**Module 1:**

1.Discuss the three OOP principles.

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing procedures or methods that perform operations on the data, while object-oriented programming is about creating objects that contain both data and methods.

The Three OOP Principles

The three Object oriented programming (OOP) principles are 1)encapsulation

2) inheritance

3)polymorphism.

1) Encapsulation:

\*general meaning of encapsulation is the action of enclosing something in or enclosing some thing in a capsule.

\* Encapsulation is a mechanism that binds code and the data together, and keeps both of them safe from misuse. and outside interference

Or

**\*Encapsulation in Java** is a process of wrapping code and data together into a single unit, for example, a capsule which is mixed of several medicines.

\* The whole idea behind encapsulation is to hide the implementation details from users

\* .In encapsulation, the data in one class is hidden from other classes so data in one class cannot be accessed by other classes, so it is also known as data-hiding

\* Both Abstraction & Encapsulation works hand in hand because Abstraction says what details to be made visible & Encapsulation provides the level of access to that visible details.

To hide data from users , you must:

* declare class variables/attributes/methods as private
* provide public **get** and **set** methods to access and update the value of a private variable

**Example** :

Power steering of a car is a complex system, which internally have lots of components tightly coupled together, they work synchronously to turn the car in the desired direction. **But to the external world there is only one interface i.e., the steering wheel is available and rest of the complexity is hidden.**

Similarly in java if want to protect the data u need declare variables/attributes/methods as private .so as they are private they can be accessed in the declared class only cannot be accessed by any other classes

Advantages of Encapsulation:

• **Data Hiding:** The user will have no idea about the inner implementation of the class

• **Increased Flexibility:** We can make the variables of the class as read-only or write-only based on our requirement.

• **Reusability**: Encapsulation also improves the re-usability

• **Testing code is easy**: Encapsulated code is easy to test for unit testing.

Inheritance:

It is **the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class**.

Important terminology:

• **Super Class or a base class or a parent class**:

The class whose features are inherited is known as super class

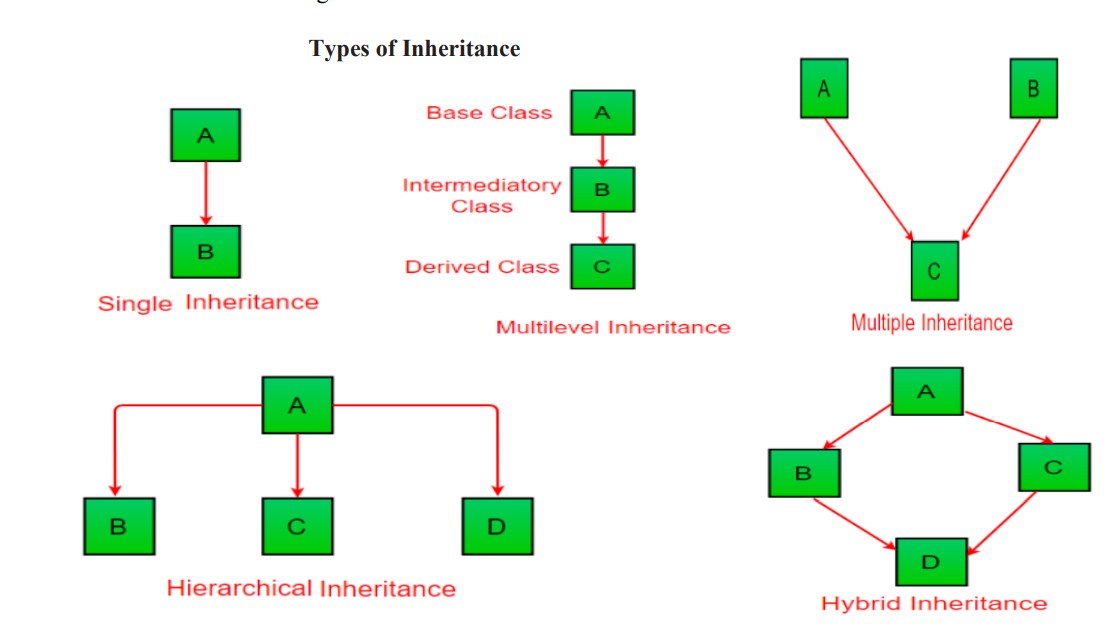
• **Sub Class or a derived class, extended class, or child class**:

\*The class that acquires or inherits the other class features is known as sub class().

\*The subclass can add its own fields and methods in addition to the superclass fields and methods.

• Reusability:

Inheritance supports the concept of ―reusability‖, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.



