";
      System.out.print(this.type + super.type);
   }

   public static void main(String[] args)
   {
      new Beta().go();
   }
}
```

Output :

Explanation :

The statement new Beta().go() executes in two phases. In first phase *Beta* class constructor is called. There is no instance member present in *Beta* class. So now *Beta* class constructor is executed. As *Beta* class extends *Alpha* class, so call goes to *Alpha* class constructor as first statement by default(Put by the compiler) is super() in the *Beta* class constructor. Now as one

instance variable(*type*) is present in *Alpha* class, so it will get memory and now *Alpha* class constructor is executed, then call return to *Beta* class constructor next statement. So *alpha beta* is printed.

In second phase *go()* method is called on this object. As there is only one variable(*type*) in the object whose value is *a*. So it will be changed to *b* and printed two times. The super keyword here is of no use.

Modify the program as shown below: (The newly added statement is shown in bold).

```
class Alpha
{
   public String type = "a ";
   public Alpha()
  {
     System.out.print("alpha ");
  }
}

public class Beta extends Alpha
{
   public String type;
   public Beta()
 {
    System.out.print("beta ");
 }

   void go()
   {
      type = "b ";
      System.out.print(this.type + super.type);
   }
   public static void main(String[] args) {
      new Beta().go();
   }
}
```

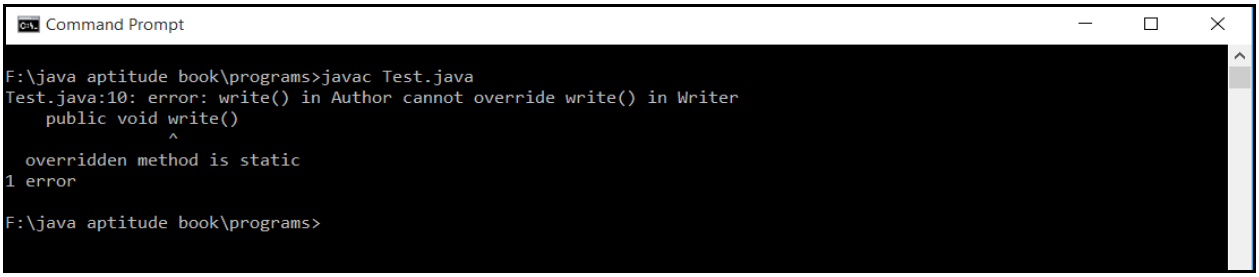
On execution of the above program now, the following output is generated.

Q. No. 74 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Writer
{
   public  static void write()
   {
System.out.println("Writing...");
   }
}
class Author extends Writer
{
   public void write()
   {
      System.out.println("Writing book");
   }
}

public class Test
{
   public static void main(String[] args)
   {
      Writer w = new Author();
      w.write();
   }
}
```

Output :



Explanation :

The given program generates a compiler error since the static methods cannot be overridden in the derived class.

Add a static qualifier to the write() method in the Author class and re-execute the program. The following output is generated.

Modify the main() method as shown below and re-execute the program.

```
public static void main(String[] args)
   {
      Author a = new Author();
      a.write();
   }
```

On execution of the program, the following output is generated.

The following table depicts the invocation of different members of the class in both static and non-static contexts.

Case 1 : write() method is declared as static in both Writer and Author classes

Statement	***Method Invoked***
Writer w = new Writer(); w.write();	write() method of Writer class
Writer w = new Author(); w.write();	write() method of Writer class

Case 2 : write() method is declared as non-static in both Writer and Author classes

Statement	***Method Invoked***
Writer w = new Writer(); w.write();	write() method of Writer class
Writer w = new Author(); w.write();	write() method of Author class

Q. No. 75 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Writer
{
   public  static void write()
   {
      System.out.println("Writing...");
   }
}

class Author extends Writer
{
   public  static void write()
   {
      System.out.println("Writing book");
   }
}

public class Programmer extends Author
{
   public  static void write()
   {
      System.out.println("Writing code");
   }

   public static void main(String[] args)
   {
      Author a = new Programmer();
      a.write();
   }
}
```

Output :

Explanation :

Since static methods can't be overridden, it doesn't matter which class object is created. As *a* is a *Author* referenced type, so always *Author* class method is called. If we remove *write()* method from *Author* class then *Writer* class method is called, as *Author* class extends *Writer* class.
The following table depicts the invocation of different members of the class in both static and non-static contexts.

Case 1 : write() method is declared as static in Writer, Author and Programmer classes

Statement	***Method Invoked***
Writer w = new Writer(); w.write();	write() method of Writer class
Writer w = new Author(); w.write();	write() method of Writer class
Writer w = new Programmer(); w.write();	write() method of Writer class

Case 2 : write() method is declared as non-static in Writer, Author and Programmer classes

Statement	***Method Invoked***
Writer w = new Writer(); w.write();	write() method of Writer class
Writer w = new Author(); w.write();	write() method of Author class
Writer w = new Programmer(); w.write();	write() method of Programmer class

If we remove write() method from the Author class and invoke write() method on the Author class as shown below, then write() method of Writer class is invoked as Author class inherits

Writer class.

```
class Writer
{
public  static void write()
   {
      System.out.println("Writing");
   }
}
class Author extends Writer
{

}

public class Programmer extends Author
{
   public  static void write()
   {
      System.out.println("Writing code");
   }

   public static void main(String[] args)
   {
      Author a = new Author();
      a.write();
   }
}
```

On execution of the above program, the following output is generated.

Q. No. 76 *Category : Inheritance*

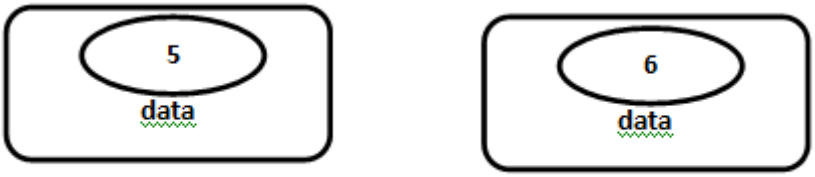
What is the output generated on execution of the following Java Program?

```
class Base
{
   private int data;

   public Base()
   {
      data = 5;
   }

   public int getData()
   {
      return this.data;
   }
}

class Derived extends Base
{
   private int data=6;
   public Derived()
   {
      super();
   }
   public int getData()
   {
      return data;
   }

   public static void main(String[] args)
   {
      Derived myData = new Derived();
      System.out.println(myData.getData());
   }
}
```

Output :

Explanation :

In the given program, each of the classes Base and Derived has its own copy of the private data member data. The main() method creates an object of Derived class which initializes its data member, data to 6 and invokes the super class constructor which initializes super class data member data to 5 as shown in the following Figure.



Base Class **Derived Class**

The subsequent statement in the main() method then prints the value of data member, data in the derived class.

Hence the program generates the output 6.

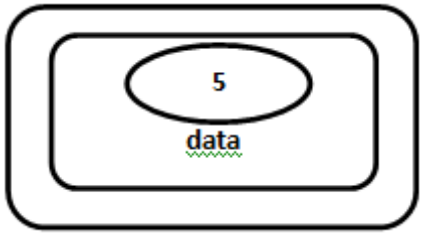
Modify the program as shown below and re-execute it.

```
class Base
{
   protected int data;

   public Base()
   {
      data = 5;
   }

   public int getData()
   {
      return this.data;
   }
}

class Derived extends Base
{

   public Derived()
   {
      super();
   }
```

```
    public int getData()
    {
        return data;
    }

    public static void main(String[] args)
    {
        Derived myData = new Derived();
        System.out.println(myData.getData());
    }
}
```

On execution of the program, the following output is generated,

In the above program, the access specifier of data member, data in the Base class is changed to protected which is inherited by the Derived class. Now, there is a single copy of data which is shared by both Base class and Derived class as shown in the following Figure.



The main() method creates an object of Derived class which invokes a super class constructor to initialize the data member, data to 5. The subsequent statement in the main() method then displays the value of data member data, inherited by Derived class.

Hence the program generates the output 5.

Q. No. 77 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Base
{
   int multiplier(int data)
   {
      data=data*5;
      return data;
   }
}

public class Derived extends Base
{
   private static int data;
   public Derived()
   {
      data = 25;
   }
   public static void main(String[] args)
   {
      Derived temp = new Derived();
      System.out.println(temp.multiplier(data));
      System.out.println(temp.multiplier(data));
   }
}
```

Output :

Explanation :

In the given program, the main() method creates an object of Derived class which invokes a one argument constructor to initialize its data member, data to 25. main() method then invokes the multiplier() method on Derived class which is inherited from Base class. In Java, primitive data types are passed by reference. Hence the multiplier() method maintains its own copy of data which is multiplied by 5 and the same is returned by the method. Hence the program outputs the value 125. The same task is repeated during the second call to multiplier() method on the

Derived class object and the program prints 125 for the second time. Hence the given program generates the output,

125
125.

Modify the program as shown below and re-execute it.

```
class Base
{

}

public class Derived extends Base
{
   private static int data;
   public Derived()
   {
      data = 5;
   }

   int multiplier()
   {
      data=data*5;
      return data;
   }
   public static void main(String[] args)
   {
      Derived temp = new Derived();
      System.out.println(temp.multiplier());
      System.out.println(temp.multiplier());
   }
}
```

In the above program, the Derived class declares a static data member, data which is initialized to 5 in the default constructor. multiplier() method is invoked two times multiplying the previous value of data by 5 in each call. Since data is declared as a static data member, its value

is maintained between function calls.

The execution of statements in the main() method is depicted in the following table.

Statement	*Description*	*Value of data*	*o/p*
Derived temp = new Derived();	Instantiates an object of Derived class and invokes a default constructor	5	
System.out.println(temp.multiplier());	Invokes a multiplier() method of Derived class	25	25
System.out.println(temp.multiplier());	Invokes a multiplier() method of Derived class Since data is a static member its value is retained in the previous call,	125	125

Q. No. 78 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Person
{
   private void who()
   {
      System.out.println("Inside private method Person(who)");
   }

   public static void whoAmI()
   {
      System.out.println("Inside static method, Person(whoAmI)");
   }

   public void whoAreYou()
   {
      who();
      System.out.println("Inside virtual method, Person(whoAreYou)");
   }
}
```

```
class Kid extends Person
{
   private void who()
   {
      System.out.println("Kid(who)");
   }

   public static void whoAmI()
   {
      System.out.println("Kid(whoAmI)");
   }

   public void whoAreYou()
   {
      who();
      System.out.println("Kid(whoAreYou)");
   }
}
public class Test52
{
   public static void main(String args[])
   {
      Person p = new Kid();
      p.whoAmI();
      p.whoAreYou();
   }
}
```

Output :

Explanation :

Static binding (or compile time) happens for static methods. Here p.whoAmI() calls the static

method so it is called during compile time hence results in static binding and prints the method in People class, whereas p.whoAreYou() calls the method in Kid class since by default Java takes it as a virtual method i.e, dynamic binding

Q. No. 79 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class A
{
  B b;
  public A()
  {
    b=new B();
  }
}

class B
{
  A a;
  public B()
  {
    a=new A();
  }
}


public class Test25
{
  public static void main(String args[])
  {
     A aref= new A();
System.out.println("Exiting main...");
  }
}
```

Output :

```
Command Prompt

F:\java aptitude book\programs>java Test24
Hello from Student...

F:\java aptitude book\programs>javac Test25.java

F:\java aptitude book\programs>java Test25
Exception in thread "main" java.lang.StackOverflowError
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
        at B.<init>(Test25.java:15)
        at A.<init>(Test25.java:6)
```

Explanation :

A circular reference happens when one object refers to another, and that other one refers to the first object.

On execution of the main method, an object of class A is instantiated which invokes constructor of class A which instantiates an object of class B invoking constructor of class B which in turn invokes constructor of class A . . . . . The process continues till call stack becomes full.

The following Java program is a modified version of the program given in Q which resolves a circular reference created above.

```
class A
{
  int x;
  B b;

  public A()
  {
  }
```

```
  public A(int x)
  {
        this.x=x;
  }
  public void display()
  {
        System.out.println("x : " + x);
        b=new B(20);
        b.display();
  }
}

class B
{
int y;
  A a;

  public B()
  {
         a=new A(10);
}
  public B(int y)
  {
        this.y=y;
}

public void display()
{
        System.out.println("y : " + y);
}
}


public class Test25
{
  public static void main(String args[])
  {
    A aref= new A(10);
    aref.display();
    System.out.println("main now exits...");
  }
}
```

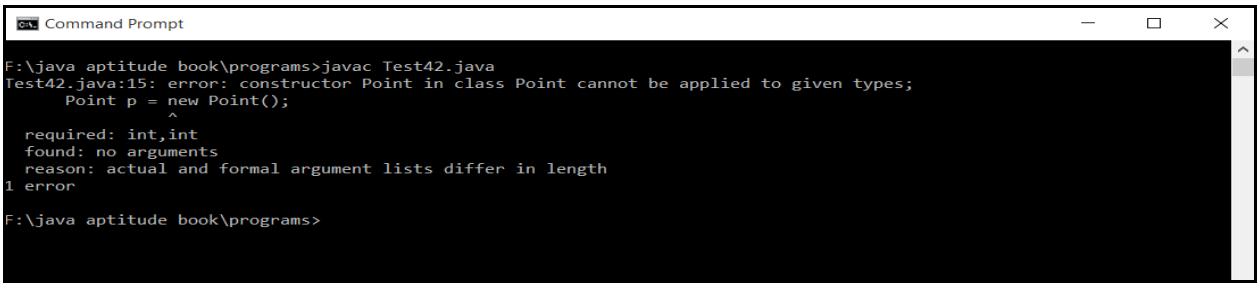
In the above example, the constructor of class A no longer instantiates an object of class B. The instantiation is deferred till the call to display method of class A is invoked which in turn invokes the display() method of class B. At the end the statement " main now exits..." is displayed.

Q. No. 80 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Point
{
   protected int x, y;
   public Point(int _x, int _y)
{
      x = _x;
      y = _y;
   }
}

public class Test42
 {
   public static void main(String args[]) {
     Point p = new Point();
     System.out.println("x = " + p.x + ", y = " + p.y);
   }
}
```

Output :



Explanation :

In the above program, there are no access permission issues because the Test and Main are in same package and protected members of a class can be accessed in other classes of same package. The problem with the code is: there is not default constructor in Point. Like C++, if we write our own parameterized constructor then Java compiler doesn't create the default constructor. So there are following two changes to Point class that can fix the above program.
1) Remove the parameterized constructor.
2) Add a constructor without any parameter.

Java doesn't support default arguments, so that is not an option.
Add the following default constructor to the class Point

```
public Point()
   {
   }
```

On re-execution of the program now, the following output is generated.

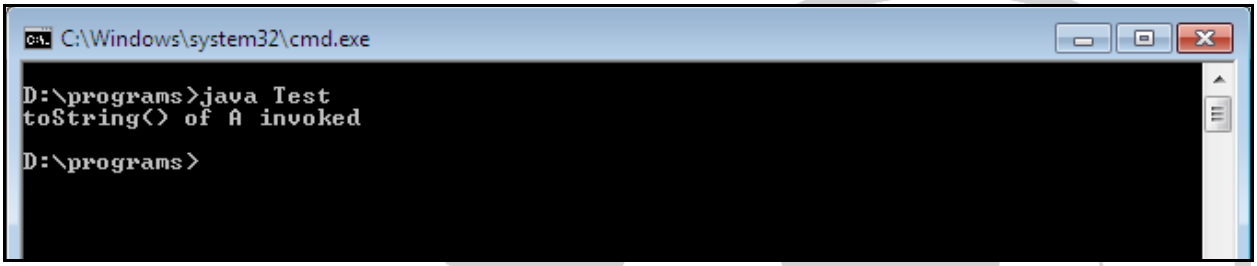
Q. No. 81 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class A
{
  public String toString()
  {
    return "toString() of A invoked";
  }
}
```

```
public class Test
{
   public static void main(String args[])
   {
     A aref=new A();
     System.out.println(aref);
   }
}
```

Output :



Explanation :

Whenever a reference is passed as a parameter to the System.out.println() method, toString() method of the corresponding class is invoked to generate a formatted output conforming to that class.

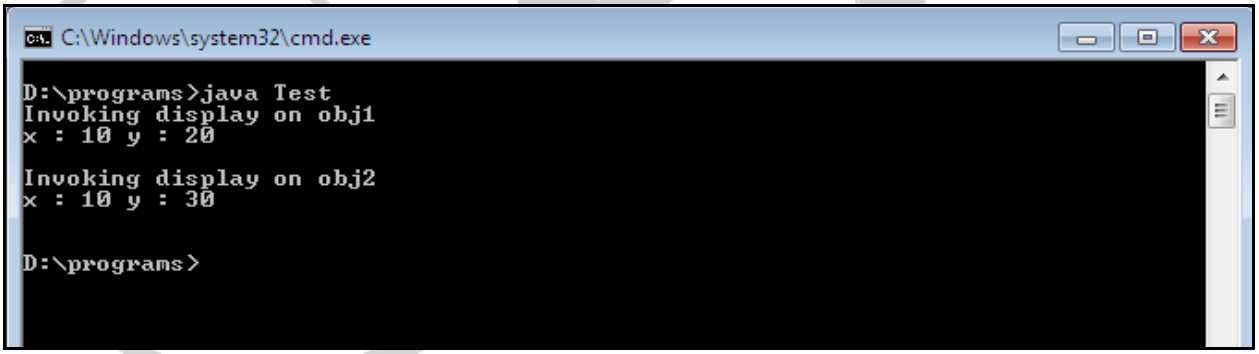
Q. No. 82 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   // Initializer block
   int x,y;
   {

      x=10;
   }

   public Test()
   {
      y=20;

   }
```

```
    public Test(int y)
    {
      this.y=y;
    }

    void display()
    {
      System.out.printf("x : %d y : %d\n\n",x,y);

    }

    public static void main(String arr[])
    {
        Test obj1, obj2;
        obj1 = new Test();
        System.out.println("Invoking display on obj1");
        obj1.display();
        obj2 = new Test(30);
        System.out.println("Invoking display on obj2");
        obj2.display();
    }
}
```

Output :



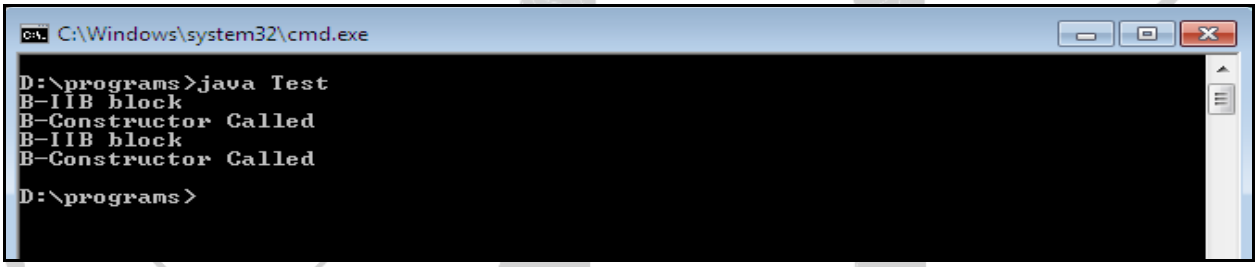
Explanation :

Initializer code is executed before every constructor on instance per basis.

Consider the following java program:

```
class B
{
   B()
   {
      System.out.println("B-Constructor Called");
   }

   {
      System.out.println("B-IIB block");
   }
}

class Test
{
   public static void main(String[] args)
   {
      B b1 = new B();
      B b2 = new B();
   }
}
```

On execution of the above program, the following output is generated.

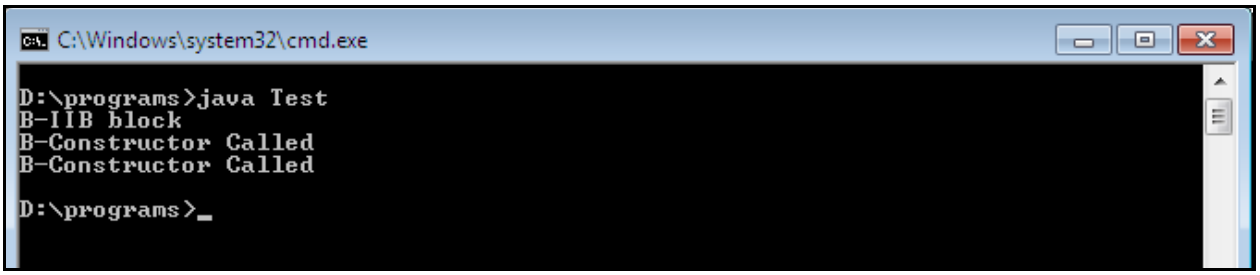


For every instance of B created, first instance initializer block is invoked followed by a default constructor.

Replace the instance initializer block in class B with the static block as shown below and re-execute the program.

