Introduction to OOP, procedural programming languages and object oriented language, principles of OOP, applications of OOP, history of java, java features, JVM, program structure.

Variables, primitive data types, identifiers, literals, operators, expressions, precedence rules and associativity, primitive type conversions and casting, flow of control.

Introduction to OOP – Principle of OOP

1Q: Briefly explain the basic concepts of OOPS (OR)

Explain the need for object oriented programming (OR)

What is OOP? Explain the principles of object oriented programming languages (10M) (OR) Give the characteristics of OOPs in detail.(8M) (OR)

Compare inheritance with polymorphism (4M)

Object Oriented Programming - Def

Object Oriented Programming is a technique that builds the program using the objects along with a set of well defined interfaces. It describes the way in which elements within a computer program must be organized and how these elements can interact with each other. Object oriented programming uses bottom-up approach and also manages the increasing complexity.

The following are the basic concepts of OOPS

1. **Object**
2. **Class**
3. **Data abstraction**
4. **Encapsulation**
5. **Inheritance**
6. **Polymorphism**

Object

- **An object can be defined as an instance of the class which is used to access the members of a class.**
- **An object is the real world entity.**
- **An object is the representative of class.**

- **Syntax:** **class-name object-name;**
- **Eg:** **Student s;**
- **Here, Student is the class name and " s " is the object name.**

Class

- **A class can be defined as a template that groups data and its associated functions.**
- **The class contains two parts namely,**
  a) **Declaration of data members.**
  b) **Declaration of member functions.**
- **The data members of a class explain about the state of the class and the member functions explain about the behavior of the class.**
- **There are three types of variables for a class. They are,**
  a) **Local variables – declared inside the class.**
  b) **Instance variables – declared inside the class but outside of the methods.**
  c) **Class variables – declared inside the class with static modifier and they reside outside of the method.**

Data abstraction (3M)

- **Representing the essential features without including its background details is called data abstraction.**
- **It is the mechanism that hides the implementation details and shows only the functionality.**

Encapsulation

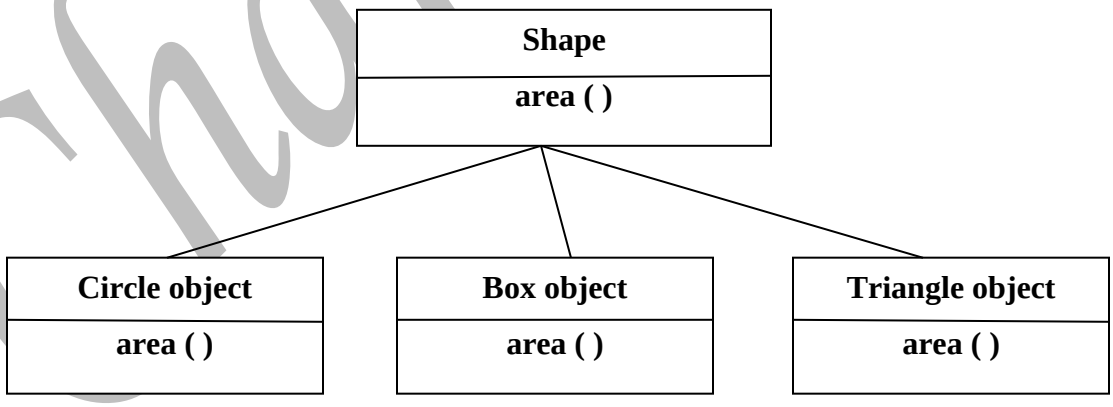
- **Encapsulation is the mechanism of binding data members and corresponding methods into a single module or class in-order to protect them from being accessed by the outside code.**
- **The data and functions in a class are called as members of the class. The data defined in the class are called data members and the functions defined are called member functions.**
- **The main idea behind the concept of encapsulation is to obtain high maintenance and to handle the application's code.**
- **It is the most striking feature of the class.**

Inheritance

- **Acquiring or getting properties from base class to the derived class is called as inheritance.**
- **The class which gives properties to the other classes is called base class and the class which accepts properties from the other class is called derived class.**
- **The main advantage of inheritance is code reusability.**
- **The following are the types of inheritance**
  a) **Single level inheritance**
  b) **Multi-level inheritance**
  c) **Multiple inheritance**
  d) **Hybrid inheritance**
  e) **Hierarchical inheritance**

Polymorphism (4M)

- **Poly means many and morph means forms. So the ability to make more than one form is called polymorphism.**
- **It is used in inheritance programs.**
- **It can be divided as two ways**
  a) **Static polymorphism (Compile-time polymorphism)**
  b) **Dynamic polymorphism (Run-time polymorphism)**
- **The following diagram represents that a single function name can be used for different purposes with different types of arguments.**



Procedural language and object oriented language

2Q: Discuss about Procedural languages Vs OOP (OR)

Differentiate between procedure oriented and object oriented programming (8M) (OR)

List the major differences between C and JAVA (OR)

Differentiate between structured programming and object oriented programming. (OR)

What are the problems with procedure languages? How object oriented languages overcome the problems of procedural languages? (10M) (OR)

What are the drawbacks of procedural languages? Explain the need of object oriented programming. (10M) (OR)

What is procedural language? Differentiate between procedural language and OOP.(8M)

Procedure oriented programming	Object oriented programming
1. C is structured programming language.	1. JAVA is object oriented programming language.
2. It supports top-down approach.	2. It supports bottom-up approach.
3. In this, set of functions/procedures are used to perform a task.	3. In this, objects are used.
4. In this, data and functions are considered as different entities.	4. In this, data and functions are encapsulated into a single entity called class.
5. It does not support features such as inheritance, encapsulation and polymorphism.	5. It supports features such as inheritance, encapsulation and polymorphism.
6. It is not user friendly.	6. It is user friendly.
7. Complexity in this is more.	7. Complexity in this is less.
8. It supports pointers.	8. It does not support pointers.
9. In this, header files are included.	9. In this, packages are imported.

Applications of OOP

3Q: Discuss the applications of OOPS. (3M) (OR)

List and explain the applications of OOPs. (8M)

- **Mobile computing.**
- **Real time systems.**
- **Neural networks.**
- **Image processing.**
- **Artificial intelligence.**
- **Web based applications.**
- **Database management.**
- **Business process re-engineering.**
- **Enterprise resource planning.**
- **Data warehousing and data mining.**

History of java

4Q: Briefly give the history of JAVA.

- **Java language was introduced by James Gosling at Sun Microsystems.**
- **Java was invented in 1991. Initially, Java language was referred as "Oak" and later renamed as Java in 1995.**
- **The primary motive of Java language was to prepare software that can be embedded in different computer applications.**
- **Gosling and others stared working on portable and cross-platform language which results software that can be executed on different CPU's under various environments. This leads to the invention of Java.**
- **In the initial stage of Java, an important factor World Wide Web (WWW) has not taken during the implementation of Java.**
- **But in 1993, the focus of Java is shifted to internet programming due to the occurrence of portability problems during the creation of code for the internet.**
- **Java can be inherited from the syntax of C and object oriented concepts are from C++.**
- **Hence, it is easy to learn Java when one is familiar with C or C++. So, Java was implemented for internet programming using C++ concepts.**

JVM

5Q: Discuss about Java Virtual Machine (JVM) (6M) (OR)

What is the role and responsibility of JVM in program execution? (6M)

Define java byte code. Why java generates byte code? (8M) (OR)

What is the significance of Java's byte code? (3M) (OR)

What is byte code? How it will be generated? (3M) (OR)

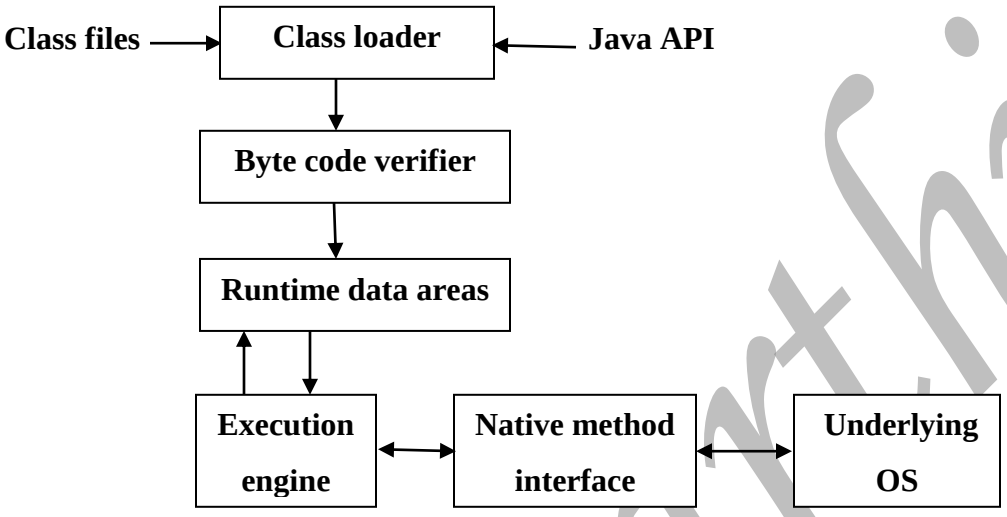
"Java is called Machine Independent language" - Justify this statement with proper explanation. (8M)

- **Java Virtual Machine (JVM) is the most important part of java technology.**
- **JVM and Java API together, form a platform for all java programs to run.**
- **JVM and Java API is known as Java Runtime Environment (JRE).**
- **Java provides both compiler and a software machine called Java Virtual Machine (JVM) for each computer machine.**
- **The java compiler translates java source code into an intermediate code known as byte code which executes on a special type of machine. This machine is called Java Virtual Machine and exists only inside the computer memory.**

- **Java interpreter reads byte code and translates into a language that the computer can understand and it can execute the code in any system.**

- **JVM loads the java classes, verifies the byte code, interprets and executes it. Additionally, it provide functions like security management and garbage collection.**

- **The following is the internal architecture of JVM**



Class Loader

It performs loading of classes and interfaces into JVM. When a program attempts to invoke a method of certain class, then JVM checks whether that class is already loaded or not. If not, JVM uses class loader to load the binary representation of that class.

Byte code verifier

After loading the class, byte code verifier checks whether the loaded representation is well-formed, whether it follows the semantic requirements of java programming language and JVM. It also checks whether the code contains proper symbol table or not. If a problem occurs during verification, then an error is thrown.

Runtime data areas

These are special memory areas which JVM maintains to store temporarily byte codes and other information like loaded class files, objects, parameters to methods, return values, local variables and results etc.,

Execution engine

It is responsible for executing instructions contained in the methods of loaded classes.

Native method interface

If any java program calls a non-java API method or platform-specific native method then, the code of these native methods is obtained from underlying OS through the use of native method interface.

Java features

6Q: Briefly explain the features or properties of JAVA (8M) (OR)

List various types of statements and quote suitable examples for each type. (9M) (OR)

Java was used for internet applications. Why? (4M)

Support the statement "java byte code gives high performance". (4M)

Support the statement "java is dynamic". Discuss. (4M)

Support the statement "java is Architecture-Neutral" (4M)

Support the statement "java is robust". Discuss. (4M)

The following are the features of JAVA.

i. **Object oriented**
ii. **Compiled and interpreted**
iii. **Platform independent and portable**
iv. **Distributed**
v. **Robust and secure**
vi. **Familiar, simple and small**
vii. **Multi-threaded and interactive**
viii. **Dynamic and extensible**
ix. **High performance**

i) Object oriented

- **Almost everything in JAVA is in terms of object.**
- **Complete program code and data placed within objects and classes.**
- **Java is said to be a true object oriented language.**
- **It comes with an extensive set of classes, packages which we can use in the programs by inheritance.**
- **In Java, the object model is not only very simple but also very easy to extend.**

ii) Compiled and interpreted

- **Generally, a computer language will be either compiled or interpreted.**
- **But Java combines both these approaches.**

- **In the first stage, java compiler translates source code into byte code instructions. But byte code instructions are not machine instructions.**
- **So, java interpreter generates machine code in the second stage which can be directly executed by the machine which is running the java program.**
- **Hence, java is not only a compiled but also an interpreted language.**

iii) Platform independent and portable (or) Architecture-neutral (4M)

- **Java supports portability. Ie java programs can be moved from only system to another, anywhere and anytime easily.**
- **Changes and upgrades in processors and operating system will not force any changes in java programs.**
- **Java programs can run on any platform.**

iv) Distributed

- **Java is distributed language.**
- **It not only has the ability to share data but also programs.**
- **Java applications can open and access remote objects on internet very easily.**
- **This enables many programmers placing at different locations to work on a single project.**

v) Robust and secure (4M)

- **Java provides many safeguards in order to ensure reliable code.**
- **Java has strict compile time and run time checking for data types and it also incorporates the concept of exception handling that captures errors and eliminates the risk of system crash.**
- **Hence java is said to be robust language.**
- **For a language which is used for programming on internet, security is very important.**
- **Java not only verifies the memory access but also ensures no viruses communicate with an applet.**

vi) Familiar, simple and small

- **Java is familiar language. It is modeled on C and C++ languages.**
- **Java uses several features of C and C++ and therefore java code looks like a C++ code.**
- **Java is simplified version of C++.**

- **Java is not only small but also a simple language.**
- **Java does not use pointer, goto statement, operator overloading, multiple inheritance etc.,**

vii) Multithreaded and interactive

- **Java supports multithreading ie., it is not necessary for an application to finish one task before starting another. In this way, several tasks can handle simultaneously.**
- **The java run-time comes with tools which supports multiprocessor synchronization and construct interactive systems running smoothly.**

viii) Dynamic and extensible (4M)

- **Java is a dynamic language which is capable of dynamically linking new class libraries, methods and objects.**
- **Java also supports extensibility. It support functions written in other languages like C and C++.**
- **This makes the programmers to use the efficient functions available in these languages.**

ix) High performance (4M)

- **Because of using intermediate byte code, java performance is excellent for an interpreted language.**
- **In order to reduce overhead during runtime, java architecture is designed carefully.**
- **Multithreading improves overall execution speed of java programs.**

Program structure

7Q: Explain the general syntax of writing an application program in Java. Also explain the steps to run an application Java program. (OR)

How do you write a Java program? Explain compilation and execution procedure with example. (OR)

Explain the Java program structure. (8M) (OR)

Discuss the lexical issues of Java.(6M)

A java program consists of one or more classes. Only one of these classes defines the main( ) method. The class consists of data and methods that operate on the data of a class.

Syntax:

Documentation section
↓
Package statement
↓
Import statement
↓
Interface statement
↓
Class definitions
↓
Main method class definition
{
main( ) method definition
}

Documentation

This section consists of a set of comments about the program. This is suggested and it is optional.

Package statement

This is the first statement in every java program, if needed. This statement tells the compiler that the classes defined here belongs to this package. This statement is optional.

Import statement

The import statement tells the interpreter to include the classes from the package defined. This is the next statement after the package declaration but should be written before defining a class. There may be number of import statements. This statement is optional. For example,

```
import java.io.*;
```

Interface statement

The interface statement defines method declaration without body for the subclasses. This is optional. It is used in inheritance programs.

Class definition

This section consists of a number of class definitions where each class consists of data members and methods. This is optional.

Main method class

This is an essential section of a java program. Every java program must have a class definition that defines the main( ) method. This is essential because main( ) is the starting point for running java programs. The program terminates on reaching the end of the main( ) method.

Example program

- **JAVA program is typed in text editor (notepad) and make corrections if necessary.**
- **The program name is same as the class name and stores with extension .java in the secondary storage device.**

  Syntax: Filename.java

  Eg: Simple.java
- **The starting letter of the class name will be the capital and same as program name. It is the conventional rule in java.**

  For example,

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

- **public is an access specifier, which defines who can access this method. Public means that this method will be accessible by any class.**
- **static is a keyword which identifies the class related thing. This means the given method or variable is not instance related but class related. It can be accessed without creating the instance of a class.**
- **void is used to define the Return Type of the method. It defines what the method can return. Void means the method will not return any value.**
- **main is the name of the method. This method name is searched by JVM as a starting point for a program.**
- **For compilation, give the following command**
  Syntax: javac Filename.java
  Eg: javac Simple.java

- **At this stage, java compiler translates the java program into byte code which is understood by java interpreter.**
- **If the program compiles correctly, a file called Filename.class is created. This is the file containing byte code that will interpret during the execution of a program. For example, after compilation of Simple.java program, Simple.class file is created.**
- **Java interpreter reads byte code and translates into a language that the computer can understand and it can execute the code with the following command.**

  Syntax: java Filename

  Eg: java Simple

8Q: Explain about Installation of JDK 1.6

JDK means Java Development Kit. JDK is a collection of classes, java compiler and java virtual machine interpreter.

Installation of JDK 1.6

- **Run the installation process by double clicking on jdk 1.6.exe**
- **A window is displayed showing the license. Click on Accept. A custom setup dialog is shown.**
- **Click on Next to install the JDK. This will display JRE custom setup dialog.**
- **Click on Next to install JRE. (Java Runtime Environment)**
- **After the completion of the installation, click on Finish button.**

Configuration of JDK 1.6

- **Right click on MyComputer icon. This will display a context menu.**
- **Select properties from the context menu. This will show environment variables window.**
- **Select PATH from the user variables section and click on Edit if PATH is user variable already. Otherwise, click on New to enter new user.**
- **A window shown empty fields for variable name and variable value. Type PATH in variable name and C:\ProgramFiles\jdk1.6\bin;**
- **Click on OK.**

Testing the installation

- **Open the command prompt.**
- **Type javac and then press enter to test whether java software is available or not.**

Variables

9Q: What is a variable? Explain the declaration and initialization of variables.

Variable:

Variable is an identifier which is used to store the value and it can be changed during the execution of a program.

Declaration of variables

- **Syntax:**

  datatype variable-name;

- **Example:**

  int k;

- **Morethan one variables of the same data type can be declared by using comma. For example,**

  int a,b,c;

Initialization of variables

- **Initialization of variable can be defined as a process of assigning a value to the variable. This can be done directly during the declaration of a variable or after the declaration of variable.**

  Syntax:

  datatype variable-name=value; (OR)

  datatype variable-name;

  variable-name=value;

  Example:

  int k=10; (OR)

  int k;

  k=10;

- **Multiple variables that are declared with similarly type can be initialized simultaneously. This can be done by using comma operator.**

  Example:

  int a,b,c,k=0;

Dynamic initialization of variable

- **In Java, it is possible to initialize the variables dynamically. Instead of using two step process i.e. declaring variable first and then initializing. We can combine these two steps into one.**
- **The following is the example for dynamic initialization of variables:**

```
//program to read two integers and print sum
class Sum
{
    public static void main(String [ ] args)
    {
            int a,b,sum;
            a=Integer.parseInt(args[0]);
            b=Integer.parseInt(args[1]);
            sum=a+b;
            System.out.println("sum of a and b="+sum);
    }
 }
```

javac Sum.java (for compilation)

java Sum 10 20 (for execution)

- **In the above example, the variables "a" and "b" are initialized dynamically. After compilation and execution, value of sum is 30.**

Primitive data types

10Q: List the primitive data types of Java. Explain each of them in detail. (8M)

Java supports eight types of primitive data types which are grouped into four types as shown below:

i) Integers

ii) Floating point numbers

iii) Characters

iv) Boolean

i) Integers

- **The integer types are the numbers without fractional part. The following are the four integer types:**
  - **a) byte**
  - **b) short**
  - **c) int**
  - **d) long**
- **All these are signed, positive and negative values. Java does not support unsigned data types.**

a) byte :

- **It is the smallest integer type. It occupies one byte in memory.**
- **Syntax: byte var-name;**
- **Example: byte b;**

b) short :

- **It occupies 2 bytes in memory.**
- **Syntax: short var-name;**
- **Example: short s;**

c) int :

- **It occupies 4 bytes in memory.**
- **Syntax: int var-name;**
- **Example: int i;**

d) long :

- **It occupies 8 bytes in memory.**
- **Syntax: long var-name;**
- **Example: long l;**

ii) Floating point numbers

- **When we want to hold numbers containing fractional part, we use floating point numbers. There are two types of floating points. They are**
  - **a) float**
  - **b) double**

a) float :

- **It specifies a single precision value which uses 32 bits of storage. For example, 10.25**
- **Syntax: float var-name;**
- **Example: float f;**

b) double :

- **It specifies double precision value which uses 64 bits of storage. It is the good choice when we need to maintain accuracy over many iterative calculations or manipulating large valued numbers.**
- **Syntax: double var-name;**
- **Example: double d;**

iii) Characters

- **It is used to store characters. It occupies 2 bytes in memory.**
- **Syntax: char var-name;**
- **Example: char ch;**

iv) Boolean :

- **It is used for logical values. It has only one of the two possible values, true or false. Default value is false. It uses only one byte of storage.**
- **Syntax: boolean var-name;**
- **Example: boolean k;**

Identifiers

11Q: Briefly discuss about Java identifiers. (OR)
What are the naming conventions for java identifiers? (4M)

Identifier is the name given to the variables, class, methods, objects, packages. It is the sequence of characters which contain alphabets, digits, underscore and dollar sign. Length of the variable name is not limited.

