MAHARASHTRA STATE BOARD OF TECHNICAL EDUCATION
(Autonomous)
(ISO/IEC - 27001 - 2005 Certified)
WINTER-15 EXAMINATION
Model Answer Paper

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

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 judgment 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.

Marks

1. (A) Attempt any THREE of the following: **3×4 = 12**

(a) Explain any four features of Java.
(Any four features - 1 Mark each)

Ans:

1. **Compile & Interpreted**: Java is a two staged system. It combines both approaches. First java compiler translates source code into byte code instruction. Byte codes are not machine instructions. In the second stage java interpreter generates machine code that can be directly executed by machine. Thus java is both compile and interpreted language.

2. **Platform independent and portable**: Java programs are portable i.e. it can be easily moved from one computer system to another. Changes in OS, Processor, system resources won't force any change in java programs. Java compiler generates byte code instructions that can be implemented on any machine as well as the size of primitive data type is machine independent.

3. **Object Oriented**: Almost everything in java is in the form of object. All program codes and data reside within objects and classes. Similar to other OOP languages java also has basic OOP properties such as encapsulation, polymorphism, data abstraction, inheritance etc. Java comes with an extensive set of classes (default) in packages.

4. **Robust & Secure**: Java is a robust in the sense that it provides many safeguards to ensure reliable codes. Java incorporates concept of exception handling which captures errors and eliminates any risk of crashing the system. Java system not only verify all memory access but also ensure that no viruses are communicated with an applet. It does not use pointers by which you can gain access to memory locations without proper authorization.

5. **Distributed:** It is designed as a distributed language for creating applications on network. It has ability to share both data and program. Java application can open and access remote object on internet as easily as they can do in local system.

6. **Multithreaded:** It can handle multiple tasks simultaneously. Java makes this possible with the feature of multithreading. This means that we need not wait for the application to finish one task before beginning other.

7. **Dynamic and Extensible**: Java is capable of dynamically linking new class library's method and object. Java program supports function written in other languages such as C, C++ which are called as native methods. Native methods are linked dynamically at run time.

(b) What is exception? How it is handled? Explain with suitable example.
(Definition – 1 Mark, Listing of keywords – 1 Mark, Example – 2 Marks)

Ans:

Exception: An exception is an event, which occurs during the execution of a program, that stop the flow of the program's instructions and takes appropriate actions if handled. .i.e. It is erroneous situation encounter during course of execution of program.
Exception handling in java is done by 5 keywords as:
1) try 2) catch 3) finally 4) throw 5) throws
Example:

```
class DemoException
{
```

```
public static void main(String args[])
 {
   try
     {
          int b=8;
           int c=b/0;
         System.out.println("answer="+c);
       }
     catch(ArithmeticException e)
     {
      System.out.println("Division by Zero");
           }
          }
         }
```

(c) Describe break and continue statement with example.

(Each explanation - 1 Mark, Example - 1 Mark)

[**Note: any other relevant example can be considered]

Ans:

Break:

The break keyword is used to stop the entire loop. The break keyword must be used inside any loop or a switch statement.

The break keyword will stop the execution of the innermost loop and start executing the next line of code after the block.

Example:

```
public class TestBreak
{
public static void main(String args[])
   {
         int [] numbers = {10, 20, 30, 40, 50};
         for(int x : numbers )
         {
          if( x == 30 )
             {
                  break;
              }
            System.out.println( "value of x-"+x );
```

```
            }
          }
  }
```

continue:

The continue statement skips the current iteration of a for, while , or do-while loop. The unlabeled form skips to the end of the innermost loop's body and evaluates the boolean expression that controls the loop.

A labeled continue statement skips the current iteration of an outer loop marked with the given label.

Example:

```
public class TestContinue
{
public static void main(String args[])
{
int [] numbers = {10, 20, 30, 40, 50};
for(int x : numbers )
 {
      if( x == 30 )
         {
          continue;
           }
           System.out.print( x );
          System.out.print("\n");
     }
   }
}
```

(d) What are streams? Write any two methods of character stream classes.

(Definition of Streamclass - 2 Marks, two methods of character stream class - 2 Marks)

Ans:

Java programs perform I/O through streams. A stream is an abstraction that either produces or consumes information (i.e it takes the input or gives the output). A stream is linked to a physical device by the Java I/O system. All streams behave in the same manner, even if the actual physical devices to which they are linked differ. Thus, the same I/O classes and methods can be applied to any type of device.

Java 2 defines two types of streams: byte and character. Byte streams provide a convenient means for handling input and output of bytes. Byte streams are used, for example, when reading or writing binary data. Character streams provide a convenient means for handling input and output of characters. They use Unicode and, therefore, can be internationalized. Also, in some cases, character streams are more efficient than byte streams.

The Character Stream Classes

Character streams are defined by using two class hierarchies. At the top are two abstract classes, **Reader** and **Writer**. These abstract classes handle Unicode character streams. Java has several concrete subclasses of each of these.

Methods of Reader Class

1) **void mark(int *numChars*) :** Places a mark at the current point in the input stream that will remain valid until numChars characters are read.

