**MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION**
(Autonomous)
**(ISO/IEC - 27001 - 2013 Certified)**

**WINTER– 18 EXAMINATION**

**Subject Name: Java Programming** **Model Answer** **Subject Code: 17515**

**Important Instructions to examiners:**

1) The answers should be examined by key words and not as word-to-word as given in the model answer scheme.
2) The model answer and the answer written by candidate may vary but the examiner may try to assess the understanding level of the candidate.
3) The language errors such as grammatical, spelling errors should not be given more Importance (Not applicable for subject English and Communication Skills.
4) While assessing figures, examiner may give credit for principal components indicated in the figure. The figures drawn by candidate and model answer may vary. The examiner may give credit for any equivalent figure drawn.
5) Credits may be given step wise for numerical problems. In some cases, the assumed constant values may vary and there may be some difference in the candidate's answers and model answer.
6) In case of some questions credit may be given by judgement on part of examiner of relevant answer based on candidate's understanding.
7) For programming language papers, credit may be given to any other program based on equivalent concept.

<table>
<tr><th>Q. No.</th><th>S. Q. No.</th><th>Answer</th><th>Marking Scheme</th></tr>
<tr><td>1</td><td>A</td><td><b>Attempt any three of the following.</b></td><td>3x4=12</td></tr>
<tr><td></td><td>a)</td><td><b>Explain following terms related to Java features.<br>i) Object Oriented ii) Complied and interpreted.</b></td><td>4</td></tr>
<tr><td></td><td>Ans:</td><td><b>i) Object Oriented:</b><br>• Almost everything in java is in the form of object.<br>• All program codes and data reside within objects and classes.<br>• Similar to other OOP languages java also has basic OOP properties such as encapsulation, polymorphism, data abstraction, inheritance etc.<br>• Java comes with an extensive set of classes (default) in packages.<br><br><b>ii) Compiled and Interpreted:</b><br>• Java is a two staged system. It combines both approaches.<br>• First java compiler translates source code into byte code instruction. Byte codes are not machine instructions.<br>• In the second stage java interpreter generates machine code that can be directly executed by machine. Thus java is both compile and interpreted language.</td><td>2 Marks for each)</td></tr>
<tr><td></td><td>b)</td><td><b>Differentiate between Input stream class and Reader class.</b></td><td>4</td></tr>
<tr><td></td><td>Ans:</td><td><table><tr><th>Input Stream Class</th><th>Reader Class</th></tr><tr><td>Input Streams are used to read bytes from a stream.</td><td>Reader classes are used to read character streams.</td></tr><tr><td>Input Stream class useful for binary data such as images, video</td><td>Reader classes are best used to read character data.</td></tr></table></td><td>any 4 Correct Points. 1 Mark for one point)</td></tr>
</table>

| | |
|---|---|
| and serialized objects. | |
| Input Stream classes are used to read 8 bit bytes. | Reader class is used to read 16 bit Unicode character stream. |
| An Input Stream is byte-oriented. | A Reader is character-oriented. |
| Constructor: InputStream() | Constructor:<br>Reader() and Reader(Object lock) |
| Methods:<br>read();<br>read(byte[] b);<br>read(byte[] b, int off, int len); | Methods:<br>read();<br>read(char[] buf);<br>read(char[] buf, int off, int len);<br>read(charBuffer dest); |

**c)** **Explain any two logical operators in java with example.** **4**

**Ans:** **Logical Operators:** Logical operators are used when we want to form compound conditions by combining two or more relations. Java has three logical operators:

**2 Marks for each operator. Any 2 operators)**

&& : Logical AND
|| : Logical OR
! : Logical NOT

**Example:**

```
public class Test
{
public static void main(String args[])
{
boolean a = true;
boolean b = false;
System.out.println("a && b = " + (a&&b));
System.out.println("a || b = " + (a||b) );
System.out.println("!(a && b) = " + !(a && b));
}
}
```

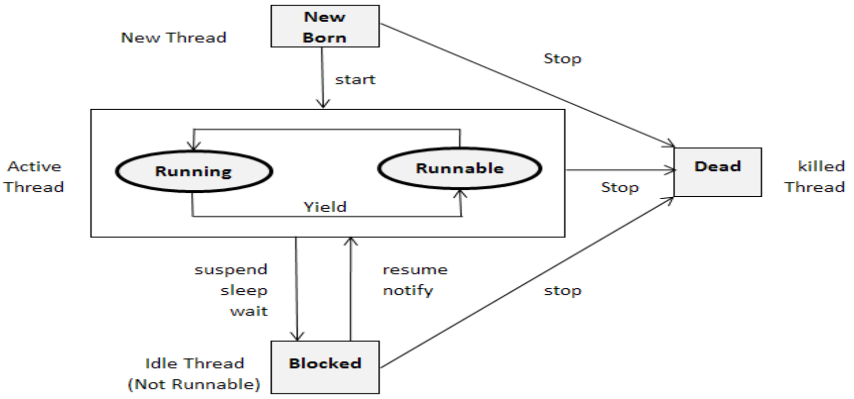
**Output:**

```
a && b = false
a || b = true
!(a && b) = true
```