```
   static
   {
      System.out.println("B-IIB block");
   }
```

On execution of the program, the following output is generated.



Now, the static block is invoked only once when the first instance is created. The purpose of the static block is to initialize the static members of the class.

Q. No. 83 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class B
{
   B()
   {
      System.out.println("B-Constructor Called");
   }

   {
      System.out.println("B-IIB block");
   }

}
// Child class
class A extends B
{
   A()
   {
      super();
      System.out.println("A-Constructor Called");
   }
   {
      System.out.println("A-IIB block");
   }
   // main function
   public static void main(String[] args)
   {
      A a = new A();
   }
}
```

Output :

Explanation :

The given program contains the following class hierarchy

The super class B contains an instance initializer block and a constructor. Similarly, the derived class A contains an instance initializer block and a default constructor. The order in which they are executed is

i. All initializer blocks of super class.
ii. Default constructor of super class
iii. All initializer blocks of derived class.
iv. Default constructor of derived class

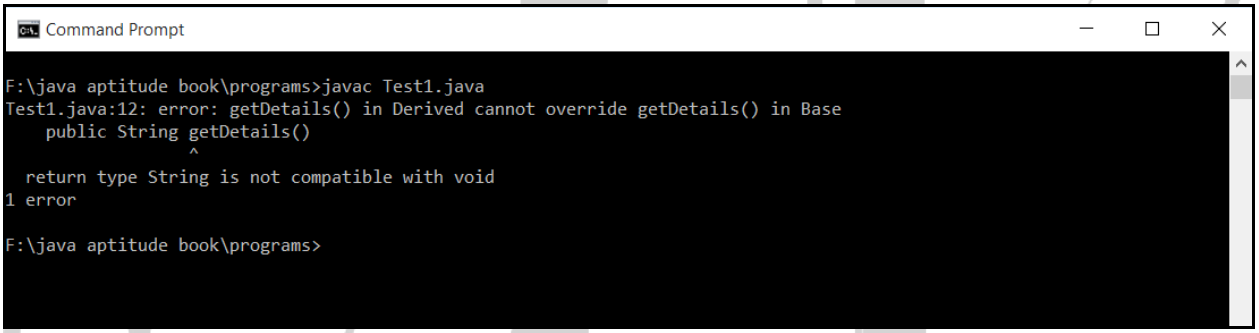
Q. No. 84 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Base
{
   public void getDetails()
   {
      System.out.println("Base class getDetails() -  ");
   }
}
```

```
class Derived extends Base
{
   public String getDetails()
   {
      return "Derived class getDetails()";
   }
}

public class Test1
{

   public static void main(String[] args)
   {
      Base obj = new Derived();
      obj.getDetails();
   }
}
```

Output :



Explanation :

getDetails() method of Base class is inherited in the Derived class. Since getDetails() method in the Derived class does not match the corresponding getDetails() method in the Base class in method signature, it is not the case of method overriding. Since the signature of the methods are same, it is not the case of method overloading. Hence the compiler generates an error message.

Consider the following version of the above program:

```
class Base
{
   public void getDetails()
   {
      System.out.println("Base class getDetails() -  ");
   }
}


class Derived extends Base
{
   public void getDetails(String temp)
   {
      System.out.println("Derived class getDetails()" + temp);
   }
}

public class Test1
{

   public static void main(String[] args)
   {
      Base obj = new Derived();
      obj.getDetails();
      ((Derived)obj).getDetails("SIBER");
   }
}
```

Now, there are two versions of getDetails() method available to the Derived class.

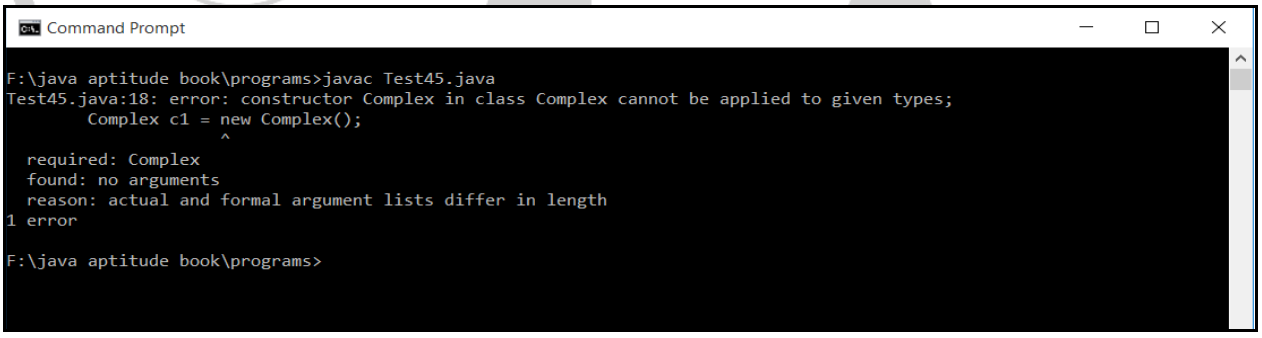
- getDetails() method inherited from Base class.
- getDetails(String) method added by Derived class.

Q. No. 85 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class Complex
{
   private double re, im;
   public String toString()
   {
      return "(" + re + " + " + im + "i)";
   }

   Complex(Complex c)
 {
      re = c.re;
      im = c.im;
   }
}

public class Test45
{
   public static void main(String[] args)
   {
      Complex c1 = new Complex();
      Complex c2 = new Complex(c1);
      System.out.println(c2);
   }
}
```

Output :



Explanation :

In Java, if we write our own copy constructor or parameterized constructor, then compiler

doesn't create the default constructor. This behavior is same as C++.

Add the copy constructor to the class as shown below and re-execute the program,

```
Complex()
{
}
```

On re-execution of the program, the following output is generated.

Q. No. 86 *Category : Interfaces*

Which two of the following are legal declarations for non nested classes and interfaces?

1. final abstract class Test {}

2. public static interface Test {}

3. final public class Test {}

4. protected abstract class Test {}

5. protected interface Test {}

6. abstract public class Test {}

A. 1 and 4

B. 2 and 5

C. 3 and 6

D. 4 and 6

Output :

Option C is correct.

Explanation :

(1) is wrong because a class cannot be abstract and final at the same time, as there would be no

way to use such a class. (2) is wrong because interfaces and classes cannot be marked as static. (4) and (5) are wrong because classes and interfaces cannot be marked as protected.

Q. No. 87 *Category : Interfaces*

Which of the following class level (nonlocal) variable declarations will not compile?

A. protected int a;

B. transient int b = 3;

C. private synchronized int e;

D. volatile int d;

Output :

Option C

Explanation :

Since the synchronized modifier applies only to methods and not to the variable declarations.

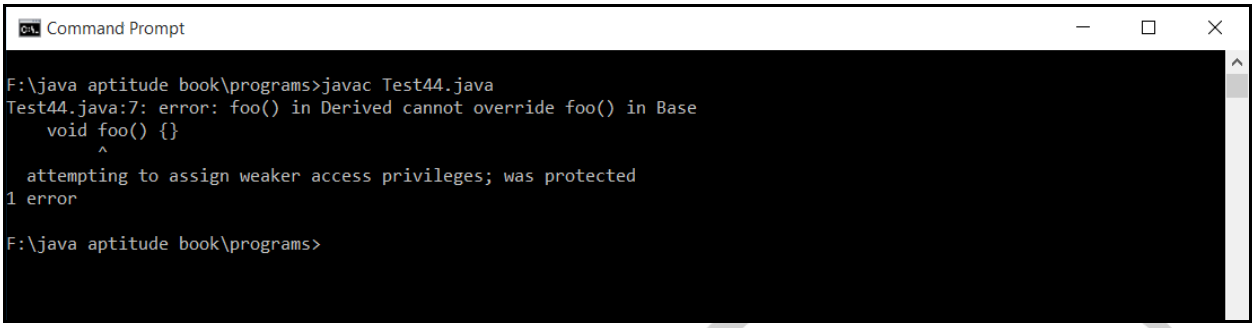
Q. No. 88 *Category : Interfaces*

What is the output generated on execution of the following Java Program?

```
class Base
{
  protected void foo()
  {
  }
}
class Derived extends Base
 {
   void foo()
  {
  }
}

public class Test44
{
   public static void main(String args[]) {
      Derived d = new Derived();
      d.foo();
   }
}
```

Output :

Compiler Error



Explanation :

foo() is protected in Base and default in Derived. Default access is more restrictive. When a derived class overrides a base class function, more restrictive access can't be given to the overridden function. If we make foo() public, then the program works fine without any error. The behavior in C++ is different. C++ allows to give more restrictive access to derived class methods.

Change the declaration of foo() method in derived class as shown below and re-execute the application.

```
public void foo()
{
}
```

On re-execution of the application, the following output is generated.

Q. No. 89 *Category : Interfaces*

What is the output generated on execution of the following Java Program?

```
public class Test46
{
   public static void print(String s)
   {
      System.out.println("String");
   }
   public static void print(Object o)
   {
      System.out.println("Object");
   }

   public static void main(String args[])
   {
      print(null);
   }
}
```

Output :

Explanation :

In case of method overloading, the most specific method is chosen at compile time. As 'String' is a more specific type than 'java.lang.Object'. In this case the method which takes 'String' as a parameter is chosen.

Modify the main() method as shown below and re-execute the application.

```
public static void main(String args[])
   {
      print(new Object());
   }
```

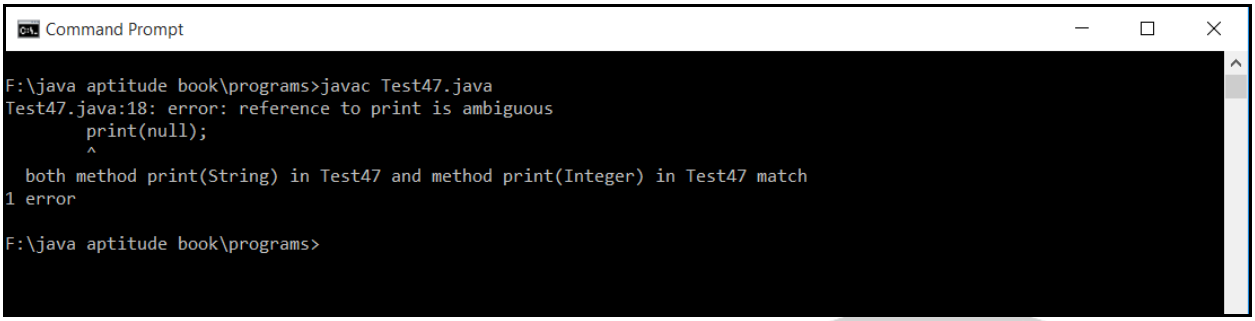
On re-execution of the application, the following output is generated.

Q. No. 90 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
public class Test47
{
   public static void print(String s)
   {
      System.out.println("String");
   }
   public static void print(Object o)
   {
      System.out.println("Object");
   }
   public static void print(Integer i)
   {
      System.out.println("Integer");
   }

   public static void main(String args[])
   {
      print(null);
   }
}
```

Output :



Explanation :

In this case of method Overloading, the most specific method is chosen at compile time. As 'String' and 'java.lang.Integer' is a more specific type than 'java.lang.Object', but between 'String' and 'java.lang.Integer' none is more specific. In this case the Java is unable to decide which method to call.

Q. No. 91 *Category : Interfaces*

Which is a valid declaration within an interface?

A. public static short stop = 23;
B. protected short stop = 23;
C. transient short stop = 23;
D. final void madness(short stop);

Output :

Option A

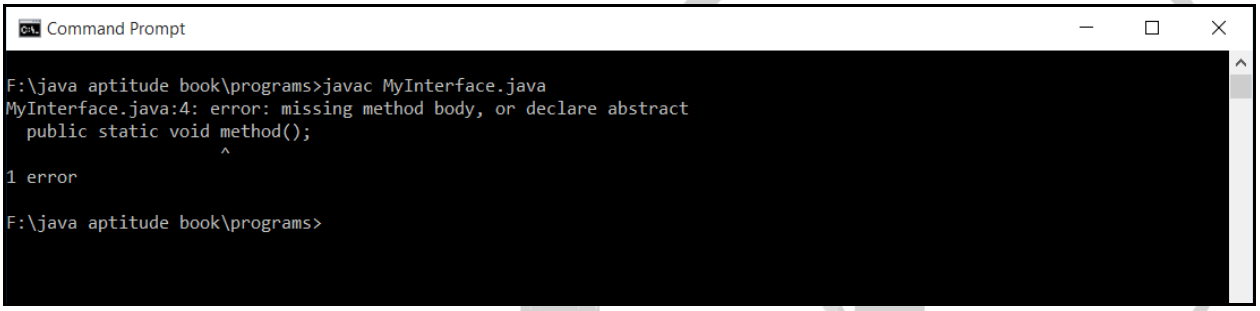
Explanation :

(B) and (C) are incorrect because interface variables cannot be either protected or transient. (D) is incorrect because interface methods cannot be final or static.
In interface declaration, data members can be declared as static but the member functions cannot be static.

Consider the following interface declaration.

```
interface MyInterface
{
  public static short stop = 23;
  public static void method();
}
```

When the above interface is compiled, the following compiler error is generated.



Q. No. 92 *Category : Object Oriented Programming*

What is the output generated on execution of the following Java Program?

```
class Employee
{
   int empId;
   String name;
   float basic;

   Employee(int empId,String name, float basic)
   {
        this.empId=empId;
        this.name=name;
        this.basic=basic;
   }

   public Object clone()throws CloneNotSupportedException
   {
        return super.clone();
   }
```

```
    public void display()
    {
          System.out.println("Employee Info..");
          System.out.println(" Emp Id : " + empId);
          System.out.println(" Name   : "+name);
          System.out.println(" Basic  : "+basic);
    }
}

public class CloneTest
{
    public static void main(String args[])
    {
      try
      {
       Employee e1=new Employee(1234,"Mohan",8000);
       Employee e2=(Employee)e1.clone();
       System.out.println("Original Employee...");
       e1.display();
       System.out.println();
       System.out.println("cloned Employee...");
       e2.display();
      }
      catch(CloneNotSupportedException e)
      {
          System.out.println(e);
      }
   }
}
```

Output :

The application will successfully compile and the following error message is displayed at runtime.

Explanation :

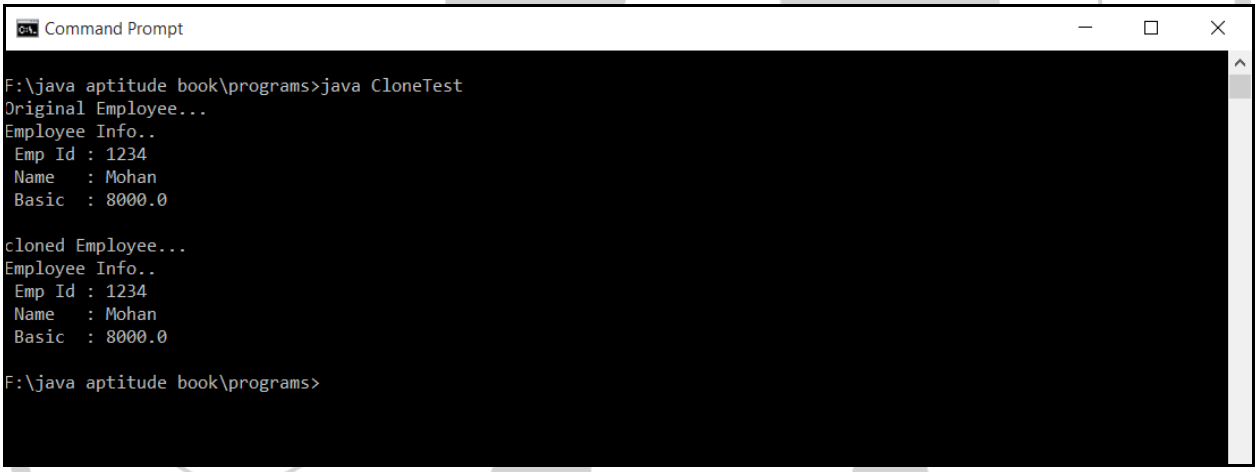
Object cloning is a mechanism to create exact copy of an object. For this purpose, java.lang.Object class supports a clone() method. The syntax of the clone() method is

protected Object clone() throws CloneNotSupportedException

The java.lang.Cloneable interface must be implemented by the class for enabling object cloning. If the Cloneable interface is not implemented, clone() method throws

CloneNotSupportedException.

Insert "implements Cloneable" clause in front of Employee class declaration and re-execute the application. The following output is generated.



Q. No. 93 *Category : Interfaces*

Consider the following interface in Java.