2) **boolean markSupported( ) :** Returns **true** if **mark( )** / **reset( )** are supported on this stream.
3) **int read( ) :**Returns an integer representation of the next available character from the invoking input stream. –1 is returned when the end of the file is encountered.

4) **int read(char *buffer*[ ]) :** Attempts to read up to buffer. Length characters into buffer and returns the actual number of characters that were successfully read. –1 is returned when the end of the file is encountered.

5) **abstract int read(char *buffer*[ ],int *offset*,int *numChars*):** Attempts to read up to numChars characters into buffer starting at buffer[offset], returning the number of characters successfully read.–1 is returned when the end of the file is encountered.

6) **boolean ready( ):** Returns **true** if the next input request will not wait. Otherwise, it returns **false**.

7) **void reset( ):** Resets the input pointer to the previously set mark.

8) **long skip(long *numChars*)** :- Skips over *numChars* characters of input, returning the number of characters actually skipped.

9) **abstract void close( )** :- Closes the input source. Further read attempts will generate an IOException

[Note: any two methods from above list to be considered]**

Writer Class
Writer is an abstract class that defines streaming character output. All of the methods in this class return a **void** value and throw an **IOException** in the case of error

Methods of Writer class are listed below: -

1) abstract void close( ) : Closes the output stream. Further write attempts will generate an **IOException**.

2) abstract void flush( ) : Finalizes the output state so that any buffers are cleared. That is, it flushes the output buffers.

3) void write(int*ch*): Writes a single character to the invoking output stream. Note that the parameter is an **int**, which allows you to call **write** with expressions without having to cast them back to **char**.

4) void write(char *buffer*[ ]): Writes a complete array of characters to the invoking output stream

5) abstract void write(char *buffer*[ ],int *offset,* int *numChars*) :- Writes a subrange of *numChars* characters from the array *buffer*, beginning at *buffer*[*offset*] to the invoking output stream.

6) void write(String *str*): Writes *str* to the invoking output stream.

7) void write(String *str,* int *offset,*int *numChars*): Writes a sub range of *numChars* characters from the array *str*, beginning at the specified *offset*.

[Note: any two methods from above list to be considered]**

(B) Attempt any ONE of following: 1×6 = 6

(a) What is package? How to create package? Explain with suitable example.
(Definition of package - 1 Mark, Package creation - 2 Marks, Example - 3 Marks)

[Note: Any relevant example can be considered]**

Ans:

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.

The syntax for creating package is:

package *pkg*;

Here, *pkg* is the name of the package

eg : package mypack;

Packages are mirrored by directories. Java uses file system directories to store packages. The class files of any classes which are declared in a package must be stored in a directory which has same name as package name. The directory must match with the package name exactly. A hierarchy can be created by separating package name and sub package name by a period(.) as pkg1.pkg2.pkg3; which requires a directory structure as pkg1\pkg2\pkg3.

The classes and methods of a package must be public.

Syntax:

To access package In a Java source file, **import** statements occur immediately following the **package** statement (if it exists) and before any class definitions.

Syntax:

import *pkg1*[.*pkg2*].(*classname*|*);

Example:

package1:

```
package package1;
public class Box
{
int l= 5;
int b = 7;
int h = 8;
    public void display()
    {
        System.out.println("Volume is:"+(l*b*h));
        }
    }
}
```

S***ource file:***

```
import package1.Box;
class   VolumeDemo
{
    public static void main(String args[])
    {
Box b=new Box();
b.display();
    }
}
```

(b) State the use of 'super' and 'final' keyword w.r.t inheritance with example.
(super with example - 3 Marks , final with example - 3 Marks)
[Note any relevant example can be considered]**

Ans:

when you will want to create a superclass that keeps the details of its implementation to itself (that is, that keeps its data members private). In this case, there would be no way for a subclass to directly access or initialize these variables on its own. Whenever a subclass needs to refer to its immediate super class, it can do so by use of the keyword **super**. As constructer can not be inherited, but derived class can called base class constructer using super ()

super has two general forms.

The first calls the super class constructor. (super() method)

The second is used to access a member of the super class that has been hidden by a member of a subclass.

Using super () to Call Super class Constructors

A subclass can call a constructor method defined by its super class by use of the following form of **super**:

super(*parameter-list*);

Here, *parameter-list* specifies any parameters needed by the constructor in the super class. **super( )** must always be the first statement executed inside a subclass' constructor.

A Second Use for super

The second form of **super** acts somewhat like **this**, except that it always refers to the super class of the subclass in which it is used. This usage has the following general form:

super.member

Here, member can be either a method or an instance variable. This second form of **super** is most applicable to situations in which member names of a subclass hide members by the same name in the super class.

