**Lecture 1**

**To create a projet**

File->New->Java Project->Project Name(Give a project name)->you can use default location or custom location->use an execution environment (JavaSE-1.8)->Finish

**To Create Class**

First letter of class name should always be capital

Left click on your created project-> create a class (keep in mind file name is the class name)-> You can select if you want the main method in the class and other option

**To print integer float string etc**

|  |
| --- |
| package lab1;  public class Test1 {   public static void main(String[] args) {    /\*To print something \*/  System.out.println("Hello"); //to print a string  System.out.println(2); //to print an integer/float/double  System.out.println(2.32);     }  } |

**To add/subtract two numbers**

package lab1;

public class Test1 {

public static void main(String[] args) {

int a = 5;

int b = 10;

int sum = a + b;

System.***out***.println("The sum is: "+sum);

//or

System.***out***.println("The sum is: "+(a+b));

}

}

**Lecture 2**

**To take user input:**

If you want to take multiple inputs and if you want to take string as input you have to make sure you take string input first before taking input of other data types. Or you have to create a separate object of the scanner class if you want to take string input later. Or you have to use “sc.nextLine” to discard the remaining characters of previous input by creating an empty space.

|  |
| --- |
| package lab2;  //to take user input first you need to import the Scanner class from the util library import java.util.Scanner;  public class Test {   public static void main(String[] args) {    //create an object for scanner class   Scanner sc= new Scanner(System.in);         System.out.print("Enter integer: ");  int a = sc.nextInt();  System.out.println("You have input: "+a);    System.out.print("Enter float: ");  float f = sc.nextFloat();  System.out.println("You have input: "+f);    sc.nextLine();    System.out.println("Enter String: ");  String s = sc.nextLine();  System.out.println("You have input: "+s);     // Scanner sc1= new Scanner(System.in);  // System.out.print("Enter String: "); // String s1 = sc1.nextLine(); // System.out.println("You have input: "+s1);     //after using the object for taking input if you have no further use of it   //close the object  sc.close(); // sc1.close();      } } |

**IF else**

|  |
| --- |
| package lab2;  //to take user input first you need to import the Scanner class from the util library import java.util.Scanner;  public class Test {   public static void main(String[] args) {    //create an object for scanner class   Scanner sc= new Scanner(System.in);     System.out.print("Enter integer: ");  int a = sc.nextInt();  System.out.println("You have input: "+a);    sc.close();    if(a%2 == 0) {  System.out.println("The number is even");  }  else {  System.out.println("The number is odd");  }    } } |

**Switch**

|  |
| --- |
| package lab2; import java.util.Scanner;  public class Test {   public static void main(String[] args) {    //create an object for scanner class   Scanner sc= new Scanner(System.in);     System.out.print("Enter integer: ");  int a = sc.nextInt();  System.out.println("You have input: "+a);    sc.close();    int b = a%2;    switch(b) {  case 0:  System.out.println("Even");  break;  default:  System.out.println("Odd");  }  } } |

**Loop**

|  |
| --- |
| package lab2; import java.util.Scanner;  public class Test {   public static void main(String[] args) {    //create an object for scanner class   Scanner sc= new Scanner(System.in);     System.out.print("Enter integer: ");  int a = sc.nextInt();  System.out.println("You have input: "+a);    sc.close();    for(int i=1; i<=10; i++) {  System.out.println(a+" X "+i+" = "+ a\*i);  }  } } |

**LAB2**

**Task 2**

1. Write a program that takes an integer and determines if it’s prime or not. A number is prime if it is divisible by 1 and itself only, i.e. 2, 3, 11, 37 etc.

|  |
| --- |
| package lab2; import java.awt.desktop.AboutHandler; import java.util.Scanner;  public class Test {   public static void main(String[] args) {    //create an object for scanner class   Scanner sc= new Scanner(System.in);     System.out.print("Enter integer: ");  int a = sc.nextInt();  sc.close();    boolean b= false;    for(int i=0; i<=a; i++) {  if(a == 0 || a==1) {  System.out.println("Not a prime");  b=true;  break;  }  else if(i>1 && i!=a && a%i==0) {  System.out.println("Is not prime");  b=true;  break;  }    }    if(b == false) {  System.out.println("Is prime");  }  } } |

**Task 3**

3.Write a program that prints the multiplication table from 1 to 5.