```
interface Base
{
    boolean m1 ();
    byte m2(short s);
}
```

which two code fragments will compile?

1. interface Base2 implements Base {}

```
2. abstract class Class2 extends Base
{
      public boolean m1()
      {
        return true;
      }
  }

3. abstract class Class2 implements Base {}
4. abstract class Class2 implements Base
{
 public boolean m1()
{
  return (7 > 4); }
}

5. abstract class Class2 implements Base
{
protected boolean m1()
{
   return (5 > 7);
}
}
```

A. 1 and 2
B. 2 and 3
C. 3 and 4
D. 1 and 5

Output :

Option C is correct.

Explanation :

Fragment	Description
1	Wrong because one interface extends another interface.
2	Wrong because class does not extend but implements an interface.
3	Correct

4	Correct
5	Interface methods are implicitly public, so the methods being implemented must be public. Attempting to execute the application with fragment 5 generated the following compiler error. Base.java:10: m1() in Class2 cannot implement m1() in Base; attempting to assign weaker access privileges; was public protected boolean m1() ^ 1 error

Q. No. 94 *Category : Access Specifiers*

What is the output generated on execution of the following Java Program?

```
class Test
{
   protected int x=10, y=20;
}

public class Test38
{
   public static void main(String args[])
   {
      Test t = new Test();
      System.out.println(t.x + " " + t.y);
   }
}
```

Output :

Explanation :

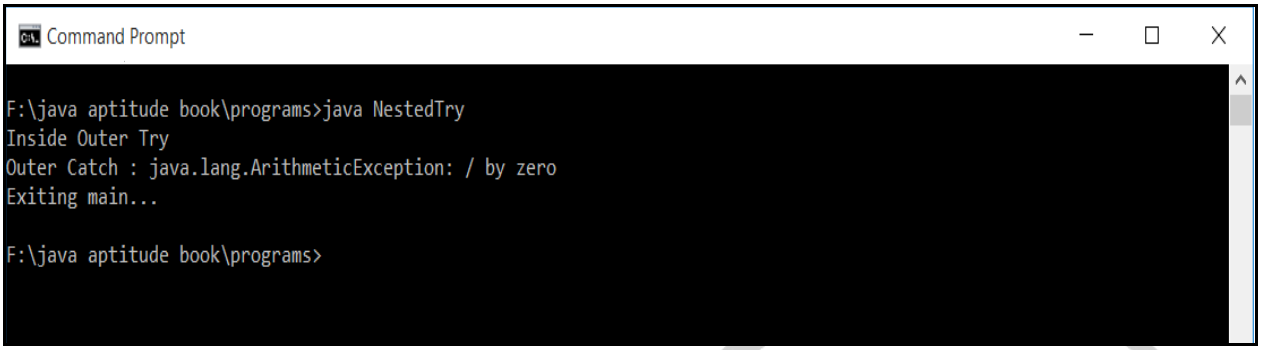
In Java, a protected member is accessible in all classes of same package and in inherited classes of other packages. Since Test and Main are in same package, no access related problem in the above program

Q. No. 95 *Category :Exception Handling*

What is the output generated on execution of the following Java Program with no command-line arguments?

```
public class NestedTry
{
  public static void main(String[] args) {
    try
    {
      System.out.println("Inside Outer Try");
      int a=args.length;
      int b=50/a;
      try
      {
        System.out.println("Inside Inner Try");
        if (a==1) a=a/(a-a);
        if (a==2)
        {
           int c[]={1};
          c[50]=100;
        }

        if(a==3)
          throw new RuntimeException();
      }
      catch(ArrayIndexOutOfBoundsException e)
      {
          System.out.println("Inner Catch : "+e);
      }
    }
    catch(ArithmeticException e)
    {
         System.out.println("Outer Catch : "+e);
    }
        System.out.println("Exiting main...");
  }
}
```

Output :



Explanation :

When the above program is executed with no command-line arguments, the ArithmeticException is raised in the outer try block which is caught in the outer catch block.

When the program is executed with a single command-line argument, the ArithmeticException is raised in the inner try block and there is no matching inner catch block to handle the exception. Stack is wound and outer catch statements are examined and since the catch statement for ArithmeticException is found, the exception is processed by the application.

When the program is executed with two command-line arguments, ArrayIndexOutOfBoundsException is raised in the inner try block and is processed in the inner catch block.

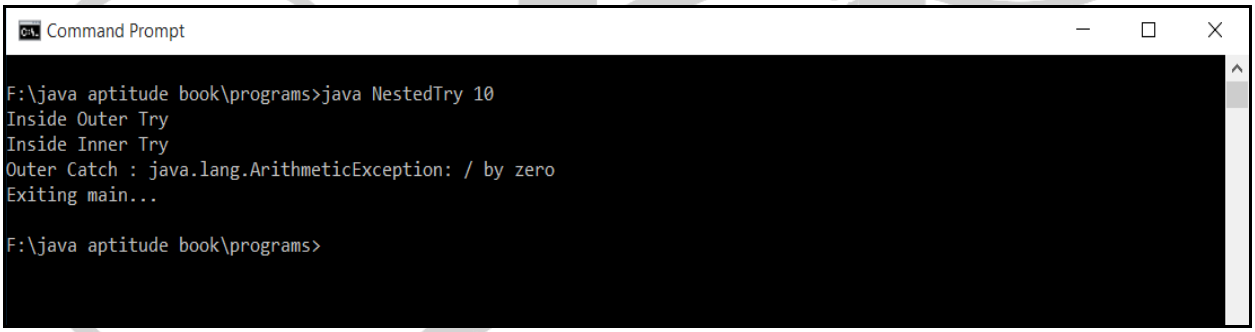
When the application is executed with three command-line arguments, RuntimeException is raised and there is no matching inner or outer catch block. Hence the exception is processed by the Java runtime system. The following table summarizes various cases.

No. of command-line Arguments	Type of Exception raised	Exception is raised in		Exception is processed in		
		Inner try block	Outer try block	Inner catch block	Outer catch block	System
0	ArithmeticException		✓		✓	
1	ArithmeticException	✓			✓	
2	ArrayIndexOutOfBoundsException	✓		✓		
3	RuntimeException	✓				✓

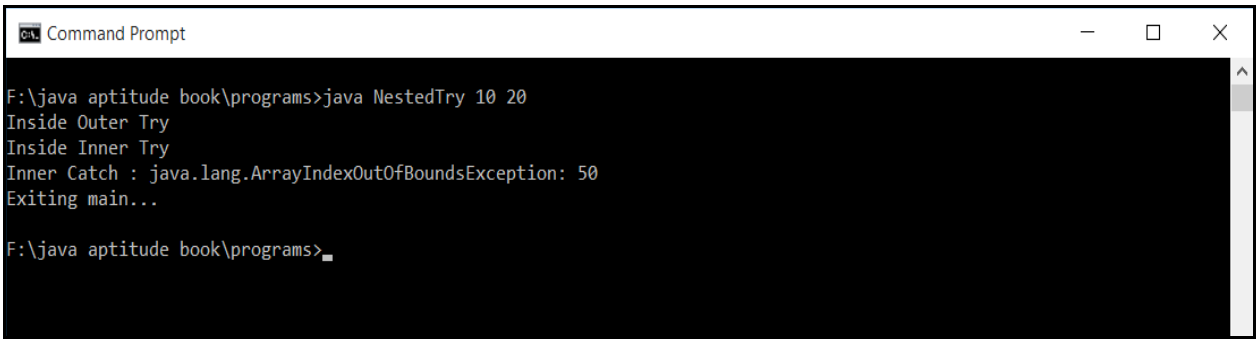
The program is executed with the following test cases:

Test Case 1: No command-line arguments.

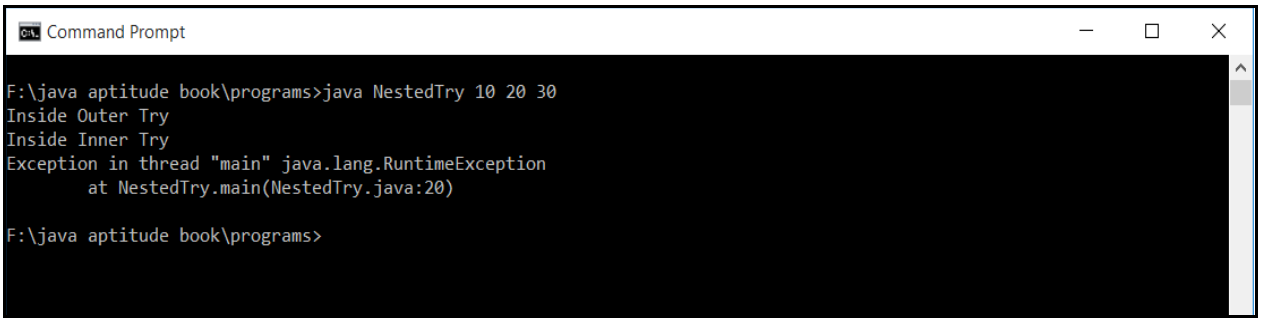
Test Case 2: Single command-line arguments.



Test Case 3: Two command-line arguments.



Test Case 4: Three command-line arguments.



Q. No. 96 *Category :Exception Handling*

What is the output generated when the following Java program is executed with one and two command-line arguments?

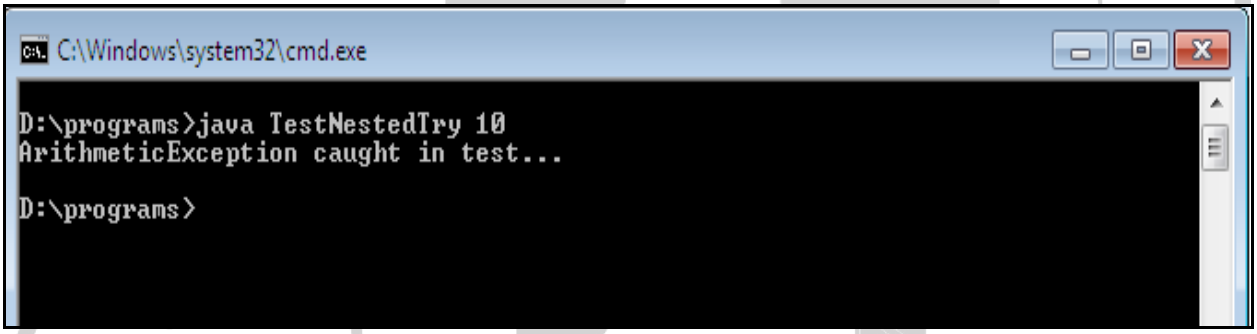
```
public class TestNestedTry
{
  static void test(int x)
  {
    try
    {
      if (x==1)
        throw new ArithmeticException();
      else
        throw new ArrayIndexOutOfBoundsException();
    }
    catch(ArithmeticException e)
    {
       System.out.println("ArithmeticException caught in test...");
    }
  }

  public static void main(String[] args) {
     try
     {
       int a=args.length;
       test(a);
     }
     catch(ArrayIndexOutOfBoundsException e)
     {
        System.out.println("ArrayIndexOutOfBoundsException caught in main...");
     }
  }
}
```

Output :

Explanation :

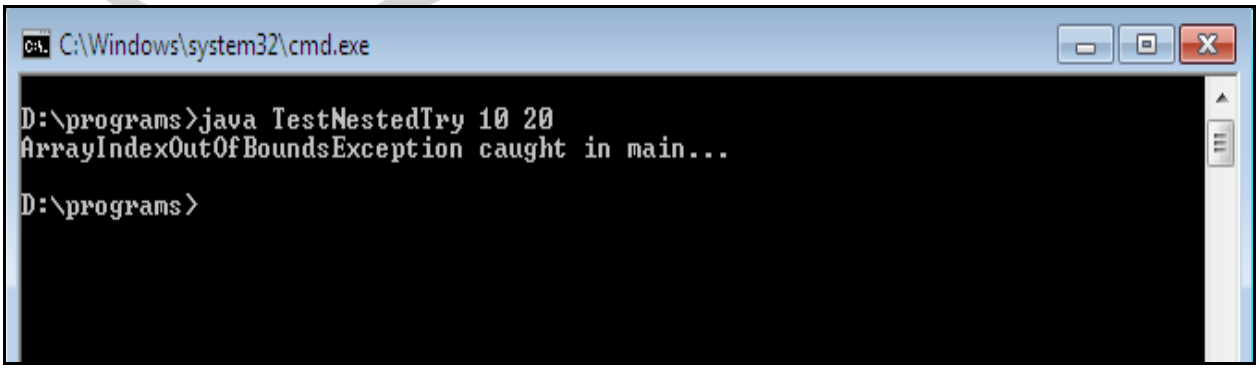
The above example illustrates an implicit method of nesting of try statements. The try..catch blocks provided by test() method are implicitly nested inside try block of main() method. Hence if the exception is raised inside try block of test() method and if there is no matching catch block inside test() method, the stack is would and catch blocks of main() method are examined for matching exception type.

When the program is executed with a single command-line argument, the following output is generated.



When the program is executed with a single command-line argument, main() method invokes the method test() with argument 1, which throws ArithmeticException and is caught by catch block inside test() method itself.

When the program is executed with two command-line arguments, the following output is generated.



When the program is executed with a two command-line arguments, ArrayIndexBoundsException is thrown inside try block of test() method and there is no matching catch block for catching the generated exception. However, the invoking main() method contains the required catch block for catching the exception. Hence the exception is caught in the outer catch block provided by main() method.

Q. No. 97 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
class Test53
{
   public static void main(String args[])
   {
      try
      {
         System.out.println("First statement of try block");
         int num=45/3;
         System.out.println(num);
      }
      catch(Exception e)
      {
         System.out.println("Gfg caught Exception");
      }
      finally
      {
         System.out.println("finally block");
      }
      System.out.println("Main method");
   }
}
```

Output :

Explanation :

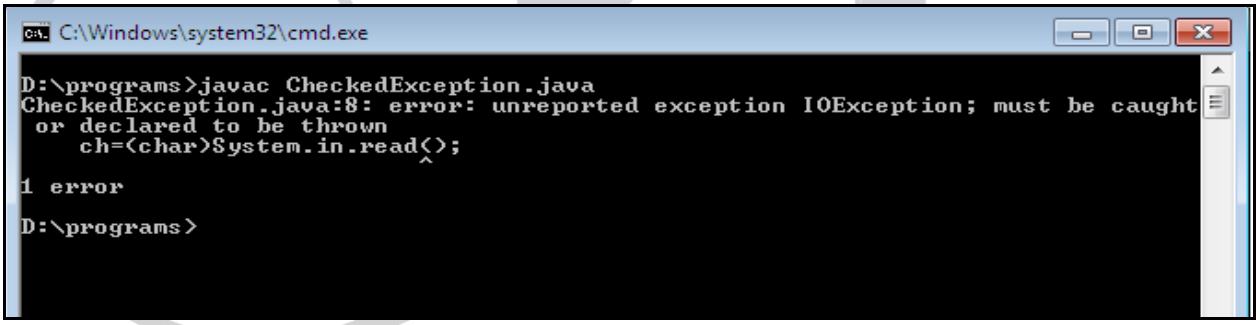
Since there is no exception, the catch block is not called, but the *finally* block is always executed after a try block whether the exception is handled or not.

Q. No. 98 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
import java.io.*;

public class CheckedException
{
public static void main(String[] args)
{
   char ch='A';
   System.out.print("Enter any character : ");
   ch=(char)System.in.read(); ←Statement 8
    System.out.println("You Entered : "+ch);
 }
}
```

Output :



Explanation :

There are two ways of fixing the error.

Method 1 :

Include the statement in a try block and add appropriate catch block for processing checked exception.

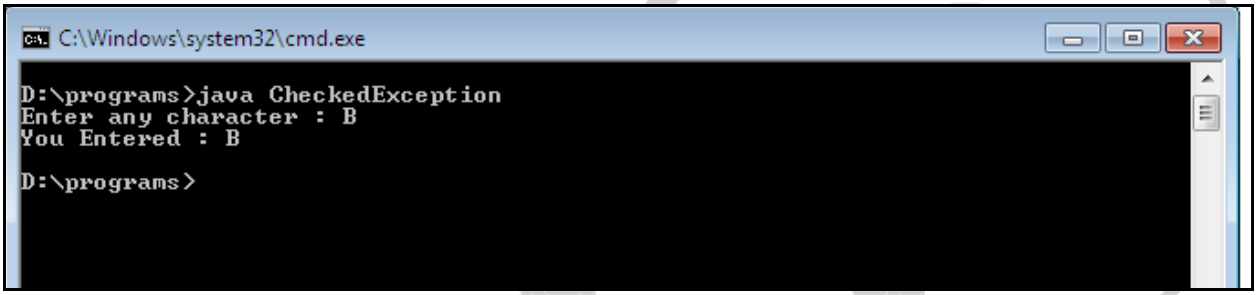
Method 2 : Add a throws clause in a method declaration.

Both the methods are illustrated here.

Replace statement 8 in the above program with the block given below and re-execute the program.

```
try
  {
        ch=(char)System.in.read();
  }
  catch(IOException e)
 {
        System.out.println(e);
  }
```

On execution the following output is generated.



The second alternative is replace the main method with the one given below and re-execute the program.

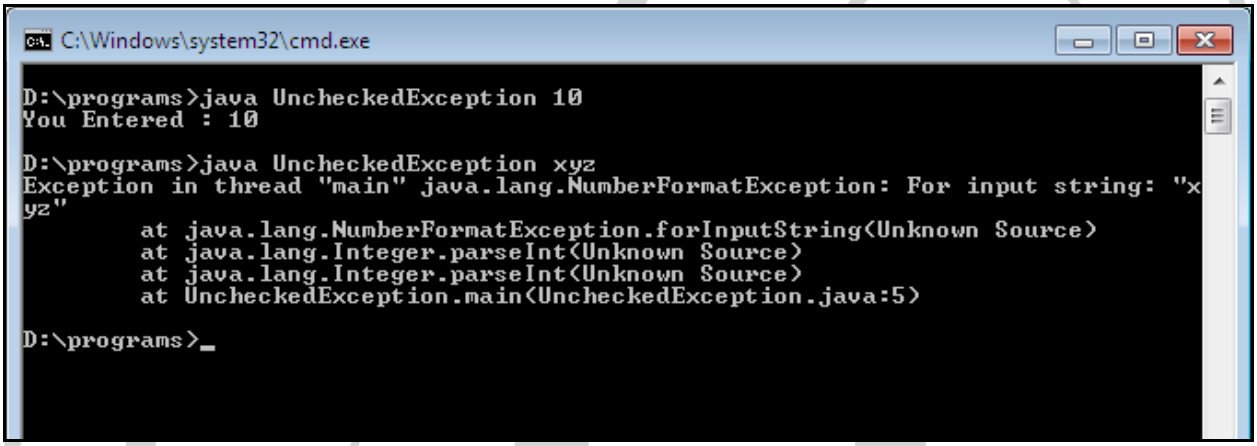
```
public static void main(String[] args) throws IOException
 {
   char ch;
   System.out.print("Enter any character : ");
   ch=(char)System.in.read();
   System.out.println("You Entered : "+ch);
 }
```

On execution of the above program the same output as above is generated.

Consider the following program.