Example:

```
// Using super to overcome name hiding.
class A
   {
    int i;
}
// Create a subclass by extending class A.
   class B extends A
   {
      int i; // this i hides the i in A
     B(int a, int b)
     {
        super.i = a; // i in A
       i = b; // i in B
      }
    void show()
     {
       System.out.println("i in superclass: " + super.i);
      System.out.println("i in subclass: " + i);
      }
}
class  UseSuper
{
public static void main(String args[])
{
   B subOb = new B(1, 2);
   subOb.show();
 }
}
```

Final keywords

The keyword **final** has three uses. First, it can be used to create the equivalent of a named constant.(in interface or class we use final as shared constant or constant.)
other two uses of **final** apply to inheritance

Using final to Prevent Overriding

While method overriding is one of Java's most powerful features, there will be times

When you will want to prevent it from occurring. To disallow a method from being overridden, specify **final** as a modifier at the start of its declaration. Methods declared as **final** cannot be overridden. The following fragment illustrates **final**:

```
class A
 {
   final void meth()
  {
   System.out.println("This is a final method.");
    }
}
class B extends A
 {
void meth()
{
// ERROR! Can't override.
    System.out.println("Illegal!");
  }
}
```

As base class declared method as a final , derived class can not override the definition of base class methods.

2. Attempt any TWO of the following : 2×8 = 16

(a) Write a program to create a vector with seven elements as (10, 30, 50, 20, 40, 10, 20). Remove element at 3rd and 4th position. Insert new element at 3rd position. Display the original and current size of vector.
(Vector creation with elements – 2 Marks, Remove elements – 1 Mark, Insert new element – 1 Mark, Show original size – 1 Mark, show current size – 1 Mark)

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(20));
                v.addElement(new Integer(30));
                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());
        }
}
```

(b) What is meant by an interface? State its need and write syntax and features of an interface. Give one example.

(Definition - 1 Mark syntax - 1 Mark, Any two features - 2 Marks, Any two needs - 2 Marks, Example - 2 Marks)

Ans:

Defining an Interface:

Interface is also known as kind of a class. So interface also contains methods and variables but with major difference the interface consist of only abstract method (i.e.methods are not defined,these are declared only) and final fields(shared constants). This means that interface do not specify any code to implement those methods and data fields contains only constants. Therefore, it is the responsibility of the class that implements an interface to define the code for implementation of these methods. An interface is defined much like class.

Syntax:

```
access interface InterfaceName
{
        return_type method_name1(parameter list);
        ….
        return_type  method_nameN(parameter list);
        type final-variable 1 = value1;
        ….
```

```
        type final-variable N = value2;
    }
```

Features:

1. Variable of an interface are explicitly declared final and static (as constant) meaning that the implementing the class cannot change them they must be initialize with a constant value all the variable are implicitly public of the interface, itself, is declared as a public
2. Method declaration contains only a list of methods without anybody statement and ends with a semicolon the method are, essentially, abstract methods there can be default implementation of any method specified within an interface each class that include an interface must implement all of the method

Need:

1. To achieve multiple Inheritance.
2. We can implement more than one Interface in the one class.
3. Methods can be implemented by one or more class.

Example:

```
interface sports
{
        int  sport_wt=5;
        public void disp();
}
class Test
{
        int  roll_no;
        String name;
        int m1,m2;
        Test(int r, String nm, int m11,int m12)
        {
        roll_no=r;
        name=nm;
        m1=m11;
        m2=m12;
        }
}
class Result extends Test implements sports
{
Result (int r, String nm, int m11,int m12)
{
        super (r,nm,m11,m12);
```

```
}
public void disp()
{
System.out.println("Roll no : "+roll_no);
System.out.println("Name : "+name);
System.out.println("sub1 : "+m1);
System.out.println("sub2 : "+m2);
System.out.println("sport_wt : "+sport_wt);
        int t=m1+m2+sport_wt;
        System.out.println("total : "+t);
}
public static void main(String args[])
{
        Result r= new Result(101,"abc",75,75);
        r.disp();
}
}
```

(c) Write syntax and example of following Graphics class methods:

(i) drawOval( )

(ii) drawPolygon( )

(iii)drawArc( )

(iv)drawRect( )

(Each method syntax – 1 Mark, Example – 1 Mark)

[Note: common program using above all methods can be considered]**

Ans:

(i) drawOval( )

Drawing Ellipses and circles:

To draw an Ellipses or circles used drawOval() method can be used.

Syntax: void drawOval(int top, int left, int width, int height)

The 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 a circle or filled circle, specify the same width and height the following program draws several ellipses and circle.

Example:

g.drawOval(10,10,50,50);

(ii) drawPolygon

drawPolygon() method is used to draw arbitrarily shaped figures.

Syntax: void drawPolygon(int x[], int y[], int numPoints)
The polygon's end points are specified by the co-ordinates pairs contained within the x and y arrays. The number of points define by x and y is specified by numPoints.
Example:

```
int xpoints[]={30,200,30,200,30};
int ypoints[]={30,30,200,200,30};
int num=5;
g.drawPolygon(xpoints,ypoints,num);
```

(iii)drawArc( )
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, 30, 40, 40, 90);
```