1X1=1

1X2=2

…………

5X10=50

|  |
| --- |
| package lab2; import java.util.Scanner;  public class Test {   public static void main(String[] args) {    for(int a=1; a<=5; a++) {  for(int i=1; i<=10; i++) {  System.out.println(a+" X "+i+" = "+ a\*i);  }  System.out.println("------------\n");  }    } } |

**Task 4**

Write a program that takes an integer and prints its divisors, i.e. divisors of 12 are 1, 2, 3, 4, 6.

|  |
| --- |
| package lab2; import java.awt.desktop.AboutHandler; import java.util.Scanner;  public class Test {   public static void main(String[] args) {    //create an object for scanner class   Scanner sc= new Scanner(System.in);     System.out.print("Enter integer: ");  int a = sc.nextInt();  sc.close();      for(int i=1; i<a; i++) {  if(a%i==0) {  System.out.println(i);    }  }  } } |

**LAB3**

**Task 1**

0. Generate a random integer n between 5 and 20 (inclusive). Use **for** loop to print all integers from 0 to n separated by a space. To generate a random integer between min and max range:

int n = (int)(min+Math.random()\*(max-min+1))

|  |
| --- |
| package lab2; import java.awt.desktop.AboutHandler; import java.util.Scanner;  public class Test {   public static void main(String[] args) {  int min,max;  min=5;  max=20;    int n = (int)(min+Math.random()\*(max-min+1));    System.out.println(n);    for(int i=0;i<=n;i++) {  System.out.print(i+" ");  }    } } |

**Task 3**

3.Write a program which will use a while loop to print all the integers between 100 and 150 which are divisible by 8 in descending order.

Output: 144 136 128 120 112 104

|  |
| --- |
| package lab3; import java.util.Scanner;  public class task3 {  public static void main(String[] args) {  int i = 150;  while (i >= 100) {  if (i % 8 == 0) {  System.out.print(i + " ");  }  i--;  }  } } |

**Task 2**

2. Print the following

12345

1234

123

12

1

|  |
| --- |
| package lab3; import java.util.Scanner; public class task2 {  public static void main(String[] args) {    for(int i=5; i>=1; i--) {  for(int j=1; j<=i; j++) {  System.out.print(j);  }  System.out.println("\n");  }    } } |

**Task 4**

4.Generate a random int between 1990 and 2020. Then print it check if it’s a leap year or not. Note: A leap year must satisfy any or both of the following conditions:

Divisible by 400

Divisible by 4 and not divisible by 100

package lab2;

import java.awt.desktop.AboutHandler;

import java.util.Scanner;

public class Test {

public static void main(String[] args) {

int min,max;

min=1990;

max=2020;

int n = (int)(min+Math.*random*()\*(max-min+1));

System.***out***.println(n);

if(n%400==0 || (n%4==0 && n%100!=0)) {

System.***out***.println("True");

}

else {

System.***out***.println("False");

}

}

}

**Task 5**

5.Print the following pattern:

\*

+++

\*\*\*\*\*

+++++++

\*\*\*\*\*\*\*\*\*

|  |
| --- |
| package lab3; import java.util.Scanner; public class task5 {  public static void main(String[] args) {    boolean b = true;    for(int i=1; i<=9; i= i+2) {    for(int j=1;j<=i;j++) {  if(b== true) {  System.out.print("\*");  }  else if(b==false) {  System.out.print("+");  }  }  if(b== true) {  b=false;  }  else if(b==false) {  b=true;  }  System.out.println("");  }    } } |

**Lecture 3**

**LAB4**

**Task1**

Declare an integer array of size 6, initialize it with user input, calculate and print the

average. Now calculate the percentage of numbers that are above that average.

For example: if 3 of the array elements are greater than average, percentage is: 3 \* 100 / 6 = 50%

|  |
| --- |
| import java.util.Scanner; public class Task1 {   public static void main(String[] args) {    Scanner sc = new Scanner(System.in);    int[] arr = new int[6];    float sum=0;    System.out.println("Enter Array content: ");    for(int i=0; i<6; i++) {  arr[i]= sc.nextInt();  sum += arr[i];  }    float avg = sum/6;  System.out.println("Average is: "+avg);      int count=0;  for(int i=0; i<6; i++) {  if(arr[i]>avg) {  count+=1;  }  }      float prcnt = (count\*100)/6;  System.out.println("Percentage of numbers that are above the average is: "+prcnt+"%");     } } |

**Task2**

Take an integer from user, generate that many fibonacci numbers and store in an

array. Display the array.

