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.

Q.1.

a) Attempt any three of following:

a) Explain :

1) Platform independence

2) Compiled and interpreted features of Java.

(2 M – each feature)

1) Platform independent

Java programs are platform independent and portable. That is they can be easily moved from one computer system to another. Changes in operating systems, processors, system resources will not force any change in java programs. Java compilers generate byte code instructions that can be implemented on any machine as well as the size of primitive data type is machine independent. In this sense, Java programs are platform independent.

2) Compiled & interpreted features of Java

Java is a two staged system. It combines both approaches. First java compiler translates source code into byte code instructions. In the second stage java interpreter generates machine code that can be directly executed by machine. Thus java is both compiled and interpreted language.

b) Explain serialization in relation with stream class.

(4 M Explanation, example not expected)

- 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.

c) Write any four mathematical functions used in Java.
(1 M – each method's syntax and use)
Note: Any four methods can be considered.

1) min() :
Syntax: static int min(int a, int b)
Use: This method returns the smaller of two int values.

2) max() :
Syntax: static int max(int a, int b)
Use: This method returns the greater of two int values.

3) sqrt()
Syntax: static double sqrt(double a)
Use : This method returns the correctly rounded positive square root of a double value.

4) **pow()** :
Syntax: static double pow(double a, double b)
Use : This method returns the value of the first argument raised to the power of the second argument.

5) exp()
Syntax: static double exp(double a)
Use : This method returns Euler's number e raised to the power of a double value.

6) round() :
Syntax: static int round(float a)
Use : This method returns the closest int to the argument.

7) abs()
Syntax: static int abs(int a)
Use : This method returns the absolute value of an int value.

d) Explain following methods related to threads:

1) suspend()
2) resume()
3) yield()
4) wait()

(1 M – each method's syntax and use)

1) **suspend()** -
syntax : public void suspend()
This method puts a thread in suspended state and can be resumed using resume() method.

2) resume()
syntax : public void resume()
This method resumes a thread which was suspended using suspend() method.

3) yield()
syntax : public static void yield()
The yield() method causes the currently executing thread object to temporarily pause and allow other threads to execute.

4) wait()
syntax : public final void wait()
This method causes the current thread to wait until another thread invokes the notify() method or the notifyAll() method for this object.

b) Attempt any one of following:

a) What is difference between array and vectors? Explain any 2 methods of vector with example.

(4M- for any 4 points of differences, 1M each - for any 2 methods of Vector class)

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[] arraname= 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.

Note : Any other methods of Vector class can also be considered.

Vector class methods :

1) void addElement(Object obj)

Adds the specified component to the end of this vector, increasing its size by one.

2) int capacity()

Returns the current capacity of this vector.

3) void clear()

Removes all of the elements from this Vector.

4) boolean contains(Object elem)

Tests if the specified object is a component in this vector.

5) void copyInto(Object[] anArray)

Copies the components of this vector into the specified array.

6) Object elementAt(int index)

Returns the component at the specified index.

7) boolean equals(Object o)

Compares the specified Object with this Vector for equality.

8) int indexOf(Object elem)

Searches for the first occurence of the given argument, testing for equality using the equals method.

9) void insertElementAt(Object obj, int index)

Inserts the specified object as a component in this vector at the specified index.

10) boolean removeElement(Object obj)

Removes the first (lowest-indexed) occurrence of the argument from this vector.

11) void removeElementAt(int index)

removeElementAt(int index)

12) int size()

Returns the number of components in this vector.

b) What is constructor? Demonstrate the use of parameterized constructor with suitable example.
(1M – What is constructor, 2M –features and example of constructor,
1M – parameterized constructor ,2M – example)

- 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.

Eg :

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

Output :
Area : 20

- **Parameterized constructor :**

When constructor method is defined with parameters inside it, different value sets can be provided to different constructor with the same name.

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));
}
}
```

Output :
Area : 20
Area : 42

Q.2. Attempt any two of following:

a) Explain abstract class with suitable example.

(4M- explanation, 4M – example)

- A class that is declared with abstract keyword is known as abstract class in java. It can have abstract and non-abstract methods (method with body).
- Abstraction is a process of hiding the implementation details and showing only functionality to the user. Another way, it shows only important things to the user and hides the internal details.
- A method must be always redefined in a subclass of an abstract class, as abstract class does not contain method body. Thus abstract makes overriding compulsory.
- Class containing abstract method must be declared as abstract.
- You can not declare abstract constructor, and so, objects of abstract class cannot be instantiated.
- Syntax :

```
abstract class < classname>
{
.
.
abstract method1(…);
method2(….);
.
.}
```

Example :

```
abstract class A
{
        abstract void disp();
        void show()
        {
        System.out.println("show method is not abstract");
        }
}


class B extends A
{
void disp()
{
        System.out.println("inside class B");
}
}

class test
{
        public static void main(String args[])
        {
        B b = new B();
        b.disp();
```