The following are the rules for identifiers:

- **It should begin with a letter or underscore.**
  For example: total, _total

- **It should not begin with a digit.**

  For example: 007 is invalid
- **It is case sensitive.**

  For example: TOTAL is not same as total
- **It should not be a reserve keyword**

  For example: int int; is invalid.

  int k; is valid.
- **It should not contain any spaces. Use underscore symbol to join two words.**

  For example: int total amount; is invalid.

  int total_amout; is valid.

12Q: List out different keywords in Java (OR) Java Buzzwords (8M)

abstract	**default**	**Do**	**case**	**import**
continue	**goto**	**if**	**enum**	**static**
for	**package**	**private**	**try**	**void**
new	**synchronized**	**this**	**final**	**class**
switch	**boolean**	**break**	**char**	**finally**
assert	**instanceof**	**public**	**interface**	**native**
double	**return**	**throws**	**float**	**while**
implements	**transient**	**catch**	**short**	**long**
protected	**byte**	**extends**	**strictfp**	**volatile**
throw	**else**	**int**	**const**	**super**

Literals

13Q: Discuss about literals in Java.

There are five types of literals in Java.

i) **Integer literals.**

ii) **Floating point literals.**

iii) **Boolean literals.**

iv) **Character literals.**

v) **String literals.**

i) Integer literal

- **Integer may be decimal, octal and hexadecimal value.**
- **Decimal numbers have base 10 and they do not have leading zeros. For example, 1, 55.**
- **Octal numbers have base 8 and they range from 0 to 7. They are denoted by a leading zero. For example, 00, 05, 07.**
- **Hexadecimal numbers have base 16 and they range from 0 to 15. The hexadecimal numbers from 10 to 15 are represented by alphabets A to F respectively.**

ii) Floating point literal

- **The numbers that contains a decimal value followed by a fraction component is called floating point literal.**
- **It can be expressed in two notations:**
  - **a) Standard notation: It represents the floating point literal in the form of whole number component and a fractional component separated by decimal point.**

    For example, 10 . 75

    whole number decimal point fractional part
  - **b) Scientific notation: It consists of a standard floating point number followed by suffix. Suffix specifies a power of 10 by which the number should be multiplied. The exponent is denoted by either E or e followed by negative or positive decimal number. For example, 7.324E32, 3e+1000. In java, the default float literal is double.**

iii) Boolean literal

- **Boolean literal can take only two logical values "true" or "false". Both of them cannot be converted to numerical representation. Default value is "false".**

iv) Character literals

- **In java, character literals are indices into the Unicode character set. These literals are represented within single quotes.**
- **They use \ (slash) for entering the ASCII characters which cannot be entered directly such as newline character '\n' and tab character '\t'.**
- **They allow us to enter value of character in both octal and hexadecimal notation. For example, octal notation is '\141' and hexadecimal notation is '\u0061'.**

v) String literals

- **It consists of sequence of characters enclosed within double quotes. For example, "JAVA"**

Operators

14Q: Discuss operators in Java. (8M)

Operators are used in the expressions to perform operations on the operands. Java provides various operators and they are classified into three types. They are,

A) Unary operators.

B) Binary operators.

C) Ternary operator.

A) Unary operators:

Unary operators perform operations on a single operand. The various types of unary operators are as follows:

i) ++ (Increment) : The ++ is an operator that adds the value 1 to its operand (same as a=a+1). It can be applied in two ways:

a) Pre-Increment – If the operand is preceded by increment operator, then it is said to be pre-increment. For example, ++a.

b) Post-Increment – If the operand is succeeded by increment operator, then it is said to be post-increment. For example, a++.

ii) - - (Decrement) : The - - is an operator that subtracts the value 1 from its operand (same as a=a-1). It can be applied in two ways:

a) Pre-decrement – If the operand is preceded by decrement operator, then it is said to be pre-decrement. For example, - - a.

b) Post-decrement – If the operand is succeeded by decrement operator, then it is said to be post-decrement. For example, a- -.

B) Binary operators:

Binary operators perform operations on two operands. The various types of binary operands are as follows:

i) Arithmetic operators – The operators that can perform arithmetic operations are called arithmetic operators.

The following are the arithmetic operators:

Operator	Meaning
+	Addition
-	Subtraction
*	Multiplication
/	Division
%	Modulus

ii) Relational operators – The operators that can perform comparisons between two operands are called relational operators.

The following are the relational operators:

Operator	Meaning
<	Less than
< =	Less than or equal to
>	Greater than
> =	Greater than or equal to
= =	Equal to
!=	Not equal to

iii) logical operators – The operators than can perform logical operations between two operands are called logical operators.

The following are the logical operators:

Operator	Meaning
&&	Logical AND
||	Logical OR
!	Logical NOT

iv) Assignment operators - **An assignment operator can be defines as an operator which assigns a value to the variable or operand. The syntax is**

variable-name=value;

k=10;

v) Bitwise operators – Bitwise operators can manipulate the set of bits associated with the operands. These can be applied to integer and character type values and cannot be applied to floating point and Boolean type values.

Operator	Meaning
&	Bitwise AND
|	Bitwise OR
^	Bitwise exclusive OR
~	One's complement
<<	Left shift
>>	Right shift

v) Special operators - The special operators of java are

- **Instanceof operator – It can be defined as an object reference operator which returns true when the left hand side value is an instance of the class which is given on the right hand side. Simply, this operator is used to find that object in left hand side belongs to specified class or not.**

  For example: student instanceof BTech

  If student belongs to that class BTech, then it returns true otherwise returns false.
- **Dot operator – It can be defined as an operator which accesses the instance methods and data of a class. The dot operator can be represented as " . "**

  For example: Student s;

  s.getdata( );

C) Ternary operator

It performs operation on three operands. It is also called as conditional operator. The following is the syntax:

condition ? expr1 : expr2;

eg: a > b ? a : b;

It means, if the condition is true then " a " is big. Otherwise " b " is big.

Expressions – precedence rules and associativity

15Q: What is an expression? Explain about precedence rules and associativity. (8M) (OR)

Write the table that shows the precedence of operators in java. (3M)

Expression

An expression can be defined as combination of operators, variables and constants.

For example, c=a+b;

c=5;

Precedence and associativity

- **If an expression contains two operands with different precedence, then the order of their evaluation is determined using precedence rules.**
- **If an expression contains two operands with same precedence, then their order of evaluation is determined using associativity rules.**
- **There are two types of associativity rules names left associativity and right associativity.**
- **The left associativity determines the operators to be evaluated from left to right.**
- **The right associativity determines the operators to be evaluated from right to left.**
- **Precedence and associativity override each other using parenthesis.**
- **The below tables explains the associativity of the operators:**

Operators	Associativity
. , [ ], ( ), i++, i- -	L -> R
++i, - - i, ~ , !	R -> L
new, (type)	R -> L
* , / , %	L -> R
+ , -	L -> R
<< , >>	L -> R
< , > , < = , > =, instanceof	Non associative
= = , !=	L -> R
&	L -> R
^	L -> R
|	L -> R
&&	L -> R
||	L -> R
? :	R -> L

In the above table, the operators are listed from highest precedence to lowest precedence. In the second column, L -> R indicates left to right associativity and R -> L indicates right to left associativity.

Primitive Type Conversion and Casting

16Q: Explain Primitive type conversion and casting with examples. (8M) (OR)
Describe about type conversion. Also explain how casting is used to perform type conversion between incompatible types.

Type conversion

Def:-

Type conversion refers to the conversion of data from one data type into another data type.

- **It is one of the most important aspects of java programming.**
- **Consider an example for type conversion,**

```
int x;
float y;
x=5;
y=x;          // integer value 5 is assigned to float. Output is 5.00
```

- **If compatible types are assigned to variables, the right-hand side expression is assigned directly to the left hand side expression.**
- **But it is not possible to convert all data types since java support strict type checking.**
- **Similarly, all implicit type conversions are not supported.**
- **An automatic type conversion can be done only if the following conditions are satisfied:**
  i) The data type of the variable and value must be compatible.
  ii) The data type of left-hand side must be large in size compared to the right-hand side.
- **The int data-type can hold byte, short values since it is higher than these types, whereas long values cannot be assigned to int.**
- **The integer types and floating point types are compatible with each other.**
- **Numeric types cannot be compatible with Boolean or char. Similarly, char and Boolean are also compatible.**

Type casting

Def:-

Type casting is an explicit conversion of one type of value into another type. Simply, the data-type is stated using parenthesis before the value or expression.

- **Type casting in java must follow the given rules:**

  i) Type casting cannot be performed on Boolean variables.

  ii) Type casting of integer data-type into any other data-type is possible. But, if the casting into smaller type is performed, it results in loss of data.

  iii) Type casting of floating point type into integer type is possible, but with loss of data.

  iv) Type casting of char type into integer type is possible, but with loss of data.

- **The general form of type casting is as follows:**

  variable = (data-type) variable or expression;

- **For example, consider that there are two integer variables namely a=2, b=3. To perform b/a, actual result is 1.5. But the java compiler gives the result of b/a as integer, discarding the decimal values. Hence, type conversion is necessary to produce accurate results.**

```
int a,b;
float c;
a=2, b=3;
c = (float) b/a;        // output is 1.5
```

Flow control

17Q: Explain control structures in JAVA.

The following are the control structures in Java:

1) if (simple if)
2) if-else
3) else if
4) nested if
5) switch

} Conditional or Branching statements

6) while
7) do-while
8) for

} Looping statements

9) break
10) continue

1) if statement (simple if)

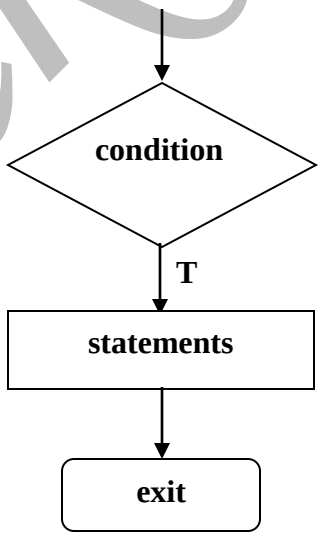
It is used to check only one condition in a program. Ie only true condition.

Syntax:

```
if(condition)
        statements;
```

If the condition is true, then the corresponding statements will be executed.

Flowchart:



Example:

```
if(x > y)
        System.out.println("x is big");
```

2) if-else statement

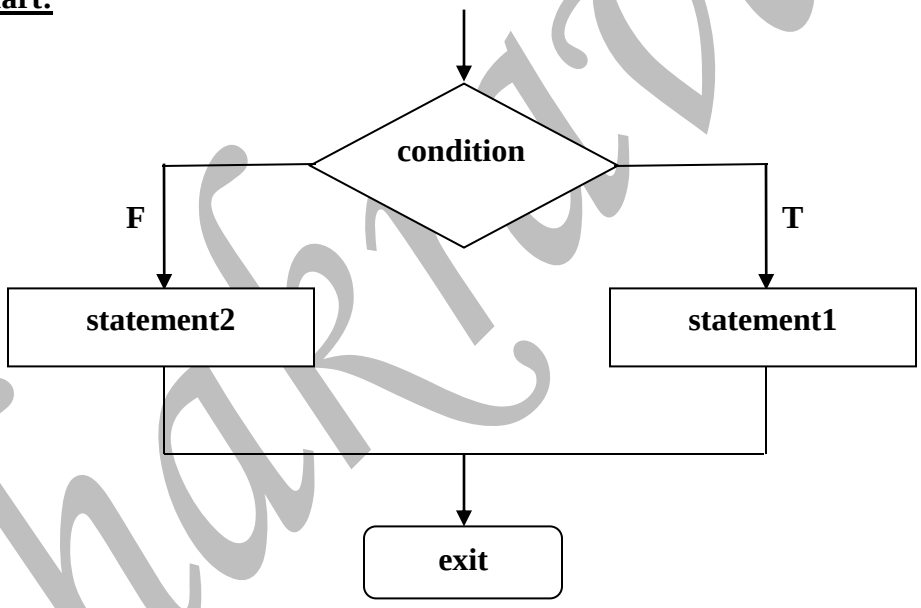
It is used to check two conditions in a program.

Syntax:

```
if(condition)
        statement1;
else
        statement2;
```

If the condition is true, then statement1 will be executed. Otherwise, statement2 will be executed.

Flowchart:



Example:

```
if( x > y )
        System.out.println("x is greater than y");
else
        System.out.println("y is greater than x");
```

3) else-if statement

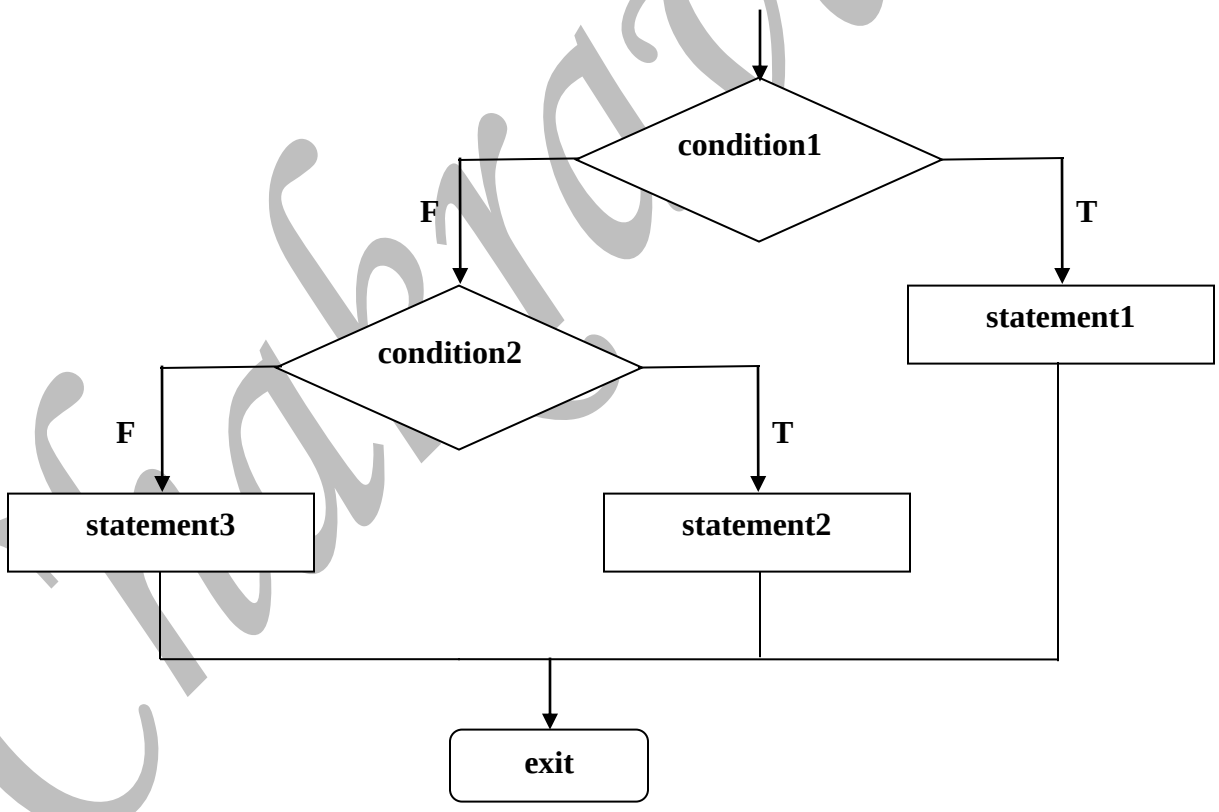
It is used to check multiple conditions in a program. Three conditions are preferable.

Syntax:

```
if( condition1 )
        statement1;
else if( condition2 )
        statement2;
else
        statement3;
```

If the condition1 is true, then statement1 will be executed. If fails, condition2 is checked. If it is true, then statement2 will be executed. If fails, statement3 will be executed.

Flowchart:



Example:

```
if ( x > y && x > z )
        System.out.println("x is biggest");
else if ( y > z )
        System.out.println("y is biggest");
else
        System.out.println("z is biggest");
```

4) nested if statement

Writing an if statement within another if statement is known as nested if. It is essential in programming since they yield a follow-up selection depending on the result of previous selection.

Syntax:

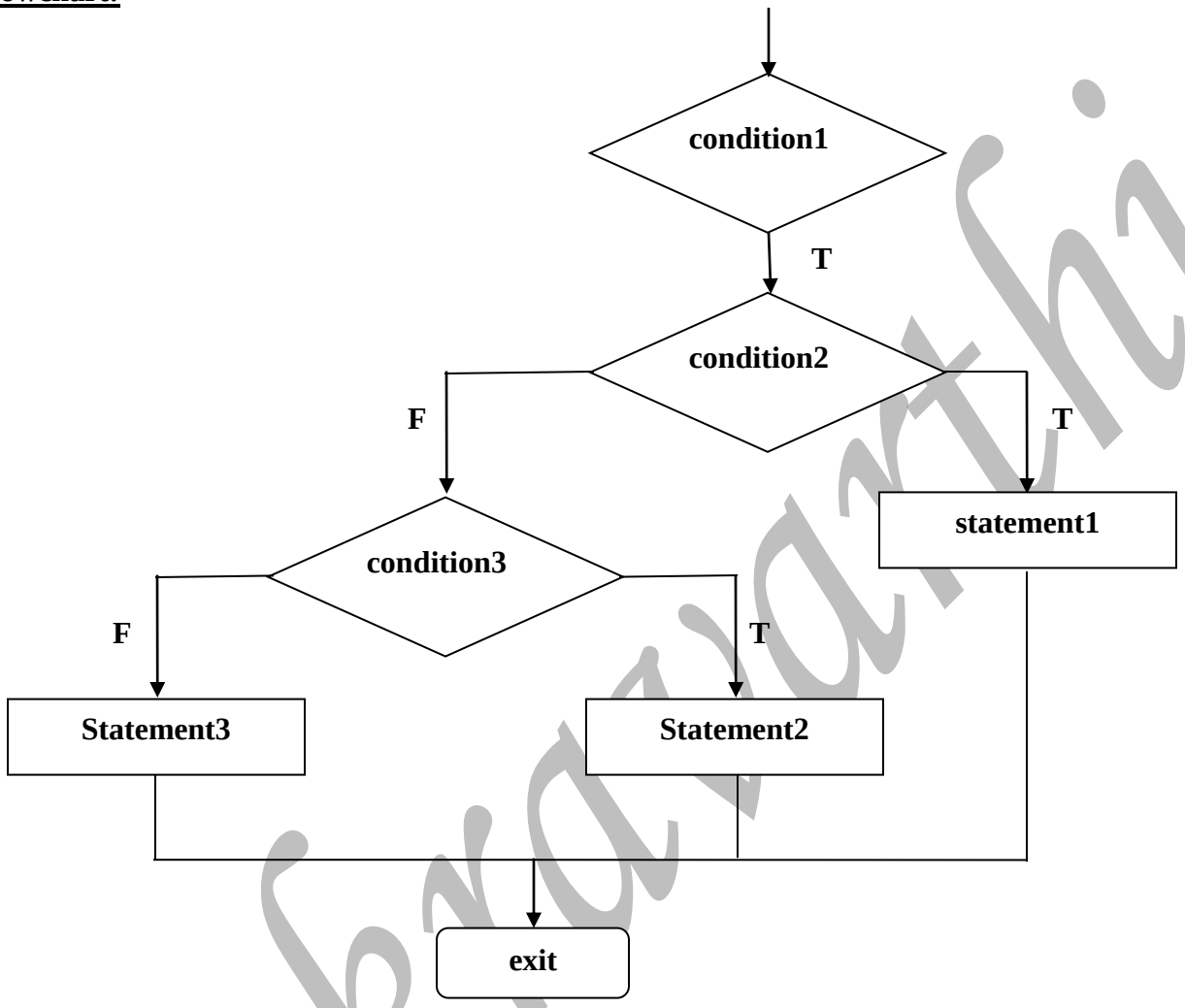
```
if( condition1 )
        if( condition2 )
                statement1;
else if( condition3 )
        statement2;
else
        statement3;
```

First condition1 is checked. If it is true, then condition2 is checked. If both the conditions are true, only then statement1 is executed. If the condition3 is true, then statement2 is executed otherwise, statement3 will be executed.