Sample output:

Enter a number: 8

First 8 Fibonacci numbers: 0 1 1 2 3 5 8 13

|  |
| --- |
| import java.util.Scanner; public class Task2 {   public static void main(String[] args) {    Scanner sc = new Scanner(System.in);    System.out.print("Enter an integer: ");  int n = sc.nextInt();    int[] arr = new int[n];    for(int i = 0; i<n; i++) {  if(i==0) {  arr[i] = 0;  }  else if(i ==1) {  arr[i] = 1;  }  else {  arr[i]= arr[i-1]+arr[i-2];  }  }    System.out.print("First "+n+" Fibonacci Numbers: ");  for(int i = 0; i < n; i++) {  System.out.print(arr[i]+" ");  }  }  } |

**Task3**

Take a 3X3 array and initialize it with these values:

3 4 9

2 9 11

4 6 0

Calculate and print the sum for each row, column and both diagonals

|  |
| --- |
| import java.util.Scanner;  public class Task3 {  public static void main(String[] args) {  int sumRow, sumCol, LD = 0, RD = 0;  int a[][] = {  {3, 4, 9},  {2, 9, 11},  {4, 6, 0}  };   int row\_length = a.length;  int col\_length = a[0].length;    for (int i = 0; i < row\_length; i++) {  sumRow = 0;  for (int j = 0; j < col\_length; j++) {  sumRow = sumRow + a[i][j];  }  System.out.println("Sum of row " + (i + 1) + ": " + sumRow);  }     for (int i = 0; i < col\_length; i++) {  sumCol = 0;  for (int j = 0; j < row\_length; j++) {  sumCol = sumCol + a[j][i];  }  System.out.println("Sum of column " + (i + 1) + ": " + sumCol);  }    for (int k = 0; k < row\_length; k++) {  for (int l = 0; l < col\_length; l++) {  if (k == l) {  LD = LD + a[k][l];  }  if ((k + l) == (a.length) - 1) {  RD = RD + a[k][l];  }  }  }   System.out.println("Sum of left diagonal: " + LD);  System.out.println("Sum of right diagonal: " + RD);  } } |

**Task4**

Take an integer array and print only the numbers that are in consecutive orders of 3.

Enter size: 12

Enter numbers: 1 2 3 2 2 2 11 4 4 4 3 3

Output: 2 4

|  |
| --- |
| import java.util.Scanner;  public class Task4 {  public static void main(String[] args) {  Scanner sc = new Scanner(System.in);   System.out.print("Enter size: ");  int size = sc.nextInt();  int[] arr1 = new int[size];  int[] arr2 = new int[size];   System.out.print("Enter numbers: "); // Change from "Enter Array content: " to "Enter numbers: "  for (int i = 0; i < arr1.length; i++) {  arr1[i] = sc.nextInt();   if (i > 1) {  if (arr1[i] == arr1[i - 1] && arr1[i] == arr1[i - 2]) {  boolean B = false;  for (int j = 0; j < arr2.length; j++) {  if (arr2[j] == arr1[i]) {  B = true;  break;  }  }  if (!B) {  arr2[i] = arr1[i];  }  }  }  }  System.out.print("Output: ");  for (int i = 0; i < size; i++) {  if (arr2[i] != 0) {  System.out.print(arr2[i] + " ");  }  }  sc.close();  } } |

**Lecture 4**

There can’t be multiple public classes in one file. In one file there’ll be only one public class.