```
        b.show();
        }
}
```

Output : show method is not abstract inside class B

b) What is interface? How it is different from class? With suitable program explain the use of interface.
(2M – what is interface, 3M – 3 points of differences between interface and class, 3M – example)

Interface:

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.

Difference between Interface and class

Class	Interface
A Class is a full body entity with members, methods along with their definition and implementation.	An Interface is just a set of definition that you must implement in your Class inheriting that Interface
A Class has both definition and an implementation of a method	Interface has only definition. That is, it contains only abstract methods.
A Class can be instantiated.	An Interface cannot be instantiated
A sub class can be extended from super class with 'extends'.	You can create an instance of an Object of a class that implements the Interface with 'implements'

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();
}
}
```

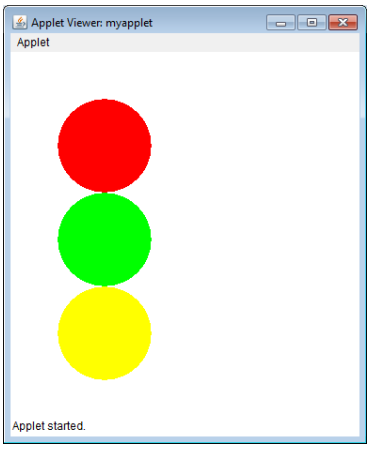
Output :

```
D:\>java result
Roll no : 101
Name : abc
sub1 : 75
sub2 : 75
sport_wt : 5
total : 155
```

c) Design an applet which displays three circle one below the other and fill them red, green and yellow color respectively. ***(3M- correct logic, 2M – correct use of class, packages and <applet> tag , 3M – correct syntaxes)***

```
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);
        }}
```

Output :



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

Q.3. Attempt any four of following:

a) Explain use of following methods:

1) **indexOf()**
2) **charAt()**
3) **substring()**
4) **replace()**

(1 mark each for each method use with parameters and return type.)
(Any one syntax of each method should be considered)

1. **indexOf():**
   int indexOf(int ch): Returns the index within this string of the first occurrence of the specified character.
   int indexOf(int ch, int fromIndex): Returns the index within this string of the first occurrence of the specified character, starting the search at the specified index.
   int indexOf(String str): Returns the index within this string of the first occurrence of the specified substring.
   int indexOf(String str, int fromIndex): Returns the index within this string of the first occurrence of the specified substring, starting at the specified index.
2. **charAt():**
   char charAt(int index): Returns the char value at the specified index.
3. **substring():**
   String substring(int beginIndex): Returns a new string that is a substring of this string.
   String substring(int beginIndex, int endIndex): Returns a new string that is a substring of this string. The substring begins at the specified beginIndex and extends to the character at index endIndex - 1
4. **replace():**
   String replace(char oldChar, char newChar): Returns a new string resulting from replacing all occurrences of oldChar in this string with newChar.

b) Write a program to find sum of digits of number.
(A program in which the student has assumed the number or has accepted the number in any other way may also be considered.)
(1 mark for syntax and 3 marks for logic.)

```
import java.io.*;
class SumOfDigits
{
        public static void main(String args[])
         {
        BufferedReader b = new BufferedReader(new InputStreamReader(System.in));
                int n;
                try
                {
                        System.out.println("Enter the number");
                        n = Integer.parseInt(b.readLine());
                        int sum = 0, n1;
                        while (n > 0)
```

```
                {
                        n1 = n%10;
                        n = n/10;
                        sum = sum+n1;
                }
                System.out.println("The sum of the digits of the number is :"+sum);
        }
        catch(Exception e)
        {
                System.out.println(e);
        }
    }
}
```

Or

```
class SumOfDigits
 {
    public static void main(String args[])
     {
        int n = 1234;    //Student may have assumed any other number
        int sum = 0, n1;
        while (n > 0)
        {
                n1 = n%10;
                n = n/10;
                sum = sum+n1;
        }
        System.out.println("The sum of the digits of the number is :"+sum);
    }
}
```

c) What is use of stream classes? Write any two methods FileReader class.

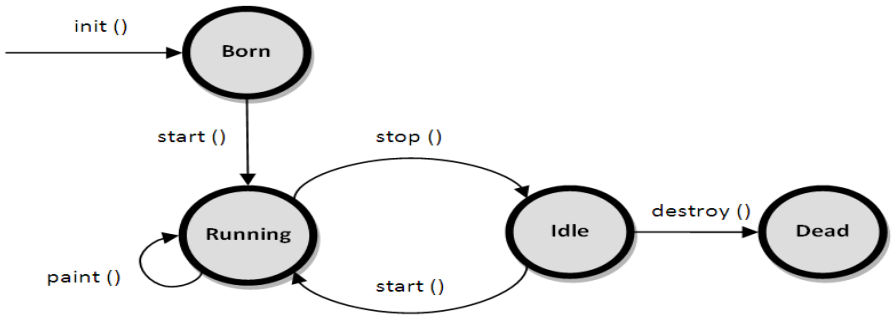
(2 marks for use of stream class, 2 marks for any two methods of FilReader class.)

