**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 it manipulates 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 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 as private if u do like that then any another class or any other code which is not a member of class cannot access the variable/attribute/method

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:

In general inheritance means acquiring traits by child from parents

It is **the mechanism in java by which one class is allowed to inherit the features(attributes 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

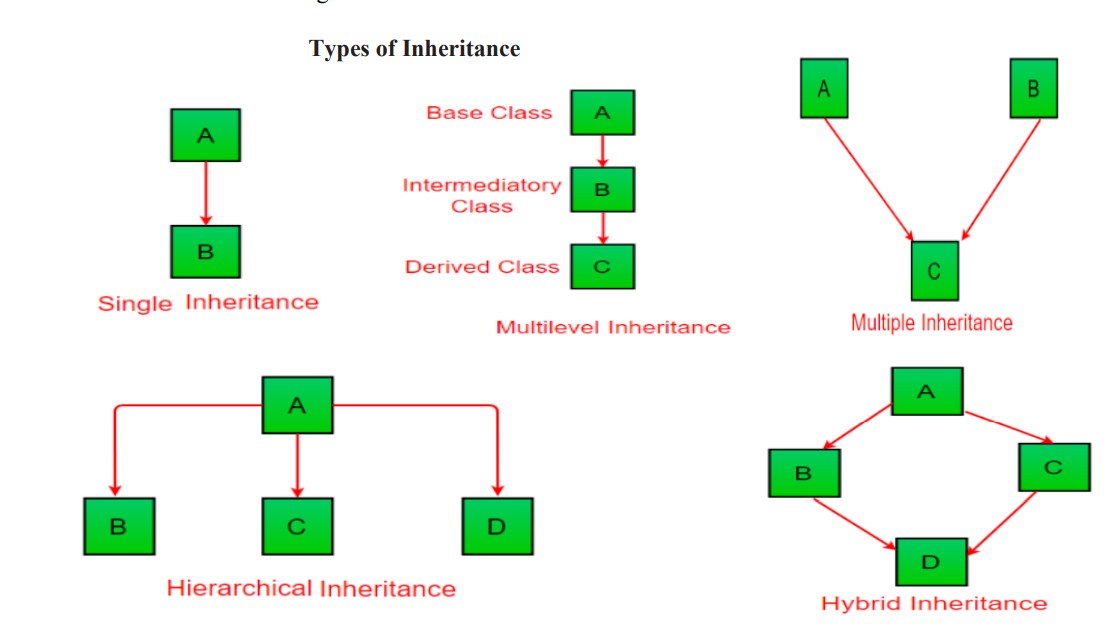
• **Sub Class or a derived class, extended class, or child class**:

\*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 Vehicle //superclass  {  public String company = "Ford"; // Vehicle attribute  public void method1() //note no static keyword // Vehicle method  {  System.out.println("Tuut, tuut!");  }  }  //subclass  class Car extends Vehicle  {  private String modelName = "Mustang"; // Car attribute  public static void main(String[] args)  {  // Creating a myCar object  Car myCar = new Car();  // Calling the method1() method (from the Vehicle class) on the myCar object  myCar.method1();  // Displaying the value of the company attribute (from the Vehicle class) and the value of the attribute modelName from the Car class  System.out.println(myCar.company + " " + myCar.modelName);  }  } |

Output:

|  |
| --- |
| Tuut, tuut! Ford Mustang |

Polymorphism

Polymorphism means "many forms", and it occurs when we have many classes that are related to each other by inheritance.

As we know [**Inheritance**](https://www.w3schools.com/java/java_inheritance.asp) lets us inherit attributes and methods from another class. **Polymorphism** uses those methods to perform different tasks. This allows us to perform a single action in different ways.

For example, think of a superclass called Animal that has a method called animalSound(). Subclasses of Animals could be Pigs, Cats, Dogs and they also have their own implementation of an animal sound (the pig oinks, and the cat meows

//NOTE:

In all classes method name is same

|  |
| --- |
| class Animal {  public void animalSound() {  System.out.println("The animal makes a sound");  }  }  class Pig extends Animal {  public void animalSound() {  System.out.println("The pig says: wee wee");  }  }  class Dog extends Animal {  public void animalSound() {  System.out.println("The dog says: bow wow");  }  }  class Main {  public static void main(String[] args) {  Animal myAnimal = new Animal(); // Create a Animal object  Pig myPig = new Pig(); // Pig mypig= new Pig() will alsoworks  Dog myDog = new Dog(); // Create a Dog object  myAnimal.animalSound();  myPig.animalSound();  myDog.animalSound();  }  } |

Output:

|  |
| --- |
| The animal makes a sound The pig says: wee wee The dog says: bow wow |

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

ARRAY:

1)Normally, an array is a collection of similar type of elements which are stored in contiguous memory location

2) **Java array** is an object which contains elements of a similar data type. Additionally, The elements of an array are stored in a contiguous memory location. It is a data structure where we store similar elements. We can store only a fixed set of elements in a Java array.