```
import java.io.*;
public class UnCheckedException
{
  public static void main(String[] args)
{
    int num=Integer.parseInt(args[0]);
    System.out.println("You Entered : "+num);
  }
}
```

parseInt() method of Integer class throws NumberFormatException which is an unchecked exception. Hence the above program will compile successfully and if executed with non-numeric argument will throw NumberFormatException which is caught by Java runtime system as shown below: The program is executed with valid and invalid command-line arguments.



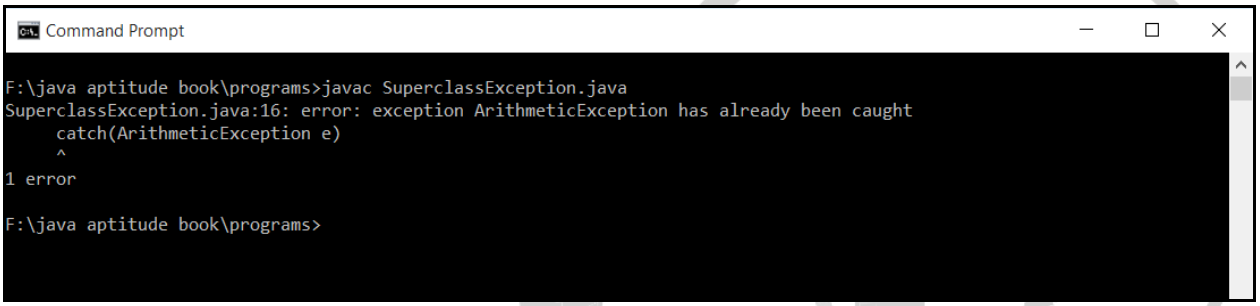
Q. No. 99 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class SuperclassException
{
  public static void main(String args[])
{
    int x=10;
    int y=0;
    int z;
    try
    {
      z=x/y;
    }
```

```
    catch(Exception e)
    {
      System.out.println(e);
    }
    catch(ArithmeticException e)
    {
      System.out.println(e);
    }
  }
}
```

Output :



Explanation :

Exception class is the super class of ArithmeticException. Hence ArithmeticException is-a Exception. Hence all exceptions are caught in the catch block processing Exception. Thus, in the above program, second catch block is unreachable and in Java unreachable code generates an error. As a result, we arrive at the following rule.

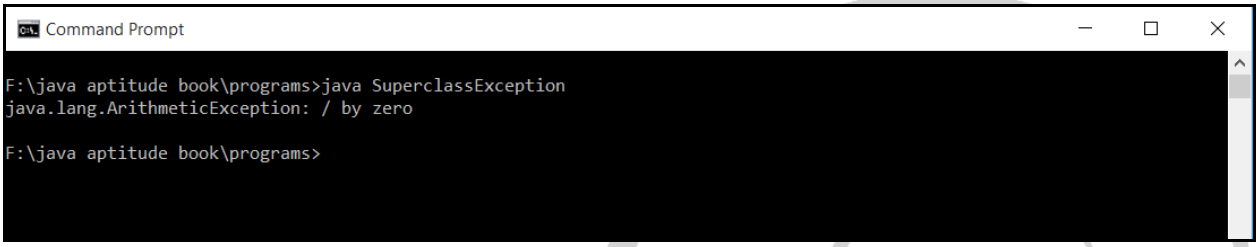
Rule : All catch blocks for subclass exceptions must come before catch blocks for super class exception.

In order to fix the error, interchange the catch blocks as shown below.

```
    catch(ArithmeticException e)
    {
      System.out.println(e);
    }
```

```
catch(Exception e)
{
  System.out.println(e);
}
```

On executing the program the following output is generated.

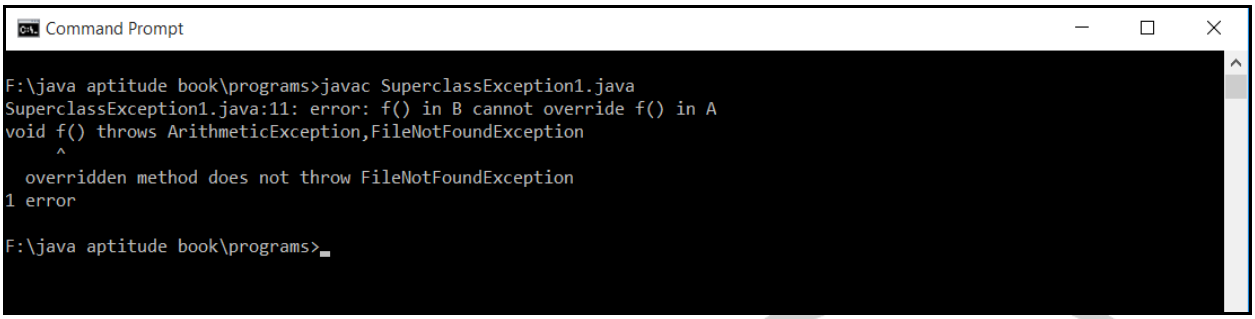


Q. No. 100 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
import java.io.*;

class A
{
        void f() throws ArithmeticException
        {
        }
}
class B extends A
{
        void f() throws ArithmeticException, FileNotFoundException ← Statement 9
        {
          System.out.println("Inside f of B..");
        }
}
public class SuperclassException1
{
        public static void main(String args[]) throws FileNotFoundException
        {
          B obj=new B();
          obj.f();
        }
}
```

Output :



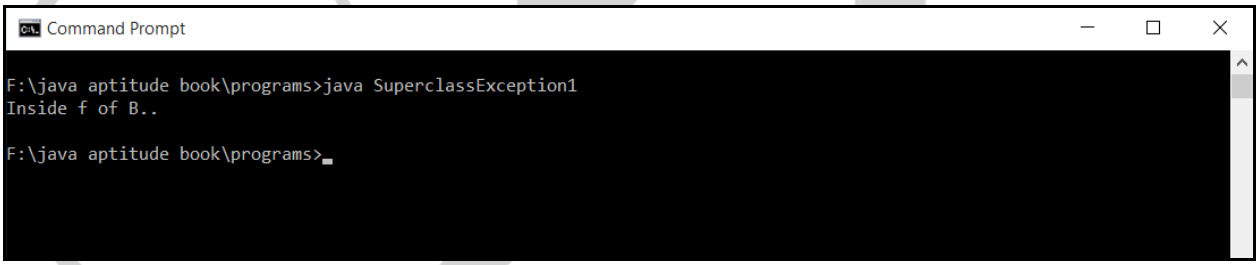
Explanation :

The reason for this error is that the overridden method cannot throw a checked exception that is not thrown by the corresponding base class method. However, the overridden method is allowed to throw any number of unchecked exceptions that are not thrown by the corresponding base class method.

Hence, if the statement 9 in the above program is replaced with the following statement

void f() throws ArithmeticException, NumberFormatException

and the program is re-executed, the following output is generated.



Hence, we arrive at the following rules :

Rule 1. The overridden method in the subclass cannot throw a checked exception that is not thrown by the corresponding base class method.

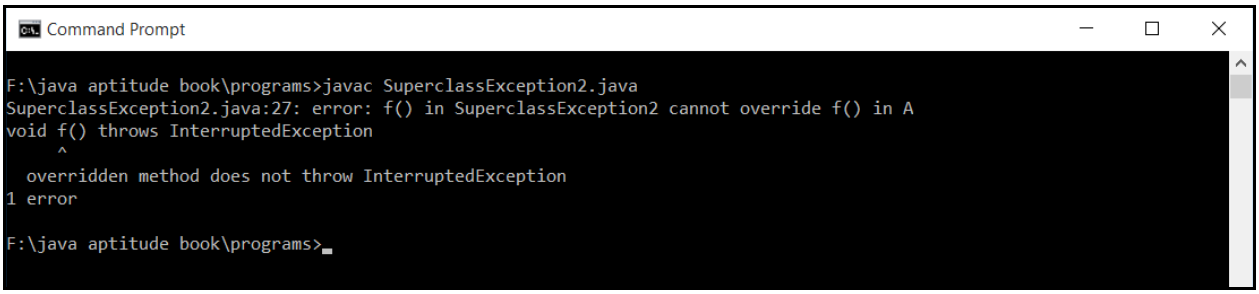
Rule 2. The overridden method in the sub class is allowed to throw any number of unchecked exceptions that are not thrown by the corresponding base class method.

Q. No. 101 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
import java.io.*;

class A
{
        void f() throws ArithmeticException
        {

        }
}

public class SuperclassException2 extends A
{
        public static void main(String args[]) throws FileNotFoundException
        {
                A obj=new SuperclassException2();

                try
                {
                obj.f();
                }
                catch(ArithmeticException e)
                {
                return;
                }
                catch(Exception e)
                {
                System.out.println(e);
                throw new RuntimeException("Something wrong here..");
                }
                }
                void f() throws InterruptedException ← Statement 27
                {
                System.out.println("Inside f...");
                }
}
```

Output :

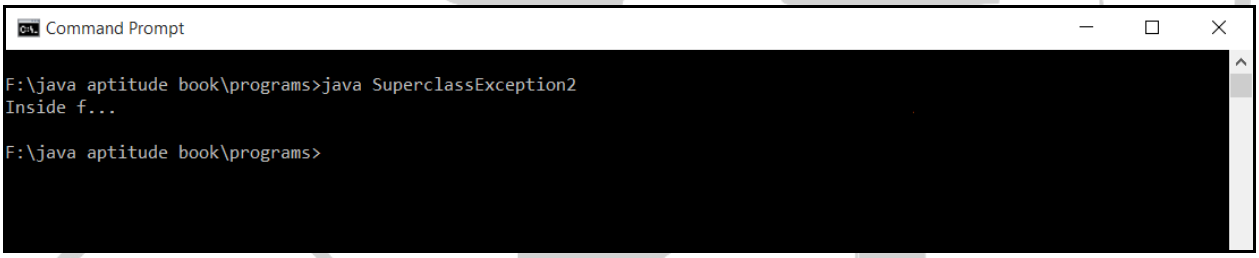


Explanation :

Replace statement 27 in the above program, with the statement

void f()

and re-execute the program. The program successfully compiles and generates the following output.



Hence, we arrive at the following rule.

Rule : The overridden method in the subclass may or may not throw the same checked exceptions as thrown by the corresponding base class method.

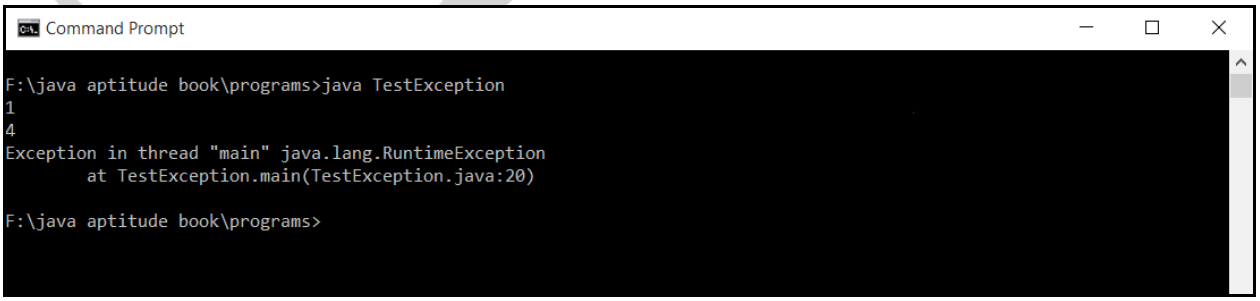
Q. No. 102 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
import java.io.*;

public class TestException
{

  static void f() throws InterruptedException
  {
    throw new InterruptedException();
  }
```

```
public static void main(String[] args)
{
  try
  {
    f();
  }
  catch (InterruptedException e)
  {
   System.out.println("1");
   throw new RuntimeException();
  }

  catch (RuntimeException e)
  {
   System.out.println("2");
   return;
  }

  catch(Exception e)
  {
   System.out.println("3");
  }
  finally
  {
    System.out.println("4");
  }
  System.out.println("5");
 }
}
```

Output :



Explanation :

main() invokes a method f(), which throws InterruptedException which is caught in the

following catch block, which prints "1" and in turn throws a RuntimeException. Only the exceptions thrown in the try block are caught in the matching catch blocks. Hence the RuntimeException is not caught in the program but is handled by the Java runtime system, but before that finally block is executed which prints "4". This is followed by a system exception.

Q. No. 103 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
import java.io.IOException;

import java.util.EmptyStackException;

public class Test58
{
   public static void main(String[] args)
   {
      try
      {
         System.out.printf("%d\n", 1);
         throw(new Exception());
      }
      catch(IOException e)
      {
         System.out.printf("%d\n", 2);
      }
      catch(EmptyStackException e)
      {
         System.out.printf("%d\n", 3);
      }
      catch(Exception e)
      {
         System.out.printf("%d\n", 4);
      }
      finally
      {
         System.out.printf("%d\n", 5);
      }
   }
}
```

Output :

Explanation :

The catch statements are written in the order: more specific to more general. In the code above, a new exception of type Exception is thrown. First the code jumps to first catch block to look for exception handler. But since the IOException is not of the same type it is moves down to second catch block and finally to the third, where the exception is caught and 4 is printed. Therefore, the answer is 145, as the order of execution in terms of blocks is: try->catch->finally

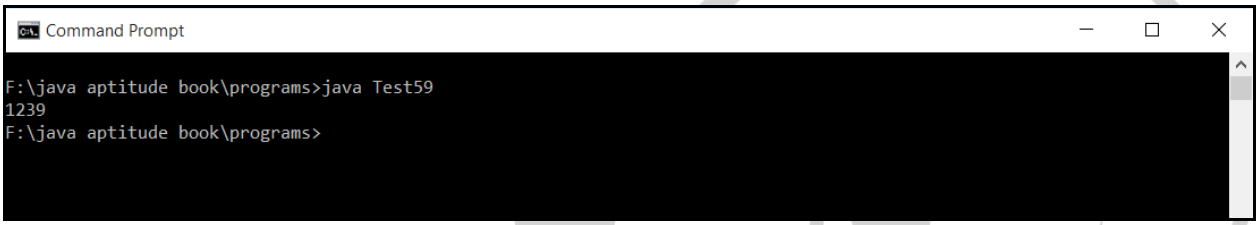
Q. No. 104 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test59
{
   static
   {
      System.out.printf("%d", 1);
   }
   static
   {
      System.out.printf("%d", 2);
   }
   static
   {
      System.out.printf("%d", 3);
   }
   private static int myMethod()
   {
      return 4;
   }
```

```
    private int function()
    {
        return 5;
    }

    public static void main(String[] args)
    {
        System.out.printf("%d", (new Test59()).function() + myMethod());
    }
}
```

Output :



Explanation :

static blocks in Java are executed even before the call to main is made by the compiler. In the main method, a new object of java class is made and its function() method is called which return 4. Number 5 is returned by a call to static function myMethod(). 4+5 = 9. Therefore, the output of the program is 1239, because 123 is printed on the console even before main method executes and main method on execution returns 9.

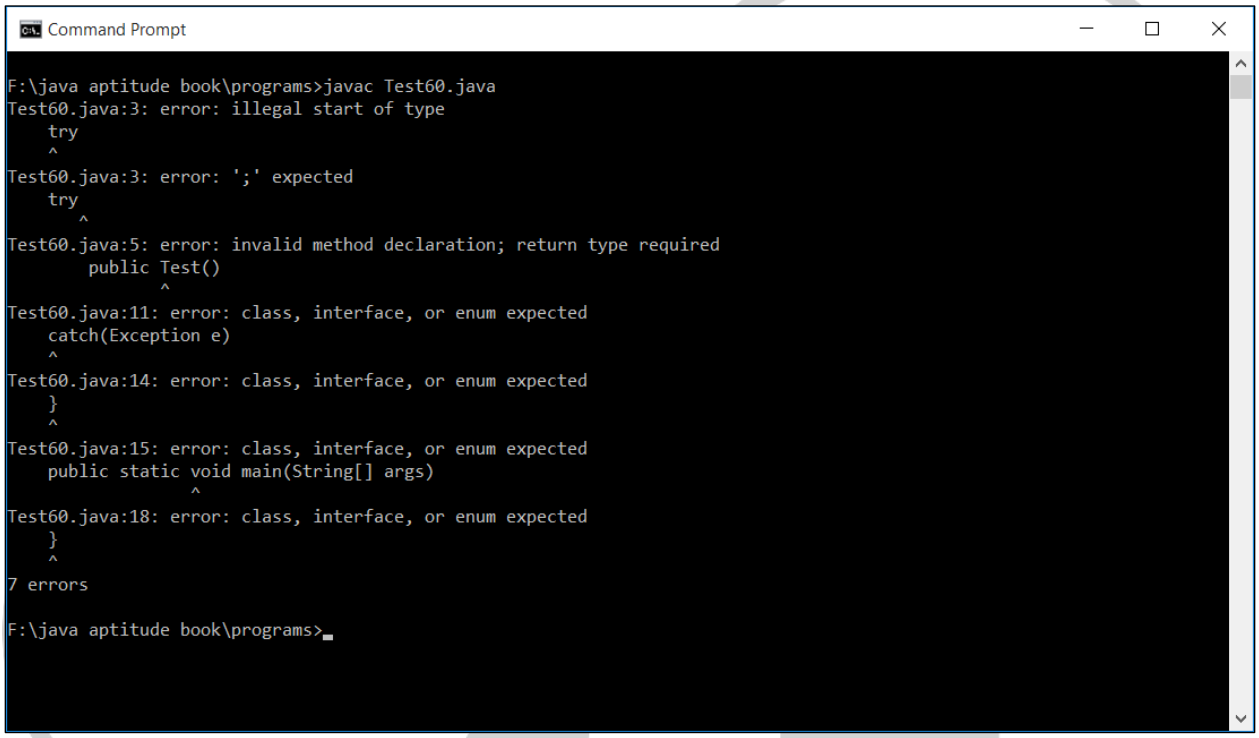
Q. No. 105 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test60
{
    try
    {
        public Test()
        {
            System.out.println("SIBER");
            throw new Exception();
        }
    }
```

```
    catch(Exception e)
    {
       System.out.println("GFG");
    }
    public static void main(String[] args)
    {
       Test60 test = new Test60();
    }
}
```