- An I/O Stream represents an input source or an output destination.
- A stream can represent many different kinds of sources and destinations, including disk files, devices, other programs, and memory arrays.
- Streams support many different kinds of data, including simple bytes, primitive data types, localized characters, and objects.
- Some streams simply pass on data; others manipulate and transform the data in useful ways. Java's stream based I/O is built upon four abstract classes: InputStream, OutputStream, Reader, Writer.
- They are used to create several concrete stream subclasses, the top level classes define the basic functionality common to all stream classes.
- InputStream and OutputStream are designed for byte streams and used to work with bytes or other binary objects.
- Reader and Writer are designed for character streams and used to work with character or string.

1. public int read()throws IOException - Reads a single character.
2. public int read(char[] cbuf, int offset, int length) throws IOException - Reads characters into a portion of an array.
3. public void close()throws IOException - Closes the stream and releases any system resources associated with it. Once the stream has been closed, further read(), ready(), mark(), reset(), or skip() invocations will throw an IOException. Closing a previously closed stream has no effect
4. public boolean ready()throws IOException - Tells whether this stream is ready to be read. An InputStreamReader is ready if its input buffer is not empty, or if bytes are available to be read from the underlying byte stream
5. public void mark(int readAheadLimit) throws IOException -Marks the present position in the stream. Subsequent calls to reset() will attempt to reposition the stream to this point. Not all character-input streams support the mark() operation.
6. public void reset()throws IOException - Resets the stream. If the stream has been marked, then attempt to reposition it at the mark. If the stream has not been marked, then attempt to reset it in some way appropriate to the particular stream, for example by repositioning it to its starting point. Not all character-input streams support the reset() operation, and some support reset() without supporting mark().

d) Describe applet life cycle with suitable diagram.

(1 mark for diagram 3 marks for explanation)



Applets are small applications that are accessed on an Internet server, transported over the Internet, automatically installed, and run as part of a web document. The applet states include:

- Born or initialization state
- Running state
- Idle state
- Dead or destroyed state

Initialization state: Applet enters the initialization state when it is first loaded. This is done by calling the init() method of Applet class. At this stage the following can be done:

- Create objects needed by the applet
- Set up initial values
- Load images or fonts
- Set up colors

Initialization happens only once in the life time of an applet.

```
public void init()
{
        //implementation
}
```

Running state: applet enters the running state when the system calls the start() method of Applet class. This occurs automatically after the applet is initialized. start() can also be called if the applet is already in idle state. start() may be called more than once. start() method may be overridden to create a thread to control the applet.

```
public void start()
{
        //implementation
}
```

Idle or stopped state: an applet becomes idle when it is stopped from running. Stopping occurs automatically when the user leaves the page containing the currently running applet. stop() method may be overridden to terminate the thread used to run the applet.

```
public void stop()
{
        //implementation
}
```

Dead state: an applet is dead when it is removed from memory. This occurs automatically by invoking the destroy method when we quit the browser. Destroying stage occurs only once in the lifetime of an applet. destroy() method may be overridden to clean up resources like threads.

```
public void destroy()
{
//implementation
}
```

Display state: applet is in the display state when it has to perform some output operations on the screen. This happens after the applet enters the running state. paint() method is called for this. If anything is to be displayed the paint() method is to be overridden.

```
public void paint(Graphics g)
{
//implementation
}
```

e) What is final variable and methods? How it is different from abstract method?
(1 mark for explanation of final variable, 1 mark for explanation of final method. 2 marks for difference)

- All variable and methods can be overridden by default in subclass. In order to prevent this, the final modifier is used.
- Final modifier can be used with variable, method or class.

final variable: the value of a final variable cannot be changed. final variable behaves like class variables and they do not take any space on individual objects of the class.
Eg of declaring final variable: final int size = 100;

final method: making a method final ensures that the functionality defined in this method will never be altered in any way, ie a final method cannot be overridden.
Eg of declaring a final method: final void findAverage() {
//implementation

}

- abstract is a modifier used with methods and classes.
- An abstract method is a method that is declared without an implementation.
- The implementation is to be done in the subclass making overriding compulsory.
- It is opposite of final modifier.
- If a class includes abstract methods, then the class itself must be declared abstract.
- When an abstract class is subclassed, the subclass usually provides implementations for all of the abstract methods in its parent class. However, if it does not, then the subclass must also be declared abstract.