(iv)drawRect()
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);
```

3. Attempt any FOUR of the following: 4×4 = 16

(a) What is constructor? Describe the use of parameterized constructor with suitable example.
(Constructor – 1 Mark, use of parameterized constructor – 1 Mark, example of parameterized constructor – 2 Marks)

[Note: any relevant example can be considered]**

Ans:

Constructor:

- A constructor is a special method which initializes an object immediately upon creation.
- It has the same name as class name in which it resides and it is syntactically similar to any method.
- When a constructor is not defined, java executes a default constructor which initializes all numeric members to zero and other types to null or spaces.
- Once defined, constructor is automatically called immediately after the object is created before new operator completes.
- Constructors do not have return value, but they don't require "void" as implicit data type as data type of class constructor is the class type itself.

Parameterized constructor:

It is used to pass the values while creating the objects

Example:

```
class  Rect
 {
        int length, breadth;
        Rect(int l, int b) // parameterized constructor
         {
                length=l;
                breadth=b;
         }
public static void main(String args[])
{
        Rect r = new Rect(4,5); // constructor with parameters
        Rect r1 = new Rect(6,7);
        System.out.println("Area : " +(r.length*r.breadth));
        System.out.println("Area : " +(r1.length*r1.breadth));
 }
}
```

(b) Describe?,: (Ternary operator) in Java with suitable example.
(Explanation of Ternary operator – 2 Marks & Example – 2 Marks)

[Note: any relevant example can be considered]**

Ans:

The ternary operator?: is an operator that takes three arguments. The first argument is a comparison argument, the second is the result upon a true comparison, and the third is the result upon a false comparison. If it helps you can think of the operator as shortened way of writing an if-else statement. It is often used as a way to assign variables based on the result of an comparison. When used correctly it can help increase the readability and reduce the amount of lines in your code

Syntax:

expression1? expression2: expression3

Expression1 can be any expression that evaluates to a Boolean value.

If expression1 is true, then expression2 is evaluated; otherwise, expression3 is evaluated.

The result of the? Operation is that of the expression evaluated.

Both expression2 and expression3 are required to return the same type, which can't be void.

Example:

```
class Ternary
{
public static void main(String args[])
        {
        int i, k;
        i = 10;
        k = i < 0 ? -i : i; // get absolute value of i
        System.out.print("Absolute value of ");
        System.out.println(i + " is " + k);
        i = -10;
        k = i < 0 ? -i : i; // get absolute value of i
        System.out.print("Absolute value of ");
        System.out.println(i + " is " + k);
        }
}
```

(c) What is difference between array and vector? Explain elementAt( ) and addElement( ) method.
(Any 2 Points - 2 Marks, Each Method - 1 Mark)

Ans:

Array	Vector
Array can accommodate fixed number of elements	Vectors can accommodate unknown number of elements
Arrays can hold primitive data type & objects	Vectors can hold only objects.
All elements of array should be of the same data type. i.e. it can contain only homogeneous elements.	The objects in a vector need not have to be homogeneous.
Syntax : Datatype[] arrayname= new datatype[size];	Syntax: Vector objectname= new Vector();
For accessing elements of an array no special methods are available as it is not a class , but derived type.	Vector class provides different methods for accessing and managing Vector elements.

1) elementAt( ): Returns the element at the location specified by index.
Syntax: Object elementAt(int index)
Example:
Vector v = new Vector();
v.elementAt(2); //return 2nd element from vector

2) addElement ( ): Adds the specified component to the end of this vector, increasing its size by one.
Syntax: void addElement(Object element)
Example:
Vector v = new Vector();
v.addElement(new Integer(1)); //add integer object 1 to vector

(d) Write any two methods of File and FileInputStream class each.
(An y two methods of each class - 1 Mark)

Ans:

File Class Methods

1. String getName( ) - returns the name of the file.
2. String getParent( ) - returns the name of the parent directory.
3. boolean exists( ) - returns true if the file exists, false if it does not.
4. void deleteOnExit( ) -Removes the file associated with the invoking object when the Java Virtual Machine terminates.
5. boolean isHidden( )-Returns true if the invoking file is hidden. Returns false otherwise.

FileInputStream Class Methods:

1. **int available( )**- Returns the number of bytes of input currently available for reading.
2. **void close( )**- Closes the input source. Further read attempts will generate an IOException.
3. **void mark(int numBytes)** -Places a mark at the current point in the inputstream that will remain valid until numBytes bytes are read.
4. **boolean markSupported( )** -Returns true if mark( )/reset( ) are supported by the invoking stream.
5. **int read( )**- Returns an integer representation of the next available byte of input. –1 is returned when the end of the file is encountered.
6. **int read(byte buffer[ ])**- Attempts to read up to buffer.length bytes into buffer and returns the actual number of bytes that were successfully read. –1 is returned when the end of the file is encountered.

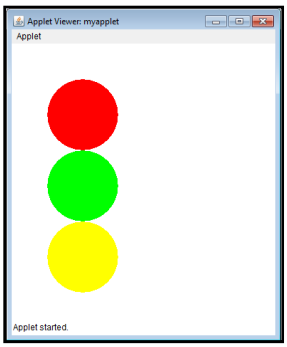
(e) Write a program to design an applet to display three circles filled with three different colors on screen.
(Correct logic – 2 Marks, Applet tag – 1 Mark, Package imported – 1 Mark)
[**Note: any other relevant logic can be considered, output not necessary]

Ans:

```
import java.awt.*;
import java.applet.*;
public class MyApplet extends Applet
```

```
 {
public void paint(Graphics g)
{
g.setColor(Color.red);
g.fillOval(50,50,100,100);
g.setColor(Color.green);
g.fillOval(50,150,100,100);
g.setColor(Color.yellow);
g.fillOval(50,250,100,100);
}
}
/*<applet code= MyApplet width= 300 height=300></applet>*/
```

Output:



4. (A) Attempt any THREE of the following: 3×4 = 12

(a) Write a program to check whether the entered number is prime or not.
(Accept No. from user - 1 Mark, Prime No. logic - 3 Marks)

Ans:

```
import java.io.*;
class  PrimeNo
{
public static void main(String args[]) throws IOException
 {
BufferedReader bin=new BufferedReader(new InputStreamReader(System.in));
        System.out.println("Enter number: ");
        int  num=Integer.parseInt(bin.readLine());
```

```
        int flag=0;
        for(int i=2;i<num;i++)
        {
                if(num%i==0)
                 {
                System.out.println(num + " is not a prime number");
                flag=1;
                break;
                }
         }
        if(flag==0)
                System.out.println(num + " is a prime number");
}
}
```

(b) Explain the following clause w.r.t. exception handling:

(i) try

(ii) catch

(iii) throw

(iv) finally

(Each keywords syntax with explanation - 1 Mark)

Ans:

i. **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.

Syntax:

```
try
{
// block of code to monitor for errors
 }
```

ii. **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 too can have one or more statements that are necessary to process the exception.

Syntax:

catch (*ExceptionType1 exOb*)

```
{
// exception handler for ExceptionType1
}
```

iii. throw: It is possible for a program to throw an exception explicitly, using the throw statement.
The general form of throw is :

throw new *ThrowableInstance*;
or
throw Throwableinstance;

throw statement explicitly throws an built-in /user- defined exception. When throw statement is executed, the flow of execution stops immediately after throw statement, and any subsequent statements are not executed.

iv. finally: It can be used to handle an exception which is not caught by any of the previous catch statements. finally block can be used to handle any statement generated by try block. It may be added immediately after try or after last catch block.

Syntax:

```
finally
{
   // block of code to be executed before try block ends
 }
```

(c) Explain any two bit-wise operators with example.
(Any 2 (Each Bitwise operator explanation - 1 Mark, Example - 1 Mark))

Ans:

1) Bitwise NOT (~): called bitwise complement, the unary NOT operator, inverts all of the bits of its operand.

e.g~ 0111 (decimal 7) = 1000 (decimal 8)

2) Bitwise AND (&):

the AND operator, &, produce a 1 bit if both operands are also 1, A zero is produced in all the cases.

e.g0101 (decimal 5) &0011 (decimal 3) = 0001 (decimal 1)

3) Bitwise OR (|) :

the OR operator, | , combines bits such that if either of the bits in the operand is a 1, then the resultant bit is a 1

e.g0101 (decimal 5) |0011 (decimal 3) = 0111 (decimal 7)

4) Bitwise XOR (^): the XOR operator, ^, combines bits such that if exactly one operand is 1, then the result is 1. Otherwise result is zero.
e.g0101 (decimal 5) ^ 0011 (decimal 3) = 0110 (decimal 6)

5) The Left Shift (<<): the left shift operator, <<, shifts all of the bits in a value' to the left a specified number of times specified by num'

General form: value <<num

e.g**.** x << 2 (x=12)

0000 1100 << 2 = 0011 0000 (decimal 48)

6) The Right Shift (>>): the right shift operator, >>, shifts all of the bits in a value' to the right a specified number of times specified by num'

General form: value >>num.

e.g. x>> 2 (x=32)

0010 0000 >> 2 = 0000 1000 (decimal 8)

7) Unsigned Right Shift (>>>) : >>> always shifts zeros into high order bit.

e.g. int a= -1

a=a>>>24

11111111 11111111 11111111 11111111 (-1 in binary as int) >>> 24

00000000 00000000 00000000 11111111(255 in binary as an int)

8) Bitwise Operators Compound Assignments: All of the above binary bitwise operators have a compound form similar to that of the algebraic operators, which combines the assignment with the bitwise operation. These two statements are equivalent.

e.g.a = a >>4 ;

a >> = 4;

(d) Explain all attributes available in <applet>tag.
(Any 4 attributes - 1 Mark each)

Ans:

APPLET Tag: The APPLET tag is used to start an applet from both an HTML document and from an appletviewer will execute each APPLET tag that it finds in a separate window, while web browser will allow many applets on a single page the syntax for the standard APPLET tag is:

```
<APPLET
[CODEBASE=codebaseURL]
 CODE =appletfileName
[ALT=alternateText]
 [NAME=applet_instance_name]
WIDTH=pixels HEIGHT=pixels
 [ALIGN=aligment]
[VSPACE=pixels]       [HSPACE=pixels]
>
[<PARAM NAME=attributeName1 VALUE=attributeValue>]
 [<PARAM NAME=attributeName2 VALUE=attributeValue>]