In the example below, the Car class (subclass) takes or inherits the attributes and methods from the Vehicle class (superclass)

|  |
| --- |
| class hans//super class  {      int k=1;      void sum()      {          System.out.println(k+1);      }  }  class chinnu extends hans  //sub class  {      int r=4;      public static void main(String[] args) {          chinnu obj=new chinnu();          System.out.println(obj.r+obj.k);//adding var of super class and sub class          obj.sum();//calling method of super class      }  }  Output:  5  2 |

Polymorphism

Refer below link

[link](https://www.programiz.com/java-programming/polymorphism#:~:text=Note%3A%20The%20print()%20method,Method%20Overriding)

Polymorphism is an important concept of object-oriented programming. It simply means more than one form.

That is, the same entity (method or operator or object) can perform different operations in different scenarios.

## Example: Java Polymorphism

class Polygon {

// method to render a shape

public void render() {

System.out.println("Rendering Polygon...");

}

}

class Square extends Polygon {

// renders Square

public void render() {

System.out.println("Rendering Square...");

}

}

class Circle extends Polygon {

// renders circle

public void render() {

System.out.println("Rendering Circle...");

}

}

class Main {

public static void main(String[] args) {

// create an object of Square

Square s1 = new Square();

s1.render();

// create an object of Circle

Circle c1 = new Circle();

c1.render();

}

}

[Run Code](https://www.programiz.com/java-programming/online-compiler)

**Output**

Rendering Square...

Rendering Circle...

In the above example, we have created a superclass: Polygon and two subclasses: Square and Circle. Notice the use of the render() method.

The main purpose of the render() method is to render the shape. However, the process of rendering a square is different than the process of rendering a circle.

Hence, the render() method behaves differently in different classes. Or, we can say render() is polymorphic.

### Why Polymorphism?

Polymorphism allows us to create consistent code. In the previous example, we can also create different methods: renderSquare() and renderCircle() to render Square and Circle, respectively.

This will work perfectly. However, for every shape, we need to create different methods. It will make our code inconsistent.

To solve this, polymorphism in Java allows us to create a single method render() that will behave differently for different shapes.

2) What is an array? Implement java program to search an element in the array using for each loop

* An *array is a group of like-typed variables that are referred to by a common name.*
* *Arrays of* any type can be created and may have one or more dimensions.
* A specific element in an array is accessed by its index.
* Arrays offer a convenient means of grouping related information.
* **One-Dimensional Arrays**
  + The general form of a one-dimensional array declaration is

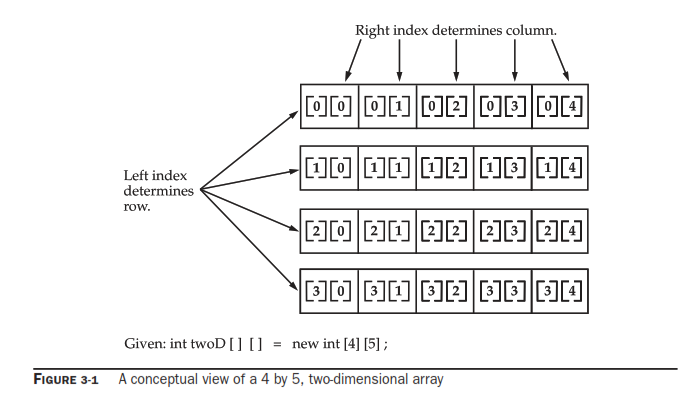
*type var-name[ ];*

* + Here, *type declares the base type of the array.*
  + *Ex:* int month\_days[];
  + This declaration establishes the fact that month\_days is an array variable,
  + No array actually exists.
* To link month\_days with an actual, physical array of integers, you must allocate one using new and assign it to month\_daysThe general form of **new as it applies to one-dimensional arrays appears** as follows:
  + *array-var = new type[size];*
* to use **new to** allocate an array, you must specify the type and number of elements to allocate.
* The elements in the array allocated by **new will automatically be initialized to zero.**
* ***So,*** Arrays will be created in two steps, First array variable is created, then it will be physically created by using new keyword.

int month\_days[];

Month\_days=new int[12];

* **In Java, multidimensional arrays are actually arrays of arrays.**
* To declare a multidimensional array variable, specify each additional index using another set of square brackets.
* **Ex: int twoD[][]=new int[4][5];**



java program to search an element in the array using for each loop.

|  |
| --- |
| import java.util.**\***;  class foreachlooparray {      public static void main(String *args*[]) {          Scanner sc1 = new Scanner(System.in);          System.out.println("enter the siaze of the array note enter size of array is less than 100");          int size = sc1.nextInt();          int h = 0;          int array1[] = new int[100];          for (int i = 0; i < size; i++) {              System.out.println("enter the element to be inserted at " + i);              array1[i] = sc1.nextInt();          }          System.out.println("enter the element u wanted to search for");          int ele = sc1.nextInt();          for (int k : array1) {              if (array1[k] == ele) {                  h = h + k;              }          }          if (h == 0) {              System.out.println("element not found");          }          else{              System.out.println("element is found at"+h);          }      }  }  } |

3)Explain the switch statement with example.