Example:

```
if ( x > y )
        if ( x > z )
                System.out.println("x is biggest");
else if ( y > z )
        System.out.println("y is biggest");
else
        System.out.println("z is biggest");
```

Flowchart:



5) switch statement

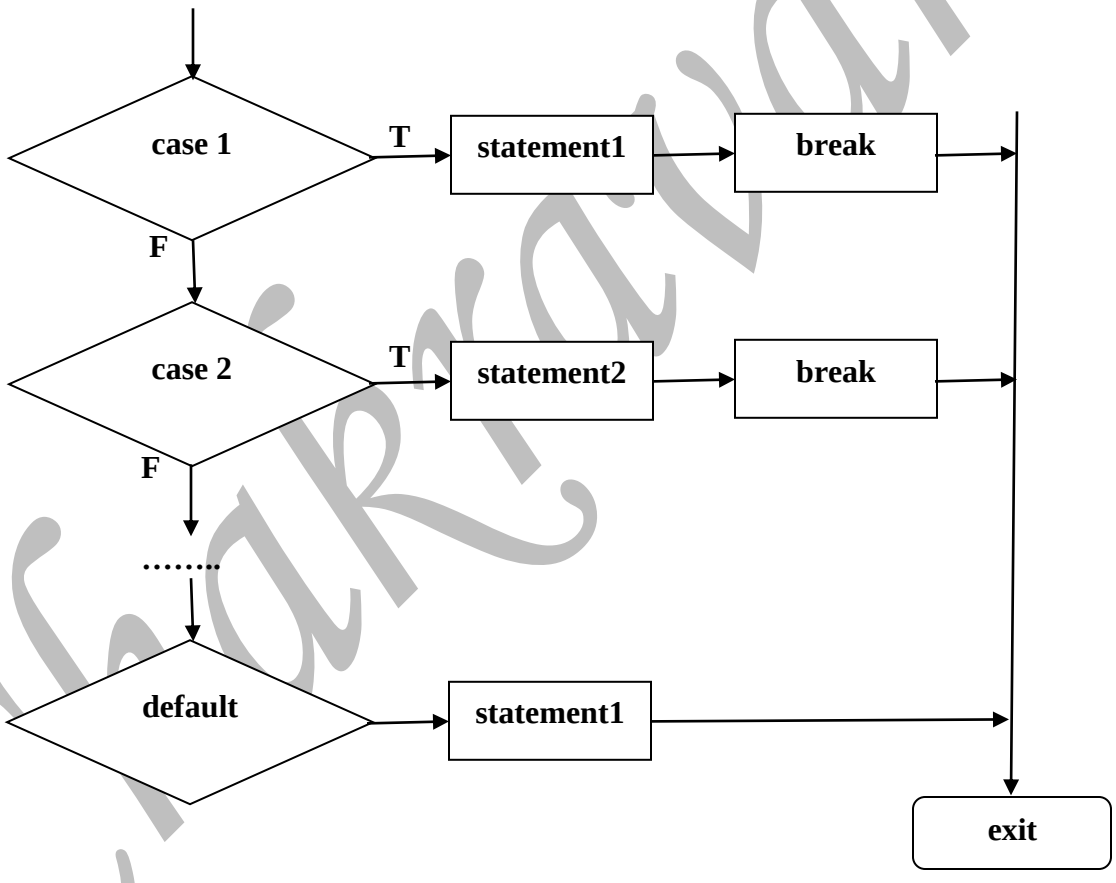
It is called as multi-way branching statement. It is used for checking multiple conditions or cases in a program, provides multiple alternatives and the user can select the required option. The break statement is used for immediate exit from the switch statement. If all the conditions are failed to execute, then default condition will be executed.

Syntax:

```
switch( variable )
{
        case 1:
                statement1;
                break;
```

```
        case 2:
                statement2;
                break;
        …
        …
        default:
                statement n;
}
```

Flowchart:



Example:

```
for( int i=1; i <= 7; i++ )
{
    switch ( i )
    {
        case 1:
            System.out.println("Sunday");
            break;
        case 2:
            System.out.println("Monday");
            break;
        case 3:
            System.out.println("Tuesday");
            break;
        case 4:
            System.out.println("Wednesday");
            break;
        case 5:
            System.out.println("Thursday");
            break;
        case 6:
            System.out.println("Friday");
            break;
        case 7:
            System.out.println("Saturday");
            break;
        default:
            System.out.println("wrong selection");
    }
}
```

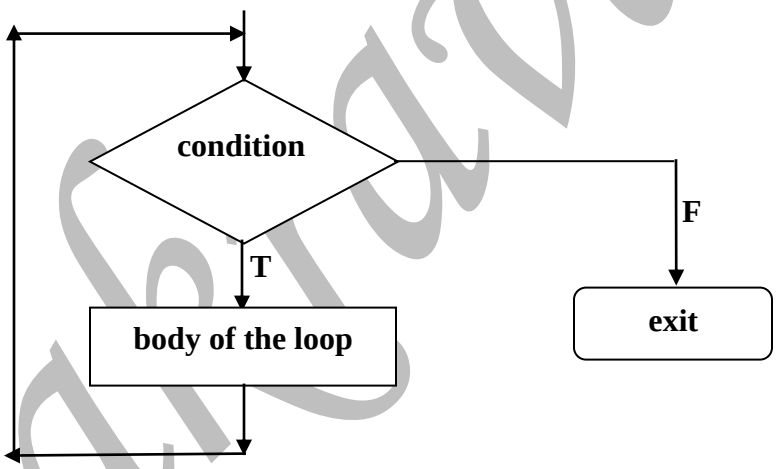
6) while loop

It will be executed repeatedly as long as the condition remains true. Before executing body of the loop, the condition is checked whether it is true or not. Hence, it is called pre-testing loop. If condition is true, then body of the loop will be executed, till the condition becomes false. Otherwise, body of the loop is not executed.

Syntax:

```
while( condition )
{
        // body of the loop;
}
```

Flowchart:



Example:

```
while ( i<= 10 )
{
        System.out.println(+i);
        i++;
}
```

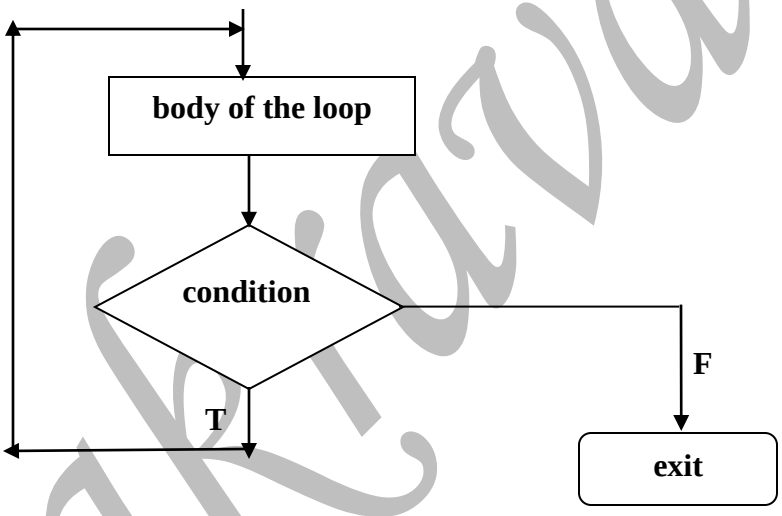
7) do-while loop

It is similar to that of while loop except that it is executed at least once. So it is called as post-testing loop. The test of condition for repeating is done after each time boy of loop is executed.

Syntax:

```
do
{
        // body of loop;
} while ( condition );
```

Flowchart:



Example:

```
do
{
        System.out.println(+i);
        i++;
}while(i<=10);
```

8) for loop

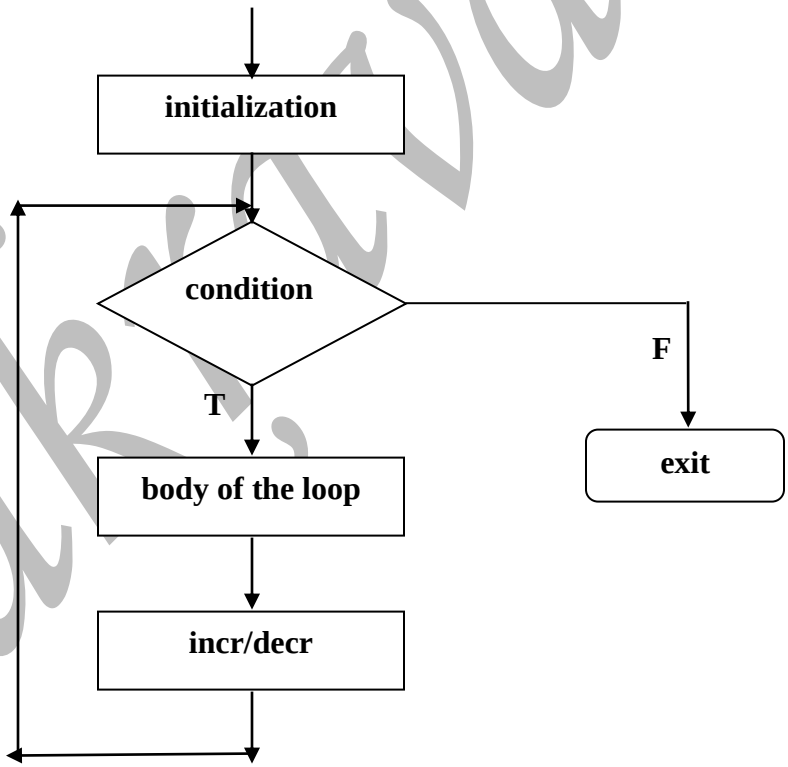
It is a flexible control structure. The length of source code can be reduced by using "for" loop. The body of the loop is executed repeatedly till the condition is false.

Syntax:

```
for( initialization ; condition ; incr/decr )
{
        // body of loop;
}
```

It contains three sections. Initialization section consists of beginning value which is assigned to the variable. Condition section species when should be the condition gets failed. Third section specifies increment or decrement value to the variable.

Flowchart:



Example:

```
for( int i = 1; i <= 10; i ++ )
        System.out.println( +i );
```

9) break

It is a reserve keyword. It can be used not only in switch statement but also in any type of control structures. It is used for immediate exit from the switch statement or iteration.

For example,

```
n=10;
for( int i=0; i<n; i++ )
{
        b = i * i;
        if( b >= n )
            break;
        System.out.println(+b);
}
```

10) continue

It is a reserve keyword. It can be used in any type of control structure. Incontrast to the break statement, the continue statement does not exist from the loop but transfers the control to the testing condition.

For example,

```
int x=0;
while( x < 15 )
{
        if( (x % 2)!= 0)
            continue;
        System.out.println(+x);
        x++;
}
```

Classes and Objects, class declaration, creating objects, methods, constructors and constructor overloading, Garbage collector, importance of static keyword, this keyword, arrays, command line arguments, nested classes or inner classes.

Classes

1Q: What is class? Write the general form of a class with example.

Class

- **A class can be defined as a template that groups data and its associated functions.**
- **The class contains two parts namely**
  a) **Declaration of data members (variables).**
  b) **Declaration of member functions.**
- **The data members of a class explain about the state of the class and the member functions explain about the behavior of the class.**
- **There are three types of variables for a class. They are**
  a) **Local variables – variables declared inside the methods.**
  b) **Instance variables – variables declared inside the class, but outside the methods.**
  c) **Class variables – variables declared inside the class with static keyword and they placed outside of the method.**

General form of a class

- **A class is declared using "class" keyword followed by name of the class.**
- **Syntax**

```
class class-name
{
    //instance variables
    datatype var;
    …
    //class variables
    static datatype var;

    datatype method1(args-list)
    {
            //local variables
            datatype var;
            …
```

```
            //body of function
        }
        datatype method2(args-list)
        {
            //local variables
            datatype var;
            …
            //body of function
        }
        …
    }
```

For example,

```
class Student
{
    char ch;                // instance variable
    static int count;       // class variable
    void getdata( )         // method
    {
        int rno=10;     // local variable
        System.out.println(+rno);
    }
}
```

Objects, creating objects

2Q: Give a brief description about object. Also explain how an object can be declared and initialized. (OR) How to assign the values to the variables in the class at the time of creation ofobject to that class? Explain with example. (8M) (OR) What is an Object? How to allocate memory for objects? (4M)

Object:-

An object can be defined as an instance of a class which is used to access the members of a class.

Declaration of an object:-

An object is similar to that of variable declaration.

Syntax:

class-name object-name;

Example:

Student s;

Here Student is the class name and " s " is the reference variable which represents object.

Initialization of Object:-

When a reference variable is declared, it is necessary to assign memory to that object. This can be done by using "new" operator. This operator allocates memory dynamically with the help of a default constructor.

Syntax:

class-name object-name = new class-name( );

Example:

Student s = new Student ( );

Accessing class members:-

An object can access the members of a class using dot (.) operator. Class members are the data members and member functions.

Syntax:

Object-name . member-name;

Example:

```
class Student
{
        int rno;
        void getdata( int r )
        {
                rno=r;
        }
        void display( )
        {
                System.out.println(+rno);
        }
}
class College
{
        public static void main(String args[ ] )
        {
                int x;
                Student s = new Student( );
                s.getdata(35);
                s.display( );
        }
}
```

Methods

3Q: What is a Method? How a method is used in the class? Explain.

Method

A method contains a set of statements which define certain actions. Each method performs a particular task. Methods are defined inside the class.

Syntax:

```
return-type method-name ( args-list )
{
        //body of method
}
```

In the above syntax, the "return-type" indicates the type of value that the method returns. The "method-name" indicates the name given to the method. It should be valid identifier. Arguments are enclosed with parenthesis. The body of method contains the operations that are to be performed on the data.

A method can be called by an object.

Syntax: `object-name . method-name(args-list)`

Example:

```
class Student
{
        int rno;
        void getdata( int r )
        {
                rno=r;
        }
        void display( )
        {
                System.out.println(+rno);
        }
}
class College
{
        public static void main(String args[ ] )
        {
                int x;
                Student s = new Student( );
                s.getdata(35);
                s.display( );
        }
}
```

4Q: Explain about various parameter passing techniques in Java.

Parameter passing:-

- **It is a technique that passes data from one function to another function.**
- **There are two parameter passing techniques:**
  a) **Call by value.**
  b) **Call by reference.**

1. Call by value:

In this method, the values of actual parameters in the calling function are copied into the corresponding parameters in the called function. But the modifications in the called function are not reflected.

Program to demonstrate call by value

```
class Sample
{
        void swap(int x,int y )
        {
                int temp;
                temp = x;
                x = y;
                y = temp;
        }
}
class  Callbyvalue
{
        public static void main(String[] args)
        {
                Sample s=new Sample();
                int a=4,b=7;
                System.out.println("Before swapping,a="+a+ "b="+b);
                s.swap(a,b);
                System.out.println("After swapping,a="+a+ "b="+b);
        }
}
```

```
javac  Callbyvalue.java
java Callbyvalue
```

Output:

```
Before calling, a=4 b=7
After calling, a=4 b=7
```

2. Call by reference:

In this method, the addresses of actual parameters are copied into the corresponding parameters in the "called function". Hence, any modifications done to the formal parameters in the "called function" cause to change.

Program to demonstrate call by reference

```
class Sample
{
        int x,y;
        Sample(int i,int j)
        {
                x =i;
                y =j;
        }
        void swap(Sample s)
        {
                int temp;
                temp = s.x;
                s.x = s.y;
                s.y = temp;
        }
}
class  Callbyref
{
        public static void main(String[] args)
        {
                Sample s=new Sample(4,5);
                System.out.println("Before swapping,a="+s.a+ "b="+s.b);
                s.swap(s);
                System.out.println("After swapping,a="+s.a+ "b="+s.b);
        }
}
```

```
javac Callbyref.java
java Callbyref
```

Output:

```
Before calling, a=4 b=5
After calling, a=8 b=2
```

Constructors and constructor overloading

5Q: Explain about Constructor Overloading in Java with example. (OR)

Write and explain the syntax of constructor with example (8M) (OR)

Illustrate constructor overloading. (8M) (OR)

Write an example program to show the calling sequence of constructors.(8M) (OR)

What is a constructor? What is its requirement in programming? Explain with program.(8M)

Def: A constructor is a special member function that gets executed automatically whenever a new objectis created.

Need for constructors:

Constructor is a special member function that avoids unexpected results caused by un-initialized variables. Generally, variables must be initialized before they are used for processing. Because, uninitialized variables contain garbage values. The constructor for the class gets automatically invoked every time new object is created.

Characteristics of constructors:

- **It has the same name as its class name.**
- **It doesn't have any return type not even void.**
- **The constructor can be declared as public.**
- **It can be overloaded.**
- **It is normally used to initialize the member of the object.**
- **There are two types of constructors. They are**
  a) **Default constructor.**
  b) **Parameterized constructor.**
- **The constructor with no arguments is called default constructor. If no constructors are defined for a class, then java automatically generates default constructor.**
- **The constructor with arguments is called parameterized constructor. If any constructors are defined for a class with parameters, then java will not create a default constructor.**
- **A class can have multiple constructors. This is called constructor overloading.**

- **All the constructors have the same name as the class name but they differ only in terms of number of arguments or datatypes or order of arguments.**

Program to demonstrate Constructor Overloading

```
class Area
{
        int a;
        Area() // default constructor
        {
                int b=10,h=5;
                a=(b*h)/2;
                System.out.println("area of triangle="+a);
        }
        Area(int s)     // parameterized constructor
        {
                a=s*s;
                System.out.println("area of square="+a);
        }
        Area(int l,int b)       // parameterized constructor
        {
                a=l*b;
                System.out.println("area of rectangle="+a);
        }
        Area(int l,int b,int h)         // parameterized constructor
        {
                a=l*b*h;
                System.out.println("area of cube="+a);
        }
}
class Constructor
{
        public static void main(String args[])
        {
                Area a1=new Area();
                Area a2=new Area(10);
                Area a3=new Area(10,20);
                Area a4=new Area(10,20,5);
        }
}
```

```
o/p:    area of traingle=25
        area of square=100
        area of rectangle=200
        area of cube=1000
```

Garbage collector

6Q: Discuss about Garbage collection. (OR) Cleaning up unused Objects. (OR)
How garbage collector plays its role? Explain. (8M) (OR)
Write about garbage collection (3M)

Dynamic allocation of objects is done using "new" keyword in java. Sometimes allocations of new objects fail due to insufficient memory. In such cases, memory space consumed by unused objects is deallocated and made available for reallocation. This is done manually in languages such as C and C++ using "delete" or "free" keyword.

In Java, a new concept called garbage collection is introduced for this purpose. It is a trouble-free approach which reclaims the objects automatically. This is done without the interference of the programmer. The objects which are not used for long time and which does not have any references are identified and their space is deallocated. This recycled space can now be used by other objects.

Garbage collection is performed during program execution. It is a time consuming process. Therefore, it is performed only when required.

Importance of static keyword and examples

7Q: Explain "static" keyword in Java with example. (OR)
How to share the data among the functions with the help of static keyword? Give example. (8M) (OR)
List the various ways of 'static' keyword usage. (4M) (OR)
Explain about the static keyword with examples. (4M) (OR)
What do you mean by static class and static method? Can we make an instance of an abstract class? Justify your answer with an example? (8M)

Static variable

- **A variable is declared with the keyword "static".**
- **Static member variable allows a common variable to be used for the entire class.**
- **The initial value assigned to the static variable is 0 as default value.**
- **Syntax: static datatype var-name;**
- **Example: static int count;**

Static methods:

- **A method is said to be static when its declaration or definition is preceded with static keyword.**
- **A static function can access only the static members of its class.**
- **Calling a static member includes name of the class as shown below:**

  class-name : : static function-name;
- **Syntax:**

  static datatype function-name ( )
  {
  …
  }

Program to demonstrate static keyword

```
class Sample
{
        static int a=10,b;
        static void printdata(int x)
        {
                System.out.println("x="+x);
                System.out.println("a="+a);
                b=a*4;
                System.out.println("b="+b);
        }
}
class Staticdemo
{
        public static void main(String[ ] args)
        {

                static.printdata(25);
        }
}
```

javac Staticdemo.java
java Staticdemo

o/p:
x=25
a=10
b=40

8Q: Explain in detail about "this" keyword. (8M or 4M)

- **Java define "this" keyword that can be used inside any method to refer the current object.**
- **It will be passed implicitly when method is called. This will be done automatically.**
- **Any member function can find the address the object and access its data by using "this" keyword.**

Program to demonstrate this keyword

```
class Area
{
	int a;
	Area()
	{
		this(10);
		int b=10,h=5;
		a=(b*h)/2;
		System.out.println("area of triangle="+a);
	}
	Area(int s)
	{
		this(10,20);
		a=s*s;
		System.out.println("area of square="+a);
	}
	Area(int l,int b)
	{
		this(10,20,5);
		a=l*b;
		System.out.println("area of rectangle="+a);
	}
	Area(int l,int b,int h)
	{
		a=l*b*h;
		System.out.println("area of cube="+a);
	}
}
class Thisdemo
{
	public static void main(String args[])
	{
		Area a1=new Area();
	}
}
```

javac Thisdemo.java
javac Thisdemo

Output:
area of cube=1000
area of rectangle=200
area of square=100
area of triangle=25

Arrays

9Q: Define Array. Explain different types of arrays in Java. (OR)

What is an array? How arrays are declared and initialized? Explain with examples. (8M)

Array

Collection of similar data items stored in continuous memory locations that share the common name is called Array. A particular value is indicated by writing the number called index or subscript in square brackets [ ] after the array name. Index starts from 0 to n - 1.