</APPLET>
```

CODEBASE: is an optional attribute that specifies the base URL of the applet code or the directory that will be searched for the applet‘s executable class file.

CODE: is a required attribute that give the name of the file containing your applet‘s compiled class file which will be run by web browser or appletviewer.

ALT: Alternate Text The ALT tag is an optional attribute used to specify a short text message that should be displayed if the browser cannot run java applets.

NAME: is an optional attribute used to specify a name for the applet instance.

WIDTH AND HEIGHT: are required attributes that give the size(in pixels) of the applet display area. **ALIGN** is an optional attribute that specifies the alignment of the applet. The possible value is: LEFT, RIGHT, TOP, BOTTOM, MIDDLE, BASELINE, TEXTTOP, ABSMIDDLE, and ABSBOTTOM.

VSPACE AND HSPACE: attributes are optional, VSPACE specifies the space, in pixels, about and below the applet. HSPACE VSPACE specifies the space, in pixels, on each side of the applet

PARAM NAME AND VALUE: The PARAM tag allows you to specifies applet-specific arguments in an HTML page applets access there attributes with the getParameter() method.

(B) Attempt any ONE of the following: 1×6 = 6

(a) Differentiate between applet and application and also write a simple applet which display message 'Welcome to Java'.
(Any 3 Points - 3 Marks, Correct Program - 3 Marks)

Ans:

Applet	Application
Applet does not use main() method for initiating execution of code	Application use main() method for initiating execution of code
Applet cannot run independently	Application can run independently
Applet cannot read from or write to files in local computer	Application can read from or write to files in local computer
Applet cannot communicate with other servers on network	Application can communicate with other servers on network
Applet cannot run any program from local computer.	Application can run any program from local computer.
Applet are restricted from using libraries from other language such as C or C++	Application are not restricted from using libraries from other language

Program:

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

import java. applet.*;
import  java.awt.*;
public class WelcomeJava extends Applet
{
          public void paint( Graphics g)
          {
             g.drawString("Welcome to java",25,50);
          }
}
```

(b) Describe the following string class methods with examples :

(i) length()

(ii) charAt()

(iii) CompareTo()

(Each Method syntax or explanation - 1 Mark, Example - 1 Mark)

Ans:

1. length():

Syntax: int length()

It is used to return length of given string in integer.

Eg. String str="INDIA"

System.out.println(str);

System.out.println(str.length()); // Returns 5

2. charAt():

Syntax: char charAt(int position)

The charAt() will obtain a character from specified position .

Eg. String s="INDIA"

System.out.println(s.charAt(2)); // returns D

3. compareTo():

Syntax: int compareTo(Object o)

or

int compareTo(String anotherString)

There are two variants of this method. First method compares this String to another Object and second method compares two strings lexicographically.

Eg. String str1 = "Strings are immutable";
String str2 = "Strings are immutable";
String str3 = "Integers are not immutable";

```
int result = str1.compareTo( str2 );
System.out.println(result);
    result = str2.compareTo( str3 );
System.out.println(result);
```

5. Attempt any TWO of the following: 2×8 = 16

(a) Write a program to accept password from user and throw 'Authentication failure' exception if password is incorrect.
(Correct logic - 5 Marks, for syntax - 3 Marks)

Ans:

```
 import java.io.*;
class PasswordException extends Exception
{
        PasswordException(String msg)
        {
        super(msg);
        }
}
class  PassCheck
{
        public static void main(String args[])
        {
        BufferedReader bin=new BufferedReader(new InputStreamReader(System.in));
        try
        {
                System.out.println("Enter Password : ");
                if(bin.readLine().equals("EXAMW15"))
                {
```

```
                    System.out.println("Authenticated ");
                }
                else
                {
                    throw new PasswordException("Authentication failure");
                }
            }
            catch(PasswordException e)
            {
                System.out.println(e);
            }
            catch(IOException e)
            {
                System.out.println(e);
            }
    }
    }
```

(b) Explain life cycle of thread with neat diagram.
(Diagram - 3 Marks, Explanation - 5 Marks)