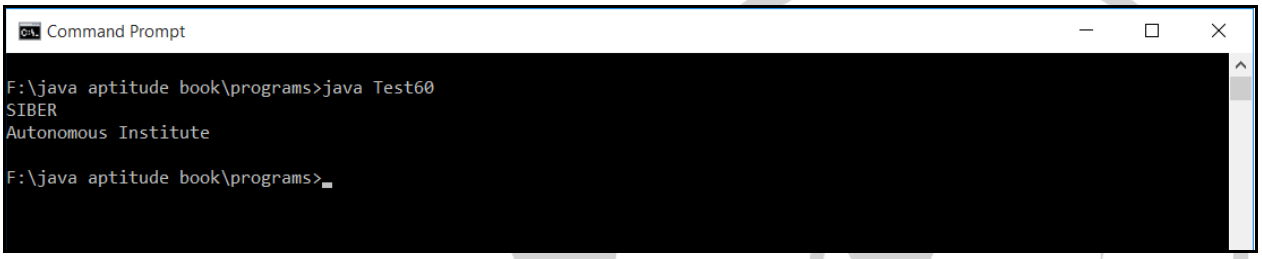
Output :



Explanation :

```
public class Test60
{

      public Test60()
      {
         System.out.println("SIBER");
          try
            {
                  throw new Exception();
           }
```

```
            catch(Exception e)
            {
                    System.out.println("Autonomous Institute");
            }
}

   public static void main(String[] args)
   {
      Test60 test = new Test60();
   }
}
```

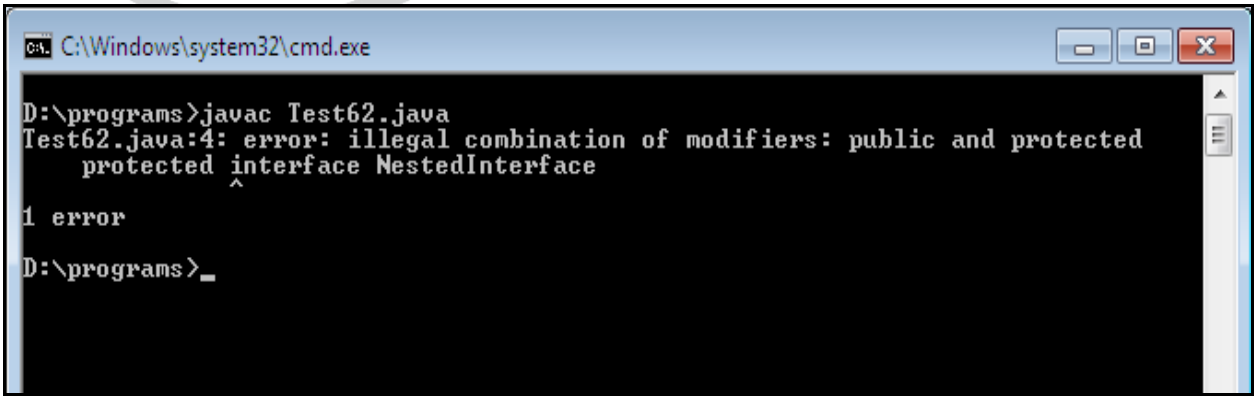


Q. No. 106 *Category :Interfaces*

What is the output generated on execution of the following Java Program?

```
public interface Test62
{
   public int calculate();
   protected interface NestedInterface
   {
      public void nested();
   }
}
```

Output :



Explanation :

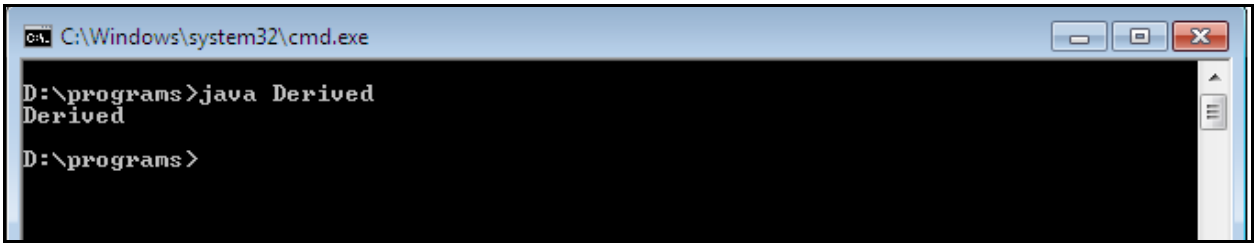
Access modifier of NestedInterface can only be public. Therefore the error:

Q. No. 107 *Category :Interfaces*

What is the output generated on execution of the following Java Program?

```
import java.io.*;

class Test62
{
   public void display() throws Exception
   {
      System.out.println("Test");
   }

}

public class Derived extends Test62
{
   public void display() throws Exception
   {
      System.out.println("Derived");
   }

   public static void main(String[] args) throws Exception
   {
      Derived object = new Derived();
      object.display();
   }
}
```

Output :



Explanation :

Overridden methods in the Derived class can declare all or few of the exceptions as that declared in the method of the Base class.

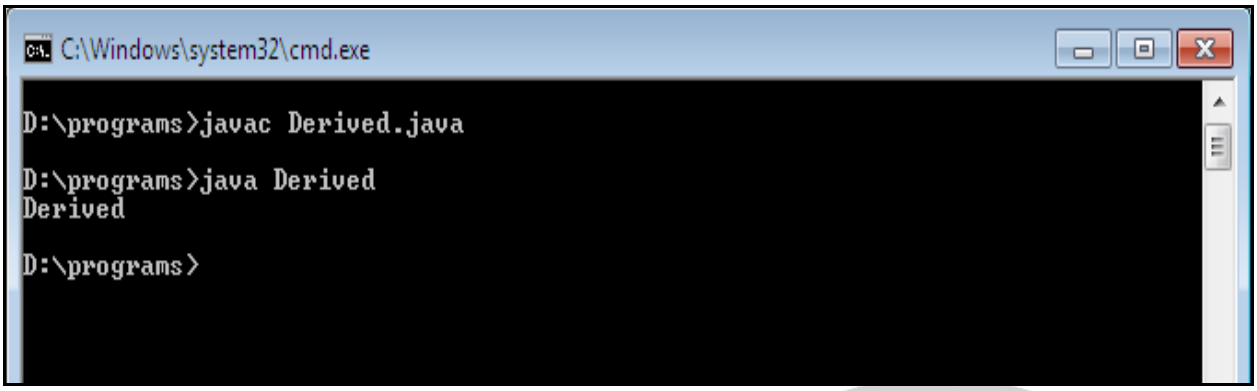
Execute the following variations of the above program.

Case 1 : Derived class overridden method throws exactly the same exception as is thrown by the corresponding base class method

<The case is considered above>

Case 2 : Derived class overridden method throws few of the exceptions as is thrown by the corresponding base class method

```
import java.io.*;

class Test62
{
   public void display() throws IOException,NullPointerException
   {
      System.out.println("Test");
   }

}

public class Derived extends Test62
{
   public void display() throws IOException
   {
      System.out.println("Derived");
   }

   public static void main(String[] args) throws Exception
   {
      Derived object = new Derived();
      object.display();
   }
}
```



Case 3 : Derived class overridden method throws the exception not thrown by the corresponding base class method

```
import java.io.*;

class Test62
{
   public void display() throws IOException,NullPointerException
   {
      System.out.println("Test");
   }

}

public class Derived extends Test62
{
   public void display() throws IOException,NumberFormatException
   {
      System.out.println("Derived");
   }

   public static void main(String[] args) throws Exception
   {
      Derived object = new Derived();
      object.display();
   }
}
```

Case 4 : Derived class overridden method does not throw the exception as is thrown by the corresponding base class method

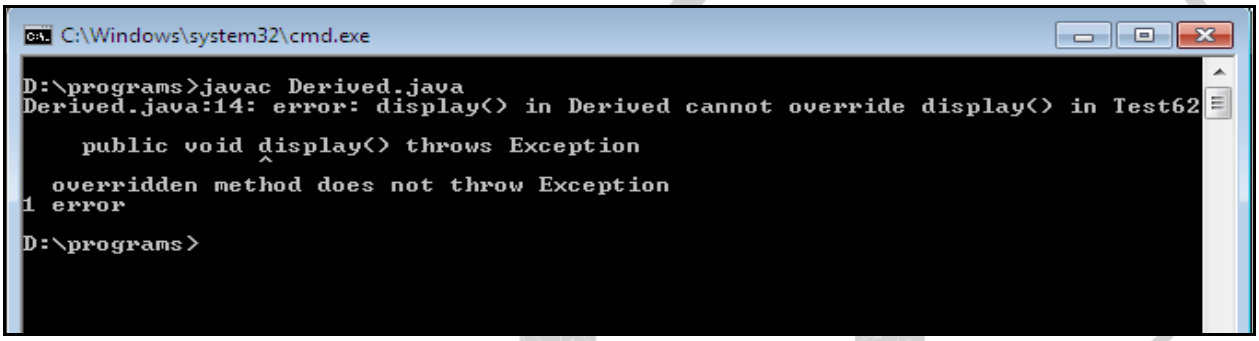
Case 5 : Derived class overridden method throws the sub class exception as is thrown by the corresponding base class method

```
import java.io.*;

class Test62
{
   public void display() throws Exception
   {
      System.out.println("Test");
   }

}


public class Derived extends Test62
{
   public void display() throws NullPointerException
   {
      System.out.println("Derived");
   }

   public static void main(String[] args) throws Exception
   {
      Derived object = new Derived();
      object.display();
   }
}
```

Case 6 : Derived class overridden method throws the super class exception as is thrown by the corresponding base class method

```
import java.io.*;

class Test62
{
   public void display() throws NullPointerException
   {
      System.out.println("Test");
   }

}
```

```
public class Derived extends Test62
{
   public void display() throws Exception
   {
      System.out.println("Derived");
   }

   public static void main(String[] args) throws Exception
   {
      Derived object = new Derived();
      object.display();
   }
}
```



The exception thrown by the overriding method should not be new or more broader checked exception. In the code above, Exception is more broader class of checked exception than IOException, so this results in compilation error

Q. No. 108 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
import java.io.*;
public class ThrowDemo
{
  static void demoproc()
  {
    try
    {
      throw new NullPointerException();
    }
```

```
  catch(NullPointerException e)
  {
    System.out.println("NullPointerException processed in demoproc..");
   throw e;
  }
 }

 public static void main(String[] args) {
  try
  {
   demoproc();
  }
  catch(NullPointerException e)
  {
     System.out.println("NullPointerException processed again in main..");
  }
 }
}
```

Output :

Explanation :

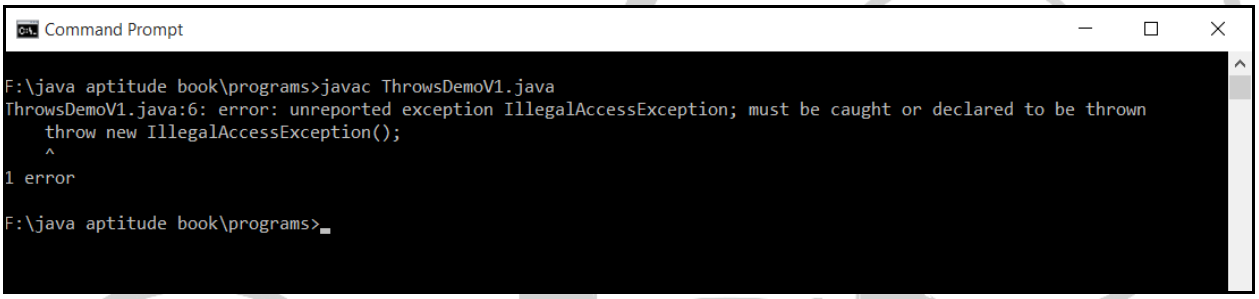
In the above Java program, the main method invokes demoproc() method, which throws a NullPointerException which is partially processed in the following catch block and is re-thrown back to the main method, which is further processed in the catch block of main() method. Hence the same exception is processed at two places, once on demoproc() method and then in the main() method.

Q. No. 108 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class ThrowsDemoV1
{
  public static void throwOne()
  {
    System.out.println("Inside throwOne..");
    throw new IllegalAccessException();
  }

  public static void main(String args[])
  {
    throwOne();
  }
}
```

Output :



Explanation :

The error can be fixed in one of the following ways.

Method 1: Using try..catch block in throwOne() method as shown below:

```
public class ThrowsDemoV2
{
  public static void throwOne()
  {
    try
    {
    System.out.println("Inside throwOne..");
    throw new IllegalAccessException();
    }
```

```
  catch(IllegalAccessException e)
  {
   System.out.println(e);
  }
 }
 public static void main(String args[])
 {
  throwOne();
 }
}
```

On execution of the above Java program the following output is generated.

Method 2 : Using throws clause in throwOne() method declaration indicating the caller of the method about the exception and using try..catch block in the main method, caller of throwOne() method.

```
public class ThrowsDemoV3
{
 public static void throwOne() throws IllegalAccessException
 {
  System.out.println("Inside throwOne..");
  throw new IllegalAccessException();
 }
```

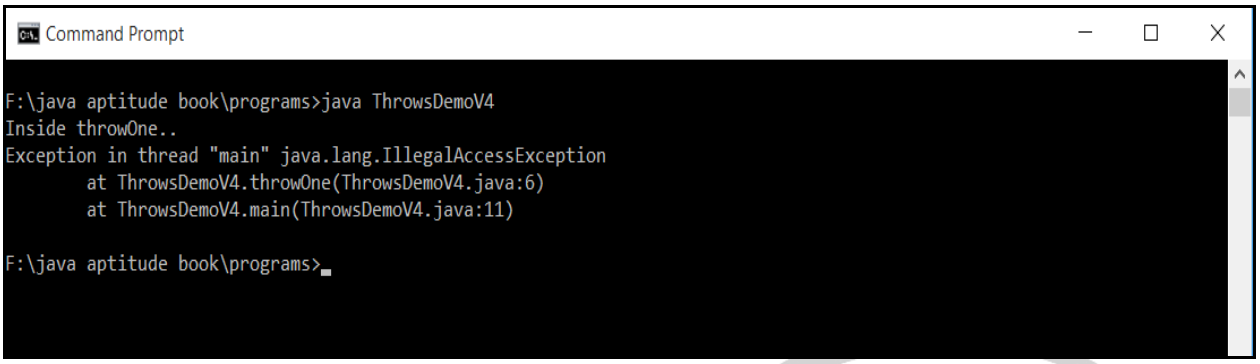
```
  public static void main(String args[])
  {
    try
    {
     throwOne();
    }
    catch(IllegalAccessException e)
    {
      System.out.println(e);
    }
  }
}
```

On execution of the above Java program the same output is generated.

Method 3 : Using throws clause in both throwOne() and main() method declarations as shown in the following program.

```
public class ThrowsDemoV4
{
  public static void throwOne() throws IllegalAccessException
  {
    System.out.println("Inside throwOne..");
    throw new IllegalAccessException();
  }
  public static void main(String args[]) throws IllegalAccessException
  {
     throwOne();
  }
}
```

On execution of the above Java program the following output is generated.

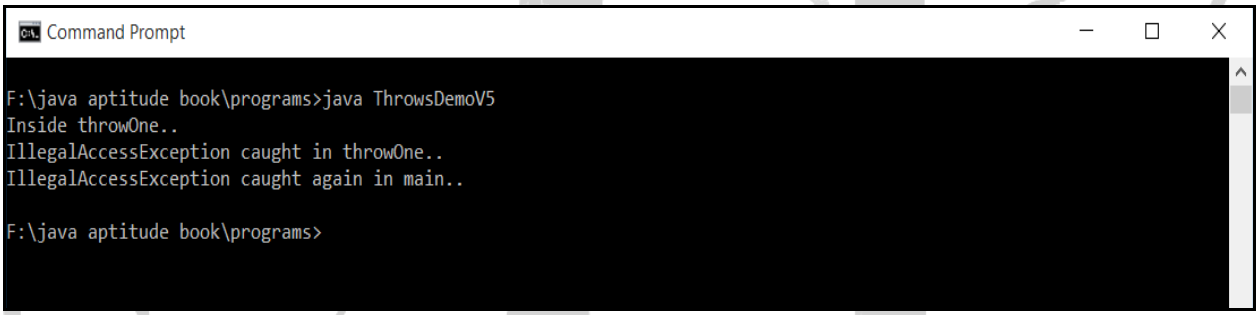


Method 4 : Processing the exception in both throwOne() and main() methods using try..catch blocks as shown in the following program.

```
public class ThrowsDemoV5
{
 public static void throwOne() throws IllegalAccessException
 {
  try
   {
        System.out.println("Inside throwOne..");
        throw new IllegalAccessException();
 }
  catch(IllegalAccessException e)
   {
        System.out.println("IllegalAccessException caught in throwOne..");
         throw e;
   }
 }
```

```
 public static void main(String args[]) throws IllegalAccessException
 {
  try
  {
   throwOne();
  }
  catch(IllegalAccessException e)
  {
        System.out.println("IllegalAccessException caught again in main..");
  }
 }
}
```

On execution of the above Java program the following output is generated.



Q. No. 109 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test27
{
public static void main(String args[])
{
        int k=0;
        try
        {
        int i=5/k; ←Statement 8
        }
```

```
        catch(ArithmeticException e)
        {
           System.out.println("1");
        }
        catch(RuntimeException e)
        {
           System.out.println("2");
        }
        catch(Exception e)
        {
           System.out.println("3");
        }
        finally
        {
           System.out.println("4");
        }
        System.out.println("5"); ←Statement 26
}
}
```

Output :

Explanation :

On execution of statement 8 ArithmeticException is thrown which is caught in the following catch block, which prints "1" and the application's normal execution continues. Next, the finally block is executed, which prints "4" , followed by statement 26 printing "5" before the termination of program.

Q. No. 110 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test28
{
        public static void main(String args[])
        {
                try
                {
                        if (args.length==0) return; ←Statement 7
                        System.out.println(args[0]); ←Statement 8
                }
                finally
                {
                        System.out.println("The End");
                }
        }
}
```

Output :

The output generated when the application is executed with and without command-line argument is shown below:

Explanation :

When the application is executed without any command-line arguments, statement 7 is executed and before the function returns, finally block is executed which prints “The End”.

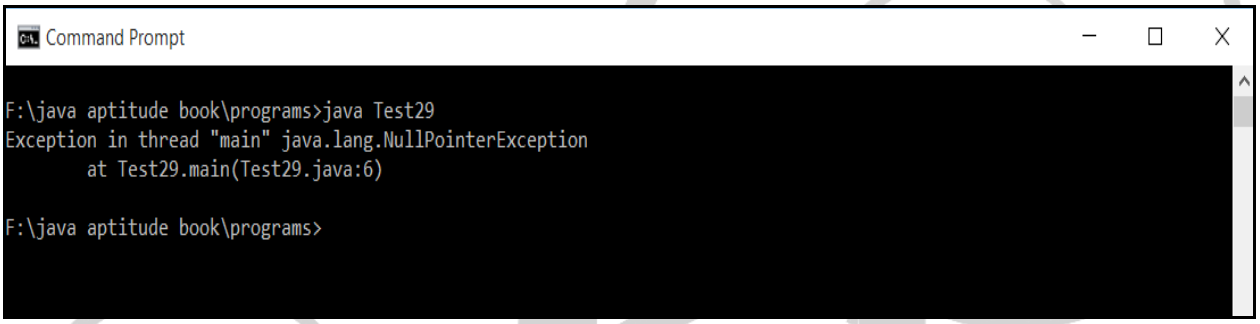
When the application is executed with a single command-line argument, statement 8 is executed, which prints the command-line argument and then the finally block is executed which prints “The End”.