Types of arrays

a) One dimensional array.

b) Two dimensional array.

c) Multi dimensional array.

a) One dimensional array

- **It is the simplest form of an array in which list of elements are stored in contingous memory locations and accessed using only one index or subscript.**
- **Syntax:**

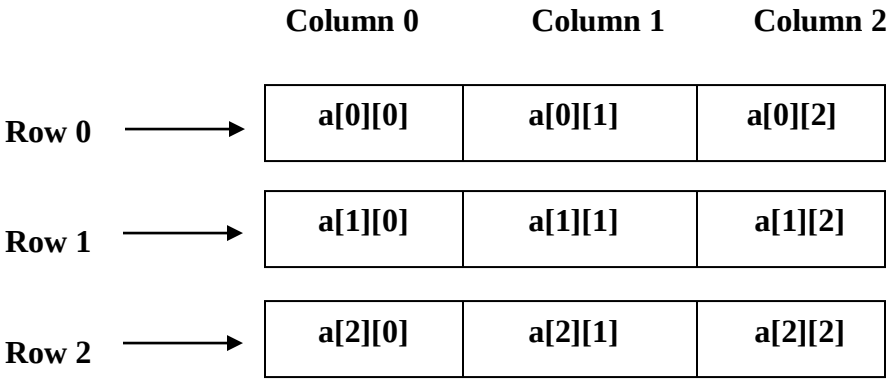
  datatype arr-name[size];
- **Example:**

  int a[5]; here, "a" is an array of 5 integers.
- **The elements are stored in memory locations as follows:**

5	10	15	20	25
a[0]	a[1]	a[2]	a[3]	a[4]

b) Tow dimensional array

- **When the data is required to store in the form of a matrix, two dimensional arrays are needed.**
- **Syntax:** **datatype arr-name[row-size][column-size];**
- **Example:** **int a[3][3];**
- **The elements are stored in memory locations as follows:**



c) Multi-dimensional array

- **The three dimensional array can be thought of as an array of arrays.**
- **It is a collection of two dimensional arrays.**
- **It consists of 2 rows and 3 columns.**
- **For example,** **int a[2] [4] [5] [6];**

Advantages of using arrays:

- **It is the simplest kind of data structure.**
- **It is easy to create, understand and implement arrays.**
- **In arrays, direct access to any element is possible.**
- **Arrays have the capability of linking data together, especially with multiple dimensions.**

Command line arguments

10Q: Discuss about command line arguments in Java.

- **In Java, the user is allowed to give input or arguments at the command prompt at the time of executing. They are called command line arguments.**

Program to read two integers and print sum

```
class Sum
{
	public static void main(String[] args)
	{
		int a,b,sum;
		a=Integer.parseInt(args[0]);
		b=Integer.parseInt(args[1]);
		sum=a+b;
		System.out.println("sum of a and b="+sum);
	}
}
```

Compilation: javac Sum.java

Execution: java Sum 10 20

Output: sum of a and b=30

Inheritance, Types of inheritance, super keyword, final keyword, overriding and abstract class. Interfaces, Creating packages, using packages, Importance of CLASSPATH and java.lang.package. Exception handling, Importance of try, catch, throw, throws, finally block, User defined exceptions, Assertions

Inheritance, Types of inheritance

Q1: Define Inheritance. What are different types of inheritance? (16M) OR
What are the different forms of inheritance? Explain. (8M) OR
What are the benefits of inheritance? Explain various forms of inheritance with suitable code segments.(16M)

- **The mechanism of deriving a new class from old class is called inheritance. Getting properties from base class to the derived class is called inheritance.**
- **The class which gives properties to the other classes is called base class or super class or parent class. The class which accepts or receives properties from the other classes is called derived class or sub-class or derived class.**
- **Inheritance allows subclasses to inherit all the data members and member functions of their parent classes.**
- **The main advantage of inheritance is code reusability.**
- **The "extends" keyword is used in the concept of inheritance. It is used to derive the classes from another class.**
- **The "extends" keyword is used in the sub class declaration and it is followed by super class name.**
- **Subclass inherits all the data members and member functions from the parent class.**

Syntax:

```
class derived-class-name extends base-class-name
{
        Variables-declaration;
        Member-functions definition;
}
```

The object must be declared to the derived class only.

Types of inheritance

1. **Single inheritance or inheritance**
2. **Multilevel inheritance**
3. **Multiple inheritance**
4. **Hybrid inheritance**
5. **Hierarchical inheritance**

1. Single inheritance

Derivation of one subclass from only one super class is called single inheritance.

Super class or base class or parent class

Subclass or derived class or child class

Syntax:

```
class A
{
        //code
}
class B extends A
{
        //code
}
```

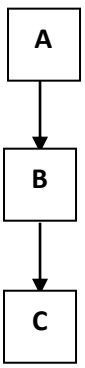
Program for single inheritance

```
class Student
{
        int rno;
        void getrno(int x)
        {
                rno=x;
        }
        void displayrno()
        {
                System.out.println("rno="+rno);
        }
}
class Marks extends Student
{
        int m1,m2,m3,total;
        void getmarks ( intx, inty, int z)
        {
                m1=x;
                m2=y;
                m3=z;
        }
        void displaymarks( )
        {
                System.out.println("m1="+m1);
                System.out.println("m2="+m2);
                System.out.println("m3="+m3);
        }
```

```
        void calculate()
        {
                total=m1+m2+m3;
                System.out.println("total="+total);
        }
}
class Single
{
        public static void main(String[] args)
        {
                Marks m=new Marks();
                m.getrno(10);
                m.getmarks(70,80,90);
                m.displayrno( );
                m.displaymarks( );
                m.calculate( );
        }
}
javac Single.java
java Single
rno=10
m1=70
m2=80
m3=90
total=240
```

2. Multilevel inheritance

Derivation of a class from another derived class is called multilevel inheritance. In this type, derived class can act as base class for some other derived class.



Base class

Derived class

Derived class

Syntax:

```
class A
{
        //code
}
```

```
class B extends A
{
        //code
}
class C extends B
{
        //code
 }
```

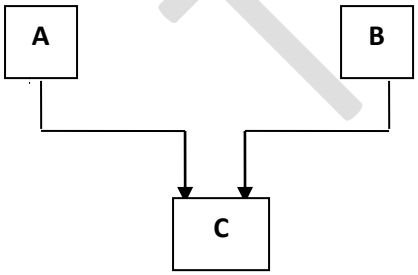
Program for multilevel inheritance

```
class Student
{
        int rno;
        void getrno(int x)
        {
                rno=x;
        }
        void displayrno()
        {
                System.out.println("rno="+rno);
        }
}
class Marks extends Student
{
        int m1,m2,m3;
        void getmarks(intx,inty,int z)
        {
                m1=x;
                m2=y;
                m3=z;
        }
        void displaymarks()
        {
                System.out.println("m1="+m1);
                System.out.println("m2="+m2);
                System.out.println("m3="+m3);
        }
}
class Result extends Marks
{
        int total;
        float avg;
        void calculate()
        {
                total=m1+m2+m3;
                System.out.println("total="+total);
                avg=total/3;
                System.out.println("average="+avg);
        }
}
```

```
class Multilevel
{
        public static void main(String[] args)
        {
                Result r=new Result();
                r.getrno(10);
                r.getmarks(70,80,90);
                r.displayrno();
                r.displaymarks();
                r.calculate();
        }
}
javac Multilevel.java
java Multilevel
rno=10
m1=70
m2=80
m3=90
total=240
average=80.0
```

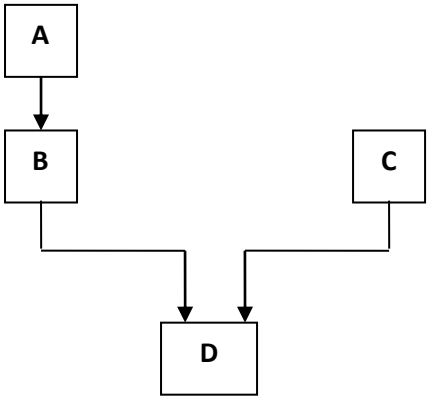
3. Multiple inheritance

Derivation of one class from two or more super classes is called multiple inheritance. But java does not support multiple inheritance directly. It can be implemented by using interface concept.



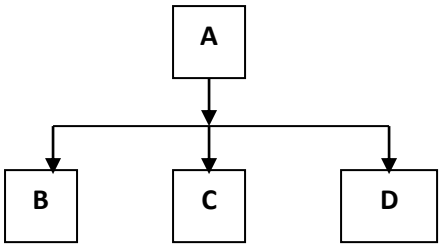
4. Hybrid inheritance

Hybrid inheritance is the combination of both single and multiple inheritances.



5. Hierarchical inheritance

Derivation of several classes from a single base class is called Hierarchical inheritance.



Syntax:

```
class A
{
        //code
}
class B extends A
{
        //code
}
class C extends A
{
        //code
}
class D extends A
{
        //code
}
```

Super keyword

2Q: Briefly discuss about "super" keyword in java

- **The "super" keyword is used to access the base class methods.**
- **If the base class and derived class have methods and variables with same name then the members of base class will be accessed by "super" keyword.**
- **Syntax:** **super.variable-name;**
  super.method-name( );
- **"super" keyword is used to access the constructor of base class.**
- **A subclass constructor is used to access the instance variable of both the subclass and the base class. The subclass constructor uses the keyword "super" to invoke the constructor method of the parent class.**
- **"super" may be used only within the subclass constructor method**
- **The call to the base class constructor must appear as the first statement with in the subclass constructor.**

- **If parameterized constructor is used, then the signature is matched with the parameters passed to the "super" method as super(parameter-list);**

Program to demonstrate "super" keyword

```
class Student
{
        intl,b;
        public Student(int x,int y) //parameterized constructor
        {
                l=x;
                b=y;
        }
        int area()
        {
                return l*b;
        }
}

class Result extends Student
{
        int h;
        public Result(intx,inty,int z) //parameterized constructor
        {
                super(x,y);
                h=z;
        }
        int volume()
        {
                return l*b*h;
        }
}

class  Superdemo
{
        public static void main(String[] args)
        {
                Result r=new Result(2,3,4);
                System.out.println("area="+r.area());
                System.out.println("volume="+r.volume());
        }
}
```

```
javac Superdemo.java
java Superdemo
area=6
volume=24
```

final keyword

3Q: Discuss about "final" keyword. (OR)

With suitable code segments illustrate various uses of 'final' keyword. (8M)

A final is a keyword used for three purposes. They are

1) final as constant:

A variable can be declared as constant with the keyword final. It must be initialized while declaring. Once we initialized the variable with final keyword, it cannot be modified in the program. This is similar to const in C language.

For example,

```
finalint x=10;
X=45; //error
```

2) final with methods (OR) final to prevent overriding:

A method that is declared as final cannot be overridden in a subclass method.

For example,

```
class A
{
    final void show( )
    {
        System.out.println("it is defined with final method");
    }
}
class B extends A
{
    void show( ) // raises compile time error
    {
        System.out.println("overriding final method");
    }
}
```

3) final with classes (OR) final to prevent inheritance:

The class that is declared as final, implicitly declares all of its methods as final. The class cannot be extended to its subclass.

For example,

```
final class A
{
    public void display( )
    {
        System.out.println("This is final class method");
    }
}
```

```
class B extends A                //error
{
        public void display( )
        {
                System.out.println("Hello");
        }
}
```

Overriding

4Q: What is Overriding or Method Overriding? Explain with example

Method overriding

The member function of a base class is said to be overridden if the function is defined in the subclass with same name as that of function of base class. It is nothing but redefining the base class function in the subclass.

Program to demonstrate Method Overriding

```
class A
{
        inti,j;
        public A(inta,int b)
        {
                i=a;
                j=b;
        }
        void show()
        {
                System.out.println("i="+i+"\n"+"j="+j);
        }
}
class B extends A
{
        int k;
        B(inta,intb,int c)
        {
                super(a,b);
                k=c;
        }
        void show()
        {
                System.out.println("k="+k);
        }
}
class  Override
{
        public static void main(String[] args)
        {
                B b=new B(1,2,3);
```

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

javac Override.java
java Override
k=3

abstract class

5Q: Write in brief about abstract class. (OR)

What is abstract class? Discuss with example. (3M)

Abstract class

- **Java defines abstract classes using the keyword "abstract". The class that doesn't have body is called abstract class.**
- **An abstract class can contain subclasses in addition to abstract methods.**
- **If the method containing in the subclasses does not override the methods of base class then they will be considered as abstract classes. So, all the subclasses must provide the implementation for all the base class's abstract methods.**
- **Abstract class cannot be instantiated.**
- **The class that contain one or more abstract methods need to be declared as abstract class.**
- **There will be no objects for abstract class because it doesn't provide complete implementation.**
- **If any attempt is made to create the object for this class, a compile-time error will be generated.**

Syntax:

```
abstract class class-name
{
        //code
}
```

Program to demonstrate abstract

```
abstract class Baseclass
{
        abstract void display( );
}
class Subclass extends Baseclass
{
        void display( )
        {
                System.out.println("Sub class");
        }
}
```

```
classAbstractdemo
{
        public static void main( String args[ ] )
        {
                Subclass s = new Subclass( );
                s.diaplay( );
        }
}
```

Method overloading

6Q: Explain about Function Overloading or Method Overloading in Java with example.

Def:- In Java, it is possible to define two or more methods within the same class that share the same name as long as these methods have different set of parameters ie based on the number of arguments, the datatypes of arguments and the order of arguments. Then the methods are overloaded and the process is referred as Method overloading or Function overloading.

- **When overload method is called, the java compiler selects the proper method by examine the number of arguments, the datatype and the order arguments in the call.**
- **Method overloading is commonly used to create several methods with same name that perform fast accessing.**

Program to demonstrate Function Overloading

```
class Function
{
        int a;
        void area()
        {
                int b=10,h=5;
                a=(b*h)/2;
                System.out.println("area of triangle="+a);
        }
        void area(int s)
        {
                a=s*s;
                System.out.println("area of square="+a);
        }
        void area(intl,int b)
        {
                a=l*b;
                System.out.println("area of rectangle="+a);
        }
        void area(intl,intb,int h)
        {
                a=l*b*h;
                System.out.println("area of cube="+a);
        }
}
```

```
class Overloading
{
        public static void main(String args[])
        {
                Function f=new Function();
                f.area();
                f.area(10);
                f.area(10,20);
                f.area(10,20,5);
        }
}
```

o/p: area of traingle=25
area of square=100
area of rectangle=200
area of cube=1000

7Q: Write the differences between method overloading and method overriding.

Method overloading	Method overriding
1. It is a method in which two or more functions have the same name, but differ in their signatures.	1. It is a method in which a derived class redefines the member function of a base class. Signatures of both the functions are same but differ in their implementations.
2. The number of parameters can vary in both the functions.	2. The number of parameters passed should be same.
3. Both functions can have different return types.	3. Derived and base member functions should have same return types.

Interfaces

8Q: Define interface. Explain how multiple inheritance is achieved using interfaces. (OR) Discuss how interfaces are defined and implemented. (OR)

What is interface? How to create it and access it? Explain with example.(8M)

Write in detail about accessing implementations through interface references

Def:

Interface is a collection of method declarations and constants that one or more classes of objects will use. Interface is same as class except that it consists of the methods that are declared have no method definition. They end with semicolon after the parameter list.

- **Syntax:**

```
access-specifier interface interface-name
{
        type var1=value;
        type var2=value
        …
        returntype method-name1(parameters-list);
        returntype method-name2(parameters-list);
        …
}
```

- **Here, access specifier is public or not used. Public access specifier indicates that the interface can be used by any class. Otherwise, the interface will be accessible to the class that are defined in the same package.**
- **"interface" keyword is used to declare the class as an interface.**
- **"interface-name" is the name of the interface and it is a valid identifier.**
- **Variables declared inside the interface are implicitly final and static. They cannot be changed in the implementation of classes.**
- **All the methods in an interface are implicitly abstract methods. Method implementation is given in next stages.**
- **The main difference between a class and interface is class contain methods with method definitions and the interface contain only abstract methods.**
- **For example,**

```
public interface Shape
{
        int radius=2;
        public void area(int a);
}
```

Implementing interfaces:

- **Once the interface has been defined, one or more classes can implement that interface. "implements" keyword used for implementing the classes.**
- **Syntax:**

```
class class-name implements interface1, interface2, …
{
        …..
        …. class body
}
```

- **If a class implements more than one interface, the interfaces are separated with a comma operator. The methods that implement an interface must be declared public.**

Program to demonstrate multiple inheritance using interfaces

```
class Marks
{
        int s1,s2,s3;
        void getdata(intx,inty,int z)
        {
                s1=x;
                s2=y;
                s3=z;
        }
        void display()
        {
                System.out.println("subject1="+s1);
                System.out.println("subject2="+s2);
                System.out.println("subject3="+s3);
        }
}
interface Sports
{
        int smarks=55;
        public void show();
}
class Result extends Marks implements Sports
{
        int total;
        public void show()
        {
                display();
                total=s1+s2+s3+smarks;
                System.out.println("Total marks="+total);
        }
}
class Multiple
{
        public static void main(String[] args)
        {
                Result r=new Result();
                r.getdata(35,67,89);
                r.show();
        }
}
javac Multiple.java
java Multiple
```

o/p: subject1=35
subject2=67
subject3=89
Total marks=246

9Q:. How interfaces can be extended? Explain.

- **In Java, interfaces are extended using "extends" keyword.**
- **The extending interface is called sub-interface and the interface that is being extended is called super-interface.**
- **The sub-interface can access all the members of its super-interface.**
- **The class which implements that sub-interface must provide the implementations for all the methods of both super-interface and sub-interface.**
- **Syntax:**

```
interface subinterface-name extends superinterface-name
{
        //code;
}
```

Program to extend interfaces

```
interface Marks
{
        void display();
}
interface Sports extends Marks
{
        void show();
}
class Student implements Sports
{
        void display( )
        {
                System.out.println("Displaying marks");
        }
        void show()
        {
                System.out.println("Displaying sports marks");
        }
}
class Sample
{
        public static void main(String[] args)
        {
                Students=new Student();
                s. display( );
                s. show( );
        }
}
javac Sample.java
java Sample
```

o/p: Displaying marks
Displaying sports marks

10Q: Write the differences between interfaces and abstract classes

Interface	Abstract class
1. It is declared using "interface" keyword.	1. It is declared using "abstract" keyword.
2. It supports multiple inheritance.	2. It does not support multiple inheritance.
3. It does not contain constructors.	3. It contains constructors.
4. Interface methods do not use any access specifier with it since all the methods are public by default.	4. Abstract methods can use public, private, protected and default specifier.
5. It uses "implements" keyword for implementing subclasses.	5. It uses "extends" keyword for extending the subclasses.
6. It does not have main( ) method.	6. It has main( ) method.
7. It can have only static, final variables by default.	7. It can have static, final variables with any access specifier.
8. It contains only abstract methods.	8. It contain both abstract and concrete methods.