**d)** **Describe life cycle of thread.** **4**

**Ans:**

**1 Mark for correct diagram, 3 M for proper explanation**



**Thread Life Cycle** Thread has five different states throughout its life.

| | | | |
|---|---|---|---|
| | | **1.** Newborn State<br>**2.** Runnable State<br>**3.** Running State<br>**4.** Blocked State<br>**5.** Dead State<br>Thread should be in any one state of above and it can be move from one state to another by different methods and ways.<br>1. **Newborn state:** When a thread object is created it is said to be in a new born state. When the thread is in a new born state it is not scheduled running from this state it can be scheduled for running by start() or killed by stop(). If put in a queue it moves to runnable state.<br>2. **Runnable State:** It means that thread is ready for execution and is waiting for the availability of the processor i.e. the thread has joined the queue and is waiting for execution. If all threads have equal priority, then they are given time slots for execution in round robin fashion. The thread that relinquishes control joins the queue at the end and again waits for its turn. A thread can relinquish the control to another before its turn comes by yield().<br>3. **Running State:** It means that the processor has given its time to the thread for execution. The thread runs until it relinquishes control on its own or it is pre-empted by a higher priority thread.<br>4 **Blocked State:** A thread can be temporarily suspended or blocked from entering into the runnable and running state by using either of the following thread method.<br>**suspend() :** Thread can be suspended by this method. It can be rescheduled by resume().<br>**wait():** If a thread requires to wait until some event occurs, it can be done using wait method and can be scheduled to run again by notify().<br>**sleep():** We can put a thread to sleep for a specified time period using sleep(time) where time is in ms. It reenters the runnable state as soon as period has elapsed /over.<br>5 **Dead State:** Whenever we want to stop a thread form running further we can call its stop(). The statement causes the thread to move to a dead state. A thread will also move to dead state automatically when it reaches to end of the method. The stop method may be used when the premature death is required | |
| **1** | **B** | **Attempt any One of the following.** | **1x6=6** |
| | **a)** | **Define a class ‘Book’ with data members bookid, bookname and price. Accept data for seven objects using Array of objects and display it.** | **6** |
| | **Ans:** | import java.lang.*;<br>import java.io.*;<br>class Book<br>{<br>String bookname;<br>int bookid;<br>int price;<br>BufferedReader br=new BufferedReader(new InputStreamReader<br>(System.in));<br>void getdata()<br>{ | **Correct program with proper logic 4 Marks** |

```
try
{
System.out.println("Enter Book ID=");
bookid=Integer.parseInt(br.readLine());
System.out.println("Enter Book Name=");
bookname=br.readLine();
System.out.println("Enter Price=");
price=Integer.parseInt(br.readLine());
}
catch(Exception e)
{
System.out.println("Error");
}
}
void display()
{
System.out.println("Book ID="+bookid);
System.out.println("Book Name="+bookname);
System.out.println("Price="+price);
}
}
class bookdata
{
public static void main(String args[])
{
Book b[]=new Book[7];
for(int i=0;i<7;i++)
{
b[i]=new Book();
}
for(int i=0;i<7;i++)
{
b[i].getdata();
}
for(int i=0;i<7;i++)
{
b[i].display();
}
}
}
```

| | | | |
|---|---|---|---|
| | **b)** | **What is interface? Describe its syntax and features.** | **6** |
| | **Ans:** | **Definition:** Java does not support multiple inheritances with only classes. Java provides an alternate approach known as interface to support concept of multiple inheritance. An interface is similar to class which can define only abstract methods and final variables.<br>**Syntax:**<br>access interface InterfaceName<br>{<br>Variables declaration; | **Definition: 1 Mark, 1 Mark for syntax and 2 Marks for Features** |

Methods declaration;
}

**Features:**

- The interfaces are used in java to implementing the concept of multiple inheritance.
- The members of an interface are always declared as constant i.e. their values are final.
- The methods in an interface are abstract in nature. I.e. there is no code associated with them.
- It is defined by the class that implements the interface.
- Interface contains no executable code.
- We are not allocating the memory for the interfaces.
- We can‘t create object of interface.
- Interface cannot be used to declare objects. It can only be inherited by a class.
- Interface can only use the public access specifier.
- An interface does not contain any constructor.
- Interfaces are always implemented.
- Interfaces can extend one or more other interfaces.

| | | | |
|---|---|---|---|
| 2 | | **Attempt any Two of following** | **2x8=16** |
| | **a)** | **Write a program to create a vector with seven elements as (10,30,50,20,40,10,20). Remove elements 3rd and 4th position. Insert new elements at 3rd position. Display original and current size of vector.** | **8** |

**Ans:**

```
import java.util.*;
public class VectorDemo
{
public static void main(String args[])
{
Vector v = new Vector();
v.addElement(new Integer(10));
v.addElement(new Integer(30));
v.addElement(new Integer(50));
v.addElement(new Integer(20));
v.addElement(new Integer(40));
v.addElement(new Integer(10));
v.addElement(new Integer(20));
System.out println(v.size());            // display original size
v.removeElementAt(2);                    // remove 3rd element
v.removeElementAt(3);                   // remove 4th element
v.insertElementAt(11,2)                // new element inserted at 3rd position
System.out.println("Size of vector after insert delete operations: " + v.size());
}
}
```