Eg: abstract class Shape

```
{
  int variable1, variable2;
  void method1
        {
        //implementation
         }
abstract void method2();
 }
```

Q.4.

a) Attempt any three of following:

a. What do mean by typecasting? When it is needed?

(Explanation of type casting with types 2 marks, need 1 mark, 1 mark for program or code snippet)

The process of converting one data type to another is called casting or type casting. If the two types are compatible, then java will perform the conversion automatically. It is possible to assign an int value to long variable. However if the two types of variables are not compatible, the type conversions are not implicitly allowed, hence the need for type casting.

Eg: int m = 50;

```
byte n = (byte) m;
long count = (long) m;
```

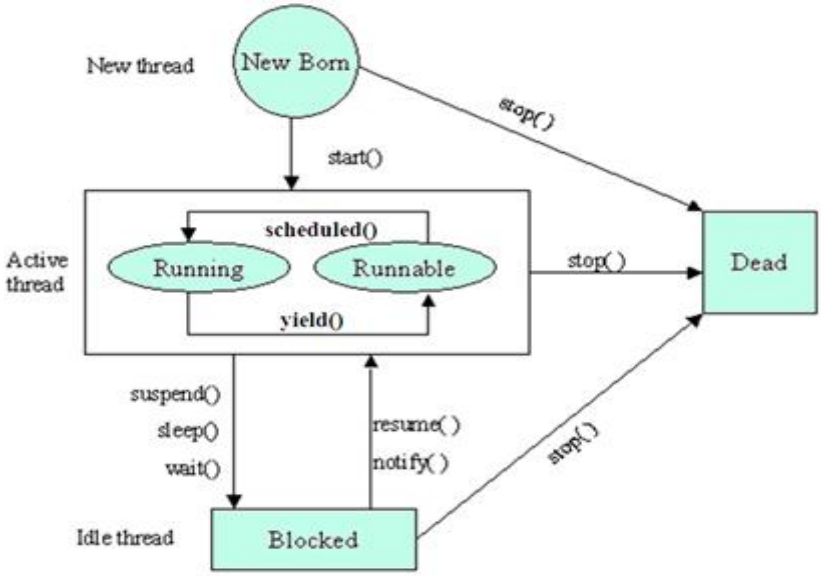
Type casting is of two types: narrowing, widening.

The process of assigning a smaller type to a larger one is known as widening and the process of assigning a larger type into a smaller one is called narrowing.

Casting is necessary when a value of one type is to be assigned to a different type of variable. It may also be needed when a method returns a type different than the one we require. Generic type casting helps in the retrieval of elements from a collection as each element in a collection is considered to be an object.

b. Describe life cycle of thread.

(1 mark for diagram, 3 marks for explanation)



A thread can be in one of the following states

1. New born state
2. Ready to run state
3. Running state
4. Blocked state
5. Dead state

New Born state

- The thread enters the new born state as soon as it is created. The thread is created using the new operator.
- From the new born state the thread can go to ready to run mode or dead state.
- If start( ) method is called then the thread goes to ready to run mode. If the stop( ) method is called then the thread goes to dead state.

Ready to run mode (Runnable State)

- If the thread is ready for execution but waiting for the CPU the thread is said to be in ready to run mode.
- All the events that are waiting for the processor are queued up in the ready to run mode and are served in FIFO manner or priority scheduling.
- From this state the thread can go to running state if the processor is available using the scheduled( ) method.
- From the running mode the thread can again join the queue of runnable threads.
- The process of allotting time for the threads is called time slicing.

Running state

- If the thread is in execution then it is said to be in running state.
- The thread can finish its work and end normally.
- The thread can also be forced to give up the control when one of the following conditions arise

1. A thread can be suspended by suspend( ) method. A suspended thread can be revived by using the resume() method.
2. A thread can be made to sleep for a particular time by using the sleep(milliseconds) method. The sleeping method re-enters runnable state when the time elapses.

3. A thread can be made to wait until a particular event occur using the wait() method, which can be run again using the notify( ) method.

Blocked state

- A thread is said to be in blocked state if it is prevented from entering into the runnable state and so the running state.
- The thread enters the blocked state when it is suspended, made to sleep or wait.
- A blocked thread can enter into runnable state at any time and can resume execution.

Dead State

- The running thread ends its life when it has completed executing the run() method which is called natural dead.
- The thread can also be killed at any stage by using the stop( ) method.

c. Write a program to generate Fibonacci series:1 1 2 3 5 8 13 21 34 55 89.
(Any other logic for generating the Fibonacci series may also be considered)
(Syntax 1 mark, logic 3 marks)

```
class FibonocciSeries
{
        public static void main(String args[])
         {
                int num1 = 1; int num2 = 1;
                System.out.println(num1);
                while (num2< 100)
                 {
                        System.out.println(num2);
                        num2 = num1+num2;
                        num1 = num2-num1;
                }
        }

}
```

d. Write syntax and function of following methods of data class:

1. **Sethme () // The method name has to be setTime()**
2. **getDay()**

(2 marks for syntax and use of each method)

1. setTime():

void setTime(long time): the parameter time - the number of milliseconds.