Packages

11Q: What is package? Explain how a package is defined, created and accessed with eg. (OR) How to create packages and use them in java? (8M)

- **Package is a collection of related classes and interfaces. It groups them based on their functionality. It acts as container for the class.**
- **Java packages are classified into two types.**
  1. **Java API**
  2. **User defined packages**
- **Java API(Application Programming Interface) packages or pre-defined packages or built-in packages are defined by the system. Some of the examples are java.util, java.lang, java.awt, java.applet etc.**
- **User defined packages are defined by the user.**

Defining and creating package:

- **To define a package, place "package" command as the first statement in the java source file. So that any class declared within that file belongs to the specified package.**

- **Syntax:** **package pack-name;**

  where pack-name is the name of the package.

- **Example:**

```
package sample;
public class Addition
{
        public void add(inta,int b)
        {
                System.out.println("Sum="+(a+b));
        }
}
```

- **The class that is defined in the package must be start with the public access specifier. So, that it can be accessible by any another of them. If it is not public, it is possible to access only in that package.**
- **Java uses file system directories to store packages. We save the program with Addition.java and complie the package is as**

  javac –d . Addition.java

  Due to this compilation, sample directory is automatically created and .class file stored in that directory.
- **Package creation has completed.**

Accessing a package or importing a package:

- **The package information is now including in our actual program by means of "import" statement. "import" is a keyword that links the package with our program. It is placed before the class definitions.**

  import sample.*;

  or import sample.Addition;

Program to demonstrate package

```
package sample;
public class Addition
{
        public void add(inta,int b)
        {
                System.out.println("Sum="+(a+b));
        }
}
```

javac –d . Addition.java // only compilation, no execution

C:/>dir

Observe that there is sample directory automatically created. In that, Addition.class is one member.

```
importsample.Addition;
class  PackageDemo
{
        public static void main(String[] args)
        {
                Addition a=new Addition();
                a.add(3,4);
        }
}
```

```
javac PackageDemo.java
java PackageDemo
Sum=7
```

Importance of CLASSPATH and java.lang package

12Q: What is CLASSPATH ? Explain.

The path of a third party as well as user-defined classes is called classpath. When a class file is executed, the tools such as javac and java of jdk are searched for the class in the specified classpath. The classpath will become current directory by default. If the files are not found in this path, then the classpath must be set.

Configuration of JDK 1.6

- **Right click on MyComputer icon. This will display a context menu.**
- **Select properties from the context menu. This will show environment variables window.**
- **Select PATH from the user variables section and click on Edit if PATH is user variable already. Otherwise, click on New to enter new user.**
- **A window shown empty fields for variable name and variable value. Type PATH in variable name and C:\ProgramFiles\jdk1.6\bin;**
- **Click on OK.**

13Q: What is java.langpackage ?

It holds all the classes and interfaces that are basic building blocks of java. It is imported by default when a class is created. It is not required to be imported explicitly.

Exception handling – importance of try, catch, throw, throws, finally keywords

14: What is an Exception? Explain how an exception can be handled in Java?

Def: Exception

An exception can be defined as error which can be occurred at runtime or execution time.

Def: Exception handling

Exception handling can be defined as a mechanism of handling exceptions that can be occurred at run-time. This mechanism is commonly used to prevent the malfunctions such as computer deadlock or computer hanging and some examples of runtime exceptions are divide by zero, array out of bound.

Exception handling mechanism

- **The runtime exception can be handled by using five keywords namely try, catch, throw, throws and finally.**
- **The code that generates exception is usually placed in try block.**
- **If an exception is occurred, execution flow finds the catch block and leaves the try block.**
- **The catch block handles the exception and generates message specifying the type of exception.**
- **There is no rule that try block must contain corresponding catch blocks. Each try block may contain zero or multiple catch blocks.**
- **When an exception is thrown and if it is matched with any of the catch block, then the statements of corresponding catch block can be executed.**
- **Otherwise, the catch block can be skipped and the statements present next to catch block will be executed.**
- **The finally block is declared after all the catch blocks whose code can be executed irrespective to the occurrence of exception.**
- **Finally block is optional.**
- **If a method throws an exception, then it can be handled by catch block. This exception can be thrown using "throws" clause.**
- **When control leaves the throws block, then it cannot return back to the "throws" clause.**

Syntax:

```
try
{
        //block of code to check exception
}
catch(Exception-type1 exception-object)
{
        //code to handle exceptions
}
catch(Exception-type2 exception-object)
{
        //code to handle exceptions
}
finally
{
        //code to be executed before/after try-catch block
}
```

Example program

```
classExceptionHandler
{
        public static void main( String args[ ] )
        {
                try
                {
                        int a=10,b;                          } this block may generate
                        b=a / 0;                             } exception.
                        System.out.println("b value="+b);    }
                }
                catch(ArithmaticException e)
                {
                        System.out.println("denominator must not be zero");
                }
                finally
                {
                        System.out.println("Quit");
                }
        }
}
```

15Q: Explain the usage of "try" keyword with an example program.

Example program

```
classExceptionHandler
{
        public static void main( String args[ ] )
        {
                try
                {
                        int a=10,b;                          } this block may generate
                        b=a / 0;                             } exception.
                        System.out.println("b value="+b);    }
                }
                catch(ArithmaticException e)
                {
                        System.out.println("denominator must not be zero");
                }
                finally
                {
                        System.out.println("Quit");
                }
        }
}
```

16Q: Discuss about "catch" statement with example program.

Example program

```
classExceptionHandler
{
        public static void main( String args[ ] )
        {
                try
                {
                        int a=10,b;
                        b=a / 0;
                        System.out.println("b value="+b);
                }
                catch(ArithmaticException e)
                {                                                    } exception
                        System.out.println("denominator must not be zero"); } caught
                }                                                    } here.
                finally
                {
                        System.out.println("Quit");
                }
        }
}
```

17Q: Discuss how "throw" statement is used in Java. Explain with example.

- **Syntax:**

  throwthrowable-instance;

  Here throwable instance must be an object type of throwable class.

- **There are two ways to obtain throwable object.**

  1. **Using a parameter into catch block.**

     For example,

```
catch(NullPointerException e)
{
        ….
        throw e;
}
```

  2. **Creating a throw statement with "new" operator**

     For example,

```
throw new NullPointerException;
```

Program to demonstrate throw statement

```
classDemoThrow
{
    int x=5, y=0,z;

    int division( )
    {
        try
        {
            if( y = = 0 )
                throw new ArithmeticException("Division by Zero");
            else
                return x / y;
        }
        catch(ArithmeticExceptionae)
        {
            System.out.println("Exception caught at division");
            throwae;     //throws exception explicitly
        }
    }
    public static void main( String args[ ] )
    {
        try
        {
            z=division( );
            System.out.println(+z);
        }
```

```
            catch(ArithmeticExceptionae)
            {
                  System.out.println("Exception caught again in main");
            }
      }
}
```

18Q: Explain in detail about "throws" statement along with example.

throws keyword

Generally any method causes exceptions. But some methods rise exceptions which could not be handled. If it is the case, then the method should atleast inform the method callers that it my throw exceptions. The keyword "throws" is used for this purpose.

Syntax:

return-type method-name(parameter list) throws Exception1, Exception2, …

Example program to demonstrate throws keyword

```
classThrowsDemo
{
      int x=5, y=0,z;

      int division( )
      {
            try
            {
                  if( y = = 0 )
                        throws new ArithmeticException("Division by Zero");
                  else
                        return x / y;
            }
            catch(ArithmeticExceptionae)
            {
                  System.out.println("Exception caught at division");
                  throwsae;
            }
      }
      public static void main( String args[ ] ) throws IOException
      {
            try
            {
                  z=division( );
                  System.out.println(+z);
            }
```

```
            catch(ArithmeticExceptionae)
            {
                System.out.println("Exception caught again in main");
            }
        }
}
```

19Q: How are "finally" statement used in Java? Explain in detail.

Example program

```
classExceptionHandler
{
        public static void main( String args[ ] )
        {
            try
            {
                int a=10,b;
                b=a / 0;
                System.out.println("b value="+b);
            }
            catch(ArithmaticException e)
            {
                System.out.println("denominator must not be zero");
            }
            finally
            {
                System.out.println("Quit");
            }
        }
}
```

User defined exceptions

20Q: Define user-defined exception. How it can be created? Explain

User-defined Exception

An exception which can be created by the user manually can be called as user-defined exception. These exceptions can often handled by Java. While creating user-defined exceptions, user must satisfy the below steps:

Step1: The user-defined exceptions must be created as child exception class of throwable class.

Step2: If the user-defined exception is a checked exception, the Exception class must be extended.

Step3: When the runtime exception is declared as user-defined exception, then the runtimeException class must be extended.

```
class A extends Exception
{
        A ( String s1 )
        {
                super(s1);
        }
}
class Userdefined
{
        public static void main(String args[ ])
        {
                try
                {
                        throw new A("demo");
                }
                catch( Exception e)
                {
                        System.out.println(e);
                }
        }
}
```

output:

A : demo

Assertions

21Q: Write short notes on Assertions.

- **The Boolean expressions which are used for testing the code are called Assertions.**
- **They are used for creating reliable, correct and robust programs.**
- **Assertions are generally used in testing and development phases.**
- **Programmers use them to get an assurance about certain conditions.**
- **Examples of such conditions might be about a number positive or negative, array / reference is null or not.**

- **Assertions are declared in Java using " assert " keyword. By default, assertions are disabled. They need to be enabled using the options – ea and enabled assertion can be disabled using – da option.**

Example program

```
class Assertion
{
        static void verify( int x )
        {
                assert x > 0 : "The value should be positive";
                System.out.println("value ok" +x);
        }
        public static void main( String args [ ] )
        {
                verify(Integer.parseInt(args[0])
        }
}
```

22Q: Write the differences between unchecked and checked exceptions.

Unchecked Exception	Checked Exception
1. All the subclasses of runtime exception are called unchecked exceptions.	1. All the subclasses of throwable class, except runtime exceptions are called as check exceptions.
2. Program compiles even if we do not catch the exception, whether the problem really exists in the program or not.	2. It should be thrown with keyword "throws" else the program does not compile.
3. JVM checks these and terminates the program, if exception occurs in the program, when the programmer does not provide try-catch block.	3. Programmer has to check himself, as JVM does not check these exceptions.

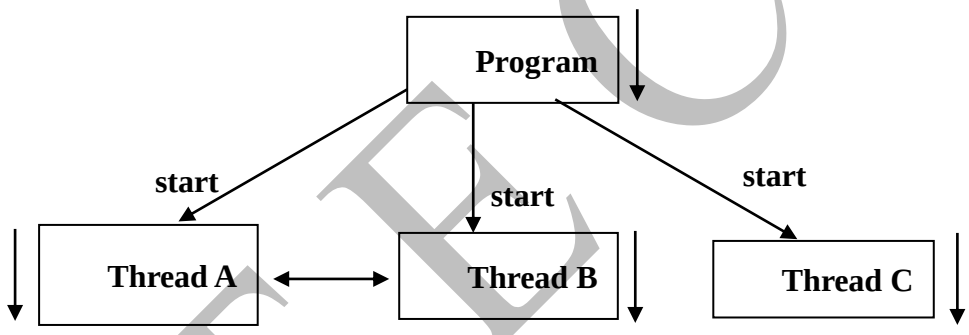
Multithreading: Introduction, thread life cycle, creation of threads, thread priorities, thread synchronization, communication between threads, reading data from files and writing data to files, random access file.

Introduction

Q1: Explain about Multithreading with example. (OR)

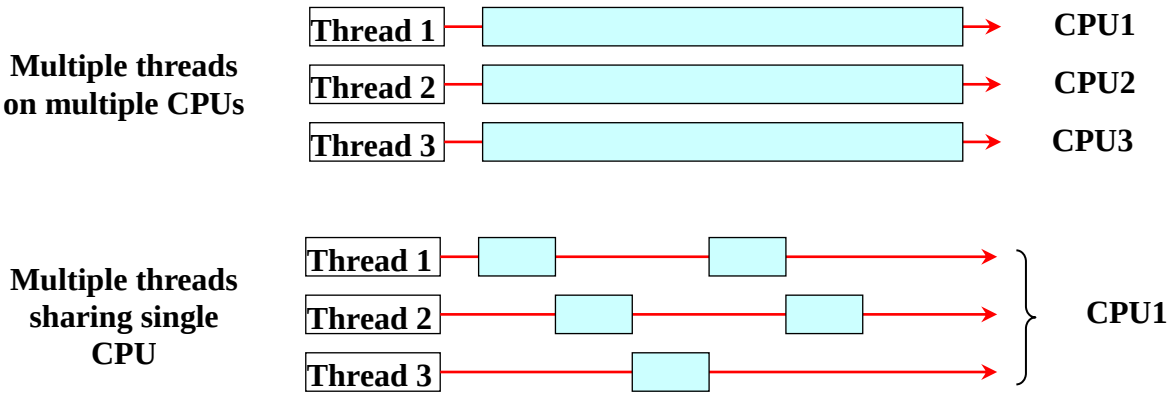
What is light weight process?. Explain.(3M)

- **Thread is a light weight process.**
- **Thread is a single sequential flow of control within a program.**
- **Thread can be defined as a set of instructions that are executed independently.**
- **Multithreading is a powerful programming tool that divides a single program into multiple subprograms and each subprogram is called thread, which are executed separately.**

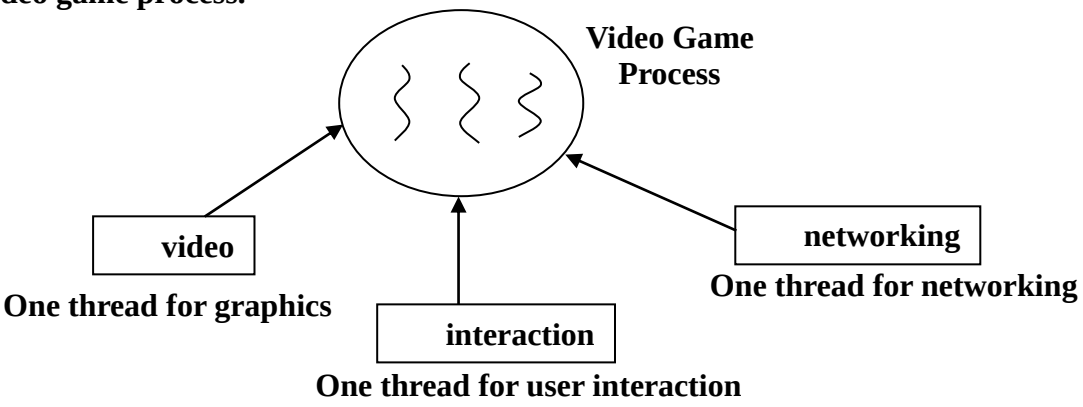


Threads may switch or exchange data/results

- **Multithreading enables programmers to write efficient programs by making maximum use of CPU.**



- **Multiple threads can work together to accomplish a common goal. For example, video game process.**



Advantages

- **Threads are used to enhance parallel processing. Hence, they decrease the execution time of a program.**
- **They are used to increase response to the user.**
- **They are used to utilize the idle time of the CPU.**
- **They are used to prioritize your work depending on priority.**
- **Thread only runs when needed. So Threads can give better performance.**
- **A thread can share the data with other threads.**

Disadvantages

- **Multiple threads can lead to deadlock.**
- **Overhead of switching between threads**

Q2: Write the differences between multi-tasking and multi-threading

Multi-tasking	Multi-threading
1. It is a process of executing multiple tasks simultaneously.	1. It is a process of dividing single program into multiple subprograms and executes them simultaneously.
2. It is an operating system concept.	2. It is a programming concept.
3. It is less efficient.	3. It is highly efficient.
4. It can implement efficient programs in operating system.	4. It can implement efficient programs in programming.
5. It executes multiple programs simultaneously.	5. It executes multiple subprograms of an individual program simultaneously.

Thread life cycle

Q3: Explain thread life cycle. (7M) (OR)

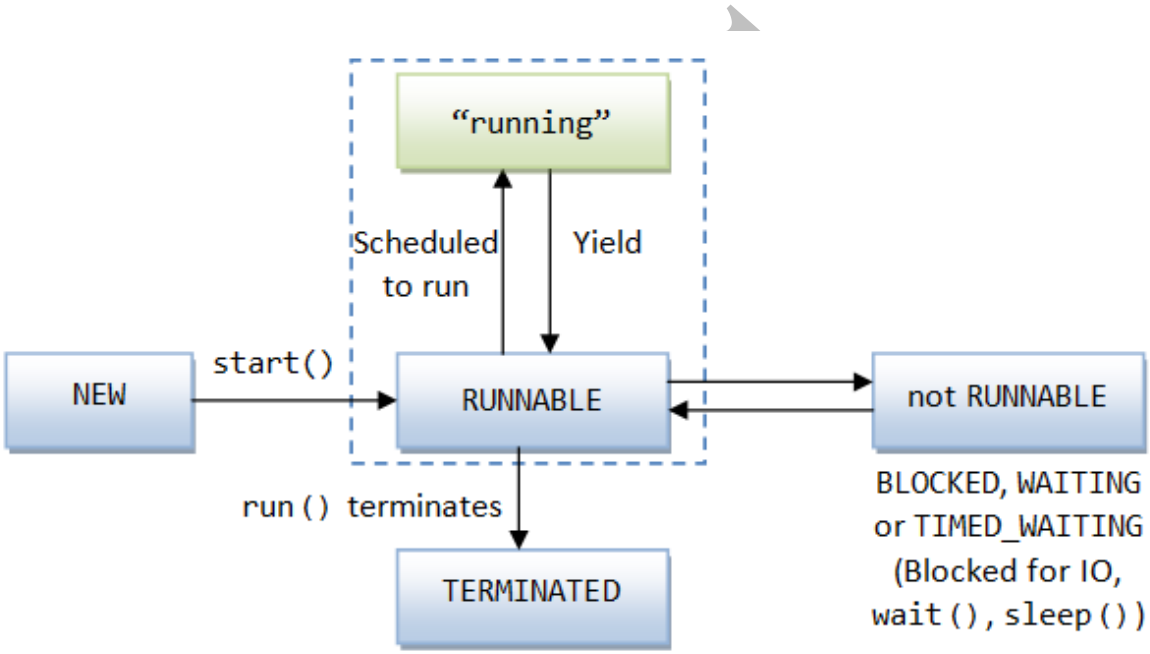
List the thread states and give state transition diagram (4M)

Thread states

Thread has 4 states.

1. **New**
2. **Runnable**
3. **Not Runnable**
4. **Terminated**

Thread life cycle



1. New

- **In this state, a new thread is created but not started.**
- **A thread is said to be born or new when a new thread object is created.**
  Thread threadobj = new ThreadDemo( );
- **In this state, no system resource is allotted to the newly born thread object.**
- **From this state, the thread can be either started using start( ) or killed using interrupt( ), moving to "Runnable" or "Terminated" state respectively.**
- **No other method apart from start( ) and interrupt( ) can be called from this state and if tried to do so, it would cause an exception, IllegalThreadStarteException.**

2. Runnable

- **In this state, a thread is ready for execution by the JVM. It represents the running state of the thread.**
- **Ready state of a thread can be defined as it is ready for execution but it might be in the queue, waiting for the operating system to provide the required resource.**
- **Once the thread is actually being executed by the processor then it is termed as "Running".**
- **From New state, the thread might move to the "Runnable" state on execution of the following statements:**

  Thread threadobj = new ThreadDemo( );

  threadobj.start( );
- **As soon as start( ) is called, system recourse is allotted to the thread as per schedule done by the Java Runtime Environment. Now thread has entered into the runnable state.**
- **There is a difference between Running thread and Runnable thread. A running thread is one which is being executed by the processor. Such a thread can be ass the "current thread". Runnable threads are those which are not actually running, but are scheduled in queue to get the processor.**

3. Not Runnable

- **This state is just a hypothetical state used by us to categorize the three valid states namely WAITING, TIMED_WAITING and BLOCKED.**
- **WAITING state, a thread is waiting indefinitely for another thread to perform a particular action (ie notify). Threads can move into this state by calling object.wait( ) or object.join( ). (without time out).**
- **TIMED_WAITING state, the thread is waiting for another thread to perform an action (notify) up to a specified waiting time. A thread can get into this state by calling either of the following methods : Thread.sleep( ), object.wait( ), Thread.join( ).**
- **BLOCKED state, a resource cannot be accessed because it is being used by another thread. A thread can enter into this state by calling object.wait( ).**

4. Terminated

This is the final state in thread life cycle. A thread may be dead in different ways.

a) **Natural death means - after completion of execution, the thread generally terminates.**
b) **Abnormal death means – the thread can be killed in born state or running state or blocked state.**

Q4: Discuss about java.lang.Thread

It is a class of java.lang package. The following are the constructors of Thread class.