The switch statement is Java‘s multiway branch statement

. It provides an easy way to skip some parts of code and execute some other parts of code based on the value of an expression.

It provides a better alternative to large series of if-else-if statements.

Here is the general form of a switch statement:

|  |
| --- |
| switch (expression or value)  {  case value1:  statement 1;  statement 2;  .  .  Statement n;  break;  case value2:  statement 1;  statement 2;  .  .  Statement n;  break;  .  .  .  .  case valueN:  statement 1;  statement 2;  .  .  Statement n;  break;  default:  statement 1;  statement 2;  .  .  Statement n;  break;  } |
|  |

Ex: switch with expression values of integers

|  |
| --- |
| import java.util.Scanner;  public class switch1 {  public static void main(String args[]) {  System.out.println("1:hi 2:bye 3:tata");  Scanner sc = new Scanner(System.in);  int a = sc.nextInt();  switch (a) {  case 1:  System.out.println("hi");  break;  case 2:  System.out.println("bye");  break;  case 3:  System.out.println("tata");  break;  }  }  } |

switch with expression values as char

|  |
| --- |
| import java.util.Scanner;  public class switch1 {     public static void main(String args[]) {        System.out.println("a:hi b:bye c:tata");        Scanner sc = new Scanner(System.in);        char a = sc.next().charAt(0);        switch (a)        {           case 'a':           {              System.out.println("hi");              break;           }             case 'b':           {              System.out.println("bye");              break;           }           case 'c':           {              System.out.println("tata");              break;           }           default:           {              System.out.println("u entered invalid value");              break;           }        }     }  } |

4)Explain while and do-while statements with an example and Write a [java a program to display Fibonacci Series based on the user input using while loop](https://beginnersbook.com/2017/09/java-program-to-display-fibonacci-series-using-loops/).

Loops:

\*Loops can execute a block of code as long as a specified condition is reached.

\*Loops are handy because they save time, reduce errors, and they make code more readable.

While loop:

\*The while loop loops through a block of code as long as a specified condition is true

Or

The while loop iterates through a block of code as long as a specified condition is true

### **Syntax**

while (condition) {

*// code block to be executed*

}

EX:

public class Main {

public static void main(String[] args) {

int i = 0;

while (i < 5)

{

System.out.println(i);

i++;

}

}

}Output:

0  
1  
2  
3  
4

**Note:** Do not forget to increase the variable used in the condition, otherwise the loop will never end!

Do/While loop:

\*The do/while loop is a variant of the while loop.

\*This loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

### **Syntax**

do {

*// code block to be executed*

}

while (condition);

The example below uses a do/while loop. The loop will always be executed at least once, even if the condition is false, because the code block is executed before the condition is tested:

### **Example**

|  |
| --- |
| public class dowhile {      public static void main(String[] args) {          int i=0;          do          {              System.out.println("hansi");          }          while(i>0);      }  } |

Output:

## Hansi

## See the output evenrthough the condition is not satisfied hansi has been printed it means that the statements in do are executed for once the condition is checked and as long as the condition is true the statements in do keep on executing

## What is Fibonacci Sequence?

The **Fibonacci sequence,**also known as Fibonacci numbers, is defined as the sequence of numbers in which each number in the sequence is equal to the sum of two numbers before it. The Fibonacci Sequence is given as:

**Fibonacci Sequence = 0, 1, 1, 2, 3, 5, 8, 13, 21, ….**

Here, the third term “1” is obtained by adding the first and second term. (i.e., 0+1 = 1)

Similarly,

“2” is obtained by adding the second and third term (1+1 = 2)

“3” is obtained by adding the third and fourth term (1+2) and so on.

For example, the next term after 21 can be found by adding 13 and 21. Therefore, the next term in the sequence is 34.

|  |
| --- |
| import java.util.Scanner;  public class fibonacciseriesusingwhile  {      public static void main(String[] args)      {          Scanner sc=new Scanner(System.in);          int fv=0;          int sv=1;          int nv=0;          System.out.println("enter no of fibonacci series elements u wanted to print");          int n=sc.nextInt();          int i=0;          while(i<n)          {              System.out.println(fv);              nv=fv+sv;              fv=sv;              sv=nv;              i++;          }      }  } |

5) Discuss if-else-if ladder with an example

A common programming construct that is based upon a sequence of nested ifs istheif-else-if ladder. The if statements are executed from the top down.