A class can have few things:  
 Attributes  
 Like in a student class student name, id etc

Constructor

**Make an employee class with different attributes of your choice.**

|  |
| --- |
| class Employee{  String name;  int ID;  double salary;    public Employee()  {    }  public Employee(String name, int ID, double salary) {  this.name = name;  this.ID = ID;  this.salary = salary;  }   }  public class Driver2 {   public static void main(String[] args) {  Employee e1 = new Employee();  e1.name = "Azmine";  e1.ID = 926;  e1.salary = 54000.55;     System.out.println("\*\*DEFAULT CONSTRUCTOR\*\*");  System.out.println("Employee Name: "+e1.name);  System.out.println("Emplyee ID: "+e1.ID);  System.out.println("Employee Salary "+e1.salary+"TK\n");    System.out.println("\*\*PARAMETERIZED CONSTRUCTOR\*\*");  //using parameterized constructor  Employee e2 = new Employee("Emon", 12836, 99999.92);  System.out.println("Employee Name: "+e2.name);  System.out.println("Emplyee ID: "+e2.ID);  System.out.println("Employee Salary "+e2.salary+"TK");            }  } |

**Create a shape class**

|  |
| --- |
| class Shape {  String name;  float length, width, radius;    Shape(String name, float length, float width){  this.name = name;  this.length = length;  this.width = width;  }    Shape(String name, float length, float width, float radius){  this.name = name;  this.length = length;  this.width = width;  this.radius = radius;  } }   public class Driver3 {   public static void main(String[] args) {    Shape rect = new Shape("Rectangle", 34.4f,33.9f);  float area1 = rect.length\*rect.width;  System.out.println("Area of the "+rect.name+": "+area1);    Shape circ = new Shape("Circle", 34.4f,33.9f,2);  double area2 = (Math.PI\*Math.pow(circ.radius,2));  System.out.println("Area of the "+circ.name+": "+area2);   }  } |

**Encapsulation and Access Modifiers:**

If any of your attributes are private you can not directly access them. But you can access them using the getter setter method. We use the getter setter method so that we can access the private attribute of one class from another class.

Getter setter method’s access must be public. If you want to modify or want to see the content of the private attributes, for each attribute you create a getter setter method.

Another name of getter setter is accessor and mutator respectively.

Public   
Private

|  |
| --- |
| class Employee{  private String name;  private int ID;  private double salary;    public Employee()  {    }  public Employee(String name, int ID, double salary) {  this.name = name;  this.ID = ID;  this.salary = salary;  }    //getter setter method  public String getName() {  return name;  }  public int getID() {  return ID;  }  public double getSalary() {  return salary;  }    public void setName(String name) {  this.name = name;  }    public void setID(int ID) {  this.ID = ID;  }    public void setSalary(double salary) {  this.salary = salary;  }   }  public class Driver2 {   public static void main(String[] args) {  Employee e1 = new Employee();  e1.setName("Azmine");   e1.setID(926);   e1.setSalary(54000.55);    System.out.println("\*\*DEFAULT CONSTRUCTOR\*\*");  System.out.println("Employee Name: "+e1.getName());  System.out.println("Emplyee ID: "+e1.getID());  System.out.println("Employee Salary "+e1.getSalary()+"TK\n");    System.out.println("\*\*PARAMETERIZED CONSTRUCTOR\*\*");  //using parameterized constructor  Employee e2 = new Employee("Emon", 12836, 99999.92);  System.out.println("Employee Name: "+e2.getName());  System.out.println("Emplyee ID: "+e2.getID());  System.out.println("Employee Salary "+e2.getSalary()+"TK\n");            }  } |

|  |
| --- |
| class Shape {  private String name;  private float length, width, radius;    Shape(String name, float length, float width){  this.name = name;  this.length = length;  this.width = width;  }    Shape(String name, float length, float width, float radius){  this.name = name;  this.length = length;  this.width = width;  this.radius = radius;  }     public String getName() {  return name;  }  public float getLength() {  return length;  }    public float getWidth() {  return width;  }    public float getRadius() {  return radius;  }    public void setName(String name) {  this.name = name;  }  public void setLength(float length) {  this.length = length;  }    public void setwidth(float width) {  this.width = width;  }    public void setRadius(float radius) {  this.radius = radius;  }       }   public class Driver3 {   public static void main(String[] args) {    Shape rect = new Shape("Rectangle", 34.4f,33.9f);  float area1 = rect.getLength()\*rect.getWidth();  System.out.println("Area of the "+rect.getName()+": "+area1);    Shape circ = new Shape("Circle", 34.4f,33.9f,2);  double area2 = (Math.PI\*Math.pow(circ.getRadius(),2));  System.out.println("Area of the "+circ.getName()+": "+area2);   }  } |