**(Correct Program with Correct logic 8 marks Creation of vector: 3Marks, removing elements 2 Marks inserting element 1 Mark Display original size : 1 Mark and Current size: 1Mark)**

| | | | |
|---|---|---|---|
| | **b)** | **What is package in Java? Write a program to create a package and** | **8** |

| | | | |
|---|---|---|---|
| | | **import the package in another class.** | |
| | **Ans:** | **Package:** Java provides a mechanism for partitioning the class namespace into more manageable parts called package (i.e package are container for a classes). The package is both naming and visibility controlled mechanism. Package can be created by including package as the first statement in java source code. Any classes declared within that file will belong to the specified package.<br>**Syntax:**<br>**package *pkg*;**<br>Here, *pkg* is the name of the package<br><br>**Program:**<br>**package1:**<br>package package1;<br>public class Box<br>{<br>int l= 5;<br>int b = 7;<br>int h = 8;<br>public void display()<br>{<br>System.out.println("Volume is:"+(l*b*h));<br>}<br>}<br>}<br><br>**Source file:**<br>import package1.Box;<br>class VolumeDemo<br>{<br>public static void main(String args[])<br>{<br>Box b=new Box();<br>b.display();<br>}<br>} | **(Definition: 2 Mark, any correct Program with proper logic: 6 Mark)** |
| | **c)** | **Write syntax and example of**<br>**i)Draw Poly ii)Draw Rect iii)Filloval iv)Draw Arc()** | **8** |
| | **Ans:** | **i) Draw Poly:** drawPoly() method is used to draw arbitrarily shaped figures.<br>**Syntax:** void drawPoly(int x[], int y[], int numPoints)<br>The polygon‟s end points are specified by the co-ordinates pairs contained within the x and y arrays.<br>The number of points define by x and y is specified by numPoints.<br>**Example:**<br>int xpoints[]={30,200,30,200,30};<br>int ypoints[]={30,30,200,200,30};<br>int num=5;<br>g.drawPoly(xpoints,ypoints,num); | **(Each one for 2 Mark)** |

**ii) Draw Rect:** The drawRect() method display an outlined rectangle.
**Syntax:** void drawRect(int top,int left,int width,int height)
The upper-left corner of the Rectangle is at top and left. The dimension of the Rectangle is specified by width and height.
**Example:** g.drawRect(10,10,60,50);

**iii)Fill Oval:** Drawing Ellipses and circles: To draw an Ellipses or circles used fillOval() method can be used. It draws the Solid Ellipses and circles.
**Syntax:** void fillOval(int top, int left, int width, int height)
The filled ellipse is drawn within a bounding rectangle whose upper-left corner is specified by top and left and whose width and height are specified by width and height to draw filled circle, specify the same width and height the following program draws several ellipses and circle.
**Example:** g.fillOval(10,10,50,50);

**iv) Draw Arc():** It is used to draw arc.
**Syntax:**
void drawArc(int x, int y, int w, int h, int start_angle, int sweep_angle);
where x, y starting point, w & h are width and height of arc, and start_angle is starting angle of arc, sweep_angle is degree around the arc
**Example:** g.drawArc(10, 10, 80, 40, 10, 90);

**3** **Attempt any four of the following:** **4x4=16**

**a) Differentiate between array and Vector.** **4**

**Ans:**

| Array | Vector |
|---|---|
| An Array is a structure that holds multiple values of same data type. | The Vector is similar to array which holds multiple values but different data types. |
| Array is a fixed-Length structure. | The size of a vector can grow or shrink as per the requirement. |
| Array is a data structure. | Vector is a Class. |
| Array is primitive data type. | Vector implements the List interface. |
| Array is a static-memory allocation. | Vector is a dynamic-memory allocation. |
| No methods provided by Array for doing operations on values. | Vector provides methods for doing operations like adding, deleting, inserting, etc… |
| Wrapper classes are not used. | Wrapper classes are used in vector. |
| Declaration of Array:<br>int a[]=new int[10]; | Declaration of vector:<br>Vector v=new Vector(4,5); |

**any 4 Correct Points. 1 Mark for one point)**

**b) Write a program to calculating area and perimeter of rectangle.** **4**

**Ans:**

```
import  java.io.*;
import  java.lang.*;
class  rect
{
public static void main(String args[]) throws Exception
```

**4m for Correct program and logic**

```
  {
DataInputStream dr=new DataInputStream(System.in);
int l,w;
System.out.println("Enter length and width:");
l=Integer.parseInt(dr.readLine());
w=Integer.parseInt(dr.readLine());
int a=l*w;
System.out.println("Area is="+a);
int p=2(l+w);
System.out.println("Area is="+p);
}
}
```

**c) Explain fileinputstream class to read the content of a file.** **4**

**Ans:** Java FileInputStream class obtains input bytes from a file. It is used for reading byte-oriented data (streams of raw bytes) such as image data, audio, video etc. You can also read character-stream data. But, for reading streams of characters, it is recommended to use File Reader class.