It looks like:

**if**(condition1)

{

//code to be executed if condition1 is true

}

**else** **if**(condition2)

{

//code to be executed if condition2 is true

}

**else** **if**(condition3)

{

//code to be executed if condition3 is true

}

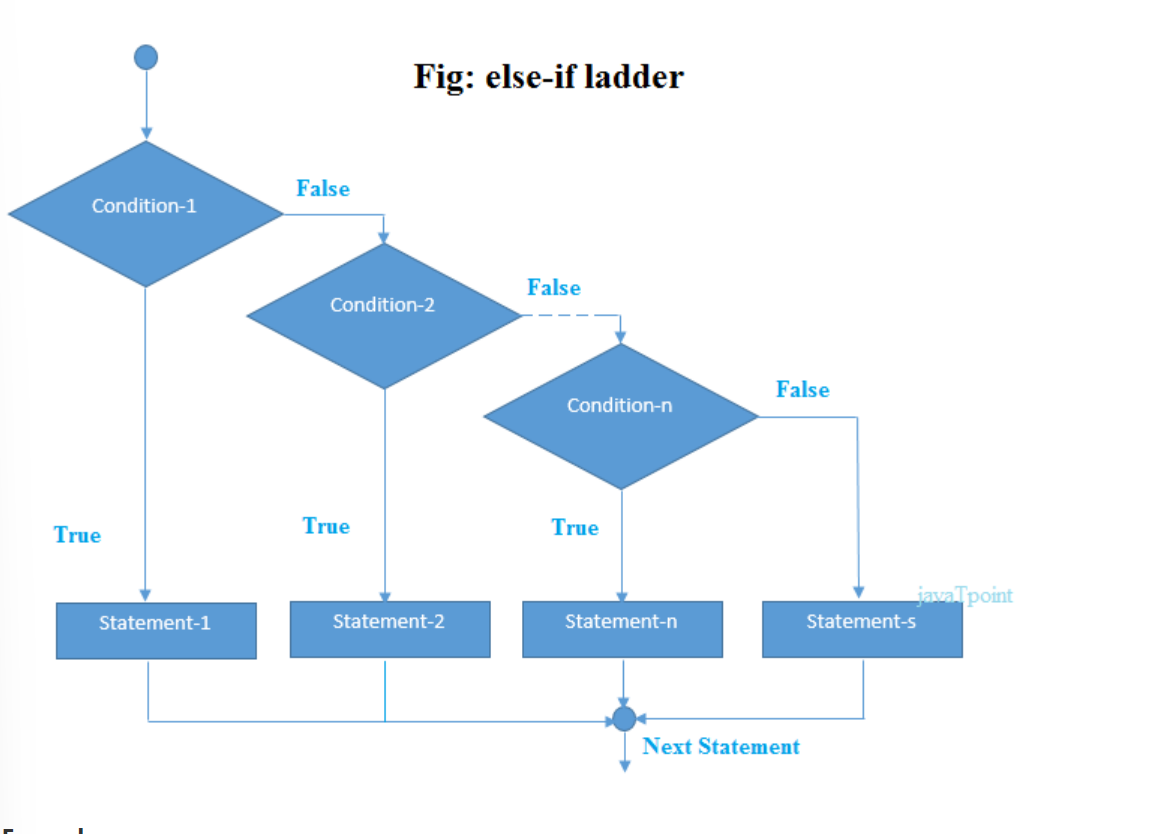
...

**Else**

{

//code to be executed if all the conditions are false

}





EXAMPLE:

|  |
| --- |
| 1. /Java Program to demonstrate the use of If else-if ladder. 2. //It is a program of grading system for fail, D grade, C grade, B grade, A grade and A+. 3. **public** **class** IfElseIfExample { 4. **public** **static** **void** main(String[] args) { 5. **int** marks=65; 7. **if**(marks<50){ 8. System.out.println("fail"); 9. } 10. **else** **if**(marks>=50 && marks<60){ 11. System.out.println("D grade"); 12. } 13. **else** **if**(marks>=60 && marks<70){ 14. System.out.println("C grade"); 15. } 16. **else** **if**(marks>=70 && marks<80){ 17. System.out.println("B grade"); 18. } 19. **else** **if**(marks>=80 && marks<90){ 20. System.out.println("A grade"); 21. }**else** **if**(marks>=90 && marks<100){ 22. System.out.println("A+ grade"); 23. }**else**{ 24. System.out.println("Invalid!"); 25. } 26. } 27. } |

Pre increment:

 we can say that the pre-increment operator increases the value of the variable first and then use it in the expression.