Q. No. 111 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test29
{
public static void main(String args[])
{
   RuntimeException re=null; ← Statement 5
   throw re;
}
}
```

Output :

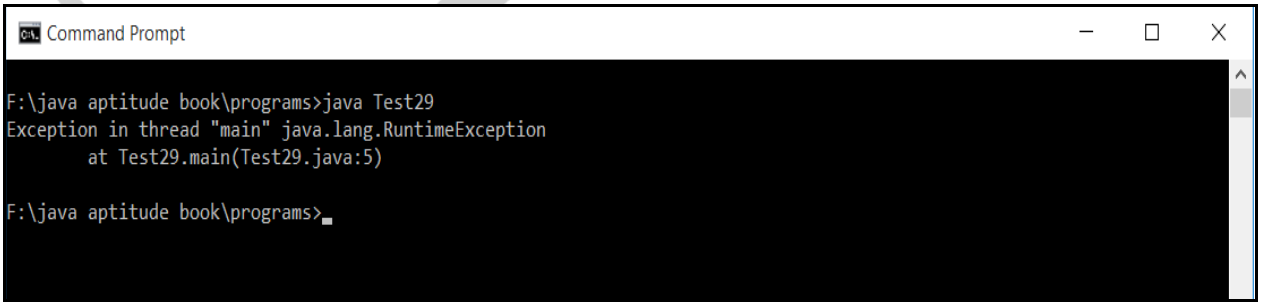


Explanation :

In the above program, replace statement 5 with the statement given below.

***RuntimeException re=new RuntimeException*();**

Re-execute the program, the following runtime error is generated.

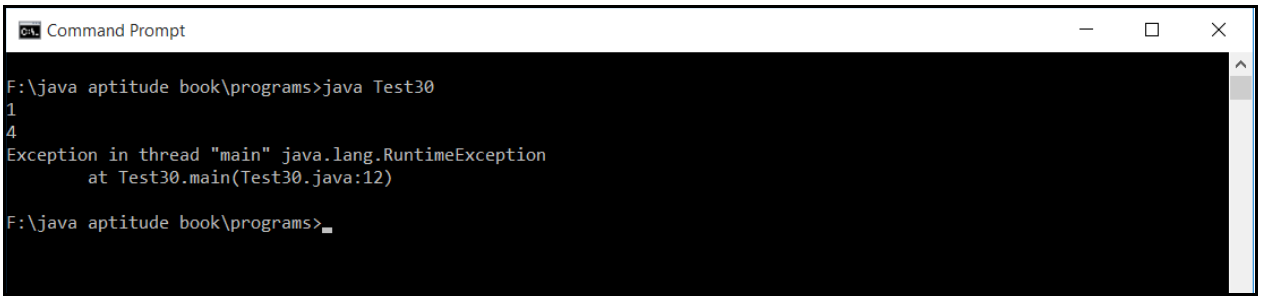


Q. No. 112 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test30
{
public static void main(String args[]) {
        try
        {
           f(); ←Statement 7
        }
        catch(InterruptedException e)
        {
           System.out.println("1");
           throw new RuntimeException();
        }

        catch(RuntimeException e)
        {
           System.out.println("2");
           return;
        }
        catch(Exception e)
        {
           System.out.println("3");
        }
        finally
        {
           System.out.println("4");
        }
        System.out.println("5");
        }

        static void f() throws InterruptedException
        {
           throw new InterruptedException("wait..");
        }
}
```

Output :



Explanation :

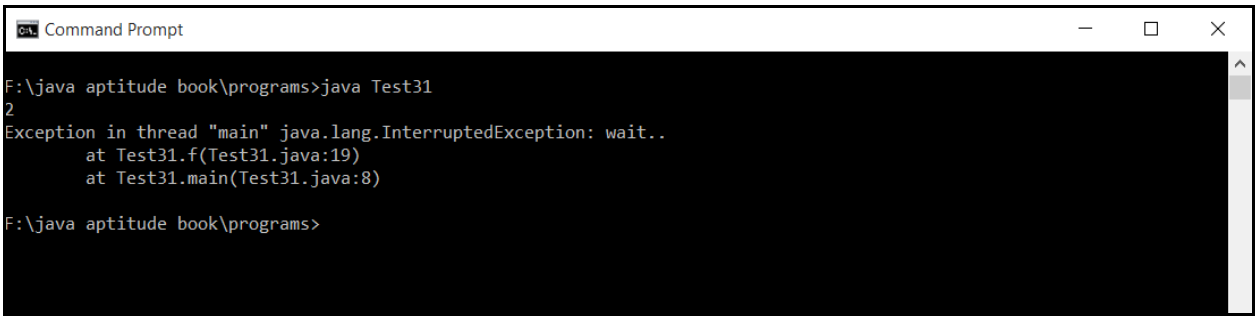
The main() method invokes the function f() in statement 7, which throws InterruptedException. There is a catch block in the main() to handle this exception which prints “1” and throws RuntimeException. Since it is not thrown in a try block, it cannot be handled in a program and is delegated to the Java runtime system for processing but before that the finally block is executed which prints “4”.

Q. No. 113 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test31
{
        public static void main(String args[]) throws InterruptedException {
                try
                {
                    f(); ←Statement 7
                   System.out.println("1");
                }
                finally
                {
                   System.out.println("2");
                }
                System.out.println("3");
        }

        static void f() throws InterruptedException
        {
           throw new InterruptedException("wait..");
        }
}
```

Output :



Explanation :

The main() method invokes the function f() in statement 7, which throws InterruptedException. Main(0 method does not have a catch block for processing it and it throws it to the caller of main(), Java runtime system. But before the unhandled exception is processed, finally block is executed which prints "2".

Q. No. 114 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test32
{
public static void main(String args[]) throws A
{
        try
        {
          f(); ←Statement 7
        }
        catch(A e)
        {
          throw e;
        }

        finally
        {
           System.out.println("Done");
        }
}

public static void f() throws B
{
   throw new B();
}
}

class A extends Throwable
{
}

class B extends A
{
}
```

Output :

Explanation :

The main() method invokes the function f() in statement 7, which throws exception B and main contains a catch block for catching an exception of type A. Since A is a super class of B, this is perfectly OK. The catch block re-throws exception B wrapped in a reference A which is processed by Java runtime system.

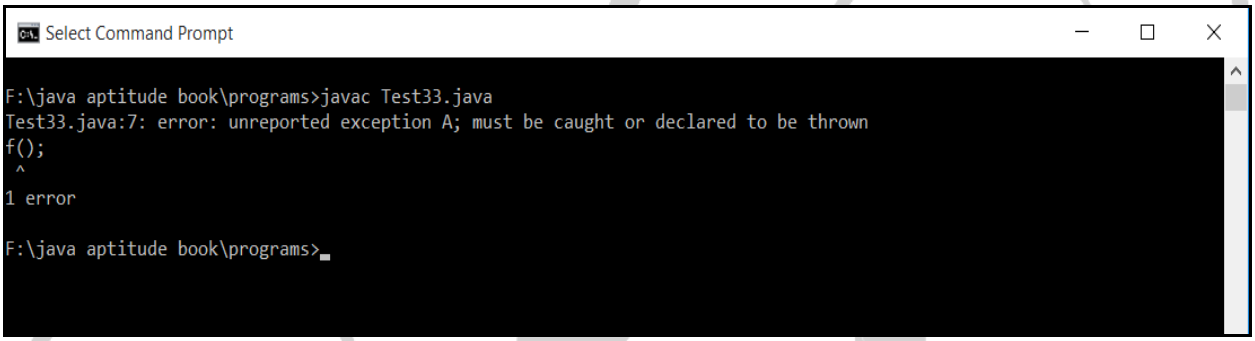
Q. No. 115 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test33
{
        public static void main(String args[]) throws B ←Statement 3
        {
                try
                {
                  f();
                }
                catch(B e)
                {
                   throw e;
                }
                finally
                {
                   System.out.println("Done");
                }
}
```

```
public static void f() throws A
{
   throw new A();
}
}

class A extends Throwable
{

}
class B extends A
{

}
```

Output :



Explanation :

In the above program, replace statement 3 with the one given below and re-execute the program.

```
public static void main(String args[]) throws A
```

On execution of the program, the following output is generated.

Q. No. 116 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test34
{
        public static void main(String args[]) throws ExceptionA
        {
                A obj=new A();
                try
                {
                   obj.f(); ←Statement 8
                }
                catch(ExceptionA e)
                {
                   System.out.println("ExceptionA caught in main..");
                   throw e;
                }
                finally
                {
                   System.out.println("Done");
                }
        }
}

class ExceptionA extends Throwable
{
}
class ExceptionB extends ExceptionA  {}
class A
{
public void f() throws ExceptionA
{
   throw new ExceptionA();
}
}
```

```
class B  extends A
{
  public void f() throws ExceptionB
  {
   throw new ExceptionB();
  }
}
```

Output :

Explanation :

In statement 8 main() invokes function f() of obj, which throws ExceptionA. The exception is caught in the main() and "ExceptionA caught in main.." is printed. The same exception is again re-thrown in main(), which is delegated to the Java runtime system and before that finally block is executed.

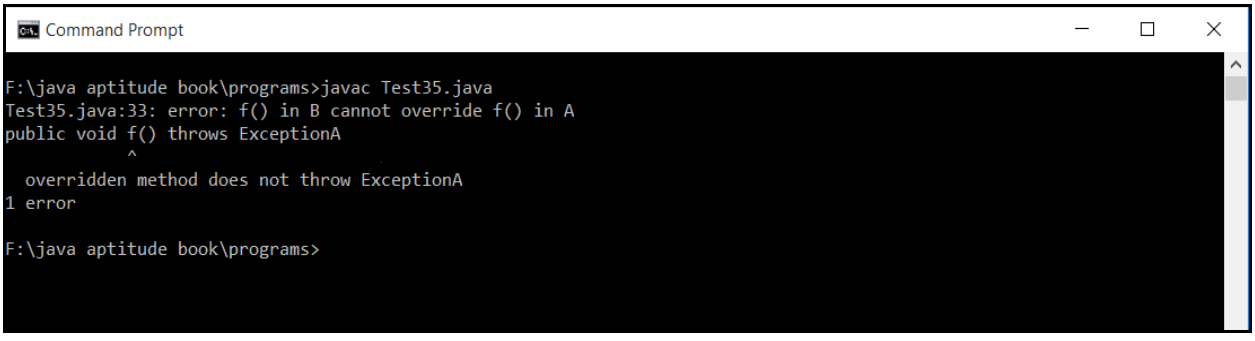
Q. No. 117 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test35
{
public static void main(String args[]) throws ExceptionA
{
        A obj=new A();
        try
        {
           obj.f();
        }
```

```
        catch(ExceptionA e)
        {
           System.out.println("ExceptionA caught in main..");
           throw e;
        }
        finally
        {
            System.out.println("Done");
        }
}
}

class ExceptionA extends Throwable
{ }

class ExceptionB extends ExceptionA
{ }

class A
{
        public void f() throws ExceptionB
        {
          throw new ExceptionB();
        }
}

class B  extends A
{
        public void f() throws ExceptionA
        {
          throw new ExceptionA();
        }
}
```

Output :



Explanation :

Replace the definitions of class A and class B as shown below:

```
class A
{
  public void f() throws ExceptionA
  {
    throw new ExceptionA();
  }
}

class B  extends A
{
  public void f() throws ExceptionB
  {
    throw new ExceptionB();
  }
}
```

Re-execute the application. The following output is generated.

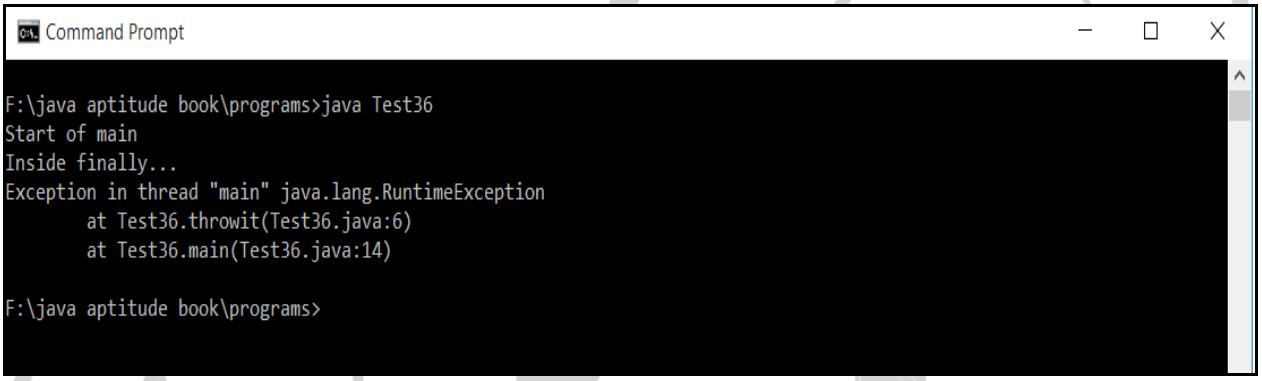
Q. No. 118 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test36
{
  public static void throwit()
  {
    throw new RuntimeException();
  }
```

```
 public static void main(String[] args)
 {
  try
  {
   System.out.println("Start of main");
   throwit();
   System.out.println("End of Try Block");
  }
  finally
  {
    System.out.println("Inside finally...");
  }
 }
}
```

Output :



Explanation :

On the execution of the program, the main() method prints

Start of main

and invokes throwit() method. throwit() method throws RuntimeException which it does not handle. Hence when the control returns to the main method, RuntimeException is uncaught and main() method either does not have a suitable catch block for catching an exception of type RuntimeException. Hence the exception is finally caught by java runtime, but before that finally block is executed which prints

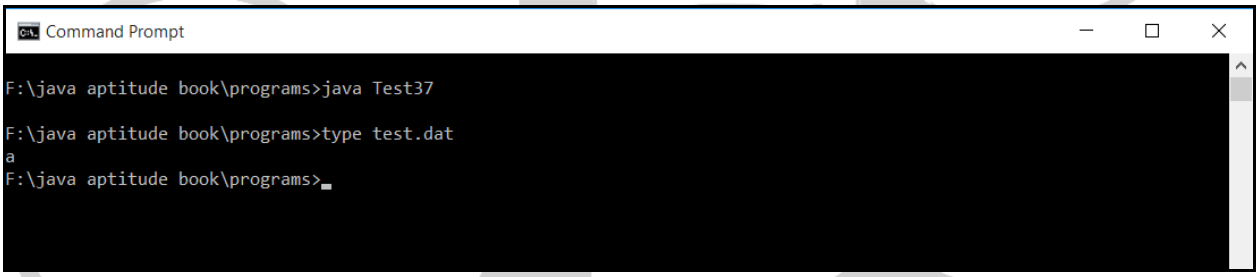
Inside finally...

Q. No. 119 *Category :Exception Handling*

What is the output generated on execution of the following Java Program?

```
import java.io.*;
public class Test
{
    public static void main(String args[])
    {
      try
      {
        FileOutputStream fos=new FileOutputStream("test.dat");
        fos.write(0x01234561);
        fos.close();
      }
      catch(IOException e)
      {
        System.out.println(e);
      }
    }
}
```

Output :



Explanation :

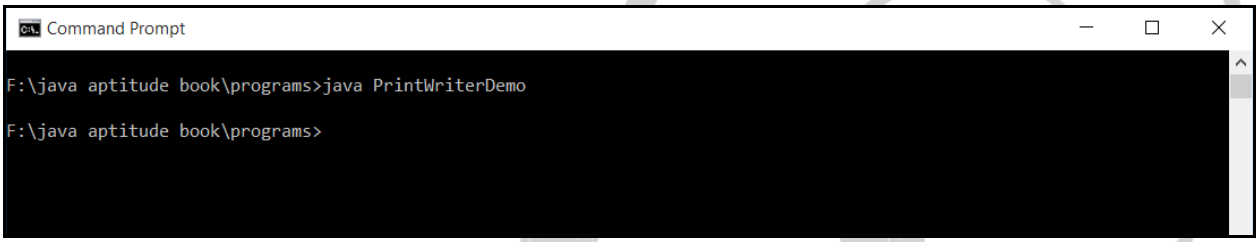
The write() method of OutputStream only writes bytes. When given an int, it will only write the 8 least significant bits. Hence the above write() statement will write ASCII code corresponding to the hexadecimal number 61 (decimal equivalent is 97 which is the ASCII code for ‘a’) to the destination of the stream.

Q. No. 120 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
import java.io.*;
public class PrintWriterDemo
{
public static void main(String[] args)
{
        PrintWriter pw=new PrintWriter(System.out,false); ←Statement 6
        pw.println("This is a string");
        int i=-7;
        pw.println(i);
        double d=1.2345;
        pw.println(d);
}
}
```

Output :



Explanation :

No output is generated when the above Java program is executed. The second parameter of a two argument PrintWriter constructor accepts a boolean parameter which turn on and off auto flush when set tot true and false, respectively.

There are two ways in which the application can be modified to generate output.

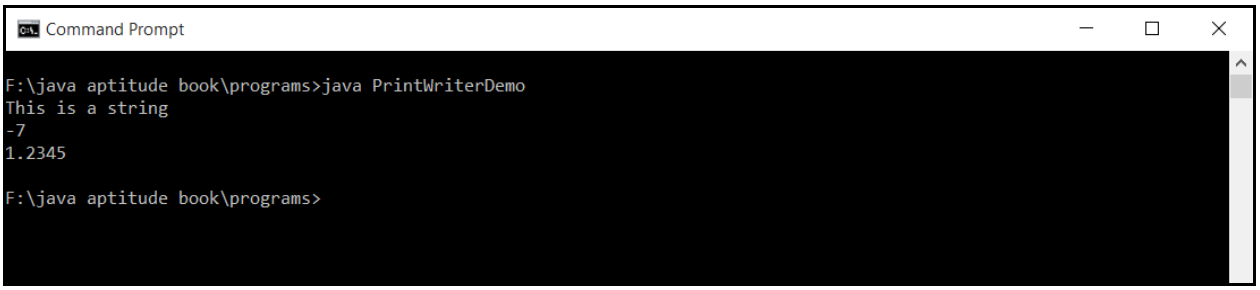
Method 1 :

Invoke flush() method of PrintWriter class for manually flushing the buffer.

Add the following statement at the end of the program and re-execute the program.