3) Array in Java is index-based, the first element of the array is stored at the 0th index, 2nd element is stored on 1st index and so on

4) Unlike C/C++, we can get the length of the array using the length member. In C/C++, we need to use the sizeof operator.

5) The **size** of an array must be specified by int or short value and not long.

|  |
| --- |
| public class Main {  public static void main(String[] args) {  String[] cars = {"Volvo", "BMW", "Ford", "Mazda"};  System.out.println(cars.length);  }  } // Outputs 4 |



### Creating, initializing and accessing an Array

### **One-Dimensional Arrays:**

The general form of a one-dimensional array declaration is

Syntax:

Datatype var-name[];

Or

Datarype[] var-name;

Explanation of syntax:

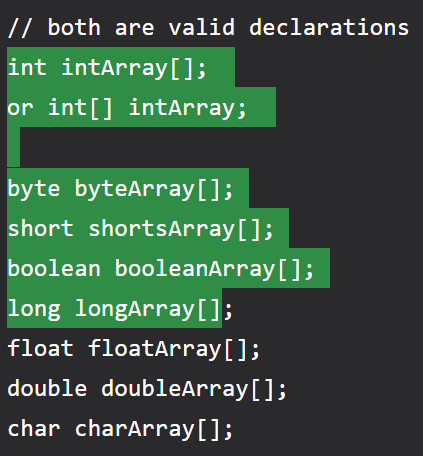
An array declaration has two components:

1)data type:

It means that as array holds similar data type elements u need to give datatype of variable which u wanna store In the array

2)var-name

Name of the array u can name it as ur wish but u need to keep rules to name variables In mind



### Instantiating an Array in Java

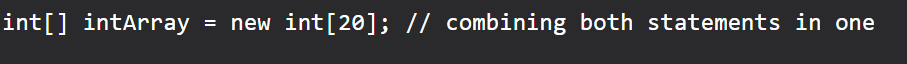
When an array is declared, only a reference of an array is created. To create or give memory to the array, you create an array like this: The general form of *new* as it applies to one-dimensional arrays appears as follows:

Text

Description automatically generated

Here, *type* specifies the type of data being allocated, *size* determines the number of elements in the array, and *var-name* is the name of the array variable that is linked to the array. Text

Description automatically generated**.or**



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) {                System.out.println("element is found at"+h);  h = h + 1;              }          }          if (h == 0) {              System.out.println("element not found");          }      }  }  } |

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)  {  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

|  |
| --- |
| public class switchwithchar {  public static void main(String[] args) {  char num = 'a';  switch (num) {  case 'a':  System.out.println("number is 0");  break;  case 'b':  System.out.println("number is 1");  break;  default:  System.out.println(num);  }  }  } |

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.

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 condition 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 Main {  public static void main(String[] args) {  int i = 0;  do {  System.out.println(i);  i++;  }  while (i < 5);  }  } |

Output:

0  
1  
2  
3  
4

## 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.**\***;  public class fibonacciseries {      public static void main(String *args*[]) {          int n;          Scanner sc1 = new Scanner(System.in);          System.out.println("enter the number of numbebers fibonacci series sequence ");          n = sc1.nextInt();          int i = 1;          int firstterm = 0;          int secondterm = 1;          while (i <= n) *//here <= bcz above i declared i as 1 it is not 0*  {              System.out.println(firstterm);              int nextterm = firstterm + secondterm;              firstterm = secondterm;              secondterm = nextterm;              i = i + 1;          }      }  } |

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

A common programming construct that is based upon a sequence of nested ifs is the if-else-if ladder.