**Syntax:**

b = ++a;

**For example**, if the initial value of a were 5, then the value 6 would be assigned to b.

Post increment:

 in post-increment value is first used in the expression, and then it is incremented.

**Syntax:**

b = a++;

For example, assume the initial value of a to be 5. Then after executing the above statement the final value of b will be 5 as the value of a will be incremented after performing the expression.

6)Explain the working of short circuit logical operators with an examples.

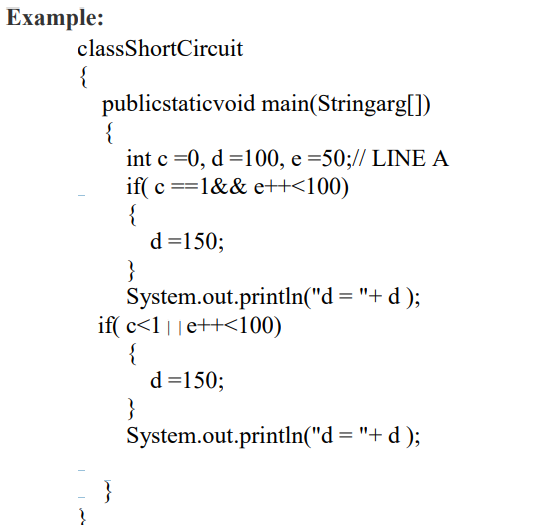
**Short Circuit Logical Operators**

\* Short circuit logical operators evaluate second expression only if it is needed.

\* When short-circuit AND (&&) is used, if the first value is false, second value is not evaluated as the result is false irrespective of the second value.

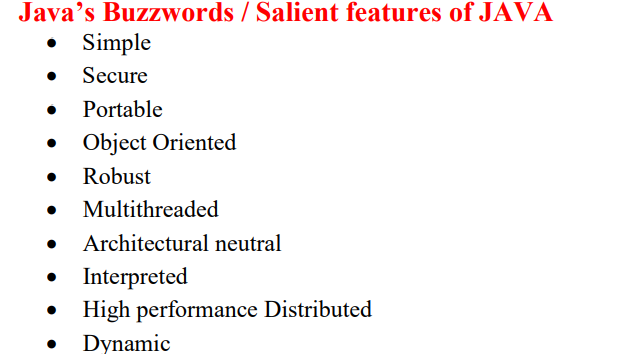
\* Similarly for short-circuit OR (||), if the first value is true, then second value is not evaluated as the result is true irrespective of the second value..

\* These short-circuit operators will be useful when we want to control the evaluation of right hand operand.



Or

7) Explain Java’s buzzwords or salient features of Java



**Simple:**

Java was designed to be easy for the professional programmer to learn and use effectively. If one has some programming experience, he will not find Java hard to master.If you already understand the basic concepts of object-oriented programming, learning Javawill be even easier.

**Secure :**

Java is secure bcz if u are using java compatible webbrowser if u download something from the internet there are chances of viral infection which may steal the personal information like bank account numbers ,credit card details etc which are on our local system.so java supportable browser will protects us by downloading applets with out fear of viral infection bcz java achieved this protection by allowing an applet to the Java execution environment and not allowing it access to other parts of the computer

**Portable :**

Java is portable means programs written in java will work on different types of computers as well as operating systems.

**Object Oriented**

Java is purely object oriented programming language because without class and object it is impossible to write any Java program. Java is not pure object oriented programming language. because java supports primitive datatypes like int ,float ,boolean, double, long etc.

**Robust :**

Java is a robust language for effective memory management and exception handling.

For example, in C/C++, the programmer must manually allocate and free all dynamic memory. This sometimes leads to problems, because programmers might forget to free. Java on its own eliminates these problems by managing memory allocation and deallocation automatically

Exceptional conditions arise in situations such as division by zero or file not found and so on. Java handles these types of exceptions effectively at run time.

**Multithreaded :**

Java was designed to meet the real-world requirement of creating interactive, networked programs. To accomplish this, Java supports multithreaded programming, which allows us to write programs that do many things simultaneously.

**Distributed** :

Java is designed for the distributed environment of the Internet because it handles TCP/IP protocols. Java also supports Remote Method Invocation (RMI). This feature enables a program to invoke methods across a network.