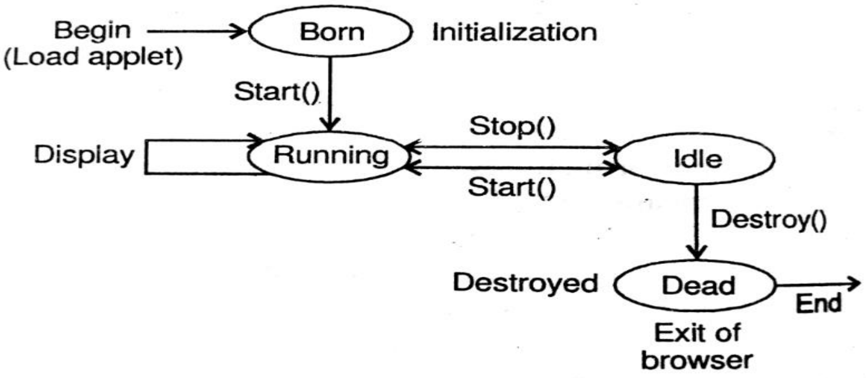
**4m for proper explanation**

Java FileInputStream class methods

| **Method** | **Description** |
|---|---|
| int available() | It is used to return the estimated number of bytes that can be read from the input stream. |
| int read() | It is used to read the byte of data from the input stream. |
| int read(byte[] b) | It is used to read up to b.length bytes of data from the input stream. |
| int read(byte[] b, int off, int len) | It is used to read up to len bytes of data from the input stream. |
| long skip(long x) | It is used to skip over and discards x bytes of data from the input stream. |
| FileChannel getChannel() | It is used to return the unique FileChannel object associated with the file input stream. |
| FileDescriptor getFD() | It is used to return the FileDescriptor object. |
| protected void finalize() | It is used to ensure that the close method is call when there is no more reference to the file input stream. |
| void close() | It is used to closes the stream. |

**Program for Reading contents from file:**

```
import  java.io.*;
public class f_demo
{
public static void main(String args[]) throws Exception
{
File f=new File("c:/input.txt");
DataInputStream dr=new DataInputStream(new FileInputStream(f));
while(dr.available() !=0)
```

| | | | |
|---|---|---|---|
| | | {<br>System.out.println(dr.readLine());<br>}<br>dr.close();<br>}<br>} | |
| | **d)** | **Explain applet life cycle with suitable diagram**. | **4** |
| | **Ans:** | Java *applet* inherits features from the class *Applet.* Thus, whenever an *applet* is created, it undergoes a series of changes from initialization to destruction. Various stages of an *applet* life cycle are depicted in the figure below:<br><br> Begin<br>(Load applet)<br>Born<br>Initialization<br>Start()<br>Display<br>Running<br>Stop()<br>Start()<br>Idle<br>Destroy()<br>Destroyed<br>Dead<br>End<br>Exit of<br>browser<br>**Applet Life Cycle**<br>**Initial State:** When a new *applet* is born or created, it is activated by calling *init()* method. At this stage, new objects to the *applet* are created, initial values are set, images are loaded and the colors of the images are set. An *applet* is initialized only once in its lifetime.<br>It's general form is:<br>public void init( )<br>//Action to be performed<br>}<br>**Running State**: An applet achieves the running state when the system calls the start() method. This occurs as soon as the applet is initialized. An applet may also start when it is in idle state. At that time, the start() method is overridden.<br>It's general form is:<br>public void start( )<br>{<br>//Action to be performed<br>}<br>**Idle State**: An applet comes in idle state when its execution has been stopped either implicitly or explicitly. An applet is implicitly stopped when we leave the page containing the currently running applet. An applet is explicitly stopped when we call stop() method to stop its execution.<br>It's general form is:<br>public void stope<br>{<br>//Action to be performed<br>}<br>**Dead State**: An applet is in dead state when it has been removed from the | **1m for diagram and 3 marks for explanation** |

| | | | |
|---|---|---|---|
| | | memory. This can be done by using destroy() method.<br><br>It's general form is:<br>public void destroy( )<br>{<br>//Action to be performed<br>}<br>**Display State (paint ()):** Apart from the above stages, Java applet also possess *paint( )* method. The paint() method is used for applet display on the screen. This method helps in drawing, writing and creating colored backgrounds of the applet. It takes an argument of the graphics class. To use The graphics, it imports the package java.awt.Graphics | |
| | **e)** | **What is use of super and final with respect to inheritance.** | **4** |
| | **Ans:** | **Super Keyword:** The **super** keyword in Java is a reference variable which is used to refer immediate parent class object. Whenever you create the instance of subclass, an instance of parent class is created implicitly which is referred by super reference variable.<br>**Usage of Java super Keyword**<br>• super can be used to refer immediate parent class instance variable.<br>• super can be used to invoke immediate parent class method.<br>• super() can be used to invoke immediate parent class constructor.<br>**Example:**<br>class A<br>{<br>int i;<br>A(int a, iont b)<br>{<br>i=a+b;<br>}<br>void add()<br>{<br>System.out.println("sum of a and b="+i);<br>}<br>}<br>class B extends A<br>{<br>int j;<br>B(int a,int b, int c)<br>{<br>super(a,b);<br>j=a+b+c;<br>}<br>void add()<br>{<br>super.add();<br>System.out.println("Sum of a, b and c is:" +j);<br>}<br>} | **2m for Use of super And 2m for Final keyword** |

**Final Keyword:** A parameter to a function can be declared with the keyword "final".
This indicates that the parameter cannot be modified in the function.
The final keyword can allow you to make a variable constant.