Or

If u want the sequence of if statements then it is nothing but the if-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

}

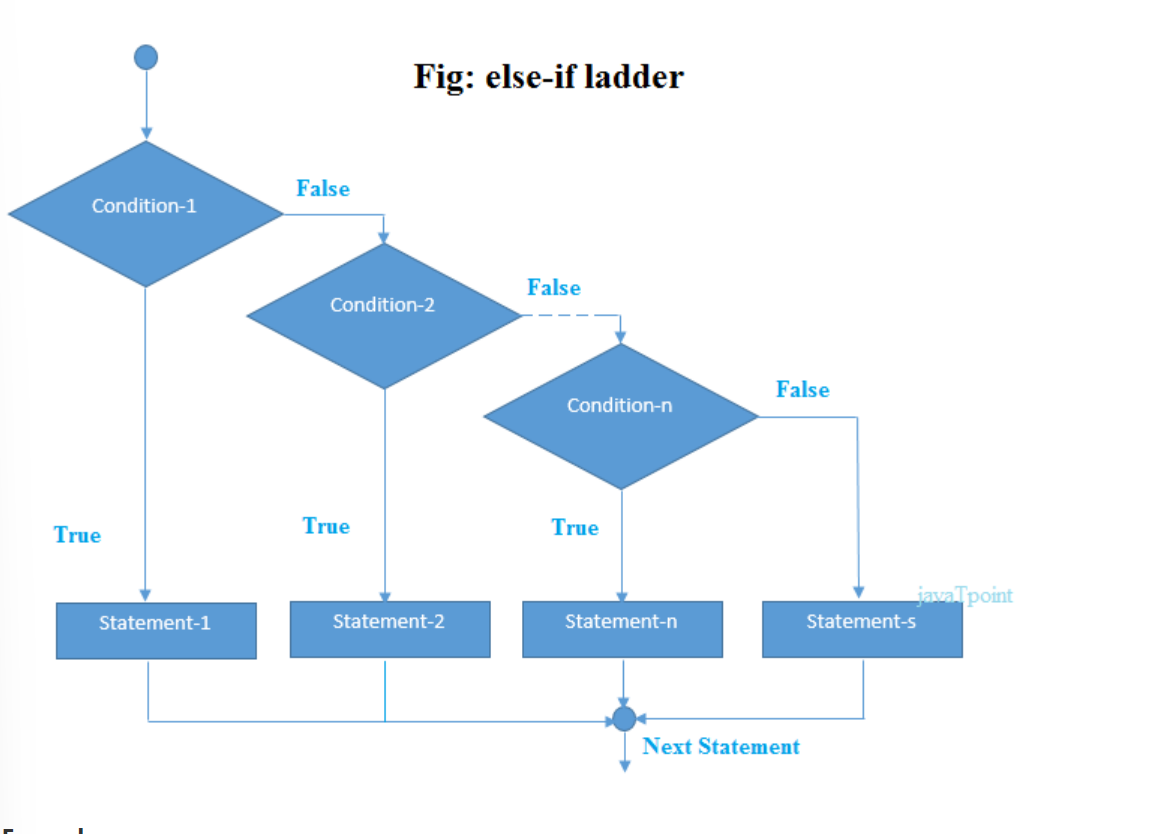
...

**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.

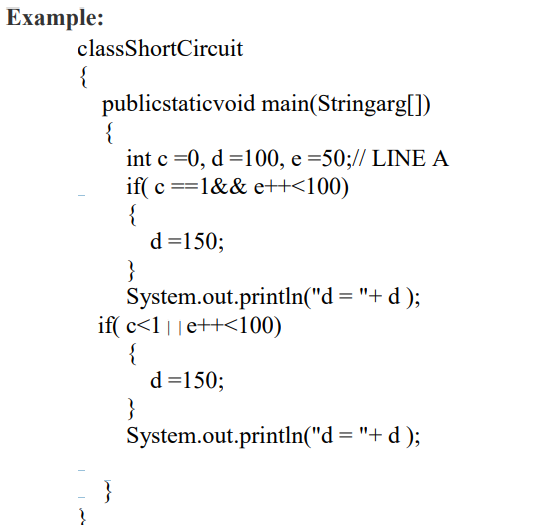
**Short Circuit Logical Operators**

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

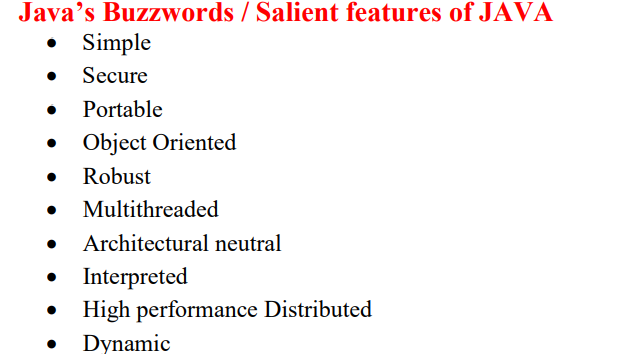
\* Similarly for short-circuit OR (||), if the first value is true, then second value is not evaluated as the result is true irrespective of the second value..