Sets this Date object to represent a point in time that is time milliseconds after January 1, 1970 00:00:00 GMT

2.getDay()

int getDay():Returns the day of the week represented by this date. The returned value (0 = Sunday, 1 = Monday, 2 = Tuesday, 3 = Wednesday, 4 = Thursday, 5 = Friday, 6 = Saturday) represents the day of the week that contains or begins with the instant in time represented by this Date object, as interpreted in the local time zone.

b) Attempt any one of the following:

a. Write the syntax and example for each of following graphics methods:

1. drawPoly ()
2. drawReact ()
3. drawOval ()
4. fillOval()

(1 mark each for each method example with syntax, 2 marks for program. Students may write different program for each method or may include all methods in one program.)

(Any one syntax and use of the method may be considered)

1. **void drawPolygon(int[] xPoints, int[] yPoints, int nPoints):** Draws a closed polygon defined by arrays of x and y coordinates. The number of points defined by x and y is specified by numpoints.
2. **void drawRect(int top, int left, int width, int height):** Draws a rectangle with upper left corner of the rectangle is top, left. Dimensions of the rectangle are specified by width and height.
3. **void drawOval(int top, int left, int width, int height):** Draws an oval within a bounding rectangle whose upper left corner is specified by top, left. Width and height of the oval are specified by width and height.
4. **void fillOval(int top, int left, int width, int height):** Draws an oval within a bounding rectangle whose upper left corner is specified by top, left. Width and height of the oval are specified by width and height.

Eg.

```
import java.applet.*;
import java.awt.*;
/*
<applet code = DrawGraphics.class height = 500 width = 400></applet>
*/
public class DrawGraphics extends Applet
 {
    public void paint(Graphics g)
    {
            int x[] = {10, 170, 80};
            int y[] = {20, 40, 140};
            int n = 3;
            g.drawPolygon(x, y, n);
            g.drawRect(10, 150,100, 80);
            g.drawOval(10, 250, 100, 80);
            g.fillOval(10, 350, 100, 80);
    }
}
```

b. What is garbage collection and finalize method in Java?
(3 marks for garbage collection, 3 marks for finalization)

Garbage collection - The objects are dynamically allocated in java by the use of the new operator. The objects which are no longer in use are to be destroyed and the memory should be released for later reallocation. Java, unlike C++, handles deallocation of memory automatically. This technique is called garbage collection. When no references to an object exist, that object is assumed to be no longer needed, and the memory occupied by the object can be reclaimed. Garbage collection occurs sporadically during the execution of programs. It will not occur just because one or more objects exist that are no longer used. Different run-time implementations will take varying approaches to garbage collection.

finalize() method: -sometimes an object will need to perform some action when it is destroyed. Eg. If an object holding some non java resources such as file handle or window character font, then before the object is garbage collected these resources should be freed. To handle such situations java provide a mechanism called finalization. In finalization, specific actions that are to be done when an object is garbage collected can be defined. To add finalizer to a class define the finalize() method. The java run-time calls this method whenever it is about to recycle an object. Inside the finalize() method, the actions that are to be performed before an object is to be destroyed, can be defined. Before an object is freed, the java run-time calls the finalize() method on the object. The general form of the finalize() method is:

```
protected void finalize()
      {
 //finalization code
      }
```

Q.5. Attempt any Two of following:

a) Write a program to create two threads; one to print numbers in original order and other to reverse order from 1 to 50.
(4-marks for Logic, 4-marks for correct Syntax)
(Any other program with correct logic may also be considered)

```
class original extends Thread
{
public void run()
{
        try
        {
        for(int i=1; i<=50;i++)
        {
                System.out.println("\t First Thread="+i);
                Thread.sleep(300);
        }
        }
        catch(Exception e)
        {}
}
}
```

```
class reverse extends Thread
{
public void run()
{
        try
        {
        for(int i=50; i>=1;i--)
        {
                System.out.println("\t Second Thread="+i);
                Thread.sleep(300);
        }
        }
        catch(Exception e)
        {}
}
}
class orgrev
{
        public static void main(String args[])
        {
                new original().start();
                new reverse().start();
                System.out.println("Exit from Main");
        }
}
```

b) What are different types of error? What is use of throw, throws and finally Statement?
[Type of Errors & explanation 2-marks, throw, throws & finally 2-mark each]

Errors are broadly classified into two categories:-

1. Compile time errors
2. Runtime errors

Compile time errors: All syntax errors will be detected and displayed by java compiler and therefore these errors are known as compile time errors.