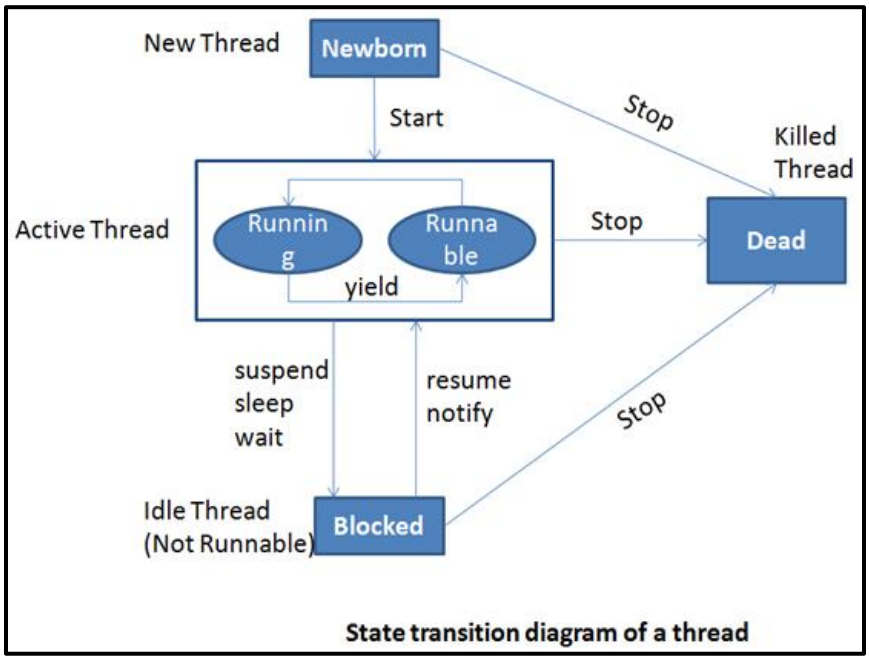
Ans:

Thread Life Cycle

Thread has five different states throughout its life.

1) Newborn State
2) Runnable State
3) Running State
4) Blocked State
5) Dead State

Thread should be in any one state of above and it can be move from one state to another by different methods and ways.



State transition diagram of a thread

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.

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().

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.

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.
suspend() : Thread can be suspended by this method. It can be rescheduled by resume().
wait(): If a thread requires to wait until some event occurs, it can be done using wait method andcan be scheduled to run again by notify().

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

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

(c) How can parameter be passed to an applet? Write an applet to accept user name in the form of parameter and print 'Hello<username>'.
(Explanation for parameter passing - 3Marks, any suitable example – 5 Marks)
(Any suitable example may be considered)

Ans:

Passing Parameters to Applet

- User defined parameters can be supplied to an applet using <PARAM…..> tags.
- PARAM tag names a parameter the Java applet needs to run, and provides a value for that parameter.
- PARAM tag can be used to allow the page designer to specify different colors, fonts, URLs or other data to be used by the applet.

To set up and handle parameters, two things must be done.

1. Include appropriate <PARAM..>tags in the HTML document.

The Applet tag in HTML document allows passing the arguments using param tag.
The syntax of <PARAM…> tag

<Applet code="AppletDemo" height=300 width=300>
<PARAM NAME = name1 VALUE = value1>
</Applet>

NAME:attribute name
VALUE: value of attribute named by corresponding PARAM NAME.

2. Provide code in the applet to parse these parameters.
   The Applet access their attributes using the getParameter method. The syntax is :
   String getParameter(String name);

```
import
java.awt.*;
import
java.applet.*;
public class HelloUser extends Applet
```

```
{
String str;
public void init()
{
        str = getParameter("username");  // Receiving parameter value
        str = "Hello "+ str;             //Using the value
}
public void paint(Graphics g)
{
        g.drawString(str,10,100);

}
}
<HTML>
<Applet code = HelloUser.class width = 400 height = 400>
          <PARAM NAME = "username" VALUE = abc>
</Applet>
</HTML>
```

OR

```
import  java.awt.*;
import java.applet.*;
/*<Applet code = HelloUser.class width = 400 height = 400>
          <PARAM NAME = "username" VALUE = abc>
</Applet>*/
public class HelloUser extends Applet
{
        String str;
        public void init()
        {
          str = getParameter("username");
          str = "Hello "+ str;
}
public void paint(Graphics g)
{
          g.drawString(str,10,100);
          }
}
```

6. Attempt any FOUR of the following : 4×4 = 16

(a) Explain method overloading with example.
(Method overloading - 1 Mark, Any relevant Example - 3 Marks)

Ans:

Method Overloading means to define different methods with the same name but different parameters lists and different definitions. It is used when objects are required to perform similar task but using different input parameters that may vary either in number or type of arguments. Overloaded methods may have different return types. It is a way of achieving polymorphism in java.

```
int add( int a, int b)              // prototype 1
int add( int a , int b , int c)     // prototype 2
double add( double a, double b)     // prototype 3
```

Example :

```
class Sample
{
    int addition(int i, int j)
    {
        return i + j ;
    }
    String addition(String s1, String s2)
    {
        return s1 + s2;
    }
    double addition(double d1, double d2)
    {
        return d1 + d2;
    }
}
class  AddOperation
{
    public static void main(String args[])
    {
        Sample sObj = new Sample();

        System.out.println(sObj.addition(1,2));
        System.out.println(sObj.addition("Hello ","World"));
        System.out.println(sObj.addition(1.5,2.2));
```

```
    }
}
```

(b) State any four system packages along with their use.
(Any 4 four, for each listing and use - 1 Mark)

Ans:

1. **java.lang** - language support classes. These are classes that java compiler itself uses and therefore they are automatically imported. They include classes for primitive types, strings, math functions, threads and exceptions.
2. **java.util** – language utility classes such as vectors, hash tables, random numbers, date etc.
3. **java.io** – input/output support classes. They provide facilities for the input and output of data
4. **java.awt** – set of classes for implementing graphical user interface. They include classes for windows, buttons, lists, menus and so on.
5. **java.net** – classes for networking. They include classes for communicating with local computers as well as with internet servers.
6. **java.applet** – classes for creating and implementing applets.