```
pw.flush();
```

On execution of the program, the following output is generated.



Method 2 :

The same output is generated on removing statement 12 and replacing statement 6, with the one given below:

PrintWriter pw=new PrintWriter(System.out,true);

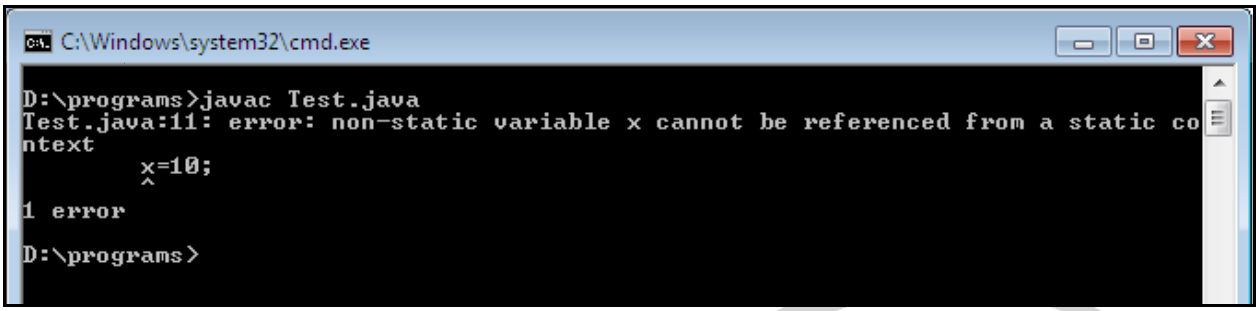
The second parameter of the constructor pertaining to autoflush on, is set to true now.

Q. No. 121 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
class B
{
   int x;
   B()
   {
      System.out.println("B-Constructor Called");
   }

   static
   {
      x=10;
   }
}

class Test
{
      public static void main(String[] args)
   {
      B b1 = new B();
      B b2 = new B();
   }
}
```

Output :



Explanation :

Non-static data member cannot be initialized in the static block.

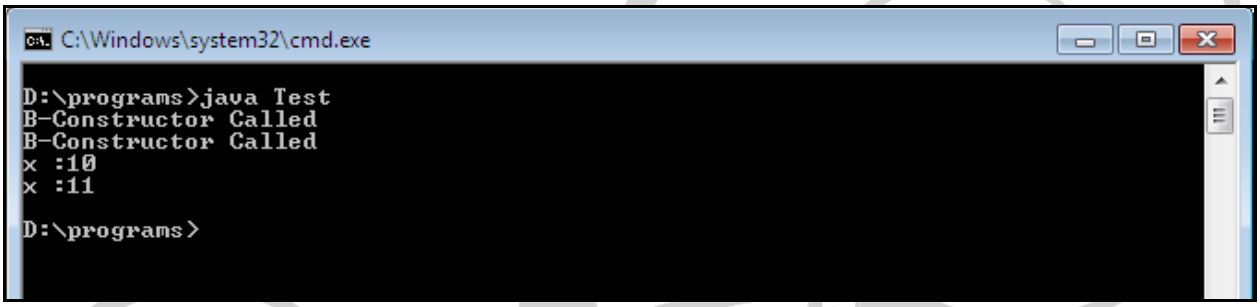
Q. No. 122 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
class B
{
   static int x;
   B()
   {
      System.out.println("B-Constructor Called");
   }

   static
   {
      x=10;
   }

   void display()
   {
      System.out.println("x :"+x);
   }

   void increment()
   {
     x++;
   }
}
```

```
class Test
{

   public static void main(String[] args)
   {
      B b1 = new B();
      B b2 = new B();
      b1.display();
      b1.increment();
      b2.display();
   }
}
```

Output :



Explanation :

Since x is declared as static in class B, all instances of class B share the static member x. There is only a single copy of x which is shared by all instances of class B. Any modification to x performed by a single instance is visible to all other instances of class B. Class B contains a static block to initialize the value of x to 10 and a display() method to display the value of x. Irrespective of number of instances of B created, static block is invoked only once to initialize the static data member x.

In the given program, two instances of B, b1 and b2 are created, in each case a default constructor is invoked which displays

B-Constructor Called

twice.

Then, display method is invoked on b1 to display the current value of x which displays

x :10

Then, b2 increments the value of x and display() method is invoked on b2 to display the current value of x again, which displays,

x : 11

Hence the program generates the output,

B-Constructor Called

B-Constructor Called

x :10

x : 11

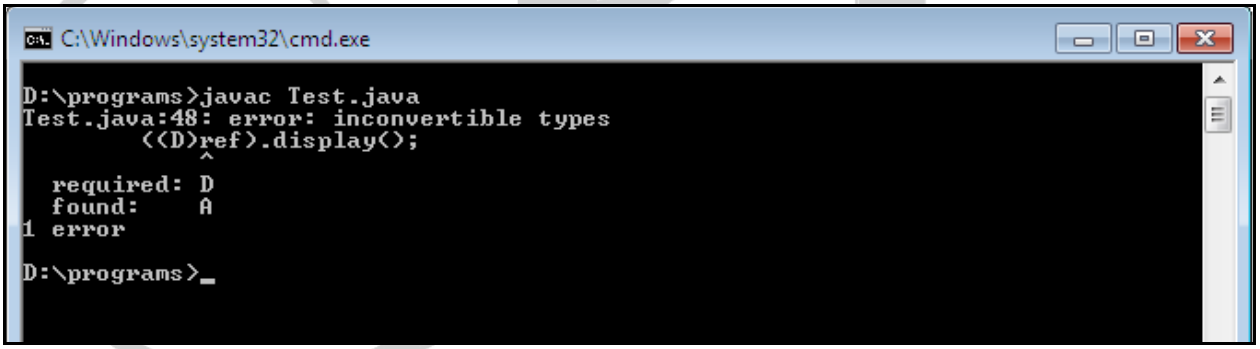
Q. No. 123 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class A
{
   void display()
   {
      System.out.println("In display of A...");
   }
}

class B extends A
{
    void display()
   {
      System.out.println("In display of B...");
   }

   void display1()
   {
      System.out.println("In display1 of B...");
   }
}

class C
{
   void display()
   {
      System.out.println("In display of C...");
   }
}
```

```
class D extends C
{
    void display()
    {
        System.out.println("In display of D...");
    }

    void display1()
    {
        System.out.println("In display1 of D...");
    }
}

public class Test
{
    public static void main(String args[])
    {
        A ref = new B();
        ((D)ref).display();
    }
}
```

Output :



Explanation :

In the given program, the following class hierarchies exist.

Class A has a display() method, which is overridden by class B. Further class B has added a new member function display1(). Similarly, C has a display() method, which is overridden by class D. Further class D has added a new member function display1().

From the class hierarchy it is evident that, reference of class A can either be initialized with an object of class A or an object of class B. Similarly, reference of class C can either be initialized with an object of class C or an object of class D. However, for accessing the new functionality added by either class B or class D, the reference must be properly cast to the appropriate class object and the inherited or overridden functionality can be accessed directly using the base class reference.

In the given program A and D are not related by any inheritance relationship. Hence the functionality of D cannot be accessed using A's reference. Trying to do it will result in "***inconvertible types***" error.

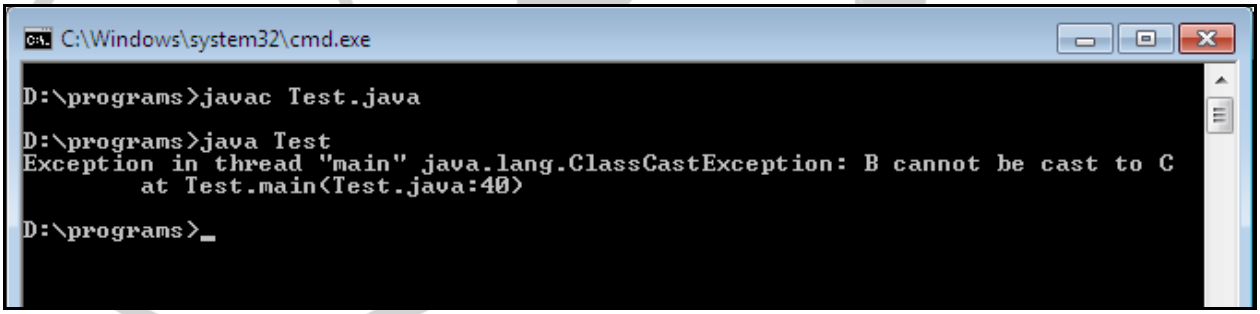
Q. No. 124 *Category : Inheritance*

What is the output generated on execution of the following Java Program?

```
class A
{
   void display()
   {
      System.out.println("In display of A...");
   }
}

class B extends A
{
    void display()
   {
      System.out.println("In display of B...");
   }

   void display1()
   {
      System.out.println("In display1 of B...");
   }
}
```

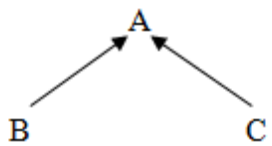
```
class C extends A
{
    void display()
    {
        System.out.println("In display of C...");
    }

    void display2()
    {
        System.out.println("In display2 of C...");
    }
}

public class Test
{
    public static void main(String args[])
    {
        A ref = new B();
        ((C)ref).display2();
    }
}
```

Output :



Explanation :

In the given program, the following class hierarchy exists.



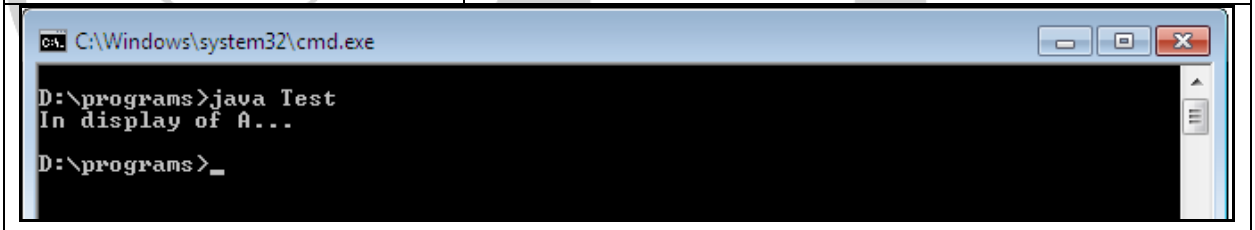
Class A has a display() method, which is overridden by classes B and C. Further, class B has added a new member function display1() and class C has added a new member function display2(). In the given program, ref is a base class reference initialized with an object of derived class B.

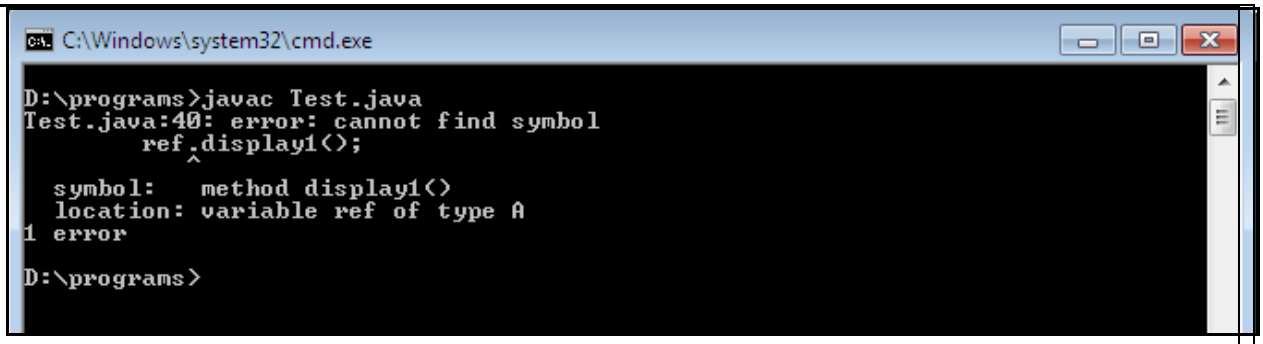
Following rules apply:

- Inherited functionality can be directly accessed through base class reference, ref
- Overridden functionality can be accesses directly through base class reference and the corresponding overridden versions in the derived class are accessed.
- New functionality added by the derived class B can be accessed by casting ref to derived class B.

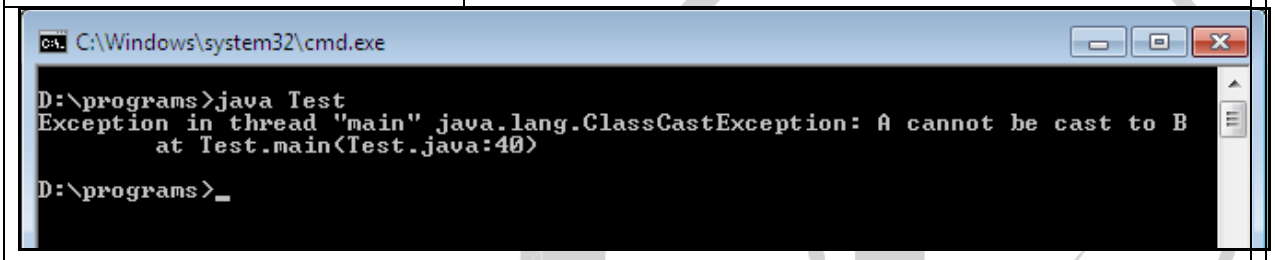
In the above inheritance tree, there is no direct relationship between classes B and C except the fact that they have a common base class. Hence A's reference initialized with an object of class B cannot be used for accessing the functionality of class C. Doing so will result in ClassCastException as revealed in the output generated by the given program.

The following table depicts the methods invoked in various cases:

A ref = new A()	
Statement	***Method Invoked***
ref.display()	A's display()
	
ref.display1()	Compiler Error

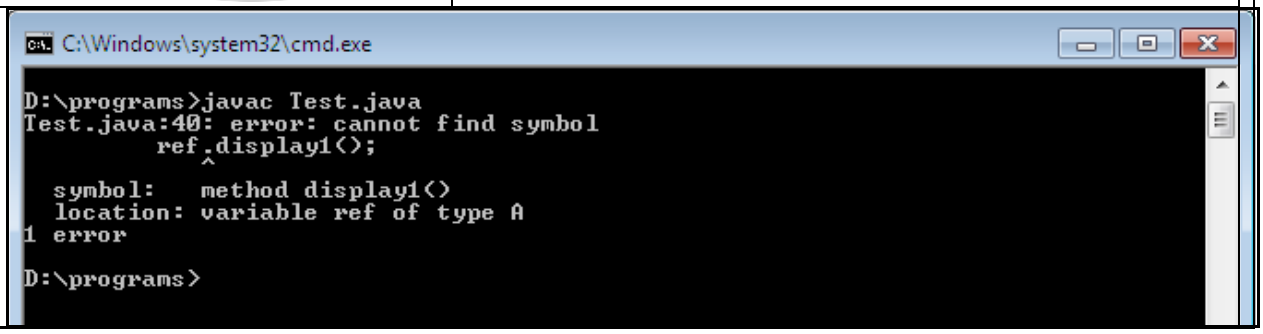


ref.display2()	<Same As Above>
((B)ref).display1()	Compiles successfully, but generates runtime error.



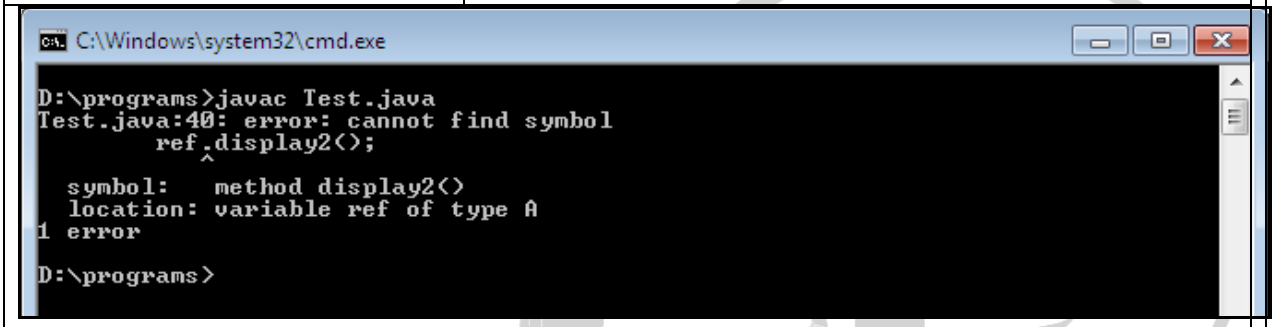
((C)ref).display2()	<Same As Above>
A ref = new B()	
ref.display()	B's display()

ref.display1()	

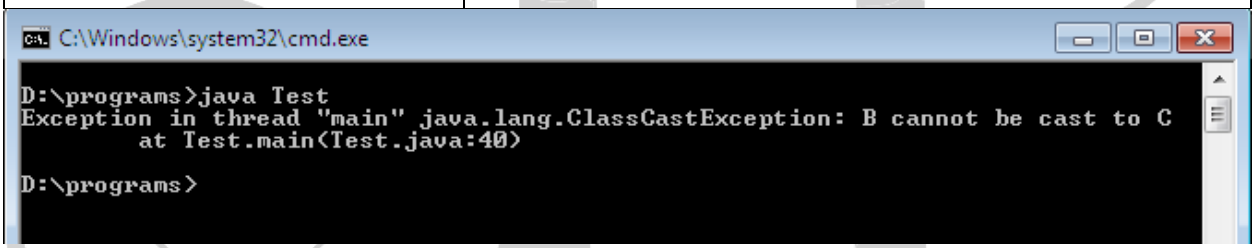


((B)ref).display1()	B's display1()

ref.display2()	Compiler Error



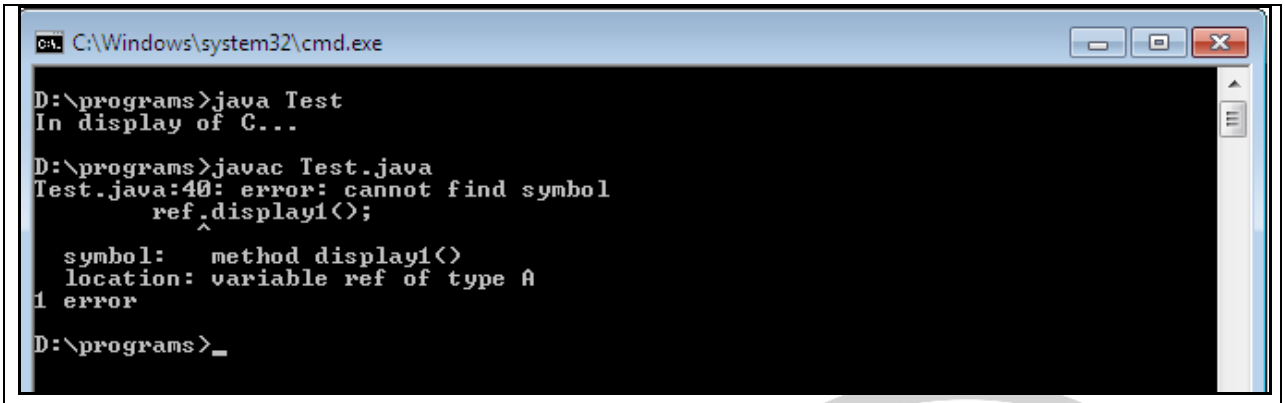
((C)ref).display2()	Compiles successfully. Generates runtime error.