- **Thread ( )**
- **Thread (String tname)**
- **Thread (ThreadGroup tgname, String tname)**

Methods of Thread class

Method	Description
1. getName( )	1. This method is used to get the thread name.
2. getPriority( )	2. This method is used to the thread's priority which would be maximum, minimum or normal.
3. isAlive( )	3. This method is used to determine if thread is still running or not. It returns true if it is active. Otherwise, false is returned.
4. join( )	4. This method waits for a method to terminate.
5. run( )	5. This is the method that starts the thread.
6. start( )	6. This method is sued to start execution of the thread by calling run( ) method.
7. sleep( )	7. This method stops the execution of a thread for a short duration.
8. setName( )	8. This method is used to assign a name to thread.
9. yield( )	9. This method is used to stop the execution of current thread temporarily and allow other threads to execute.
10. currentThread( )	10. It returns a thread object and this object encapsulates the thread which is calling this method.

Creation of threads

Q5: Explain in detail the process of creating thread with an example.

Creation of Thread

- **Thread can be created by using an object of type "Thread".**
- **A thread can be created in two ways**
  - **a) By implementing Runnable interface.**
  - **b) By extending Thread class.**

a) Implementing Runnable interface

- **The simple and easiest way for creating the thread is to create a class which implements Runnable interface.**
- **Runnable interface is abstract unit of executable code.**
- **In-order to implement Runnable, a class required to implement a single method called run( )**
- **In this method, we can define the code of a new thread.**

```
class MyThread implements Runnable
{
        …  …  …
        public void run( )
        {
            …  …
            …  …
        }
}
```

- **One of the important constructors defined by "Thread" is given below.**

  Thread(Runnable obj, String tname)

  Here, 'obj' is the instance of a class which implements Runnable interface and 'tname' is the name of new thread.

For example,

```
class MyThread implements Runnable
{
        String tname;
        MyThread(String name)
        {
            tname = name;
        }
```

```
        public void run()
        {
            try
            {
                for(;;)
                {
                    System.out.println(tname);
                    Thread.sleep(1000);
                }
            }
            catch(InterruptedException e)
            {
                System.out.println(e);
            }
        }
}
class ThreadDemo
{
        public static void main(String [] args)
        {
            MyThreadm1 = new MyThread("Hello");
            MyThreadm2 = new MyThread("There");
            Thread t1 = new Thread(m1);
            Thread t2 = new Thread(m2);
            t1.start();
            t2.start();
        }
}
```

b) Extending Thread class

- **This is another way of creating a thread. Here, a new class which extends "Thread" is created. Later, instance of the class is created.**
- **The class which is derived should override the run( ) method to switch the thread into runnable state.**
- **In-order to start execution, it should invoke start( ) method.**

For example,

```
class MyThread extends Thread
{
    String tname;
    MyThread(String name)
    {
        tname = name;
    }
```

```
    public void run()
    {
            try
            {
                for(;;)
                {
                    System.out.println(tname);
                    sleep(1000);
                }
            }
            catch(InterruptedException e)
            {
                System.out.println(e);
            }
    }
}

class DemoThread
{
    public static void main(String [] args)
    {
            MyThreadm1 = new MyThread("Hello");
            MyThreadm2 = new MyThread("There");
            m1.start();
            m2.start();
    }
}
```

Thread priorities

Q6: How do we set priorities for threads? Explain

- **Every thread has a priority which specifies amount of time used to access the CPU.**
- **The thread with highest priority contains less access time.**
- **The thread with highest priority will go to the running state first.**
- **Suppose if a thread is in running state and another thread with highest priority comes from ready state, suspend( ) method is applied to the current thread and it moves from running state to the sleeping state.**
- **After completion of highest priority thread execution, resume( ) method is applied and it moves from sleeping state to ready state and then to the running state.**
- **The threads which have equal priorities can access the CPU equally.**
- **Syntax:**

  final void setPriority(int priority)

here, setPriority( ) is the method name which can be used to set priority and priority is an integer value which can store the priority of thread. The following are various priority constrants.

Constant	Value	Meaning
MIN_PRIORITY	1	Max priority a thread can have
NORM_PRIORITY	5	Default priority a thread can have
MAX_PRIORITY	10	Min priority a thread can have

- **The priority of a thread can be acquired by using the following syntax:**

  final int getPriority( )

Example program

```
import java.io.*;
import java.util.*;
public class PriorityDemo extends Thread
{
        public void run( )
        {
                for(int i=1;i<=10;i++)
                {
                        String str = Thread.currentThread( ).getName( );
                        System.out.println(+str+ " : " +i);
                }
        }
        public static void main(String [ ] args)
        {
                PriorityDemo p1 = new PriorityDemo( );
                PriorityDemo p2 = new PriorityDemo( );
                PriorityDemo p3 = new PriorityDemo( );

                p1.setName("First Thread");
                p2.setName("Second Thread");
                p3.setName("Third Thread");

                p1.setPriority(NORM_PRIORITY);
                p2.setPriority(MAX_PRIORITY);
                p3.setPriority(MIN_PRIORITY);
                p1.start( );
                p2.start( );
                p3.start( );
                System.out.println("Priority of first thread:"+p1.getPriority( ) );
                System.out.println("Priority of second thread:"+p2.getPriority( ) );
                System.out.println("Priority of thrid thread:"+p3.getPriority( ) );
        }
}
```

In the above program, different priorities are set to three different threads. Each thread executes the loop for 10 times. The thread with highest priority that is second thread in this program will be executed first. The last three statements get the priority associated with each thread.

Thread synchronization

Q7: What is synchronization? Explain with example

Synchronization

- **Synchronization can be defined as a process of enabling single thread to access the shared resource.**
- **In multithreaded programming, synchronization threads are essential.**
- **When threads attempt to access the shared resources, data can be modified and incorrect outputs can be generated. To overcome this situation, synchronization can be utilized.**
- **In multithreaded programming, thread synchronization is used in-order to maintain the data consistency.**
- **This can be done using "synchronized" keyword.**
- **Synchronization is required in multithreaded programming when two or more threads try to access the shared resource at the same time.**
- **When a method is declared as synchronized, then the java compiler creates a monitor and provides it to the threads object created for thread class which invokes the synchronized method initially.**
- **A monitor can be defined as an object which can act as mutex or mutual exclusive lock.**
- **The thread which can be entered into the monitor can hold the lock and disables other threads to access the method.**
- **Once the execution of initial thread is complete, then the next waiting thread can enter into the monitor**
- **While executing the thread, when another thread attempts to enter the monitor, then it can be suspended until the execution of the thread in monitor gets completed.**

Example program

```
import java.io.*;
import java.util.*;
class First
{
        synchronized void call(String str)
        {
                System.out.println(" [ " +str);
                try
                {
                        Thread.sleep(1000);
                }
                catch(InterruptedException e)
                {
                        System.out.println("Interrupted");
                }
                System.out.println(" ] " );
        }
}
class Second implements Runnable
{
        String str;
        First f1;
        Thread t;
        public Second(First f2, String s)
        {
                f1=f2;
                str=s;
                t=new Thread(this);
                t.start( );
        }
        public void run( )
        {
                f1.call( str);
        }
}
class SynchornizedDemo
{
        public static void main( String args[ ] )
        {
                First f = new First( );
                Second s1 = new Second(f, "BTECH");
                Second s2 = new Second(f,"CSE");
                Second s3 = new Second(f,"BRANCH");
                try
                {
                        s1.t.join( );
                        s2.t.join( );
                        s3.t.join( );
```

```
            }
            catch(InterruputedException e)
            {
                System.out.println("Interrupted");
            }
        }
}
```

In the above program by calling sleep( ) method, the call( ) method allows another thread to get executed. This will result in the mixed output of three message strings. to prevent this, call( ) method is declared as synchronized. When the sleep( ) is called by call( ), it makes other thread to wait until the call( ) resumes and returns to the calling method. This prevents other threads to access the shared resources at the same time.

Communication between threads

Q8: How is inter-thread communication achieved? Exaplin

Inter-thread communication can be defined as multiple threads can communicate with each other by exchanging messages. Any two threads can communicate before switching to other state or after switching to other thread state. Consider that the thread in an active state can transfer the message to the thread in suspended state prior to switching to suspended state. In java, inter-thread communication can be supported by using the following methods.

a) **notifiy( )**
b) **notifyall( )**
c) **wait( )**

a) **notifiy( )**

This method is used to resume the initial thread which is in sleep mode.

syntax:

```
final void notify( )
```

b) **notifyall( )**

This method is used to resume all the threads that are in sleep mode. These threads can be executed depending on its priority.

syntax:

```
final void notifyall( )
```

c) wait( )

This method is used to send the invoking thread into sleep mode. The thread which is sent to sleep mode can be resumed with the help of notify( ) or notifyall( ) method. In addition to this, user can set time to the thread by declaring the timer as argument for wait( ) method. After completion of timer, the thread can be resumed automatically.

syntax:

final void wait( )

These three methods that are declared as final and therefore cannot be overridden. All these methods can throw InterruptedException.

Reading and writing data

Q9: Explain about reading and writing data files.

- **The data can be read/written to files, console, sockets etc using both the streams.**
- **The classes under these streams are used for reading/writing data.**

Reading/Writing Console

```
import java.util.*;
class ScannerDemo
{
        public static void main(String args[ ])
        {
                Scanner s = new Scanner(System.in);

                System.out.printl("Enter your name:");
                String name = s.nextLine( );

                System.out.println("Enter your age:");
                int age = s.nextInt( );

                System.out.println(+name);
                System.out.println(+age);
        }
}
```

- **Using Scanner class, we can directly read a primitive value form the user.**
- **The method nextLine( ) advances this scanner past the current line and returns the input as a string.**
- **The method nextInt( ) returns the next token of input as int.**
- **The following are some of the methods in Scanner class**

Methods	Description
close( )	closes this scanner
nextInt( )	returns the next token of input as int
nextFloat( )	returns the next token of input as float
nextDouble( )	returns the next token of input as double
nextLine( )	advances this canner past the current line and returns the input as a string

Reading/Writing using Buffered Byte Stream classes

- **BufferedInputStream class is used for buffering the input and it supports operation to re-read the files.**
- **BufferedOutputStream class is used to buffer the output and enhance the performance.**
- **The following are some of the methods in BufferedInputStream class.**

Methods	Description
available( )	returns the number of bytes that can be read from this input stream without blocking.
close( )	closes this input stream and releases all the resources.
read( )	reads byte of data.
reset( )	return to the first position.
mark( )	reading limited number of bytes.

- **The following are some of the methods in BufferedOutputStream class.**

Methods	Description
write( )	write the specified byte to the output stream.
flush( )	flushes the output stream.

- **Example program**

```
import java.io.*;

class BufferedOutputStreamDemo
{
    public static void main(String args[ ] ) throws IOException
    {
            FileInputStream fs = new FileInputStream(args[0]);
            BufferedInputStream bis = new BufferedInputstream(fs);
```

```
            int n = fs.available( );

            bis.mark(n);

            System.out.println("Marked the stream");
            byte b[ ] = new byte[n];
            byte b1[ ] = new byte[n];

            bis.read(b);
            System.out.println("contents of "+args[0]+ " : "+new String(b));

            System.out.println("resetting the stream");
            bis.reset( );

            System.out.println("Reading stream again from the marked point");
            bis.read(b1);
            System.out.println(+new String(b1));

            bis.close( );
            fs.close( );

            System.out.println("writing contents to "+args[1]);
            FileOutputStream fos = new FileOutputStream(args[1]);
            BufferedOutputStream out = new BufferedOutputStream(fos);
            out.write(b);
            System.out.println("contents written");

            out.close( );
            fos.close( );
        }
    }
```

- **The stream is marked. The reset method sets the pointer to the marked point.**
- **The read operation will begin from the marked point after resetting the stream.**
- **Two byte arrays have been created. One for reading the content before the stream is reset and the second one for reading the stream after it is reset.**

10Q: Explain about Random Access File

- **RandomeAccessFile gives an opportunity to read or write files from a specified location.**
- **This class has a method seek(long pos) that sets the file pointer at the specified pointer at the specified position.**
- **Now any read/write operation on the file will start from this marked position.**

- **The following are some of methods in RandomAccessFile class.**

Methods	Description
close( )	closes this random access file stream.
length( )	returns the length of the file.
read( )	reads a byte of data from the file.
readLine( )	reads the next line of the text from the file
void seek(long pos)	sets the file pointer, measured from the beginning of the file, at which the next read or write operation occurs.

- **In the following example, seek( ) method is used to set the file pointer to end of file(EOF) and then write the contents to the file, appending the file.**

```
import java.io.*;
class RandomAccessFileDemo
{
    public static void main(String args[ ]) throws IOException
    {
            System.out.println("opening the file in read / write mode");

            RandomAccessFile raf = new RandomAccessFile("sample.txt","rw");
            raf.seek(raf.length( ));

            String str = "Contents appended using Random Access File";
            System.out.println("appending contents to file");
            raf.write(str.getBytes( ));
            System.out.println("contents appended");

            System.out.println("reading the contents of the file");
            raf.seek(0);
            while(str = raf.readLine( ) )!=null )
                System.out.println(str);
            raf.close( );
    }
}
```

Applet class, Applet structure, Applet Life cycle, sample example Applet programs, Event Handling: Event delegation model, Sources of event, Event listeners, Adapter classes, Inner classes.

Applet class

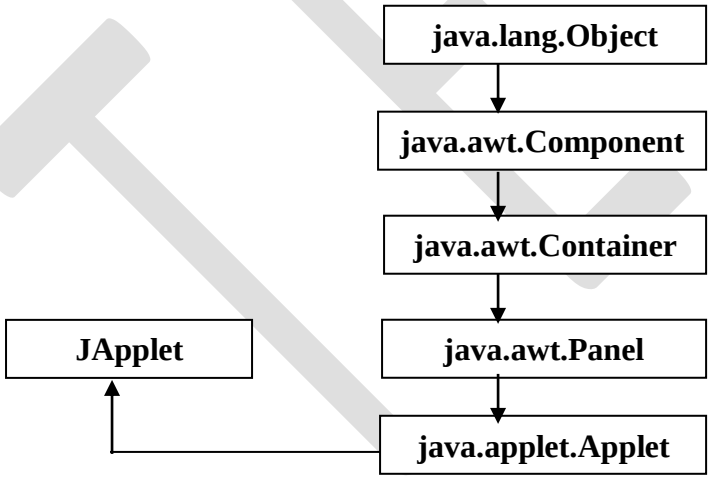
Q: Define Applet. Explain in detail about applet class.

Def: Applet

- **A java program that can be enabled in HTML document and can run on Java enabled browsers like Internet Explorer is called an Applet.**
- **It does not have main( ) method.**
- **It is derived from "Applet" class.**

applet class

- **The java.applet.Applet is the base class for all the applets.**
- **The applet can inherit all the methods of applet from java.applet package.**
- **A subclass is created from the Applet class to implement applet.**
- **The inheritance hierarchy of applet is as follows:**



Object

It is the root of complete class hierarchy. Every object implements the methods of object class. It belongs to java.lang.Object package.

Component

It is an object in the form of graphical representation that will be displayed on the screen. It is the base class of all the AWT components. It belongs to java.awt.Component package.

Container

It is an AWT (Abstract Windowing Toolkit) container object that contains all the other AWT components. It belongs to java.awt.Container package.

Panel

It is a simple container class which allows the application to attach other components or panels. It has FlowLayout as default layout manager. It belongs to java.awt.Panel package.

Applet

A java program that can be enabled in HTML document and can run on Java enabled browsers like Internet Explorer is called an Applet. It belongs to java.applet.Applet package.

JApplet

It is an applet that belongs to javax.swing.JApplet package.

Methods of Applet class

Method	Description
1. init( )	1. Applet execution begins.
2. isActive( )	2. It returns true if the applet is still running. Otherwise it returns false.
3. getParameter(String name)	3. It is used to obtain the parameter that is associated with the name of parameter.
4. resize(int width, int height)	4. It is used to resize the applet according to the given specifications.
5. start( )	5. It is used to start or resume the applet execution.
6. stop( )	6. It is used to stop or suspend the applet.
7. destroy( )	7. It is used to terminate the applet.

Applet structure

Q: Explain about Applet structure. (OR)

Q: Write an example applet program and explain how to run it.

Applet structure or skeleton

```
import java.awt.*;          // to make Graphics class available
import java.applet.*;       // to make Applet class available


public class AppletName extends Applet
{
        public void init( )
        {
                // initialization
        }
        public void start( )
        {
                // start or resume the applet execution
        }
        public void stop( )
        {
                // interrupts the execution of applet
        }
        public void destroy( )
        {
                // performs shutdown activities
        }
        public void paint(Graphics g)
        {
                // display or redisplay the output
        }
}
```

Example program

```
import java.awt.*;
import java.applet.*;

public class FirstApplet extends Applet
{
        public void paint(Graphics g )
        {
                g.drawString("Welcome to applet programming", 10,10);
        }
}
```

- **In the above program, the first statement imports the AWT package which consists of classes used to draw lines, shapes, letters, set colours and to choose fonts. This package can also act as an interface between the applet program and user.**
- **The next statement imports java.applet package which contains the Applet class. This class is a super class of every class created in applet. Therefore, an applet class must be extended in any applet application.**
- **In the next statement, a class called FirstApplet is defined. Every class defined in an applet must be declared with access specifier "public" since the class must be accessed externally.**
- **In the FirstApplete class, a method called paint( ) is declared with access specfier "public". This method belongs to the component class which can be defined in AWT package. Generally, paint( ) method is used to draw the output. While declaring paint( ) method, an argument called Graphics is used. The Graphics class contains graphics context which can describe graphics environment during the execution of applet. This can be used when the output is needed in applet window. The Graphics class consists of an object which can be used to draw the output.**
- **In the paint( ) method, another method called drawString( ) is invoked which can output a string with specified X, Y location in applet window. This method can be defined in Graphics class and can be accessed through the object of this class.**
- **In applet program, it is not mandatory to use main( ) method. The execution of an applet program can be started from the class which can be passed to the browser or to other applet-enabled program.**

Executing or Running an applet

An applet can be executed in two ways. They are as follows:

a) **Applet viewer**
b) **Browser**

a) Applet viewer

Save the file as FirstApplet.java and compile it by using javac. Now type the following HTML code in notepad editor and save the file as SampleApplet.html (here the file name is not necessarily the same as the class name).

```
<html>
<body>
<applet code = "AppletExample"  width=200 height=80>
</applet>
```

```
</body>
</html>
```

Now execute the html file by using,

```
appletviewer SampleApplet.html
```

The another way is to simply include this code in comments at the top of class declaration and below to import statements.

```
import java.awt.*;
import java.applet.*;
/*
<applet code = "AppletExample"  width=200 height=80>
</applet>
*/
public class AppletExample extends Applet
{
        public void paint(Graphics g )
        {
                g.drawString("Welcome to applet programming", 10,10);
        }
}
```

b Browser

An applet program can be executed by simply typing the URL of html file in address bar or by double clicking the html file. Then the output can be displayed.

Applet life cycle

Q: Explain about Applet life cycle.

An applet consists of various methods which can override and can be used to control the execution of applet either in applet viewer or browser. These methods can be involved in the life cycle of an applet.

Applet life cycle

The following are the methods in the applet life cycle.

a) **init( )**
b) **start( )**
c) **stop( )**
d) **destroy( )**

a) init( )

This method is used to initialize all the applet variables and also perform different start-up activities. This is the method which must be invoked initially. This method can be called only once in an applet program.

b) start( )

This method is invoked after the completion of invoking init( ) method. The start( ) method is used to start and resume the applet. If the user minimized and returned again to previous webpage which consists of an applet, then the execution process of this applet can be resumed from start( ) method. This method can be invoked multiple times during the execution of an applet program.