(c) What is use of ArrayList Class ?State any three methods with their use from ArrayList.
(Any one Use - 1 Mark, Any 3 methods - 1 Mark each)

Ans:

Use of ArrayList class:

1. ArrayList supports dynamic arrays that can grow as needed.
2. ArrayList is a variable-length array of object references. That is, an ArrayListcan dynamically increase or decrease in size. Array lists are created with an initial size. When this size is exceeded, the collection is automatically enlarged. When objects are removed, the array may be shrunk.

Methods of ArrayList class :

1. **void add(int index, Object element)**
   Inserts the specified element at the specified position index in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >size()).

2. **boolean add(Object o)**
   Appends the specified element to the end of this list.
3. **boolean addAll(Collection c)**
   Appends all of the elements in the specified collection to the end of this list, in the order that they are returned by the specified collection's iterator. Throws NullPointerException if the specified collection is null.
4. **boolean addAll(int index, Collection c)**
   Inserts all of the elements in the specified collection into this list, starting at the specified position. Throws NullPointerException if the specified collection is null.
5. **void clear()**
   Removes all of the elements from this list.
6. **Object clone()**
   Returns a shallow copy of this ArrayList.
7. **boolean contains(Object o)**
   Returns true if this list contains the specified element. More formally, returns true if and only if this list contains at least one element e such that (o==null ? e==null : o.equals(e)).
8. **void ensureCapacity(intminCapacity)**
   Increases the capacity of this ArrayList instance, if necessary, to ensure that it can hold at least the number of elements specified by the minimum capacity argument.
9. **Object get(int index)**
   Returns the element at the specified position in this list. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).
10. **int indexOf(Object o)**
    Returns the index in this list of the first occurrence of the specified element, or -1 if the List does not contain this element.
11. **int lastIndexOf(Object o)**
    Returns the index in this list of the last occurrence of the specified element, or -1 if the list does not contain this element.
12. **Object remove(int index)**
    Removes the element at the specified position in this list. Throws IndexOutOfBoundsException if index out of range (index < 0 || index >= size()).
13. **protected void removeRange(intfromIndex, inttoIndex)**
    Removes from this List all of the elements whose index is between fromIndex, inclusive and toIndex, exclusive.
14. **Object set(int index, Object element)**
    Replaces the element at the specified position in this list with the specified element. Throws IndexOutOfBoundsException if the specified index is is out of range (index < 0 || index >= size()).

15. **int size()**
    Returns the number of elements in this list.
16. **Object[] toArray()**
    Returns an array containing all of the elements in this list in the correct order. Throws NullPointerException if the specified array is null.
17. **Object[] toArray(Object[] a)**
    Returns an array containing all of the elements in this list in the correct order; the runtime type of the returned array is that of the specified array.
18. **void trimToSize()**
    Trims the capacity of this ArrayList instance to be the list's current size.

(d) Explain Serialization in relation with stream classes.
(Correct Explanation - 4 Marks)

Ans:

- Serialization in java is a mechanism of writing the state of an object into a byte stream.
- Java provides a mechanism, called object serialization where an object can be represented as a sequence of bytes that includes the object's data as well as information about the object's type and the types of data stored in the object.
- After a serialized object has been written into a file, it can be read from the file and deserialized that is, the type information and bytes that represent the object and its data can be used to recreate the object in memory.
- Classes ObjectInputStream and ObjectOutputStream are high-level streams that contain the methods for serializing and deserializing an object.
- The ObjectOutputStream class contains many write methods for writing various data types such as writeObject() method. This method serializes an Object and sends it to the output stream. Similarly, the ObjectInputStream class contains method for deserializing an object as readObject(). This method retrieves the next Object out of the stream and deserializes it. The return value is Object, so you will need to cast it to its appropriate data type.
- For a class to be serialized successfully, two conditions must be met:
  The class must implement the java.io.Serializable interface.
  All of the fields in the class must be serializable. If a field is not serializable, it must be marked transient.

(e) What is byte code? Explain any two tools available in JDK.
(Bytecode - 2 Marks, Any 2 Tools - 1 Mark each)

Ans:

Byte code: Bytecode in Java is an intermediate code generated by the compiler such as Sun's javac, that is executed by JVM. Bytecode is compiled format of Java programs it has a .class extension.

Tools	Brief Description
javac	The compiler for the Java programming language.
java	The launcher for Java applications. In this release, a single launcher is used both for development and deployment. The old deployment launcher, **jre**, is no longer provided.
javadoc	API documentation generator.
appletviewer	Run and debug applets without a web browser.
jar	Manage Java Archive (JAR) files.
jdb	The Java Debugger.
javah	C header and stub generator. Used to write native methods.
javap	Class file disassemble