\* 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** :

In java if write the programs then it 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 Main {  public static void main(String[] args) {  int time = 20;  String result;  result = (time < 18) ? "Good day." : "Good evening.";  System.out.println(result); //note u need to print the result  }  } |

If we write this in if else format:

|  |
| --- |
| public class Main {  public static void main(String[] args) {  int time = 20;  if (time < 18) {  System.out.println("Good day.");  } else {  System.out.println("Good evening.");  }  }  } |

|  |
| --- |
|  |

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);

    }

}

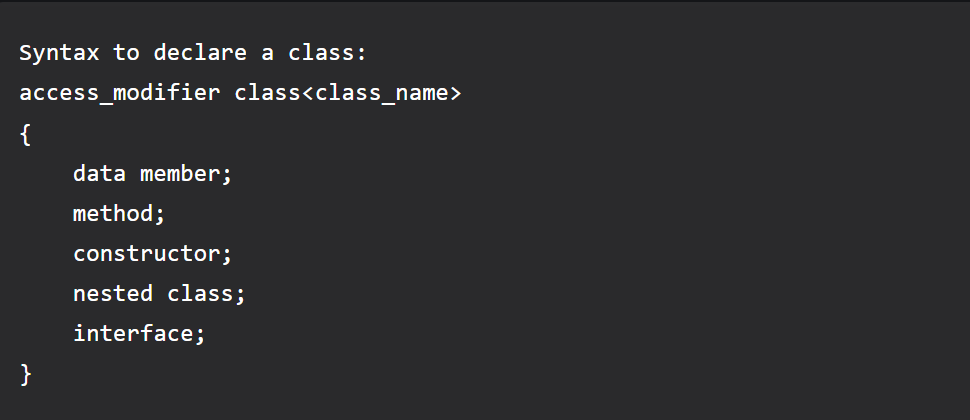
MODULE

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  
• 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);      }  } |

Output:

1

hansi

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**:

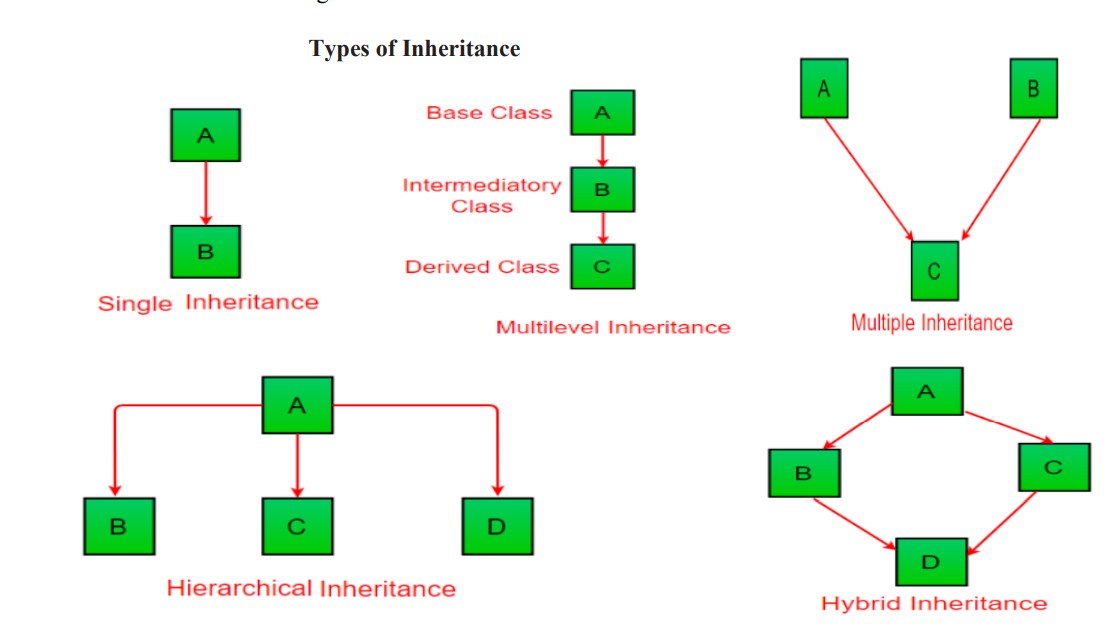
\*The class that acquires or inherits the other class features is known as sub class().