**Example:**
**Final Variable:** The final variable can be assigned only once. The value of a final variable can never be changed.
final float PI=3.14;

**Final Methods:** A final method cannot be overridden. Which means even though a sub class can call the final method of parent class without any issues but it cannot override it.
**Example:**

```
class XYZ
{
final void demo(){
System.out.println("XYZ Class Method");
}
}

class ABC extends XYZ{
void demo()
{
System.out.println("ABC Class Method");
}

public static void main(String args[])
{
ABC obj= new ABC();
obj.demo();
}
}
```

The above program would throw a compilation error, however we can use the parent class final method in sub class without any issues.

```
class XYZ
{
final void demo(){
System.out.println("XYZ Class Method");
}
}

class ABC extends XYZ
{
public static void main(String args[])
{
ABC obj= new ABC();
obj.demo();
}
}
```

**Output:**

```
XYZ Class Method
```

**final class:** We cannot extend a final class. Consider the below example:

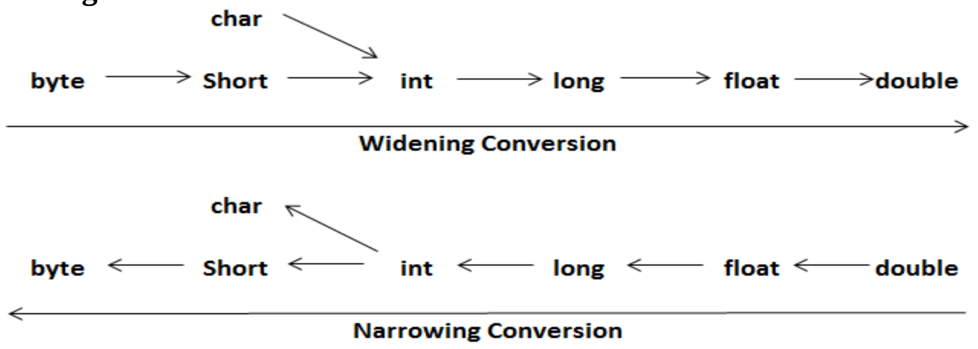
```
final class XYZ
{
}

class ABC extends XYZ
{
void demo()
{
System.out.println("My Method");
}
public static void main(String args[])
{
ABC obj= new ABC();
obj.demo();
}
}
```

**Output:**

```
The type ABC cannot subclass the final class XYZ
```

| | | | |
|---|---|---|---|
| **4** | **A** | **Attempt any THREE of following.** | **3x4=12** |
| | **a)** | **Explain type casting with suitable example.** | **4** |
| | **Ans:** | In Java, type conversion is performed automatically when the type of the expression on the right hand side of an assignment operation can be safely promoted to the type of the variable on the left hand side of the assignment. Assigning a value of one type to a variable of another type is known as **Type Casting.** | **4m for proper explanation** |



Every expression has a type that is determined by the components of the expression.

Example:

```
        double x;
        int y=2;
        float z=2.2f;
        x=y+z;
```

| | | | |
|---|---|---|---|
| | | The expression to the right of the “=“ operator is solved first and the result is stored in x. The int value is automatically promoted to the higher data type float (float has a larger range than int) and then, the expression is evaluated. The resulting expression is of float data type. This value is then assigned to x, which is a double (larger range than float) and therefore, the result is a double.<br>Java automatically promotes values to a higher data type, to prevent any loss of information.<br>**Example: int x=5.5/2**<br>In above expression the right evaluates to a decimal value and the expression on left is an integer and cannot hold a fraction. Compiler will give you following error.<br>*“Incompatible type for declaration, Explicit vast needed to convert double to int.”*<br>This is because data can be lost when it is converted from a higher data type to a lower data type. The compiler requires that you typecast the assignment.<br>Solution: int x=(int)5.5/2; | |
| | **b)** | **Explain following clause w.r.t. exception handling**<br>**i) try ii) catch iii) throw iv) finally** | **4** |
| | **Ans:** | **try:** Program statements that you want to monitor for exceptions are contained within a try block. If an exception occurs within the **try** block, it is thrown.<br>**Syntax:** try<br>{<br>// block of code to monitor for errors<br>}<br><br>**catch:** Your code can catch this exception (using **catch**) and handle it in some rational manner. System-generated exceptions are automatically thrown by the Java runtime system. A catch block immediately follows the try block. The catch block can have one or more statements that are necessary to process the exception.<br>**Syntax:** catch (*ExceptionType1 exOb*)<br>{<br>// exception handler for *ExceptionType1*<br>}<br><br>**throw:** It is mainly used to throw an instance of user defined exception.<br>***Example:***<br>throw new myException(“Invalid number”);<br>assuming myException as a user defined exception<br><br>**finally:** finally block is a block that is used to execute important code such as closing connection, stream etc. Java finally block is always executed whether exception is handled or not. Java finally block follows try or catch block.<br>***Syntax:***<br>finally<br>{<br>// block of code to be executed before try block ends | **1m for each term** |

```
}
```

**c)** **Write a program to print sum of even numbers from 1 to 20.** **4**

**Ans:**

```
public class sum_of_even
 {
   public static void main(String[] args)
 {
      int sum=0;
     for (int i=0; i<=20;i=i+2)
     {
      sum = sum+i;
     }
       System.out.println("Sum of these numbers: "+sum);
}
}
```

**Output:**

```
Sum of even numbers: 110
```

**4m for Correct program and logic**

**d)** **Differentiate between Applet and Application.** **4**

**Ans:**

| Applet | Application |
|---|---|
| Applets run in web pages. | Applications run on stand-alone systems. |
| Applets are not full featured application programs. | Applications are full featured programs. |
| Applets are the small programs. | Applications are larger programs. |
| Applet starts execution with its init(). | Application starts execution with its main (). |
| Parameters to the applet are given in the HTML file. | Parameters to the application are given at the command prompt. |
| Applet cannot access the local file system and resources. | Application can access the local file system and resources. |
| Applets are event driven | Applications are control driven. |