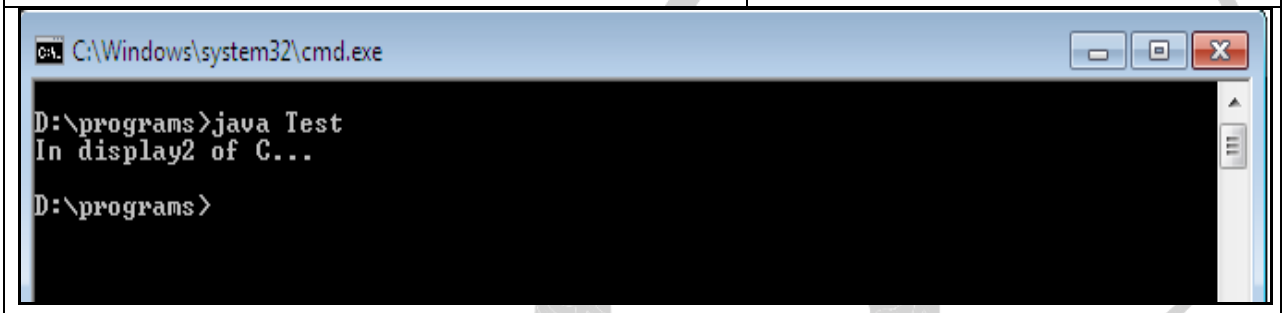
A ref = new C()

ref.display()	C's display()

ref.display1()	Compiler Error



ref.display2()	Compiler Error, Same as above
((C)ref).display2()	C's display2()

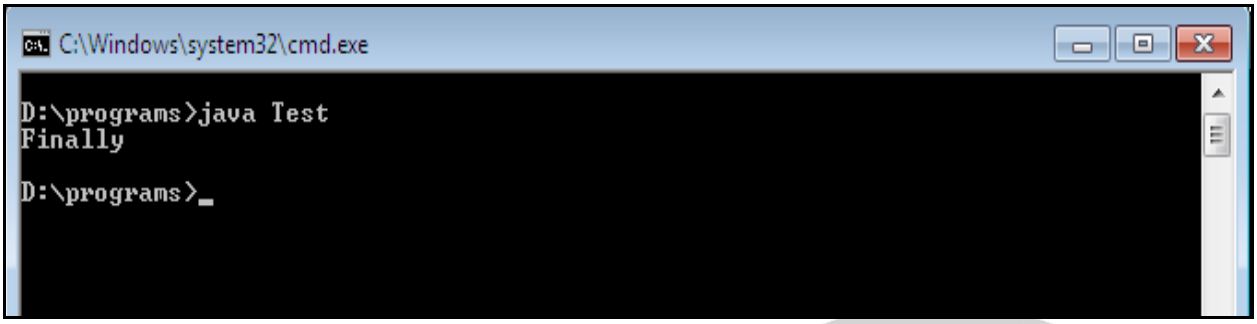


Q. No. 125 *Category : Exception Handling*

What is the output generated on execution of the following Java Program?

```
public class Test
{
   public static void main(String[] args)
   {
      try
      {
         return;
      }
      finally
      {
         System.out.println( "Finally" );
      }
   }
}
```

Output :



Explanation :

It is perfectly valid to place a finally block immediately after a try block. A try block can be followed either by one or more catch blocks and/or a finally block. A finally block if present is guaranteed to be executed under all circumstances such as

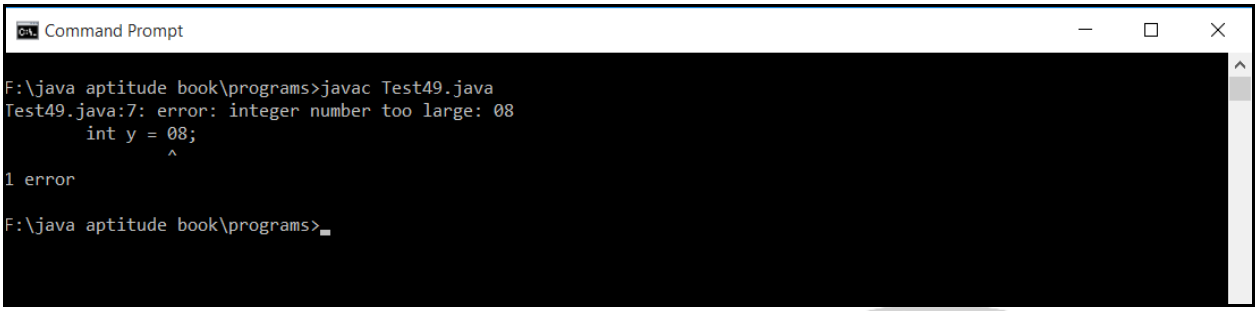
- When exception does not occur
- When exception occurs
- When function returns
- When the function abnormally terminates etc.

Q. No. 126 *Category : Data Types*

What is the output generated on execution of the following Java Program?

```
public class Test49
{
   public static void main(String args[])
   {
      short s = 0;
      int x = 07;
      int y = 08;
      int z = 112345;

      s += z;
      System.out.println("" + x + y + s);
   }
}
```

Output :



Explanation :

In Line 12 The “” in the println causes the numbers to be automatically cast as strings. So it doesn’t do addition, but appends together as string.

In Line11 the += does an automatic cast to a short. However the number 123456 can’t be contained within a short, so you end up with a negative value (-7616).

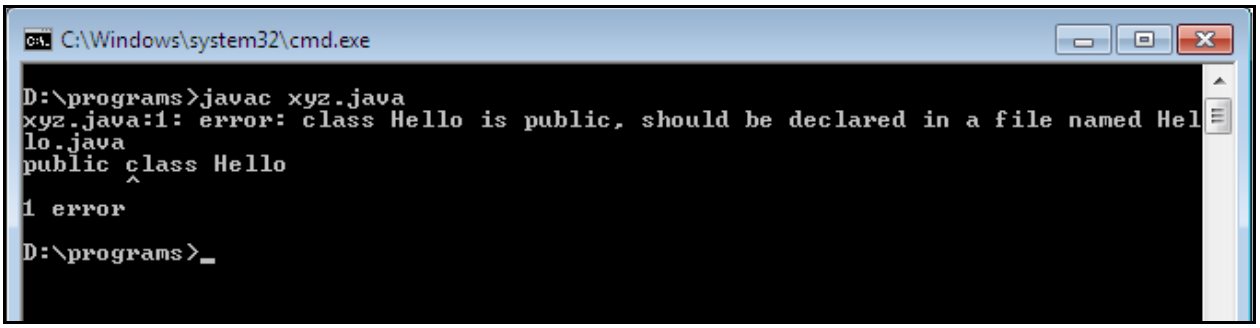
Those other two are red herrings however as the code will never compile due to line 8. Any number beginning with zero is treated as an octal number (which is 0-7).

Q. No. 127 *Category : Java Basics*

What is the output generated on execution of the following Java Program if the code is stored in a file xyz.java?

```
public class Hello
{
  public static void main(String[] args)
  {
     System.out.println("Hello from Java!");
  }
}
```

Output :

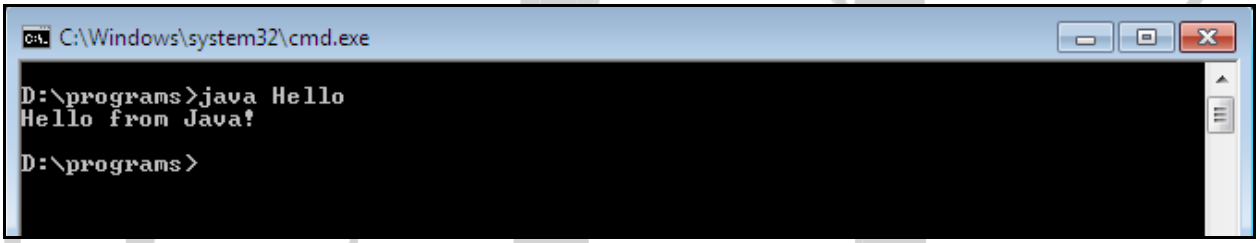


Explanation :

If the file containing the java source code contains a public class, then the name and case of the file must be the same as the public class name. Hence the above java source code must be stored in a file with the name Hello.java. The reason for this is java runtime invokes the main() method which is declared as static using the class name as

Hello.main()

On changing the file name to Hello.java and re-executing the application, the following output is generated.



Remove the public qualifier from the class name as shown below and now save the file with the name xyz.java.

```
class Hello
{
  public static void main(String[] args)
  {
     System.out.println("Hello from Java!");
  }
}
```

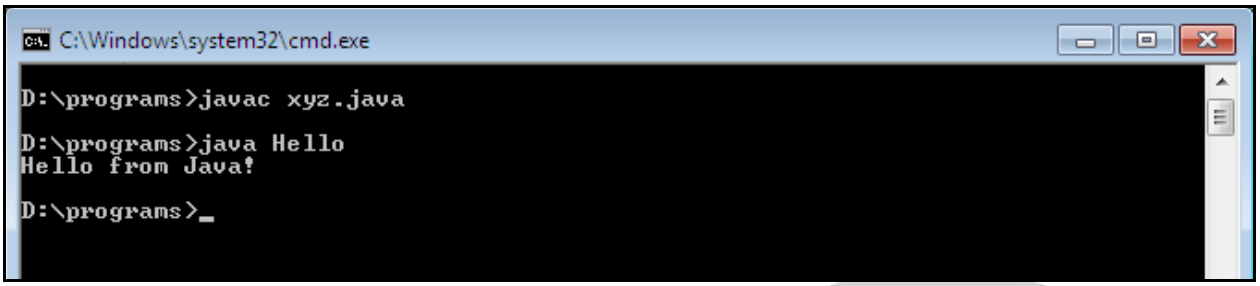
Compile the program using the command

javac xyz.java

and execute it using the command

java Hello

On execution of the program, the following output is generated.

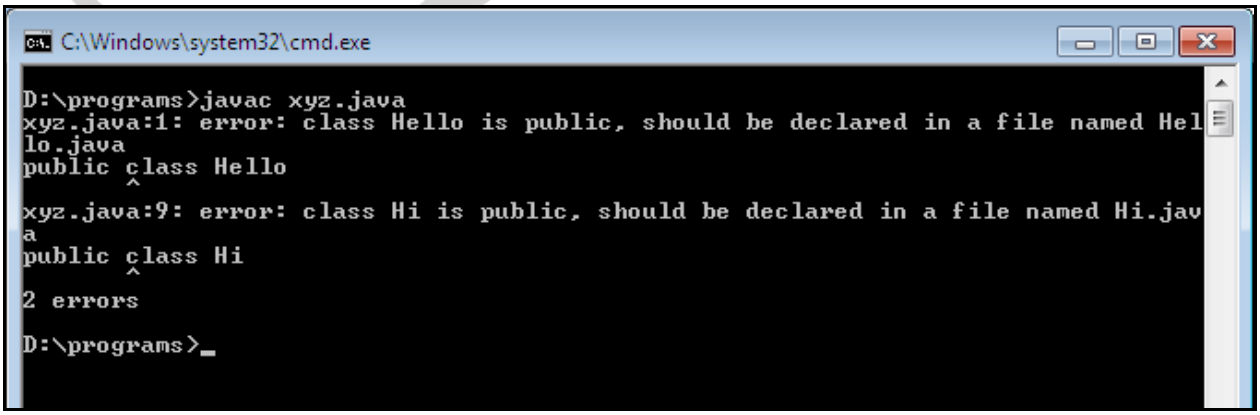


Q. No. 128 *Category : Java Basics*

What is the output generated on execution of the following Java Program if the code is stored in a file xyz.java?

```
public class Hello
{
  public static void main(String[] args)
  {
     System.out.println("Hello from Java!");
  }
}

public class Hi
{
  public static void main(String[] args){
     System.out.println("Hi from Java!");
  }
}
```

Output :



Explanation :

In a file containing a java source code there can be zero or only one public class. Remove the public qualifier from both the classes. Now the name of the class can contain any arbitrary value such as xyz.java.

Compile the program using the command

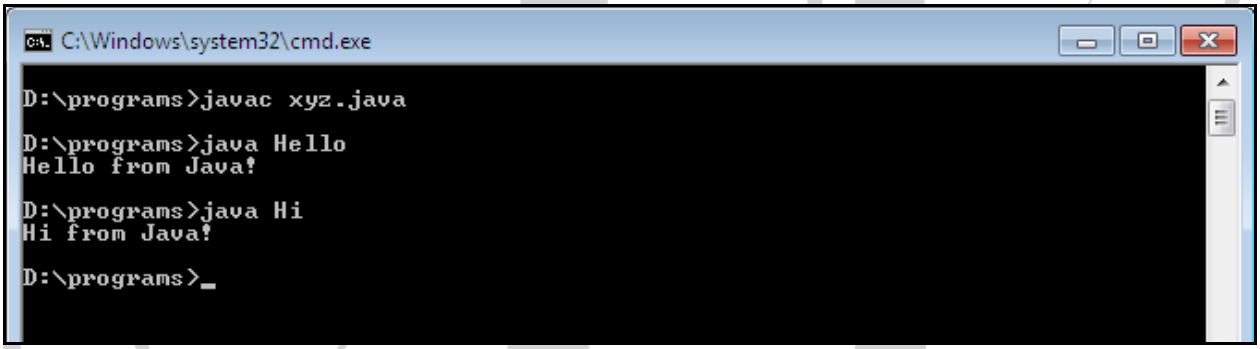
javac xyz.java

and execute it using the commands

java Hello and

java Hi

On execution of the program, the following output is generated.



There can be more than one main() method in the class. In that case if the name of the file containing the java source code is arbitrary, then the main() method invoked will depend on the name of the class used in java command. In general,

java <className>

will invoke main() method present in <className>.

Q. No. 129 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
public class Hello
{
  public static void main(String[] args)
  {
     main();
     System.out.println("Hello Again from Java!");
  }

  public static void main()
  {
    System.out.println("Hello from Java!");
  }
}
```

Output :

Explanation :

Like other methods, main() method can be overloaded. During the execution of the program, java runtime looks for the main method with the following prototype.

public static void main(String[] args)

In the given program, the main() method is overloaded. The above main() method invokes the overloaded version of main() method generating the output

Hello from Java!

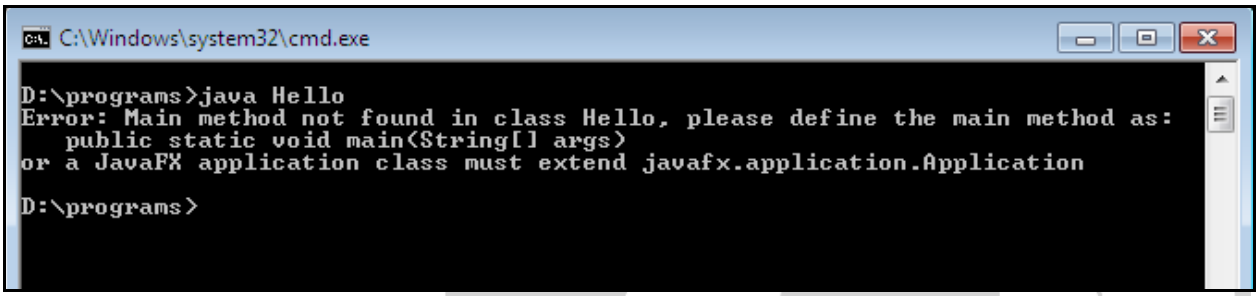
Hello Again from Java!

Q. No. 130 *Category : Java Basics*

What is the output generated on execution of the following Java Program?

```
public class Hello
{
  public static void main()
  {
    System.out.println("Hello from Java!");
  }
}
```

Output :



Explanation :

The given java program successfully compiles but generates a runtime error as shown in the output. The reason for this is that during the execution of the program, java runtime looks for the main method with the following prototype

public static void main(String[] args);

which is not available in the program. The program contains the main method with the prototype

public static void main();

References :

1. Core Java 2 Volume-I Cay S Horstmann, Fary Cornell, Sun Microsystems Press, 8thEd.
2. Core Java 2 Volume-II Cay S Horstmann, Fary Cornell, Sun Microsystems Press, 8thEd.
3. Java Complete Reference by Patric Norton
4. Core Java Vol. I, Sun Press ISBN – 981-405-861-0, Addison- Wesley.
5. Core Java Vol. II, Sun Press ISBN – 981-4058-50-5, Addison- Wesley.
6. Thinking in Java, Bruce Eckel,, Addison – Wesley, ISBN: 9814035750
7. Java 2 Programming Black Book by Steven Holzner, Dream Tech Publication
8. Learning Java by Jonathan Knudsen, Patrick Niemeyer, Publisher: O'Reilly
9. The Java FAQ by Jonni Kanerva, Pearson Education, India.
10. The Complete Reference Java –Seventh Edition – Herber Schildt, Tata McGraw Hill
11. Learning Java – An Experiential Approach by Dr. Poornima G. Naik and Dr. Kavita S. Oza, LAP Lambert Publisher, ISBN - 978-3-659-89538-8.
12. A Programmer's Guide to Java™ SCJP Certification-A Comprehensive Primer, Third Edition, Khalid A. Mughal Rolf W. Rasmussen, Addison Wesley.

Important Web Links

1. http://www.tutorialspoint.com/java/
2. http://www.javatpoint.com/java-tutorial
3. http://www.studytonight.com/java/
4. http://www.vogella.com/tutorials/JavaIntroduction/article.html
5. https://www.udemy.com/java-tutorial/
6. http://www.w3schools.in/java/intro/
7. http://www.indiabix.com/java-programming/questions-and-answers/
8. http://www.wideskills.com/java-tutorial
9. https://howtoprogramwithjava.com/programming-101-the-5-basic-concepts-of-any-programming-language/
10. https://www.cs.utexas.edu/~scottm/cs307/handouts/Eclipse%20Help/EclipseIntroduction.html

11. https://www3.ntu.edu.sg/home/ehchua/programming/java/J4c_AppletWebstart.html
12. http://www.javaprepare.com/scwd/index.html
13. http://www.freewarejava.com/
14. http://www.javagalaxy.com