\*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 Car{

public Car()

{

System.out.println("Class Car");

}

public void vehicleType()

{

System.out.println("Vehicle Type: Car");

}

}

class Maruti extends Car{

public Maruti()

{

System.out.println("Class Maruti");

}

public void brand()

{

System.out.println("Brand: Maruti");

}

public void speed()

{

System.out.println("Max: 90Kmph");

}

}

public class Maruti800 extends Maruti{

public Maruti800()

{

System.out.println("Maruti Model: 800");

}

public void speed()

{

System.out.println("Max: 80Kmph");

}

public static void main(String args[])

{

Maruti800 obj=new Maruti800(); //here only constructing obj of maruti

obj.vehicleType(); //800 instead all like ex in polymorphism bcz here

obj.brand(); //in extended class we declared main method but

obj.speed(); //In poly example we used separate Main class

}

}

Output:

|  |
| --- |
| Class Car //first 3 lines are output public class without void  Class Maruti  Maruti Model: 800  Vehicle Type: Car  Brand: Maruti  Max: 80Kmph |

5)Write a short note on finalize method.

finalize() is a method of the Object class in Java. The finalize() method is a non-static and protected method of java.lang.Object class. In Java, the Object class is **superclass** of all Java classes. Being a method of object class finalize() method is available for every class in Java. Hence, Garbage Collector can call finalize() method on any Java object for clean-up activity.

finalize() method in Java is used to release all the resources used by the object before it is deleted/destroyed by the Garbage collector. finalize is not a reserved keyword, it's a method. Once the clean-up activity is done by the finalize() method, garbage collector immediately destroys the Java object.

## Syntax of finalize() Method in Java

finalize() is a protected method of the Object class in Java. Here is the syntax:

protected void finalize() throws Throwable{}

* Protected method: protected is an access specifier for variables and methods in java. When a variable or method is protected it means it can be accessed within the class where it's declared and other derived classes(sub classes) of that class.
* In Java, all classes inherit the Object class directly or indirectly. finalize() method is protected in Object class so that all classes in Java can override and use it.

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

**\*access modifier or specifier**, is used to set the access level for classes, attributes, methods and constructors.

We divide modifiers into two groups:

* **Access Modifiers** - controls the access level
* **Non-Access Modifiers** - do not control access level, but provides other functionality

## Access Modifiers

For **classes**, you can use either public or default:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** | **Try it** |
| public | The class can be accessed by any other class | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_mod_public) |
| default | The class is only accessible by classes in the same package. This is used when you don't specify  a modifier. | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_mod_default) |

For **attributes, methods and constructors**, you can use the one of the following:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** | **Try it** |
| public | The code is accessible by all classes | [Try it »](https://www.w3schools.com/java/tryjava_multi.asp?filename=demo_mod_public2&multi=demo_mod_public2_multi) |
| private | The code is only accessible within the declared class | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_access_mod) |
| default | The code is only accessible in the same package. This is used when you don't specify a modifier. | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_mod_default2) |
| protected | The code is accessible in the same package and **subclasses**. |  |

## Non-Access Modifiers

For **classes**, you can use either final or abstract:

|  |  |  |
| --- | --- | --- |
| **Modifier** | **Description** | **Try it** |
| final | The class cannot be inherited by other classes | [Try it »](https://www.w3schools.com/java/tryjava.asp?filename=demo_inherit_final) |
| abstract | The class cannot be used to create objects |  |

7)

this

The this keyword refers to the current object in a method or constructor.

The most common use of the this keyword is to eliminate the confusion between class attributes and parameters with the same name (because a class attribute is shadowed by a method or constructor parameter). If you omit the keyword in the example above, the output would be "0" instead of "5".

|  |
| --- |
| public class Main {  int x;    // Constructor with a parameter  public Main(int x) {  this.x = x;  }  // Call the constructor  public static void main(String[] args) {  Main myObj = new Main(5);  System.out.println("Value of x = " + myObj.x);  }  } |

Output:

|  |
| --- |
| Value of x = 5 |

Super

## Definition and Usage

The super keyword refers to superclass (parent) objects.