The most of common problems are:

- Missing semicolon
- Missing (or mismatch of) bracket in classes & methods
- Misspelling of identifiers & keywords
- Missing double quotes in string
- Use of undeclared variables.
- Bad references to objects.

Runtime errors: Sometimes a program may compile successfully creating the .class file but may not run properly. Such programs may produce wrong results due to wrong logic or may terminate due to errors such as stack overflow. When such errors are encountered java typically generates an error message and aborts the program.

The most common run-time errors are:

- Dividing an integer by zero
- Accessing an element that is out of bounds of an array
- Trying to store value into an array of an incompatible class or type

- Passing parameter that is not in a valid range or value for method
- Trying to illegally change status of thread
- Attempting to use a negative size for an array
- Converting invalid string to a number
- Accessing character that is out of bound of a string

throw: If your program needs to throw an exception explicitly, it can be done using 'throw' statement. General form of throw statement is:

throw new Throwable subclass;

'throw' statement is mainly used in case there is user defined exception raised. Throwable instance must be an object of the type Throwable or a subclass of Throwable.

The flow of exception stops immediately after the 'throw' statement; any subsequent statements are not executed. The nearest enclosing 'try' block is inspected to see if it has a catch statement that matches the type of exception. If it does find a match, control is transferred to that statement. If not matching catch is found, then the default exception handler halts the program and prints the built in error message.

throws: If a method is capable of causing an exception that it does not handle, it must specify this behavior so that callers of the method can guard themselves against that exception. You do this by including a 'throws' clause in the methods declaration. A throws clause lists the types of exception that a method might throw.

General form of method declaration that includes 'throws' clause

```
Type method-name (parameter list) throws exception list
{
// body of method
}
```

Here exception list can be separated by a comma.

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

```
try                          OR
{
.....
}
finally
{
....
}
catch()
{
.....
}
```

```
try
{
.....
}
catch()
{
....
}
finally()
{
.....
}
```

c) State the use of Font class. Write syntax to create an object of Font class. Describe any 3 methods of Font class with their syntax and example of each.

[Font class 3-Marks, Syntax 2-Marks & Methods 3-Marks]

Font class: A font determines look of the text when it is painted. Font is used while painting text on a graphics context & is a property of AWT component.

The Font class defines these variables:

Variable	**Meaning**
String name	Name of the font
float pointSize	Size of the font in points
int size	Size of the font in point
int style	Font style

Syntax to create an object of Font class: To select a new font, you must first construct a Font object that describes that font.

Font constructor has this general form:

Font(String fontName, int fontStyle, int pointSize)

fontName specifies the name of the desired font. The name can be specified using either the logical or face name. All Java environments will support the following fonts: Dialog, DialogInput, Sans Serif, Serif, Monospaced, and Symbol. Dialog is the font used by once system's dialog boxes. Dialog is also the default if you don't explicitly set a font. You can also use any other fonts supported by particular environment, but be careful—these other fonts may not be universally available.

The style of the font is specified by fontStyle. It may consist of one or more of these three constants: Font.PLAIN, Font.BOLD, and Font.ITALIC. To combine styles, OR them together. For example, Font.BOLD | Font.ITALIC specifies a bold, italics style.

The size, in points, of the font is specified by pointSize.

To use a font that you have created, you must select it using setFont( ), which is defined by Component. It has this general form:

void setFont(Font fontObj)

Here, fontObj is the object that contains the desired font

Methods of Font class

1. String getFamily(): Returns the family name of this Font.
2. int getStyle():Returns the style of this Font
3. int getSize() : Returns the size of this Font
4. boolean is bold(): Returns true if the font includes the bold style value. Else returns false
5. String toString(): Returns the string equivalent of the invoking font.

Example Using Methods of Font class:

Program-1:

A program to make use of Font class methods. Which will display string "Java Programming Is Language" if font object style is BOLD else it will display string "Java Programming Is Language" with another string saying "String is not bold"

```
import java.applet.*;
import java.awt.*;
/*<applet code="FontD" width=350 height=60> </applet> */
public class FontD extends Applet {
Font f, f1;
String s="";
String msg="";
public void init()
{
f=new Font("Dialog", Font.BOLD,30);
s="Java Programming";
setFont(f);
msg="Is Language";
int a=f.getSize();
String b=f.getFontName();
int d=f.getStyle();
System.out.println("String Information: Size"+a);
System.out.println("Name:"+b);
System.out.println("Style:"+d);
}
public void paint(Graphics g) {
if(f.isBold()==true)
g.drawString(s,50,50);
else
g.drawString("String is not bold",400,400);
g.drawString(s,50,50);
g.drawString(msg,100,100);}}
```

Output:

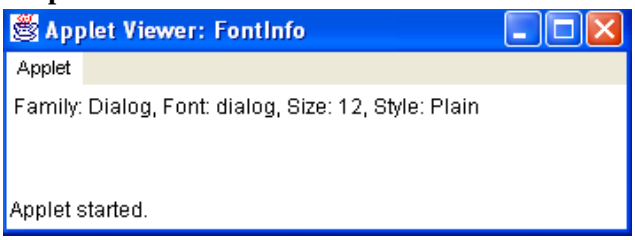

```
C:\jdk1.2.2\bin>javac FontD.java

C:\jdk1.2.2\bin>appletviewer FontD.java
String Information: Size30
Name:dialog.bold
Style:1
```

Program-2:

A program to make use of Font class methods. To displays the name, family, size, and style of the currently selected font:

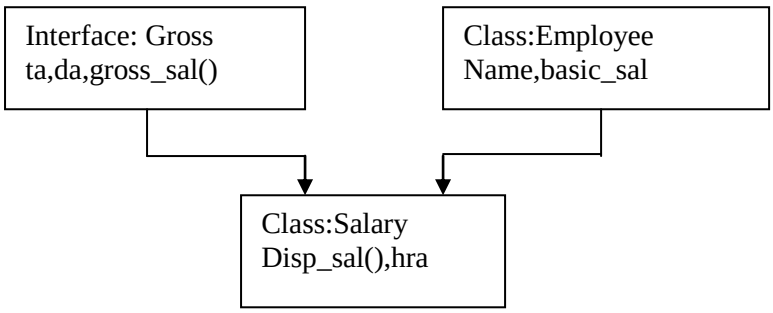
```
import java.applet.*;
import java.awt.*;
/*<applet code="FontInfo" width=350 height=60> </applet>*/
public class FontInfo extends Applet {
public void paint(Graphics g) {
Font f = g.getFont();
String fontName = f.getName();
String fontFamily = f.getFamily();
int fontSize = f.getSize();
int fontStyle = f.getStyle();
String msg = "Family: " + fontName;
msg += ", Font: " + fontFamily;
msg += ", Size: " + fontSize + ", Style: ";
if((fontStyle & Font.BOLD) == Font.BOLD)
msg += "Bold ";
if((fontStyle & Font.ITALIC) == Font.ITALIC)
msg += "Italic ";
if((fontStyle & Font.PLAIN) == Font.PLAIN)
msg += "Plain ";
g.drawString(msg, 4, 16);}}
```

Output:



Q.6. Attempt any four of following:

a)



[2-marks for Logic, 2-marks for correct Syntax]

```
interface Gross
{
double ta=500.0;
double da=1200.0;
```

```
void gross_sal();
}
class Employee
{
String name;
float basic_sal;
Employee(String n, float b)
{
name=n;
basic_sal=b;
}
void display()
{
System.out.println("Name of Employee="+name);
System.out.println("Basic Salary of Employee="+basic_sal);
}
}
class salary extends Employee implements Gross
{
float hra;
salary(String n, float b, float h)
{
super(n,b);
hra=h;
}
void disp()
{
display();
System.out.println("HRA of Employee="+hra);
}
public void gross_sal()
{
double gross_sal=basic_sal+ta+hra;
System.out.println("TA of Employee="+ta);
System.out.println("DA of Employee="+da);
System.out.println("Gross Salary of Employee="+gross_sal);
}
}
class Empdetail
{
public static void main(String args[])
{
salary s=new salary("ABC",6000,4000);
s.disp();
```

```
s.gross_sal();
}
}
```

```
C:\jdk1.2.2\bin>javac Empdetail.java

C:\jdk1.2.2\bin>java Empdetail
Name of Employee=ABC
Basic Salary of Employee=6000.0
HRA of Employee=4000.0
TA of Employee=500.0
DA of Employee=1200.0
Gross Salary of Employee=10500.0

C:\jdk1.2.2\bin>
```

b) What is use of setclass? Write a program using setclass.
(any other suitable example may also be considered.)
[2-marks for Set class, 2-marks for Program]

Note: Assuming Set interface in place of set class. Answer is as follows:

The Set interface defines a set. It extends Collection and declares the behavior of a collection that does not allow duplicate elements. Therefore, the add( ) method returns false if an attempt is made to add duplicate elements to a set.

The Set interface contains only methods inherited from Collection and adds the restriction that duplicate elements are prohibited.

Set also adds a stronger contract on the behavior of the equals and hashCode operations, allowing Set instances to be compared meaningfully even if their implementation types differ.