**any 4 Correct Points. 1 Mark for one point)**

**4** **B** **Attempt any ONE of the following.** **1x6=6**

**a)** **Write a program to create an applet for displaying circle, rectangle and triangle one below the other and filled them with red, green and yellow respectively.** **6**

```
import java.awt.*;
import java.applet.*;
/* <applet code="test.class" width=200 height=200>
</applet> */

public class test extends Applet
{
public void paint(Graphics g)
{

g.setColor(Color.RED);
g.fillOval(80,50,50,50);
```

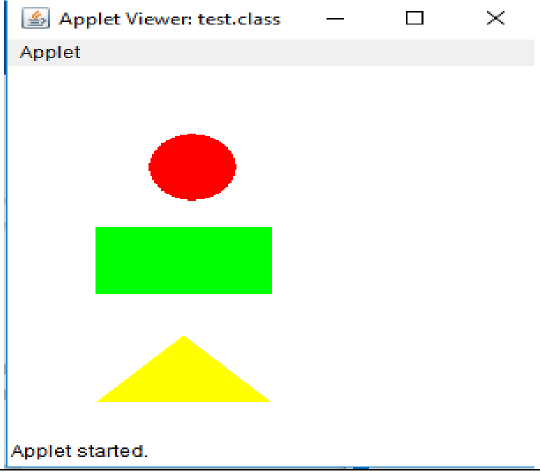
**To display circle with red color=2m**
**To display rectangle with green color=2m**
**To display triangle with yellow color=2m**

```
g.setColor(Color.GREEN);
g.fillRect(50,120,100,50);


g.setColor(Color.YELLOW);
int x1[]={50, 100, 150, 50};
int y1[]={250, 200, 250, 250};
int n1=4;
g.fillPolygon(x1, y1, n1);

}
}
```

Output:



| | | | |
|---|---|---|---|
| | **b)** | **Describe following methods related to vector addElement(), removeElement() and insertElementAt().** | **6** |
| | **Ans:** | **addElement():** Adds the specified component to the end of the vector, increasing its size by one.<br>Syntax: **void addElement(Object obj)**<br>Example:<br>v.addElement("Apple");<br>v.addElement(10);<br><br>**removeElement():**Removes the specified component from the vector. Decreasing the size of vector.<br>Syntax: **removeElement(Object obj)**<br>Example: v.removeElement("Apple");<br><br>**insertElementAt():**Inserts the item at nth position.<br>Syntax: **insertElementAt(item,n)**<br>Example: v.insertElementAt("Orange",3); | **2m for each method** |
| **5** | | **Attempt any two of the following:** | **2x8=16** |
| | **a)** | **Explain following terms: i) Thread priority ii) Types of Exception** | **8** |
| | **Ans:** | **(i) Thread Priority:** In java, each thread is assigned a priority which affects the order in which it is scheduled for running. Threads of same priority are | **Explanation of thread** |

| | | | |
|---|---|---|---|
| | | given equal treatment by the java scheduler. The Thread class defines several priority constants as<br>MIN_PRIORITY=1<br>NORM_PRIORITY=5<br>MAX_PRIORITY=10<br>Thread priorities can take values from 1 to 10.<br>To set the priority Thread class provides setPriority() method as<br>Thread.setPriority(priority value);<br>Eg: If t1 is a Thread class object then<br>T1.setPriority(Thread.MAX_PRIORITY);<br>To see the priority value of a thread the method available is getPrioriy() as<br>int Thread.getPriority();<br>eg : int p= Thread.getPriority();<br><br>**(ii) Types of Exceptions:** Java exceptions can be classified as Built-in exceptions and user defined exceptions<br>**1) Built-in Exceptions:** Built-in exceptions are the exceptions which are available in Java libraries. These exceptions are suitable to explain certain error situations. Below is the list of important built-in exceptions in Java.<br>**Arithmetic Exception:** It is thrown when an exceptional condition has occurred in an arithmetic operation.<br>**ArrayIndexOutOfBoundException:** It is thrown to indicate that an array has been accessed with an illegal index. The index is either negative or greater than or equal to the size of the array.<br>**ClassNotFoundException:** This Exception is raised when we try to access a class whose definition is not found<br>**FileNotFoundException:** This Exception is raised when a file is not accessible or does not open.<br>**IOException:** It is thrown when an input-output operation failed or interrupted<br><br>**2) User-Defined Exceptions:** Sometimes, the built-in exceptions in Java are not able to describe a certain situation. In such cases, user can also create exceptions which are called 'user-defined Exceptions'.<br>Following steps are followed for the creation of user-defined Exception.<br>The user should create an exception class as a subclass of Exception class. Since all the exceptions are subclasses of Exception class, the user should also make his class a subclass of it. This is done as:<br>**class MyException extends Exception**<br>We can create a parameterized constructor with a string as a parameter.We can use this to store exception details.<br>MyException(String str)<br>{<br>super(str);<br>}<br>To raise exception of user-defined type, we need to create an object to his exception class and throw it using throw clause, as<br>**throw new MyException("Error message here….");** | **priority=4m Explanation of Types of Exception= 4m** |
| | **b)** | **Write a program to create two threads.so one thread will print even** | **8** |