**Lecture 5**

**Class Task**



|  |
| --- |
| class Complex{  private double real, imaginary;    public Complex(){    }  public Complex(double real, double imaginary) {  this.real = real;  this.imaginary = imaginary;  }    public void add(Complex c1, Complex c2) {  double r = c1.real+c2.real;  double i = c1.imaginary+c2.imaginary;    System.out.println("Addition result: ("+c1.real+" + "+c1.imaginary+"i) + ("+c2.real+" + "+c2.imaginary+"i) = "+r+" + "+i+"i");  }  public void multiply(Complex c, double mul) {  double r = c.real\*mul;  double i = c.imaginary\*mul;    System.out.println("Multiplication result: ("+c.real+" + "+c.imaginary+"i) \* "+mul+" = "+r+" + "+i+"i");  }  public void setReal(double real) {  this.real = real;  }  public void setImaginary(double imaginary) {  this.imaginary = imaginary;  }  public double getReal() {  return real;  }  public double getImaginary() {  return imaginary;  }   }  public class Show {   public static void main(String[] args) {    Complex comp1 = new Complex(4,5);    Complex comp2 = new Complex();  comp2.setReal(12.5);  comp2.setImaginary(7);    double mul = 2;    System.out.println("First complex number: "+comp1.getReal()+" + "+comp1.getImaginary()+"i");  System.out.println("Second complex number: "+comp2.getReal()+" + "+comp2.getImaginary()+"i\n");     comp1.add(comp1, comp2);    comp1.multiply(comp1,mul);  }  } |

**Method Overloading**

Method overloading in Java refers to the ability to define multiple methods in the same class with the same name but with different parameter lists. The key is that the methods must have different types or numbers of parameters. This allows you to use the same method name for different behaviors based on the types or number of arguments passed to it.

**Constructor Overloading**

Constructor overloading in Java allows a class to have multiple constructors with different parameter lists, providing flexibility in creating objects with various initialization options.

**Create two sum method,**

Public double sum()

Print summation of two numbers

Public double sum()

Print summation of three numbers

|  |
| --- |
| package overloading;  class A{    //method overloading means the method name would be same  /\*  But one of these three condition must be satisfied:  1. Number of parameters are different   2. Atleast one Parameter type is different   3. Change the sequence of the different parameter types.  \* \*/    public void print(int a, double b) {  System.out.println("Number is: "+a);    }    public void print(int a, double b, int c) {  System.out.println("Number is: "+2+"and "+5);  }    public void print(double b, int a) {  System.out.println("Number is: "+2.3+"and "+3.3);  }    public double sum(double a, double b) {  return a+b;  }    public double sum(double a, double b, double c) {  return a+b+c;  } }  public class Overloading {   public static void main(String[] args) {  A obj = new A();    double sum1 = obj.sum(2, 3.3);  double sum2 = obj.sum(12, 3.3, 4.6);    System.out.println("Sum of two numbers: "+sum1);  System.out.println("Sum of three numbers: "+sum2);  }  } |

**Lecture 6**

In Java, static variables and methods are associated with the class rather than with any specific instance of the class. They are shared among all instances of the class and can be accessed using the class name. Here's an explanation of static variables and methods:

**Static Variables:**

* Also known as class variables, static variables are shared among all instances of a class.
* They are declared using the static keyword.
* Static variables are initialized only once, and their values are shared among all instances of the class.
* They are accessed using the class name rather than through an instance of the class.

**Static Methods:**

* Static methods are associated with the class rather than with instances of the class.
* They are declared using the static keyword.
* Static methods can be called using the class name without the need to create an instance of the class.