**Architectural Neutral:**

main issue for the Java designers was that of code longevity and portability. One of the main problems facing programmers is that no guarantee exists that if you write a program today, it will run tomorrow—even on the same machine. Operating system upgrades ,processor upgrades, and changes in core system resources can all combine to make a program malfunction. The Java designers made several hard decisions in the Java language and the Java Virtual Machine in an attempt to alter this situation. Their goal was ―write once; run anywhere, any time, forever.‖

**Interpreted and High performance** :

if u write the programs in java then they are compiled into an intermediate representation called Java bytecode. This code can be executed on any system that has the Java Virtual Machine.

Java bytecode was carefully designed so that it would be easy to translate directly into native machine code for very high performance by using a just-in-time compiler.

**Dynamic** :

Java programs carrys substantial amounts of run-time type information with them that is used to verify and resolve accesses to objects at run time. This makes it possible to dynamically link code in a safe manner. Java Development Kit (JDK) The Java Development Kit (JDK) is a software development environment used for developing Java applications and applets. It includes the Java Runtime Environment (JRE), an interpreter/loader (java), a compiler (javac), an archiver (jar), a documentation generator (javadoc) and other tools needed in Java development. JDK is a package of tools for developing Java-based software, whereas the JRE is a package of tools for running Java code.

8) Write a java program to check a number is prime or not using for loop.

|  |
| --- |
| import java.util.**\***;  public class primenumberornot {      public static void main(String *args*[]) {          System.out.println("enter the number u want to check whether prime or not");          Scanner sc1 = new Scanner(System.in);          int a = sc1.nextInt();          int i;          int j = 0;          for (i = 1; i <= a; i++) {              if (a % i == 0) {                  j = j + 1;              }          }          if (j <= 2) {              System.out.println("it is a prime number");          } else {              System.out.println("it is not a prime number");          }      }  } |

9) Discuss the ternary (?) operator with an example

short-hand [if else](https://www.w3schools.com/java/java_conditions.asp) is also known as the **ternary operator** because it consists of three operands.

It can be used to replace multiple lines of code with a single line, and is most often used to replace simple if else statements:

### **Syntax**

If variable is already declared

variable *= (*condition*) ?* expressionTrue *:*  expressionFalse*;*

if variable is not declared yet:

datatype variable *= (*condition*) ?* expressionTrue *:*  expressionFalse*;*

here:

\*variable is assigned with expesiionTrue or expressionFalse based on the condition for ex:

|  |
| --- |
| public class ternaryoperator  {      public static void main(String[] args) {          //datatype variable = (condition) ? expressionTrue :  expressionFalse;          int i=2;          String result=(i%2==0)? "even" :"odd";          System.out.println(result);      }  } |

If we write this in if else format:

|  |
| --- |
| public class Main {  public static void main(String[] args) {  int x=2;  if (x%2==0)  {  System.out.println(“ even number”);  }  else  {  System.out.println("odd number”);  }  }  } |

10) Write a j[ava a program to find factorial using while loop](https://beginnersbook.com/2017/09/java-program-to-display-fibonacci-series-using-loops/)

import java.util.**\***;

public class factorialofanumberusingforloop {

    public static void main(String[] *args*) {

        Scanner sc1 = new Scanner(System.in);

        System.out.println("enter the number to which factorial need to be finded");

        int n = sc1.nextInt();

        int i = 1;

        int fact = 1;

        while (i <= n) {

            fact = fact \* i;

            i = i + 1;

        }

        System.out.println("Factorial of " + n + " is: " + fact);

    }

}

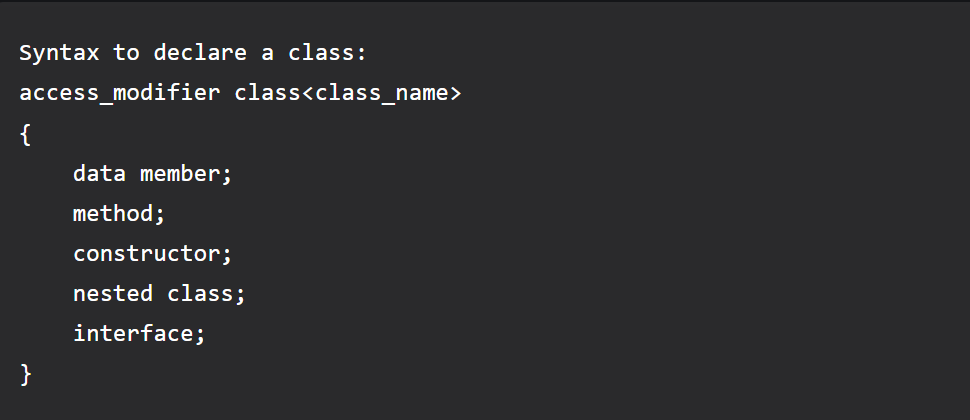
MODULE

2

1. What is a class? What are its characteristics? Give its general structure.

Class is a set of objects which shares common characteristics/ behavior and common properties/ attributes.  
2. Class is not a real world entity. It is just a template or blueprint or prototype from which objects are created.  
3. Class does not occupy memory.  
4. Class is a group of variables of different data types and group of methods.