**numbers between 1 to 10 whereas other will print odd number between 11 to 20.**

**Ans:**

```
class eventhread extends Thread
{
    public void run()
          {
          for(int i=1;i<=10;i=i+2)
          {
            System.out.println("Even no="+ i);
          }
          }
}
class oddthread  extends Thread
{
    public void run()
          {
          for(int i=11;i<=20;i=i+2)
          {
            System.out.println("Odd no="+ i);
          }
          }
}
class threadtest
{
          public static void main(String args[])
          {
          eventhread e1= new eventhread();
          oddthread o1= new oddthread();
          e1.start();
          o1.start();
          }
}
```

**Output:**

```
E:\java\bin>javac threadtest.java
E:\java\bin>java threadtest
Odd no=11
Even no=2
Odd no=13
Odd no=15
Even no=4
Even no=6
Odd no=17
Odd no=19
Even no=8
Even no=10
```

***Correct Logic : 4 M, Correct syntaxes : 4 M***

**c) How can parameters be passed to an applet? Write an applet to accept username in the parameter and print "Hello<username> ".** **8**

**Ans:** User defined parameters can be passed using <Param> tag.
Parameters are passed to an applet when it is loaded. We can call

***Parameter passing***

getParameter() method to collect the value of parameter sent from <param> tag to applet. getParameter() takes one String argument representing name of the parameter and returns the value of the same.

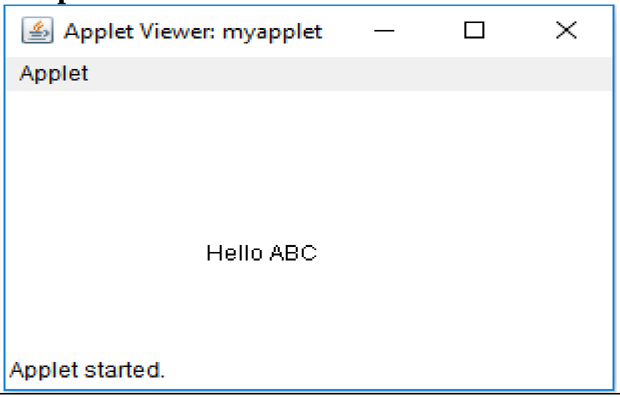
**Syntax:**

```
<Param name=”variable-name” value=”value of the variable>
String getParameter(“variable-name”);
```

**Program:**

```
import java.awt.*;
import java.applet.*;
public class myapplet extends Applet
{
        String uname="";
        public void paint(Graphics g)
        {
        uname=getParameter("username");
        g.drawString("Hello "+uname,100,100);
        }
}
/*<applet code=myapplet height=300 width=300>
<param name="username" value="ABC">
</applet>*/
```

**Output:**



```
Hello ABC
```

Applet started.

*explanation : 2M, In program : correct logic 3M, correct syntaxes : 3M*

| 6 | | **Attempt any four of the following** | **4x4=16** |
|---|---|---|---|
| | **a)** | **Demonstrate the concept of method overriding with example.** | **4** |

**Ans:**

**Method overriding:** In a class hierarchy, when method in a subclass has same name, type, & parameter list as a method in superclass then the method is said to override the method in superclass. When an overridden method is called from within a subclass it will always refer to the version of method defined by subclass. The version of method from superclass will be hidden.

**Example:**

```
class overridetest
{
int x,y;
overridetest(int a,int b)
{
x=a;
y=b;
}
```

***Explanation of method overriding: 2M, correct example: 2M***

```
void display()
{
System.out.println("x="+x);
System.out.println("y="+y);
}
}
class test extends overridetest
{
int z;
test(int a,int b,int c)
{
super(a,b);
z=c;
}
void display()                    //method overridden
{
super.display();                //call to super class display()
System.out.println("z="+z);
}
public static void main(String args[])
{
test t= new test(4,5,6);
t.display();
}
}
```

| | | | |
|---|---|---|---|
| | **b)** | **Write any two methods of file and file input stream class each.** | **4** |
| | **Ans:** | **File class methods:**<br>**boolean canExecute() :** Tests whether the application can execute the file denoted by this abstract pathname.<br>**boolean canRead()** : Tests whether the application can read the file denoted by this abstract pathname.<br>**boolean canWrite() :** Tests whether the application can modify the file denoted by this abstract pathname.<br>**int compareTo(File pathname) :** Compares two abstract pathnames lexicographically.<br>**boolean createNewFile() :** Atomically creates a new, empty file named by this abstract pathname .<br>**static File createTempFile(String prefix, String suffix) :** Creates an empty file in the default temporary-file directory.<br>**boolean delete() :** Deletes the file or directory denoted by this abstract pathname.<br>**boolean equals(Object obj) :** Tests this abstract pathname for equality with the given object.<br>**boolean exists()** : Tests whether the file or directory denoted by this abstract pathname exists.<br>**String getAbsolutePath() :** Returns the absolute pathname string of this abstract pathname.<br>**long getFreeSpace() :** Returns the number of unallocated bytes in the partition | **Any two methods from file class=2m**<br>**Any two methods from fileinput stream=2m** |