**toString Method:**

In Java, the toString() method is a method provided by the Object class, which is the root class for all Java classes. The purpose of the toString() method is to return a string representation of the object. By default, the toString() method in the Object class returns a string that consists of the class name followed by the "@" character and the object's hashcode.

|  |
| --- |
| package lab6;  class Department{  private String name;  private int id;  private static int NoWorker = 0;    public Department() {  this.name = "Anonymous";  this.id = 0;  NoWorker++;    }  public Department(String name, int id) {  this.name = name;  this.id = id;  NoWorker++;  }     public void setName(String name) {  this.name = name;  }    public void setId(int id) {  this.id = id;  }        public static int getNoWorker() {  return NoWorker;  }    public String getName() {  return name;  }    public String toString() {  return "The name of the department is: "+name+" and the id is: "+id;  } }  public class staticc {  public static void main(String[] args) {  System.out.println("No of worket in the department: "+Department.getNoWorker());  Department dept1 = new Department();  dept1.setName("BBA");  dept1.setId(010);  System.out.println("No of worket in the department: "+Department.getNoWorker());  Department dept2 = new Department("ECE",123);  System.out.println("No of worket in the department: "+Department.getNoWorker());    System.out.println(dept1);  System.out.println(dept2);    } }  // add another instance variable ID and print the both information //using both Constructor(default, parameterized) the default name is  //"Anonymous" and id is 0 |

**Lecture 7**

**Inheritance**

|  |
| --- |
| package lab;  public class Shape {  private String color;    public Shape() {  color = "Colorless";  }    public Shape(String color) {  this.color = color;  }   public String getColor() {  return color;  }   public void setColor(String color) {  this.color = color;  }   @Override  public String toString() {  return "The color of the shape is: " + color;  }    } |

|  |
| --- |
| package lab;  public class Circle extends Shape {  private double radius;    Circle(){  super();  }    Circle(String color,double radius){  super(color);  this.radius = radius;  }   public double getRadius() {  return radius;  }   public void setRadius(double radius) {  this.radius = radius;  }    public double calculateArea() {  return Math.PI\*radius\*radius;  }      @Override  public String toString() {  return super.toString()+" and the radius is: "+radius;  }      } |

|  |
| --- |
| package lab;  public class Rectangle extends Shape{  private double length, width;    public Rectangle()   {  super();  }     public Rectangle(String color, double length, double width) {  super(color);  this.length = length;  this.width = width;  }    public double getLength() {  return length;  }   public void setLength(double length) {  this.length = length;  }   public double getWidth() {  return width;  }   public void setWidth(double width) {  this.width = width;  }    public double calculateArea(double length, double width) {  return length\*width;  }    @Override  public String toString() {  return super.toString()+" And the length is : "+length+" and the width is: "+width;  }       } |

|  |
| --- |
| package lab;  public class driver {   public static void main(String[] args) {  Rectangle r1 = new Rectangle("Red", 12,4);    System.out.println(r1);    System.out.println("The area of the rectangle is: "+r1.calculateArea(r1.getLength(), r1.getWidth()));    Circle c1 = new Circle("Blue", 2);  System.out.println(c1);  System.out.println("The area of the Circle is: "+c1.calculateArea());    } } |

**Lecture 8**

**Polymorphism**

1

2

Here's a comprehensive explanation of polymorphism in Java:

Polymorphism is a fundamental concept in object-oriented programming (OOP) that allows objects of different classes to be treated as if they were objects of a common parent class. This means that the same method call can invoke different behaviors depending on the actual object type at runtime.

Types of Polymorphism in Java:

* Compile-time Polymorphism (Method Overloading):
* Multiple methods in the same class share the same name but have different parameter lists (different number of parameters or different parameter types).
* The compiler determines which method to call based on the argument types at compile time.
* Runtime Polymorphism (Method Overriding):
* A subclass redefines a method inherited from its superclass, providing a specialized implementation.
* The decision of which method to call is made at runtime based on the actual object type, even if the reference variable is of the parent class type.

Key Concepts:

* Overriding: Redefining a method in a subclass with the same signature (name, parameters, and return type) as in the superclass.
* Dynamic Method Dispatch: The mechanism that enables runtime polymorphism. The JVM determines the actual method to call at runtime based on the object's class.
* Upcasting: Assigning a subclass object to a superclass reference variable.

Benefits of Polymorphism:

* Code Reusability: Inherit common behavior from a superclass and extend it in subclasses, reducing code duplication.
* Flexibility: Write code that can work with objects of different types without knowing their exact type at compile time.
* Extensibility: Add new classes without modifying existing code, as long as they conform to the common interface.
* Maintainability: Improve code organization and modularity.

**Multi-level Inheritance:**

* It involves a chain of inheritance where a class inherits from a parent class, which in turn inherits from another class, and so on.
* It creates a hierarchical structure of classes.
* Java supports multi-level inheritance without any restrictions.

