**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:

**Inheritance in Java** is a mechanism in which one object acquires all the properties and behaviors of a parent object

or in other words

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

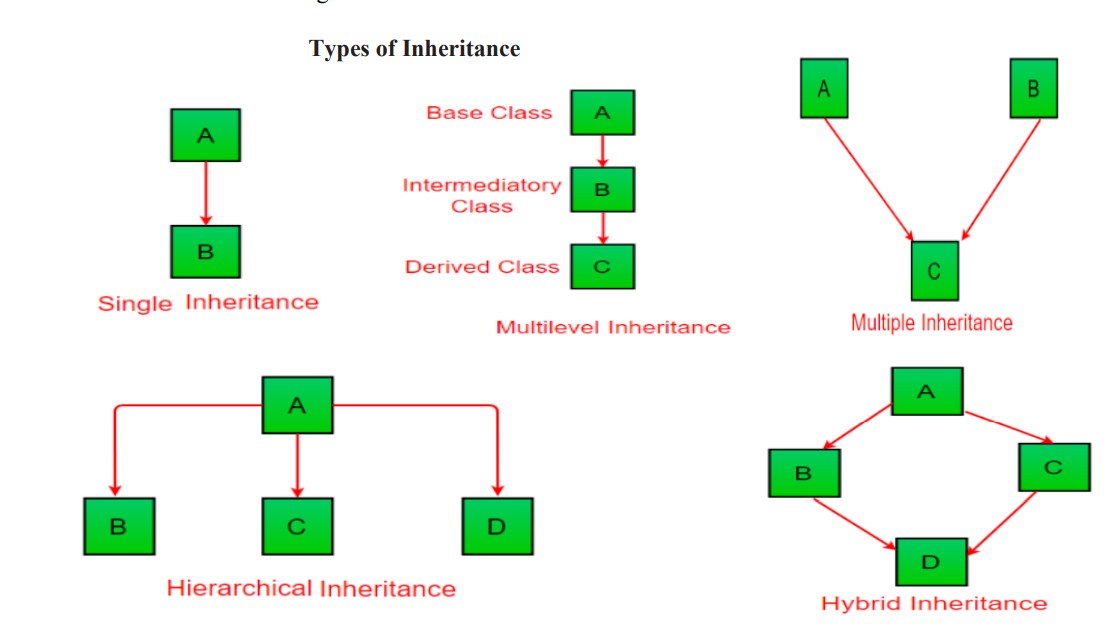
• **Sub Class or a derived class, extended class, or child class**:

\*The class that 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  Animal myPig = new Pig(); // Pig mypig= new Pig() will alsoworks  Animal 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:

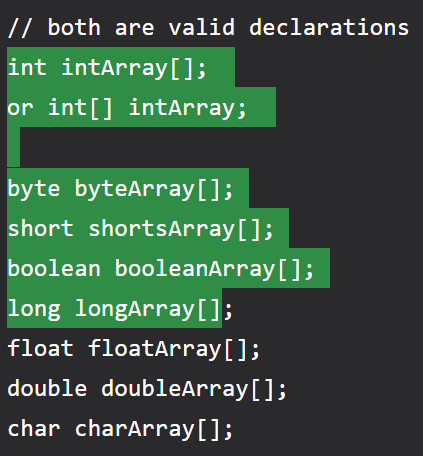
An array declaration has two components:

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

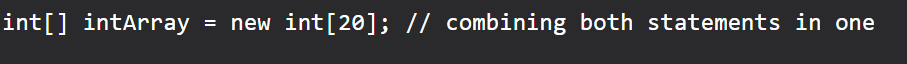
When an array is declared, only a reference of an array is created. To create or give memory to the array, you create an array like this: The general form of *new* as it applies to one-dimensional arrays appears as follows:

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) {                  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)  {  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, 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 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) {              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 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

}

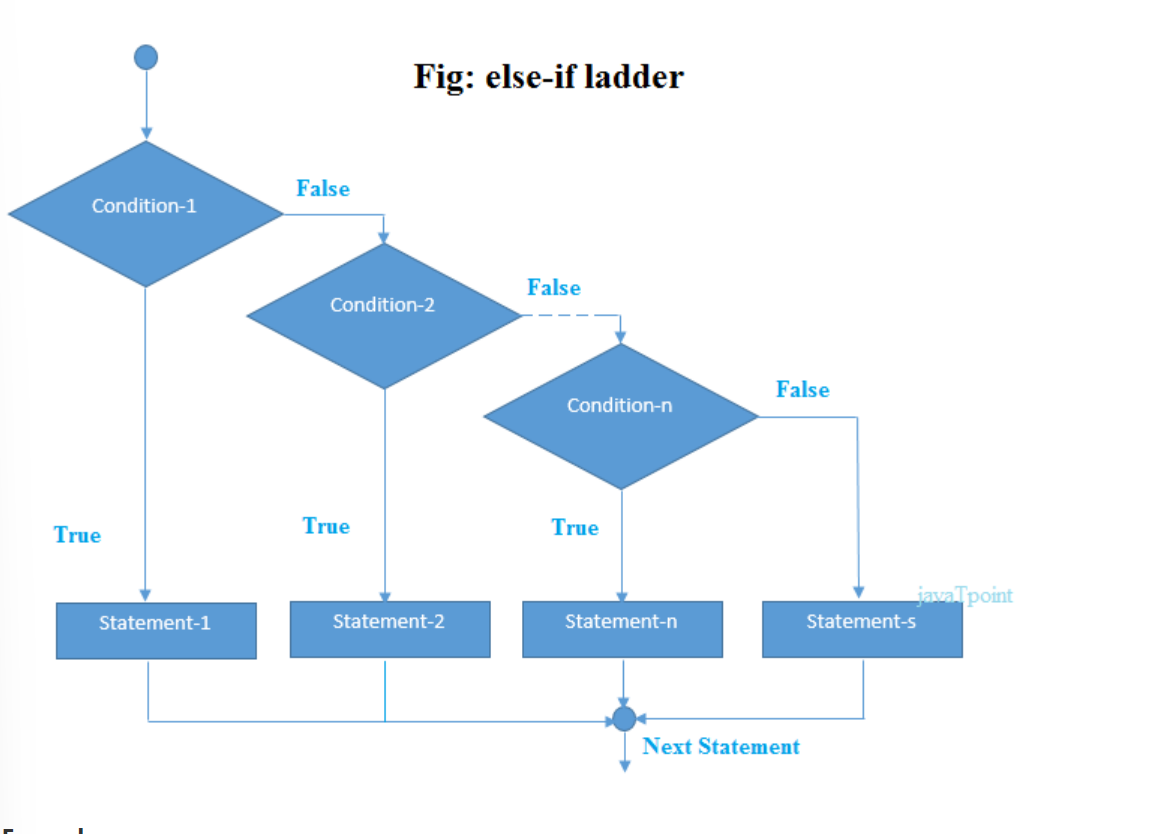
...

**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. } |

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");          }      }  } |