The methods declared by Set are summarized in the following table:

SN	Methods with Description
1	add() Adds an object to the collection
2	clear() Removes all objects from the collection
3	contains() Returns true if a specified object is an element within the collection
4	isEmpty() Returns true if the collection has no elements
5	iterator() Returns an Iterator object for the collection which may be used to retrieve an object
6	remove() Removes a specified object from the collection
7	size() Returns the number of elements in the collection

There are three general-purpose Set implementations — HashSet, TreeSet, and LinkedHashSet. Which of these three to use is generally straight forward. HashSet is much faster than TreeSet(constant-time versus log-time for most operations) but offers no ordering guarantees. If you need to use the operations in the SortedSet interface, or if value-ordered iteration is required, use TreeSet; otherwise, use HashSet. It's a fair bet that you'll end up using HashSet most of the time.

LinkedHashSet is in some sense intermediate between HashSet and TreeSet. Implemented as a hash table with a linked list running through it, it provides *insertion-ordered* iteration (least recently inserted to most recently) and runs nearly as fast as HashSet. The LinkedHashSet implementation spares its clients from the unspecified, generally chaotic ordering provided by HashSet without incurring the increased cost associated with TreeSet.

Set have its implementation in various classes like HashSet, TreeSet, LinkedHashSet. Following is the example to explain Set functionality:

```
import java.util.*;
public class SetDemo
{
  public static void main(String args[])
{
    int count[] = {34, 22,10,60,30,22};
    Set<Integer> set = new HashSet<Integer>();
    try{
      for(int i = 0; i<5; i++){
        set.add(count[i]);
      }
      System.out.println(set);
      TreeSet sortedSet = new TreeSet<Integer>(set);
      System.out.println("The sorted list is:");
      System.out.println(sortedSet);
      System.out.println("The First element of the set is: "+ (Integer)sortedSet.first());
      System.out.println("The last element of the set is: "+ (Integer)sortedSet.last());
 }
    catch(Exception e){}
  }}
```

Executing the program.

```
[34, 22, 10, 30, 60]
The sorted list is:
[10, 22, 30, 34, 60]
The First element of the set is: 10
The last element of the set is: 60
```

c) Differentiate between applet and application (any 4 points).

Ans: [4-marks for 4 Points]

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

d) What is package? How do we create it?

[Package: 3-marks Creating Packages 5-marks]

Package: Packages are used for grouping a variety of classes & interfaces together. This grouping is done according to functionality. Packages acts as containers of classes. Packages are stored in hierarchical manner & are explicitly imported into new class definitions. Following benefits are achieved by organizing classes into packages:

1. The classes contained in the packages of other programs can be easily reused.
2. In packages, classes can be unique compared with classes in other packages. That is two classes in two different packages can have same name. They may be referred by their fully qualified name, comprising package name & class name.
3. Packages provide way to hide classes thus preventing other programs or packages from accessing classes that are meant for internal use only.
4. Packages also provide a way for separating "design" from "coding". First we can design classes & decide their relationship & then we can implement Java code needed for methods. It is possible to change implementation of any method without affecting rest of the design. Java packages are therefore classified into two types. First category is known as Java API packages & second is known as User defined packages.

Creating Packages: First declare name of package using package keyword followed by package name. This must be first statement in a java source file. Here is an example:

```
package firstPackage;     //package declaration

public class FirstClass    //class definition
        {
                …..
                …..     (body of class)
                …..
        }
```

Here the package name is firstPackage. The class FirstClass is now considered a part of this package. This listing would be saved as file called FirstClass.java & located in a director named firstPackage. When source file is complied, Java will create a .class file & store it in same directory.

.class must be located in a directory that has same name as the package, & this directory should be a subdirectory of the directory where classes that will import package are located.

Creating package involves following steps:

1. Declare the package at beginning of a file using the form
   package packagename;
2. Define the class that is to be put in the package & declare it public
3. Create a subdirectory under directory where main source files are stored
4. Store listing as the classname.java file in the subdirectory created.
5. Compile the file. This creates .class file in the subdirectory

Case is significant & therefore subdirectory name must match package name exactly. Java supports the concept of package hierarchy. This is done by specifying multiple names in a package statement, separated by dots. Example: package firstPackage.secondPackage; This approach allows us to group related classes into a package & then group related packages into larger package. To store this package in subdirectory named firstPackage/secondPackage.

A java package file can have more than one class definition. in such cases only one of the classes may be declared public & that class name with .java extension is the source file name. When a source file with more than one class definition is complied, java creates independent .class files for these classes.

e) Write a program to print reverse of a number.

[2-marks for Logic, 2-marks for correct Syntax]

(Any other program with correct logic may also be considered)

```
class Reverse1
{
public static void main(String args[])
{
int num = Integer.parseInt(args[0]); //take argument as command line
int remainder, result=0;
while(num>0)
{
remainder = num%10;
result = result * 10 + remainder;
num = num/10;
}
System.out.println("Reverse number is : "+result);
}
}
```

```
C:\jdk1.2.2\bin>javac Reverse1.java

C:\jdk1.2.2\bin>java Reverse1 987654321
Reverse number is : 123456789
```