Example:

Java

class Animal { ... }

class Mammal extends Animal { ... }

class Dog extends Mammal { ... }

Use code with caution. [Learn more](https://bard.google.com/faq#coding)

content\_copy

**Multiple Inheritance (Not Supported in Java):**

* It involves a class inheriting directly from multiple parent classes.
* Java doesn't allow multiple inheritance through classes due to potential issues:

|  |
| --- |
| package lab6;   /\*Question: Assume, City bank provides 10% interest rate if a client's transaction crosses TK>1,00,000 and   \* UCB bank provides 2% interest if a client's transaction crosses TK.100. Now calculate the interest for   \* both Bank and show the result by calling interest() method from main.  \*/ class Bank {  private String name;  private String location;   private int code;    public Bank(){    }    public Bank(String name, String location, int code) {  this.name = name;  this.location = location;  this.code = code;  }      public double interest() {  return 0;  }    public String toString() {  return "The details of the bank is:"+"Name: "+name+"Location: "+location+"Code: "+code;  }   }  //Bangladesh bank has all the other bank under it //method overriding or polymorphism;  class CityBank extends Bank{    public CityBank() {  super();  }    //notice this class doesn't have any fields/ attributes. so how will you use the citybank parameterized attribute?  public CityBank(String name, String location, int code) {  super(name, location, code);    }    public double interest(double transaction\_amount) {  if(transaction\_amount>100000) {  return transaction\_amount\*0.1;  }  else  return 0;  }    public String toString() {  return super.toString();  } }  //in java multilevel inheritance works but multiple inheritance don't work class UCBank extends Bank{  private String branch;    public UCBank() {  super();  }    public UCBank(String name, String location, int code, String branch) {  super(name, location, code);  this.branch = branch;    }    public double interest(double transaction\_amount) {  if(transaction\_amount>100) {  return transaction\_amount\*0.02;  }  else {  return 0;  }  }    public String toString() {  return super.toString()+"Branch: "+branch;  } }  public class Driver {   public static void main(String[] args) {  CityBank obj1 = new CityBank("City Bank", "Gulshan",123);  System.out.println("\nCity Bank\nInterest: "+obj1.interest(4000000)+"tk");    UCBank obj2 = new UCBank("UCBank", "bashundhara",234,"bashundhara");  System.out.println("\nUCB Bank\nInterest: "+obj2.interest(1000)+"tk"); } } |

**Abstract Class**

Key Concepts:

* Abstract Class: A class declared with the abstract keyword. It acts as a blueprint for subclasses to inherit from but cannot be instantiated directly.
* Abstract Methods: Methods declared with the abstract keyword without a body. They must be implemented by non-abstract subclasses.

Purpose and Advantages:

* Blueprint for Common Structure: Establishes a general structure and behavior for a group of related classes.
* Enforces Method Implementation: Ensures that subclasses provide concrete implementations for abstract methods, guaranteeing specific functionality.
* Encapsulates Common Code: Holds shared fields and methods that can be inherited and reused by subclasses.
* Promotes Polymorphism: Enables treating objects of different subclasses as if they were objects of the abstract class, using dynamic method dispatch.

Key Characteristics:

* Cannot be Instantiated: You cannot create objects of an abstract class.
* Can Contain Concrete Methods: It can have both abstract and non-abstract (concrete) methods.
* Can Have Fields and Constructors: It can have member fields and constructors like regular classes.
* Can Extend Another Class: It can inherit from another class, abstract or non-abstract.

|  |
| --- |
| package lab6;  abstract class Shape {  private String color;  public Shape(){    }    public Shape(String color) {  this.color = color;  }    public void printInfo() {  System.out.println(this.color);  }    public abstract void show(); }  class Circle extends Shape {  private double radius;  public Circle(String color, double radius) {  super(color);  this.radius = radius;  }    public double Area() {  return 3.1416\*radius\*radius;  }    public void show() {  System.out.println("This is Circle");  } }   //design another class Rectangle extends   class Rectangle extends Shape{  private double length, width;    public Rectangle(String color, double length, double width) {  super(color);  this.length = length;  this.width = width;  }    public double Area() {  return length\*width;  }    public void show() {  System.out.println("This is Rectangle");  } }  public class Test {   public static void main(String[] args) {  // TODO Auto-generated method stub    Circle cir = new Circle("Blue",4);  System.out.println("\nArea of the circle: "+cir.Area());  cir.printInfo();  cir.show();      Rectangle rec = new Rectangle("Yellow",4,8);  System.out.println("\nArea of the Rectangle: "+rec.Area());  rec.printInfo();  rec.show();  }  } |