**String getName() :** Returns the name of the file or directory denoted by this abstract pathname.

**String getParent() :** Returns the pathname string of this abstract pathname's parent.

**File getParentFile() :** Returns the abstract pathname of this abstract pathname's parent.

**String getPath() :** Converts this abstract pathname into a pathname string.

**boolean isDirectory() :** Tests whether the file denoted by this pathname is a directory.

**boolean isFile() :** Tests whether the file denoted by this abstract pathname is a normal file.

**boolean isHidden() :** Tests whether the file named by this abstract pathname is a hidden file.

**long length() :** Returns the length of the file denoted by this abstract pathname.

**String[] list() :** Returns an array of strings naming the files and directories in the directory .

**File[] listFiles() :** Returns an array of abstract pathnames denoting the files in the directory.

**boolean mkdir() :** Creates the directory named by this abstract pathname.

**boolean renameTo(File dest) :** Renames the file denoted by this abstract pathname.

**boolean setExecutable(boolean executable) :** A convenience method to set the owner's execute permission.

**boolean setReadable(boolean readable) :** A convenience method to set the owner's read permission.

**boolean setReadable(boolean readable, boolean ownerOnly) :** Sets the owner's or everybody's read permission.

**boolean setReadOnly() :** Marks the file or directory named so that only read operations are allowed.

**boolean setWritable(boolean writable)** : A convenience method to set the owner's write permission.

**String toString() :** Returns the pathname string of this abstract pathname.

**URI toURI() :** Constructs a file URI that represents this abstract pathname.

**FileInputStream class methods :**

**int available()**: It is used to return the estimated number of bytes that can be read from the input stream.

**int read():**It is used to read the byte of data from the input stream.

**int read(byte[] b):** It is used to read up to b.length bytes of data from the input stream.

**int read(byte[] b, int off, int len):** It is used to read up to len bytes of data from the inpu stream.

**long skip(long x) :**It is used to skip over and discards x bytes of data from the input stream.

**FileChannel getChannel()** : It is used to return the unique FileChannel object associated with the file input stream.

**FileDescriptor getFD() :** It is used to return the FileDescriptor object.

**protected void finalize()** : It is used to ensure that the close method is call

when there is no more reference to the file input stream.
**void close():** It is used to closes the stream**.**

**c) Design an applet which displays rectangle filled with blue colour and display message as "MSBTE EXAM" in red colour below it.** **4**

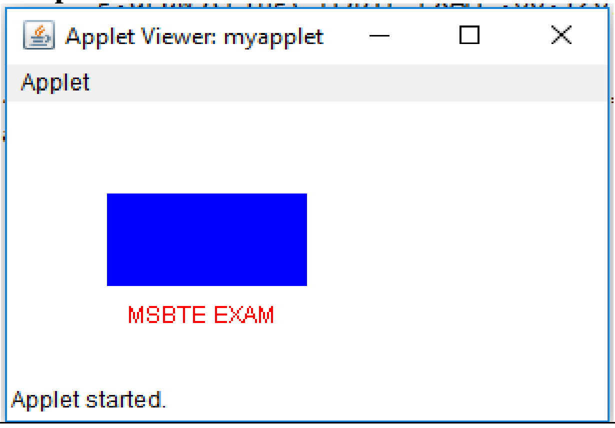
**Ans:**

```
import java.awt.*;
import java.applet.*;
public class myapplet extends Applet
{
public void paint(Graphics g)
{
g.setColor(Color.blue);
g.fillRect(50,50,100,50);
g.setColor(Color.red);
g.drawString("MSBTE EXAM",60,120);
}
}

/*<applet code=myapplet height=300 width=300>
</applet>*/
```

***Correct logic: 2M, Correct syntaxes: 2M***

**Output:**



```
MSBTE EXAM
Applet started.
```

**d) Write a program to demonstrate multiple inheritances.** **4**

**Ans:**

```
interface sports
{
  int sports_weightage=5;
  void calc_total();
}

class person
{
String name;
String category;
person(String nm,String c)
{
name=nm;
category=c;
}
```

***Program logic showing multiple inheritance : 2M, correct syntaxes : 2M***

```
}
class student extends person implements sports
{
int marks1,marks2;
student(String n,String c, int m1,int m2)
{
super(n,c);
marks1=m1;
marks2=m2;
}
public void calc_total()
{
int total;
if (category.equals("sportsman"))
total=marks1+marks2+sports_weightage;
else
total=marks1+marks2;
System.out.println("Name="+name);
System.out.println("Category ="+category);
System.out.println("Marks1="+marks1);
System.out.println("Marks2="+marks2);
System.out.println("Total="+total);
}
public static void main(String args[])
{
student s1=new student("ABC","sportsman",67,78);
student s2= new student("PQR","non-sportsman",67,78);
s1.calc_total();
s2.calc_total();
}
}
```

**e)** **Write a program to find greater number among two numbers using conditional operator.** **4**

**Ans:**

```
class greater
{
public static void main(String args[])
{
int a,b;
int bignum;
a=100;
b=150;
bignum=(a>b?a:b);                    //conditional operator
System.out.println("Greater number between "+a+ " and "+b +" is =
"+bignum);
}
}
```

***correct logic 1 M, correct use of conditional operator :2M, other correct syntaxes : 1M***

**Output:**

```
E:\java\bin>javac greater.java
E:\java\bin>java greater
```

| | | | |
|---|---|---|---|
| | | Greater number between 100 and 150 is = 150 | |
| | | | |