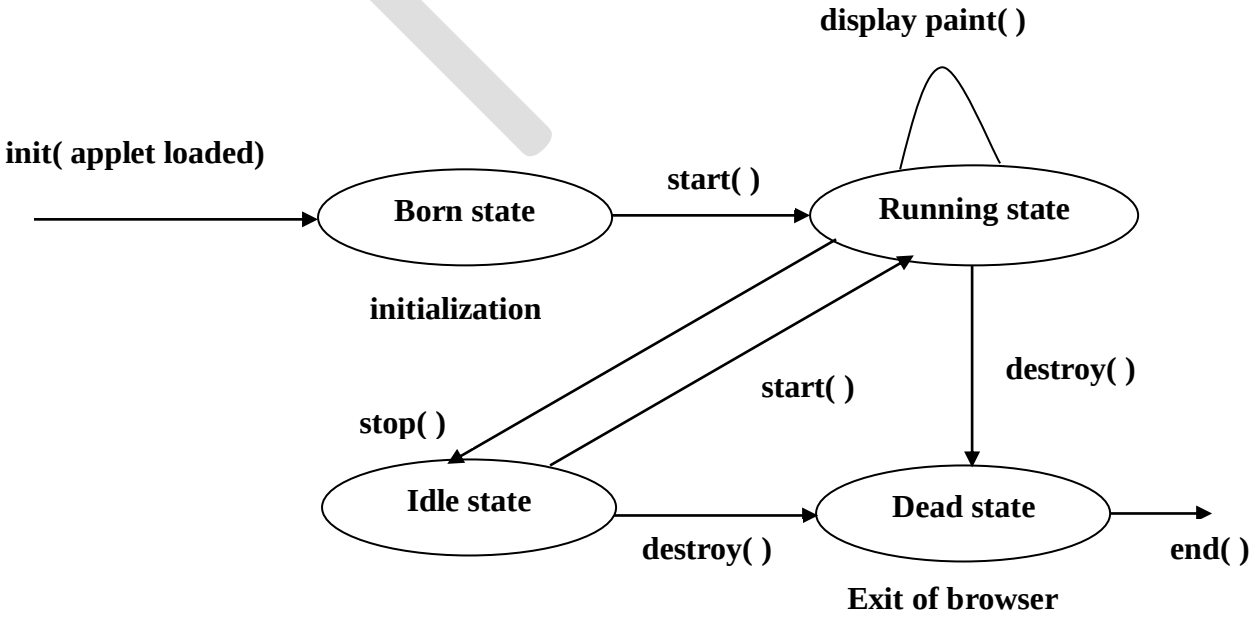
c) stop( )

This method is used to suspend the applet. This method can be invoked when the user left the current webpage which consists of an applet.

The stop( ) method can also be used to suspend the child thread which can be created by the applet and have an ability to perform any other task which can keep the applet in idle mode or sleep mode. It does not mean that the applet cannot be executed. The suspended applet or child thread can be executed by resuming the applet. This can be done by invoking the start( ) method.

d) destroy( )

This method is invoked when there is a requirement to remove the applet completely from memory. Once applet is destroyed, all the resources allocated to the applet can be released. This method can perform the shutdown operation of an applet if required.



Initially, an applet can be initialized with init( ) method and will be in born state. Then the applet can move into running state using start( ) method. In running state, output can be displayed using paint( ) method which belongs to component class of AWT package. The applet can move from running state to either idle state or dead state. If an interruption occurred during the execution of an applet, it moves to idle state using stop( ) method. If the applet is destroyed then the it enters into dead state using destroy( ) method.

Q: Explain the common methods used in Applet (displaying the output).

1. drawstring( )

This method is a member of Graphics class, used to output a string to an applet. It is typically called from within the paint( ) or update( ).

syntax:

void drawstring(String msg, int a, int b)

here, the first argument is the string output to be displayed by the applet. The second and third arguments are x and y coordinates of the window respectively where the output has to be displayed.

2. setBackground( )

This method belongs to component class. It is used to set the background color of the applet window.

syntax:

void setBackground(Color.anycolor)

Here Color is the class and it has predefined constants for each color, such as Color.red, Color.blue, Color.green and Color.pink

3. setForeground( )

This method is similar to setBackground method, except that these are used to set he color of the text to be displayed on the foreground of the applet window.

syntax:

void setForeground(Color.anycolor)

4. showStatus( )

This method is a member of Applet class. It is used to display any string in the status window of the browser or appletviewer.

syntax: void showStatus(String text)

5. paint( )

When a component needs to draw/redraw itself, paint( ) is called.

syntax:

```
public void paint(Graphics g)
{
    ....
}
```

6. repaint( )

This method is used to force a component to be repainted manually.

repaint( ) → update( ) → paint( )

Event handling

Q: Write short notes on events.

- **Any change in the state of source defined by an object is called event.**
- **Generally, an event is triggered whenever the following situations are encountered.**
  - **i) On clicking a mouse button.**
  - **ii) On entering a character through keyboard.**
- **An event is usually generated when a user directly or indirectly interact with the Interfaces.**
- **For instance, an event occurs**
  - **a) when the time expires.**
  - **b) when there is a failure in hardware or software.**
  - **c) when a counter exceeds its limited value.**

Event Delegation Model

Q: Write in detail about event delegation model.

- **The event delegation model defines standard mechanism for generating an event.**
- **Basically an event is generated by an object called source, which is later received and processed by another object called listener.**
- **The main purpose of delegation event model is to separate application logic from the user interface logic, which in turn delegate its responsibilities to the different piece of code.**
- **In this scheme, a listener can receive event notification only after registering itself with event source.**

- **This help in**
  - **i) Sending notifications only to interested listeners.**
  - **ii) Handling events efficiently.**
- **Delegation event model is mainly used for the following reasons.**
  - **i) For receiving the desired event listener**
  - **ii) For registering/unregistering with the event listener**
- **Delegation event model usually handles two commonly generated events like mouse event and keyboard event.**

Sources of Events

Q: Discuss in brief about sources of events.

Sources of events can be either components of GUI or any other class derived from a component (such as applet), which can generate event like events from keyboard and mouse. Some of the components of GUI are listed below.

Button	**Choice**	**Menu item**
Check box	**List**	**Window**
Scroll bar	**Text components**	**etc.,**

Event Listeners

Q: Explain about Event Listeners.

- **Event listeners are created by implementing one or more interfaces defined by the java.awt.event package.**
- **Whenever a source generates an event, it basically invokes the appropriate method defined in the listener interface.**
- **The method has an event object passed as an argument to it.**
- **Some of the frequently used listeners are given below.**

KeyListener	**ItemListener**	**WindowListener**
MouseListener	**ActionListener**	**ComponentListener**
MouseMotionListener	**TextListener**	**ContainerListener**
MouseWheelListner	**FocusListner**	**AdjustmentListener**

KeyListener interface

This method has three methods defined within it.

Method name	Invokes when
void KeyPressed(KeyEvent ke)	key is pressed.
void KeyReleased(KeyEvent ke)	key is released.
void KeyTyped(KeyEvent ke)	key is typed.

MouseListener interface

This interface has five methods, having signatures as follows:

Method name	Invokes when
void mouseClicked(MouseEvent me)	mouse clicked.
void mouseEntered(MouseEvent me)	mouse enters a component.
void mousePressed(MouseEvent me)	mouse is pressed but not released.
void mouseReleased(MouseEvent me)	pressed mouse is released.
void mouseExited(MouseEvent me)	mouse leaves a component.

MouseMotionListener interface

This interface has two methods having the signatures.

Method name	Invokes when
void mouseMovedMouseEvent me)	mouse is moved from one place to another.
void mouseDragged(MouseEvent me)	mouse is dragged.

ItemListner interface

This interface has only one method defined as,

Method name	Invokes when
void itemStateChanged(ItemEvent e)	the state of item changes.

ActionListener interface

This interface has only one method defined as,

Method name	Invokes when
void actionPerformed(ActionEvent e)	any action event is performed.

TextListner interface

This interface has only one method defined as,

Method name	Invokes when
void TextChanged(TextEvent e)	there is a change in text field.

FocusListener interface

This interface has two methods having the signatures.

Method name	Invokes when
void focusGained(FocusEvent e)	the component obtains keyboard focus.
void focusLost(FocusEvent e)	the component loses the keyboard focus.

WindowListener interface

This interface has seven methods having the signatures.

Method name	Invokes when
void focusGained(FocusEvent e)	the component obtains keyboard focus.
void focusLost(FocusEvent e)	the component loses the keyboard focus.

Adapter classes

Q: Explain about Adapter Classes with example.

- **It is time consuming to override all the methods of an interface to handle particular event.**
- **An Adapter class provides an empty implementations of all the methods in an event listener interface.**
- **Adapter classes are useful when we want to receive and process only some of the events that are handled by a particular event listener interface.**
- **For this, we can define a new class to act as an event listener by extending one of the adapter classes and implementing only those events in which we are interested.**
- **Consider, "MouseMotionAdapter" class has two methods, mouseDragged( ) and mouseMoved( ). If we are interested only in mouse drag events, then we extend "Mouse" and implement mouseDragged( ).**
- **The empty implementation of mouseMoved( ) would handle the mouse motion events.**

- **Adapter classes are provided by java.awt.event package.**
- **Some of the adapter classes are given below**

Adapter Class	Listener interface	Add Method
ComponentAdapter	ComponentListener	addComponentListener()
ContainerAdapter	ContainerListener	addContainerListener()
FocusAdapter	FocusListener	addFocusListener()
KeyAdapter	KeyListener	addKeyListener()
MouseAdapter	MouseListener	addMouseListener()
MouseMotionAdapter	MouseMotionListener	addMouseMotionListener()
WindowAdapter	WindowListener	addWindowListener()

Demonstration of Adapter Classes

AdapterDemo.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;
public class AdapterDemo extends Applet
{
        public void init()
        {
                addMouseListener(new My(this));
                addMouseMotionListener(new MyAdd(this));
        }
}
class My extends MouseAdapter
{
        AdapterDemo a;
        public My(AdapterDemo p)
        {
                this.a=p;
        }
        public void mouseClicked(MouseEvent me)
        {
                a.showStatus("Mouse Clicked");
        }
}
class MyAdd extends MouseMotionAdapter
{
        AdapterDemo a;
        public MyAdd(AdapterDemo p)
        {
                this.a=p;
        }
```

```
        public void mouseDragged(MouseEvent me)
        {
                a.showStatus("Mouse Dragged");
        }
}
```

Adapter.html

```
<html>
<applet code="AdapterDemo.class" width=400 height=350>
</applet>
</html>
```

Inner classes

Q: Explain about nested and inner classes or Inner classes

- **It is possible to define a class within another class. Such classes are known as nested classes. A nested class has access to all of the variables and methods of its outer class. But the outer class does not access the variables and methods of nested class.**
- **There are two types of nested classes, static and non-static.**
- **A static nested class is a class started with "static" keyword. Due to the static, it must access the members of its enclosing class through an object. Direct access cannot be possible.**
- **A non-static nested class is a class as ordinary representation of class. The most important type of the non-static class is "inner class". The inner class has access to all of the variables and methods of its outer class. Thus, inner class scope is fully within the scope of its outer class.**

Demo.java

```
class Outer
{
        int k=34;
        void test()
        {
                Inner in=new Inner();
                in.display();
        }
        class Inner
        {
                void display()
                {
                        System.out.println("value="+k);
                }
        }
}
```

```
class Demo
{
        public static void main(String args[])
        {
                Outer x= new Outer();
                x.test();
        }
}
```

Q. Write the differences between Applets and Application programs.

Applet	Application program
1. The execution of the applet does not start from main( ), as it does not have.	1. The execution of an application program starts from main( ).
2. Applets cannot run on their own. They have to be embedded inside a web page to get executed.	2. These can run on their own. In order to get executed, they need not be embedded inside any web page.
3. Applets can be only executed inside a browser or appletviewer.	3. Applications have no inherent security restrictions.
4. Applets execute under strict security limitations.	4. Applications have no inherent security restrictions.
5. Applets have their own life cycle: init( ) -> start( ) -> paint( ) -> stop( ) -> destroy( )	5. Applications have their own life cycle. Their execution begin at main( )

AWT (Abstract Window Toolkit) – introduction, components and containers, button, label, checkbox, radio buttons, list boxes, choice boxes, container classes, layouts, menu, scrollbar.

Abstract Window Toolkit

1Q. Define AWT. Discuss about components and containers of AWT.(OR)
Discuss about classes contained in java.awt package.

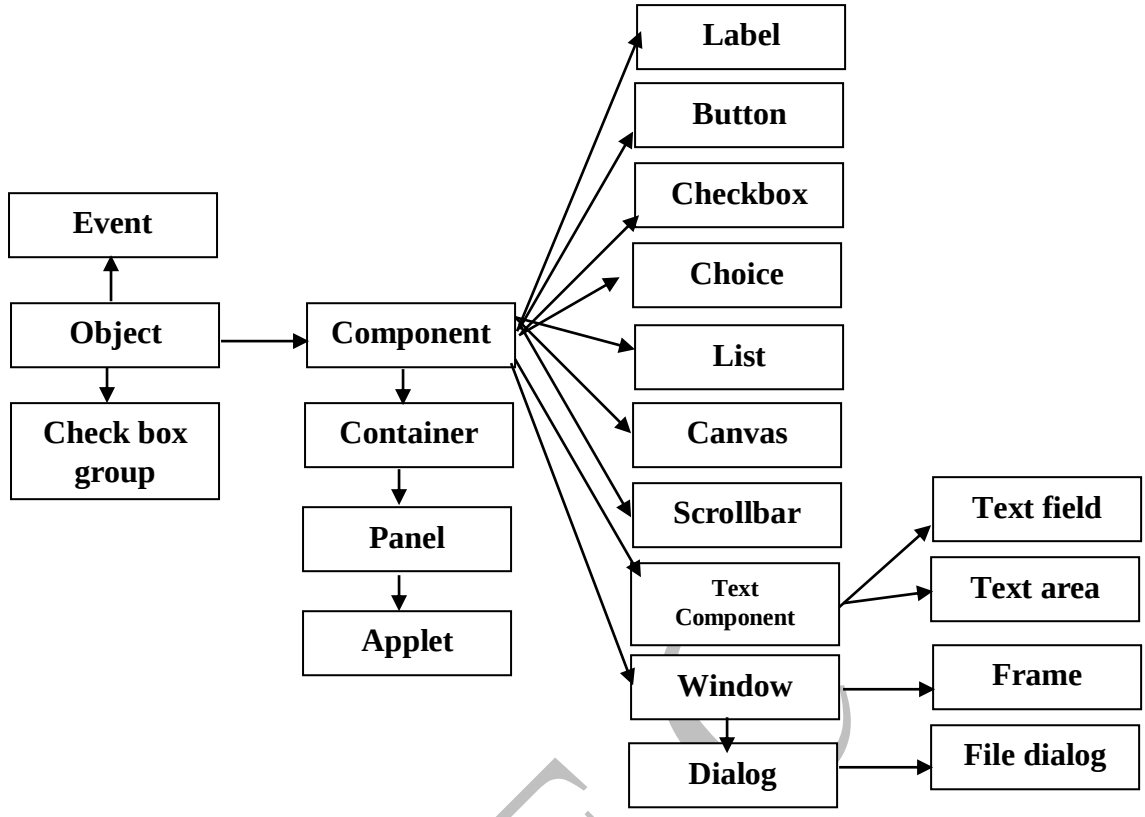
Def:

The java AWT (Abstract Window Toolkit) consists of classes that are used in building Graphical User Interface (GUI). In addition to this, they are also used in creating images, colors, fonts and graphics paintings. All these classes are contained in the package called java.awt. The layout manager is responsible for adding the components and deploying them in the containers.

Components:

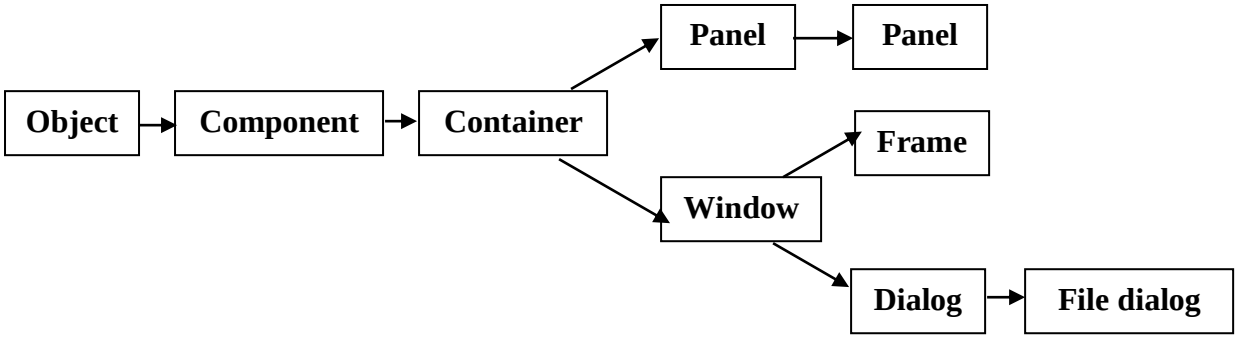
- **Components are used by the user to interact with the user interface.**
- **These components are added to the container where container is also a component.**
- **In Java, component is an abstract class that contains all the classes and methods required for managing events such as positioning and sizing the components.**
- **All the components which are showed on the screen and the components which interact with the user are derived from Component class.**
- **The Component class is derived from Object class. Button,**
- **TextField, Label etc are all the examples of component.**

- **The following figure shows component hierarchy supported by AWT.**



Container:

- **Container is derived from Component class.**
- **It contains all the components – Window, Frame, Applet, Panel etc are all containers**
- **The Container class contains additional methods that make the components to be nested within the container.**
- **It can also contain the other container objects as a component inside it.**
- **Laying out of all components of a container is the responsibility of the layout manager.**
- **The following figure shows container hierarchy.**



Buttons

2Q. Explain in detail about Buttons.

Buttons

->A push button is a component that contains a label and that generates an event when it is pressed. Push buttons are the objects of type Button.

->Button defines two constructors

1.Button( ) : it creates an empty button

2.Button(String str) : it creates a button that contains str as a label

->The following are the methods

1.void setLabel(String str) : it is used to set a label on the push button

2.String getLabel( ) : it returns the label of the push button as String

Example program

```
import java.applet.*;
import java.awt.*;
import java.awt.event.*;
/*
<applet code = "Buttons" width=200 height=100>
</applet>
*/
public class Buttons extends Applet implements ActionListener
{
        String msg=" ";
        Button b1,b2,b3;

        public void init( )
        {
                b1 = new Button("CSE");
                b2 = new Button("ECE");
                b3 = new Button("EEE");

                add(b1);
                add(b2);
                add(b3);

                b1.addActionListener(this);
                b2.addActionListener(this);
                b3.addActionListener(this);
        }
        public void action Performed(ActionEvent ae)
        {
                String s=ae.getActionCommand( );

                if(s.equals("CSE"))
```

```
                msg = "you have selected CSE branch";
        else if(s.equals("ECE")
                msg = "you have selected ECE branch";
        else
                msg = "you have selected EEE branch";

        repaint( );
    }
    public void paint(Graphics g)
    {
        g.drawString(msg,6,100);
    }
}
```

Label

3Q. Explain about Labels.

Labels:

->Lable is a text information that displays on the GUI as a string. It is a passive control that do not support any interaction with the user.

->Label defines the following constructors:

1.Label( ) : it creates a blank label

2. Label(String str) : it creates a label that contains the string specified by str (left aligned)

3.Label(String str, int align) : it creates a label that contains the string specified by str using the alignment specified by "align". The value of "align" must be one of the following three constants: Label.LEFT, Label.RIGHT and Lable.CENTER

->The following are the methods

1.void setText(String str) : it is used to change the text specified by the str argument.

2.String getText( ): it returns the text from the label.

Example program

```
import java.applet.*;
import java.awt.*;
/*
<applet code = "Labels" width=200 height=100>
</applet>
*/
public class Labels extends Applet
{
    public void init( )
    {
        Label a = new Label("Enter first value");
        Label b = new Label("Enter second value");
```