It is used to call superclass methods, and to access the superclass constructor.

The most common use of the super keyword is to eliminate the confusion between superclasses and subclasses that have methods with the same name.

|  |
| --- |
| class Animal { // Superclass (parent)  public void animalSound() {  System.out.println("The animal makes a sound");  }  }  class Dog extends Animal { // Subclass (child)  public void animalSound() {  super.animalSound(); // Call the superclass method  System.out.println("The dog says: bow wow");  }  }  public class Main {  public static void main(String[] args) {  Animal myDog = new Dog(); // Create a Dog object  myDog.animalSound(); // Call the method on the Dog object  }  } |

Output:

|  |
| --- |
| The animal makes a sound The dog says: bow wow |

## static

## Definition and Usage

The static keyword is a non-access modifier used for methods and attributes. Static methods/attributes can be accessed without creating an object of a class.

|  |
| --- |
| public class Main {  // Static method  static void myStaticMethod() {  System.out.println("Static methods can be called without creating objects");  }  // Public method  public void myPublicMethod() {  System.out.println("Public methods must be called by creating objects");  }  // Main method  public static void main(String[] args) {  myStaticMethod(); // Call the static method  Main myObj = new Main(); // Create an object of MyClass  myObj.myPublicMethod(); // Call the public method  }  } |

Final:

## Definition and Usage

The final keyword is a non-access modifier used for classes, attributes and methods, which makes them non-changeable (impossible to inherit or override).

The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...).

|  |
| --- |
| public class Main {  **final** int x = 10;  public static void main(String[] args) {  Main myObj = new Main();  myObj.x = 25; // will generate an error: cannot assign a value to a **final** variable  System.out.println(myObj.x);  }  } |

Output:

|  |
| --- |
| Main.java:6: error: cannot assign a value to final variable x     myObj.x = 25;          ^ 1 error |

8)Differentiate method overloading and overriding.

|  |  |  |
| --- | --- | --- |
| **No.** | **Method Overloading** | **Method Overriding** |
| 1) | Method overloading is used *to increase the readability* of the program. | Method overriding is used *to provide the specific implementation* of the method that is already provided by its super class. |
| 2) | Method overloading is performed *within class*. | Method overriding occurs *in two classes* that have IS-A (inheritance) relationship. |
| 3) | In case of method overloading, *parameter must be different*. | In case of method overriding, *parameter must be same*. |
| 4) | Method overloading is the example of *compile time polymorphism*. | Method overriding is the example of *run time polymorphism*. |
| 5) | In java, method overloading can't be performed by changing return type of the method only. *Return type can be same or different* in method overloading. But you must have to change the parameter. | *Return type must be same or covariant* in method overriding. |

Java Method Overloading example

**class** OverloadingExample

{

**static** **int** add(**int** a,**int** b) //here int instead of void bcz returning integer value

{

**return** a+b;

}

**static** **int** add(**int** a,**int** b,**int** c) //here int instead of void bcz returning integer value

{

**return** a+b+c;

}

}

Java Method Overriding example

**class** Animal

{

**void** eat()

{

System.out.println("eating...");

}

}

**class** Dog **extends** Animal

{

**void** eat()

{

System.out.println("eating bread...");

}

}

From the example differences are:

Overloading

|  |  |  |
| --- | --- | --- |
| **No.** | **Method Overloading** | **Method Overriding** |
| 1) | Method overloading is performed *within class*. | Method overriding occurs *in two classes* that have IS-A (inheritance) relationship. |
| 2) | Method overloading is the example of compile time polymorphism. | Method overriding is the example of run time polymorphism. |
| 3) | In method overloading just another variable is added nothing is overwritten. | In overriding if we see in example the method is overwritten with other values.so if we print in main method by writing extra code the method we will be displayed with second method values |

9) <https://www.w3schools.com/java/java_inner_classes.asp>

in Java, it is also possible to nest classes (a class within a class). The purpose of nested classes is to group classes that belong together, which makes your code more readable and maintainable.

To access the inner class, create an object of the outer class, and then create an object of the inner class:

10) Differentiate constructors and methods.