A class in java can contain:  
• data member or attribute or variable  
• method  
• constructor  
• nested class and   
• interface



EX

|  |
| --- |
| class Student {      int id = 1;*// data member (also instance variable)*      String name = "hansi"; *// data member (also instance variable)*      public static void main(String *args*[]) {          Student s1 = new Student();*// creating an object of Student*          System.out.println(s1.id);          System.out.println(s1.name);      }  } |

|  |
| --- |
| /\*  Area of circle=  Circumference of circle=2πr  Area of square =a^2  diagnol of square =√2 a  volumeof sphere =(4πr^3)/3  surface area of sphere =4πr^2  \*/  ds  import java.util.\*;  class Shape {  double dimension;  Shape() {  System.out.print("Enter Dimension : ");  Scanner sc = new Scanner(System.in);  this.dimension = sc.nextInt();  }  }  class Circle extends Shape {  public void calc\_area(){  double area = 3.14\*this.dimension\*this.dimension;  System.out.println("The Area of Circle is : " + area);  }  public void calc\_circ() {  double circ = 2\*3.14\*this.dimension;  System.out.println("The Circumference is : " + circ);  }  }  class Square extends Shape {  public void calc\_area() {  double area = this.dimension\*this.dimension;  System.out.println("The Area is of square : " + area);  }  public void diag\_len() {  double diag = this.dimension \* 0.414;//(sqrt(2)\*a)  System.out.println("The length of the diagonal is : " + diag);  }  }  class Sphere extends Shape {  public void calc\_vol() {  System.out.println("the radius val taken here is " + this.dimension);  double vol = 4\*3.14\*(this.dimension\*this.dimension\*this.dimension)/3;  System.out.println("The volume of the sphere is : " + vol);  }  public void calc\_sa() {  double sa = 4\*3.14\*this.dimension\*this.dimension;  System.out.println("The surface area of the sphere is : " + sa);  }  }  public class QBprob {  public static void main(String []args) {  System.out.println("Creating Circle");  Circle c1 = new Circle();  System.out.println("Creating Square");  Square sq1 = new Square();  System.out.println("Creating Sphere");  Sphere sp1 = new Sphere();  c1.calc\_area();  c1.calc\_circ();  sq1.calc\_area();  sq1.diag\_len();  sp1.calc\_vol();  sp1.calc\_sa();  }  } |

4)

Inheritance:

It is **the mechanism in java by which one class is allowed to inherit the features(fields and methods) of another class**.

Important terminology:

• **Super Class or a base class or a parent class**:

The class whose features are inherited is known as super class

• **Sub Class or a derived class, extended class, or child class**:

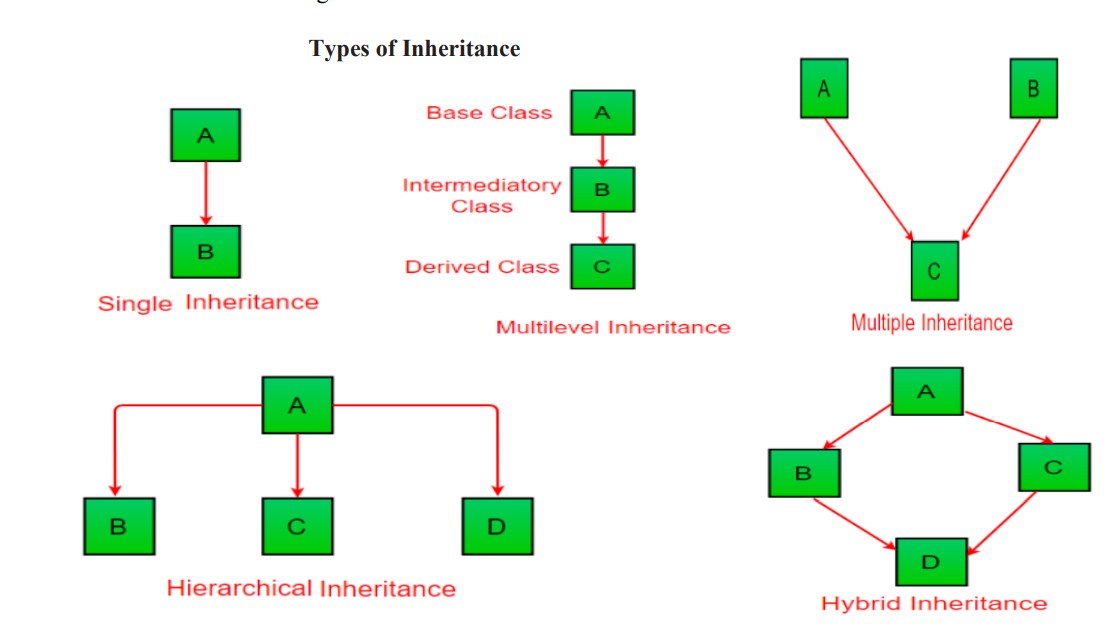
\*The class that acquires or inherits the other class features is known as sub class().