```
            add(a);
            add(b);
        }
}
```

Check box and Radio buttons

4Q. Explain about Checkbox and Radio buttons.

Check box

-> Check box is a control that is used to turn an option on or off. It consists of a small box that can either contain mark or not. There is a label associated with each check box that describes the option. It can change the state of the check box by clicking it. Check boxes are the objects of the "Checkbox" class.

-> Checkbox defines the following constructors:

1. Checkbox( ) – it creates a blank and unchecked checkbox.

2. Checkbox(String str) – it creates checkbox with the given label and state.

3. Checkbox(String str, Boolean on) – it creates checkbox with the given label and state.

-> The following are the methods.

1. String getLabel( ) – it obtains the checkbox text.

2. Boolean getState( ) – it obtains the present state of the checkbox.

3. checkboxGroup getCheckboxGroup( ) – it obtains the relative checkbox group.

4. void setLabel(String label) – it sets the checkbox label to the given text.

5. void setState(Boolean state) – it sets the state of checkbox to the given state.

Radio buttons

- **Radio buttons are created an instances of the CheckboxGroup class.**
- **They are similar to check boxes. Radio buttons have also two states selected or deselected.**
- **They are a group of buttons from which only one button can be selected at any time.**
- **If other radio button is selected then the previously selected radio button will get deselected automatically.**
- **The grouping of radio buttons are made by adding the RadioButton object to a CheckboxGroup object that will be responsible for the state of buttons in the group.**

- **There is no class provided for radio buttons in java.awt package. All the methods of java AWT checkbox can be used by radio buttons.**
- **An instance of check box group can be created as follows:**
  CheckboxGroup cb = new CheckboxGroup( );
- **This check box should be added to the CheckboxGroup using the below method:**
  add (new Checkbox("Yes/No",cb, true/flase);

List box and choice box

5Q. Explain about Listbox and choice box.

List box

- **The "List" class provides a compact, multiple-choice, scrolling section list. The list object can be constructed to show any number of choices in the visible window. It can also be created to allow multiple selections.**
- **The following are the constructors of List class**
  1. **List( ) – It creates a list which allows only one item to be selected at a time.**
  2. **List(int nr) – It creates a list with the number of rows specified by nr.**
  3. **List(int nr, Boolean ms) - It creates a list with the number of rows specified by nr. If the multiple select is true then it allows multiple items to be selected at a time.**
- **The following are the methods of List class.**
  1. **void add(String item) – It is used to append the specified item to the list.**
  2. **void deselect(int index) – It is used to deselect the item at the given index.**
  3. **String getItem(int index) –It is used to obtain the item from the given index.**
  4. **int getRows( ) – It is used to obtain the number of visible lines from the list.**
  5. **int getSelectedIndex( ) – It is used to obtain the index of the selected item.**

Choicebox

- **The "Choice" is used to create a pop-up list of items from which the user may choose. A choice control is a form of menu.**
- **To select anything from the choice, the user can just pop-down the choice list and then select from it.**
- **The choice box is a type of menu.**
- **When it is not used, it consumes less space by displaying just the selected item.**
- **The items of the choice box are aligned to left.**

- **The following is the constructor in Choice box class.**
  1. **Choice( ) – It creates an empty choice list.**
- **The following are the methods of choice box.**
  1. **void add(String name) – It is used to add items to the list.**
  2. **String getSelectedItem( ) – It is used to obtain the selected item.**

Container class

6Q. Discuss various Container classes available in AWT.

Container classes are three types.

1. **Panel**
2. **Frame**
3. **Window**

1. Panel

- **Panel is also a container without title bar, menu bar or any borders. So it does not have a physical appearance.**
- **This is the reason which makes it independent and always work under window or an applet. Hence it cannot be a top-level window.**
- **Panel is derived from Container class.**
- **It can work as a container as well as a component.**
- **As a container, it holds the components.**
- **But as a component,**
- **It can be added to other containers.**
- **Panel is a super class of applet class.**
- **Panel is declared as follows:**
  public class Panel extends Container implements Accessible
- **The following are the constructors of Panel class:**
  i) **Panel( ) – It creates a panel through default layout manager.**
  ii) **Panel(LayoutManager layout) – It creates a Panel with the given layout.**

2. Frame

- **A frame consists of borders. So it can be used as top-level window.**
- **They will be displayed when the components are added to them.**
- **Frame is a subclass of Window and has title bar, menu bar, borders and resizing corners.**

- **Frame is used to create child windows within applets and top-level or child windows for applications.**
- **The following are the constructors of Frame class:**
  - **i) Frame( ) – It creates a window without any title.**
  - **ii) Frame(String title) – It creates a window with the given title.**
- **The following are the methods of Frame class:**
  - **i) setSize( ):- this method is used to set the dimensions of a window**
  - **ii) getSize( ):- this method is used to get the current size of the window**
  - **iii) setVisible( ):- we can make a window visible by calling this method**
  - **iv) setTitle( ):- this method is used to set or change the title of a window**

3. Window

- **The window class creates a top-level window.**
- **A top-level window is not contained within any other object.**
- **It puts directly on the desktop.**
- **Generally, we don't create Window object directly. Instead, we will use a subclass of Window called Frame.**
- **Window can be defined as follows:**
  public class Window extends Container implements Accessible

Layouts

7Q. What is Layout Manager? Explain different types of layouts

Layout Manager

->A layout manager is an instance of any class that implements the "LayoutManager" interface. The layout manager is set by setLayout( ) method.

Syntax: void setLayout(LayoutManager obj)

->Each layout manager keeps track of a list of components that are stored by their names. The layout manager is notified each time we add a component to the container.

->The following are the layout types

1. FlowLayout

2. BorderLayout

3. GridLayout

4. CardLayout

1. Flow Layout

->The default layout manager is FlowLayout. It implements a simple layout style which is similar to how words flow in a text editor. Components are laid out from the upper-left corner, left to right and top to bottom. When no more components fit on a line, the next one appears on the next line. A small space is left between each component above and below, as well as left and right.

->The following are the constructors for FlowLayout.

1. FlowLayout( )

2. FlowLayout(int align)

3. FlowLayout(int align,int h, int v)

Alignment as follows.

FlowLayout.LEFT

FlowLayout.CENTER

FlowLayout.RIGHT

These values specify left, center and right alignment respectively.

2. Border Layout

->The BorderLayout implements a common layout style for top-level windows. It has narrow, fixed width components at the edges and one large are in the center. The four sides referred to as north, south, east and west.

->The following are the constructors for BorderLayout.

1. BorderLayout( ) – It creates default layout.

2. BorderLayout(int horz, int ver) – It creates border layout with a specified horizontal and vertical space between the components.

->BorderLayout defines the following constants that specify the regions.

BorderLayout.CENTER

BorderLayout.EAST

BorerLayout.WEST

BorderLayout.NORTH

BorderLayout.SOUTH

->When adding components, we will use these constants with the following form of add( ) which is defined by "Container".

void add(Component obj, Object region)

3. Grid Layout

->GridLayout lays out components in a two-dimensional grid. When we instantiate a "GridLayout". We define the number of rows and columns.

->The following are the constructors of GridLayout

1. GridLayout( ) – It creates grid layout with single column.

2. GridLayout(int rows, int columns) – It creates grid layout with given rows and columns.

3. GridLayout(int rows, int columns, int horz, int ver) – It creates grid layout with the given rows, columns and horizontal and vertical spaces.

4. Card Layout

->The CardLayout class is stored several different layouts. Each layout can be thought of as being on a separate index card in a deck that can be shuffled so that any card is on the top at a given time.

->The following are the constructors of CardLayout

1. CardLayout( ) – It creates default card layout.

2. CardLayout(int horz, int ver) – It creates card layout with specified horizontal and vertical space between the components.

Menu

8Q. Explain in detail about AWT menu.

Menu

- **Menus are mostly used in GUI environment.**
- **A menubar consist of menu objects which in turn consists of a list of MenuItem objects.**
- **MenuItems are the options provided to the user.**
- **It is possible even create nested submenus and checkable menu items.**
- **The menu options are of type CheckboxMenuItem each of which assigned a check mark when selected.**
- **To create a menubar, an instance of MenuBar must be created and then instances of Menu which define the selections displayed on the menubar must be created.**

Steps for creating menu:

i) **Creating a menu bar.**

ii) **And then add the menu bar to the frame.**

iii) **Create required number of menus.**

iv) **Add menu items to each menu.**

v) **Add all the menus to the menu bar.**

Scrollbar

9Q. Explain about scrollbars in AWT.

Scrollbar

- **It is used for selecting continues values from a range of value.**
- **A Scrollbar class is used for creating the scrollbar.**
- **It is always used with a list or text area or a window.**
- **Usually, scrollbars can either be horizontal or vertical.**
- **The purpose of scrollbar is to move along a window or a list or a text area.**
- **Horizontal scrollbar consists of left arrow, right arrow, a slider box whereas vertical scrollbar consists of upside arrow, downward arrow, a slider box.**
- **The upside arrow is used to move item by incrementally up to its extreme edge.**
- **The slider box can be dragged inorder to move randomly along the list.**
- **The following are the constructors:**
  - **i) Scrollbar( ) – It creates a vertical scrollbar by default.**
  - **ii) Scrollbar(int style) – It creates a scrollbar as specified.**
  - **iii) Scrollbar(int style, int intialvalue, int thumbsize, int min, int max) – It creates a scrollbar with specified properties.**
- **The following are the methods:**
  - **i) void setValue(int newvalue) – It is used to set the current position of the scrollbar.**
  - **ii) void setPageIncrement(int pagesize) – It is used to set the page increment.**

Q: Write a java program that creates two threads. First thread prints the numbers from 1 to 100 and the second thread prints the numbers from 100 to 1.

```
class A extends Thread
{
        public void run( )
        {
                for(int i=1; i<=100; i++)
                {
                        System.out.println("From Thread A: i="+i );
                }
                System.out.println("Exit from thread A");
}
class B extends Thread
{
        public void run( )
        {
                for(int j=100; j>=1; j-- )
                {
                        System.out.println("From Thread B: j="+j );
                }
                System.out.println("Exit from thread B");
}
class ThreadDemo
{
        public static void main( String args[ ] )
        {
                Thread A = new A( );
                A.start( );
                Thread B = new B( );
                B.start( );
        }
}
```

Q: Design and develop an applet with six buttons namely yellow, blue, green, red, black, pink. By selecting on any button, it must generate the corresponding color as output.

Colordemo.java

```
import java.awt.*;
import java.applet.*;
import java.awt.event.*;

public class Colordemo extends Applet implements ActionListener
{
        Button y,b,g,r,bl,p;

        public void init( )
        {
                y = new Button("YELLOW");
                b = new Button("BLUE");
                g = new Button("GREEN");
                r = new Button("RED");
                bl = new Button("BLACK");
                p = new Button("PINK");

                add(y);
                add(b);
                add(g);
                add(r);
                add(bl);
                add(p);

                y.addActionListener(this);
                b.addActionListener(this);
                g.addActionListener(this);
                r.addActionListener(this);
                bl.addActionListener(this);
                p.addActionListener(this);
        }

        public void actionPerfofmed(ActionEvent e)
        {
                String s = e.getActionCommand( );
                if(s.equals(“YELLOW”)
                        setBackground(Color.yellow);
                else if(s.equals(“BLUE”)
                        setBackground(Color.blue);
                else if(s.equals(“GREEN”)
                        setBackground(Color.green);
                else if(s.equals(“RED”)
                        setBackground(Color.red);
```

```
            else if(s.equals("BLACK")
                setBackground(Color.black);
            else if(s.equals("PINK")
                setBackground(Color.pink);
        }
}
```

Colors.html

```
<applet code = Colordemo width = 500 height = 500 >
</applet>
```

Cursor movement using mouse

Q: Write a JAVA program that display the x and y position of the cursor movement using Mouse.

MouseEvent.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class MousEevent extends Applet implements MouseListener,
                    MouseMotionListener
{
    String s1=" ";
    int x,y;

    public void init()
    {
        addMouseListener(this);
        addMouseMotionListener(this);
    }

    public void mouseClicked(MouseEvent me)
    {
        x=100;
        y=100;
        s1="Mouse clicked";
        repaint();
    }

    public void mouseEntered(MouseEvent me)
    {
        x=100;
        y=200;
        s1="Mouse entered";
        repaint();
    }
```

```
    public void mouseExited(MouseEvent me)
    {
        x=100;
        y=300;
        s1="Mouse exited";
        repaint();
    }

    public void mousePressed(MouseEvent me)
    {
        x=me.getX();
        y=me.getY();
        s1="Mouse Pressed";
        repaint();
    }

    public void mouseReleased(MouseEvent me)
    {
        x=me.getX();
        y=me.getY();
        s1="Mouse Realeased";
        repaint();
    }

    public void mouseDragged(MouseEvent me)
    {
        x=me.getX();
        y=me.getY();
        s1="Mouse Dragged";
        repaint();
    }

    public void mouseMoved(MouseEvent me)
    {
        x=me.getX();
        y=me.getY();
        s1="Mouse Moved";
        repaint();
    }

    public void paint(Graphics g)
    {
        g.drawString(s1,x,y);
    }
}
```

Mouse.html

```
<applet code=MouseEvent width=450 height=300>
</applet>
```

Key-up and Key-down event

Write a JAVA program that identifies key-up key-down event user entering text in a Applet.

KeyEvent.java

```
import java.awt.*;
import java.awt.event.*;
import java.applet.*;

public class KeyEvent extends Applet implements KeyListener
{
	String s1=" ";
	int x,y;
	public void init()
	{
		addKeyListener(this);
		requestFocus();
	}
	public void keyPressed(KeyEvent ke)
	{
		x=100;
		y=200;
		s1= "key pressed ";
		repaint();
	}
	public void keyReleased(KeyEvent ke)
	{
		x=100;
		y=400;
		s1= "key Released ";
		repaint();
	}

	public void keyTyped(KeyEvent ke)
	{
		s1=s1+ke.getKeyChar();
		repaint();
	}

	public void paint(Graphics g)
	{
		g.drawString(s1,x,y);
	}
}
```

Keys.html

```
<applet code="KeyEvent" width=450 height=300>
</applet>
```

Q: Program to perform arithmetic operations with text fields, labels and buttons.

```
import java.awt.*;
import java.awt.event.*;
import java.applet.Applet;

public class AwtControls extends Applet implements Actionlistener
{
        TextField t1,t2,t3;
        Button b1,b2,b3,b4;
        Label x,y,z;
        String msg=" ";

        public void init( )
        {
                x = new Label("enter first number=");
                add(x);
                t1=new TextField(15);
                add(t1);

                y = new Label("enter second number=");
                add(y);
                t2=new TextField(15);
                add(t2);

                z = new Label("Result=");
                add(z);
                t3=new TextField(15);
                add(t3);

                b1 = new Button("ADD");
                add(b1);
                b1.addActionListener(this);

                b2 = new Button("SUB");
                add(b2);
                b2.addActionListener(this);

                b3 = new Button("MUL");
                add(b3);
                b3.addActionListener(this);

                b4 = new Button("DIV");
                add(b4);
                b4.addActionListener(this);
        }
        public void actionPerformed(ActionEvent ae)
        {
                if(ae.getSource( ) == b1)
                {
```

```
                int x = Integer.parseInt(t1.getText( ));
                int y = Integer.parseInt(t2.getText( ));
                int a = x + y;
                t3.setText(“  “ + a);
            }
            if(ae.getSource( ) == b2)
            {
                int x = Integer.parseInt(t1.getText( ));
                int y = Integer.parseInt(t2.getText( ));
                int s = x - y;
                t3.setText(“  “ + s);
            }
            if(ae.getSource( ) == b3)
            {
                int x = Integer.parseInt(t1.getText( ));
                int y = Integer.parseInt(t2.getText( ));
                int m = x * y;
                t3.setText(“  “ + m);
            }
            if(ae.getSource( ) == b4)
            {
                int x = Integer.parseInt(t1.getText( ));
                int y = Integer.parseInt(t2.getText( ));
                int d = x / y;
                t3.setText(“  “ +d);
            }
            showStaturs(“Labels, Text and Buttons action performance”);
            repaint( );
        }
}
```

Controls.html

```
<html>
<applet code = “AwtControls” width=400 height=350>
</applet>
</html>
```

Q: Write a JAVA program to build calculator in swings.

Calculator.java

```
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;

class Calculator extends JPanel implements ActionListener
{
        JTextField jt = new JTextField();
        double d= 0;
        String op = "=";
```

```
boolean b1 = true;
Calculator( )
{
        setLayout(new BorderLayout());
        jt.setEditable(false);
        add(jt, "North");
        JPanel jp = new JPanel();
        jp.setLayout(new GridLayout(4, 4));
        String s1 = "789/456*123-0.=+";
        for (int i = 0; i < s1.length(); i++)
        {
                JButton b = new JButton(s1.substring(i, i + 1));
                jp.add(b);
                b.addActionListener(this);
        }
        add(jp, "Center");
}
public void actionPerformed(ActionEvent ae)
{
        String s1 = ae.getActionCommand();
        if ('0' <= s1.charAt(0) && s1.charAt(0) <= '9' || s1.equals("."))
        {
                 if (b1)
                 jt.setText(s1);
                 else
                 jt.setText(jt.getText() + s1);
                 b1 = false;
        }
        else
        {
                if (b1)
                {
                        if (s1.equals("-"))
                        {
                                jt.setText(s1);
                                b1 = false;
                        }
                        else
                        op = s1;
                }
                else
                {
double x = Double.parseDouble(jt.getText()); calculate(x);
op = s1;
b1 = true;
                }
            }
    }
```

```
        private void calculate(double n)
        {
                if (op.equals("+"))
                d += n;
                else if (op.equals("-"))
                d -= n;
                else if (op.equals("*"))
                d *= n;
                else if (op.equals("/"))
                d /= n;
                else if (op.equals("="))
                d = n;
                jt.setText("" + d);
         }
         public static void main(String[] args)
         {
                JFrame jf = new JFrame();
                jf.setTitle("calculator");
                jf.setSize(300, 300);
                Container c = jf.getContentPane();
                c.add(new calculator());
                jf.show();
        }
}
```

Displaying digital watch

Q: Write a JAVA program to display the digital watch in swing tutorial

Digitalwatch.java

```
import java.awt.*;
import java.applet.*;
import java.util.*;

public class Digitalwatch extends Applet implements Runnable
{
        Thread t,t1;
        public void start()
        {
                t = new Thread(this);
                t.start();
        }
        public void run()
        {
                t1 = Thread.currentThread();
                while(t1 == t)
                {
                        repaint();
                        try
                        {
```

```
                        t1.sleep(1000);
                    }
                    catch(Exception e)
                    {
                        System.out.println(e);
                    }
                }
        }
        public void paint(Graphics g)
        {
            Calendar cal = new GregorianCalendar();
             String  h = String.valueOf(cal.get(Calendar.HOUR));
             String m = String.valueOf(cal.get(Calendar.MINUTE));
             String s = String.valueOf(cal.get(Calendar.SECOND));
             g.drawString(h + ":" + m + ":" + s, 20, 30);
        }
}
```

watch.html

```
<applet code="digitalwatch" width=450 height=300>
</applet>
```

Ball bouncing inside a JPanel

Write a JAVA program that to create a single ball bouncing inside a JPanel.

Bounchingball.java

```
import java.awt.*;
import javax.swing.*;

public class Bouncingball extends JPanel
{
        int w,h;
        float r = 40, d= r * 2, X = r + 50, Y = r + 20,dx = 3,dy = 3;
        public Bouncingball( )
        {
            Thread thread = new Thread()
            {
                public void run()
                {
                    while (true)
                    {
                        w = getWidth();
                        h = getHeight();
                        X = X + dx ;
                        Y = Y + dy;
```

```
                        if (X - r < 0)
                        {
                                dx = -dx;
                                X = r;
                        }
                        else if (X + r > w)
                        {
                                dx = -dx;
                                X = w - r;
                        }
                        if (Y - r < 0)
                        {
                                dy = -dy;
                                Y = r;
                        }
                        else if (Y + r > h)
                        {
                                dy = -dy;
                                Y = h - r;
                        }
                        repaint();
                        try
                        {
                         Thread.sleep(50);
                        }

                        catch (Exception e)
                        {
                                System.out.println(e);
                        }
                    }
                }
            };
            thread.start();
        }
        public void paintComponent(Graphics g)
        {
            super.paintComponent(g);
            g.setColor(Color.BLUE);
            g.fillOval((int)(X-r), (int)(Y-r), (int)d, (int)d);
        }
        public static void main(String[] args)
        {
            JFrame jf = new JFrame("bouncing ball");
            jf.setSize(300, 200);
            jf.setContentPane(new bouncingball());
            jf.setVisible(true);
        }
}
```

b)Displaying a real tree

Aim: To write a JAVA program JTree as displaying a real tree upside down

Realtree.java

```
import javax.swing.*;
import javax.swing.tree.*;
class Realtree
{
        public static void main(String[] args)
        {
                JFrame jf = new JFrame();
                DefaultMutableTreeNode d1 = new
               DefaultMutableTreeNode("Color", true);
               DefaultMutableTreeNode d2 = new
               DefaultMutableTreeNode("Black");
DefaultMutableTreeNode d3 = new DefaultMutableTreeNode("Blue");
DefaultMutableTreeNode d4 = new DefaultMutableTreeNode("Navy Blue");
DefaultMutableTreeNode d5 = new DefaultMutableTreeNode("Dark Blue");
DefaultMutableTreeNode d6 = new DefaultMutableTreeNode("Green");
DefaultMutableTreeNode d7 = new DefaultMutableTreeNode("White");

               d1.add(d2);
               d1.add(d3);
               d3.add(d4);
               d3.add(d5);
               d1.add(d6);
               d1.add(d7);

               JTree jt = new JTree(d1);
               jf.add(jt);
               jf.setSize(200,200);
               jf.setVisible(true);
        }
}
```