\*The subclass can add its own fields and methods in addition to the superclass fields and methods.

• Reusability:

Inheritance supports the concept of ―reusability‖, i.e. when we want to create a new class and there is already a class that includes some of the code that we want, we can derive our new class from the existing class.

Types of inheritance:



Ex of multilevel inheritance

class hansi   //base class

{

    int a=1;

    void sum()

    {

        System.out.println(a);

    }

}

class hansi1 extends hansi //intermediate class

{

    int b=3;

    void sum1()

    {

        System.out.println(a+b);

    }

}

class hansi2 extends hansi1 //derived class

{

    int c=4;

public static void main(String[] args) {

    hansi2 obj=new hansi2();

    System.out.println(obj.a+obj.b+obj.c);

    obj.sum();//accessing base class method

    obj.sum1();//accessing intermediate class method

}

}

Output:

|  |
| --- |
| 8  1  4 |

5)Write a short note on finalize method.

## Just before destroying an object, Garbage Collector calls *finalize()* method on the object to perform clean-up activities. Once *finalize()* method completes, Garbage Collector destroys that object.

## *finalize()* method is present in Object class with following prototype

## protected void finalize() throws Throwable

## Based on our requirement, we can override *finalize()* method for perform our clean-up activities like closing connection from database

|  |
| --- |
| * public class TestGarbage1 * { * protected void finalize() * { * System.out.println("object is garbage collected"); * } * public static void main(String args[]){ * //System.gc();//if gc() will be here nothing will be done exp in 2 point of gc() * TestGarbage1 s1=new TestGarbage1(); * TestGarbage1 s2=new TestGarbage1(); * System.out.println("unreferencing s1"); * s1=null;  //unreferenceing s1 * System.out.println("unreferencing s2"); * s2=null; * int k; * //calling garbage collector explicitly * System.gc(); * } * } |

Output:

|  |
| --- |
| unreferencing s1  unreferencing s2  object is garbage collected  object is garbage collected |

 6) **Write a note on different access specifiers**

[link](https://www.w3schools.com/java/java_modifiers.asp)

**7)** Demonstrate the uses of the following: 1.this 2.static 3.super 4.final

**Comparing overriding and overloading**

| **Overriding** | **Overloading** |
| --- | --- |
| Implements “runtime polymorphism” | Implements “compile time polymorphism” |
| The method call is determined at runtime based on the object type | The method call is determined at compile time |
| Occurs between superclass and subclass | Occurs between the methods in the same class |
| Have the same signature (name and method arguments) | Have the same name, but the parameters are different |
| On error, the effect will be visible at runtime | On error, it can be caught at compile time |

10) Differentiate constructors and methods.

| **Constructors** | **Methods** |
| --- | --- |
| A Constructor is a block of code that initializes a newly created object. | A Method is a collection of statements which returns a value upon its execution. |
| A Constructor can be used to initialize an object. | A Method consists of Java code to be executed. |
| A Constructor is invoked implicitly by the system. | A Method is invoked by the programmer. |
| A Constructor is invoked when a object is created using the keyword **new**. | A Method is invoked through method calls. |
| A Constructor doesn’t have a return type. | A Method must have a return type. |
| A Constructor initializes a object that doesn’t exist. | A Method does operations on an already created object. |
| A Constructor’s name must be same as the name of the class. | A Method’s name can be anything. |
| A class can have many Constructors but must not have the same parameters. | A class can have many methods but must not have the same parameters. |
| A Constructor cannot be inherited by subclasses. | A Method can be inherited by subclasses. |