**Lecture 9**

**Iterface**

Here's a comprehensive definition of interfaces in Java:

Interfaces in Java serve as blueprints for defining a set of methods that a class must implement. They provide a way to achieve abstraction, multiple inheritance, and loose coupling in object-oriented programming.

Key characteristics of interfaces:

* Purely abstract: Contain only method signatures (no method bodies).
* Declared using the interface keyword:

|  |
| --- |
| interface MyInterface {  void method1();  int method2(String arg); } |

* Methods are implicitly public and abstract:

|  |
| --- |
| interface MyInterface {  void method1();  int method2(String arg); }1(); public abstract int method2(String arg); |

* Variables are implicitly public, static, and final:

|  |
| --- |
| int NUM = 10; // Equivalent to: public static final int NUM = 10; |

* Cannot be instantiated: You cannot create objects of interfaces directly.
* Implemented by classes:
* Java

|  |
| --- |
| class MyClass implements MyInterface {  @Override  public void method1() {  // Implementation here  }   @Override  public int method2(String arg) {  // Implementation here  } } |

Benefits of using interfaces:

* Abstraction: Hide implementation details and promote loose coupling.
* Multiple inheritance: Classes can implement multiple interfaces, gaining a diverse set of behaviors.
* Design flexibility: Allow for independent development of classes and interfaces.
* Enforce contracts: Ensure that classes adhere to a specific set of behaviors.
* Decoupling: Reduce dependencies between components, making code more modular and maintainable.

Additional features of interfaces:

* Default methods: Introduced in Java 8, allowing interfaces to have method bodies for optional implementation.
* Static methods: Introduced in Java 8, enabling interfaces to have static utility methods.

**Upcasting**

Upcasting in Java is the process of converting a subclass object reference to a superclass reference. This means treating an object of a subclass as if it were an object of its superclass. It's also known as "widening" because it involves moving up the inheritance hierarchy.

Key points about upcasting:

* Implicit conversion: Java performs upcasting automatically, without explicit typecasting, because a subclass object is always a valid instance of its superclass.
* Reference type changes, but object remains: Only the reference type changes, not the actual object itself. The object still retains its subclass-specific features.
* Access to superclass members: You can access only those members (fields and methods) that are declared in the superclass, even if the actual object has additional subclass-specific members.

|  |
| --- |
| class Animal {  public void eat() {  System.out.println("Animal is eating");  } }  class Dog extends Animal {  public void bark() {  System.out.println("Woof!");  } }  public class Main {  public static void main(String[] args) {  Dog dog = new Dog(); // Create a Dog object   Animal animalRef = dog; // Upcasting: Dog object assigned to Animal reference   animalRef.eat(); // Calls the method from Animal  // animalRef.bark(); // Compilation error: Animal doesn't have a bark() method  } } |

**Package in Java**

Here's how you can use classes from the my\_pack package in a different package:

* Import the necessary classes:
* In the Java file where you want to use the classes from my\_pack, add the following import statement at the beginning:
* Java
* import my\_pack.ClassName; // Replace ClassName with the actual class name
* If you need to use multiple classes from the package, import each one individually.
* Alternatively, you can import all classes from the package using:
* Java

|  |
| --- |
| import my\_pack.\*; |

* Access the classes directly:
* Once imported, you can use the classes from my\_pack as if they were in the same package as your current file.
* Create objects, call methods, and access fields just like any other class:
* Java

|  |
| --- |
| ClassName obj = new ClassName(); obj.method1(); System.out.println(obj.field1); |

Example:

Java

|  |
| --- |
| // my\_pack/MyClass.java package my\_pack;  public class MyClass {  public void sayHello() {  System.out.println("Hello from MyClass!");  } }  // another\_pack/Main.java package another\_pack;  import my\_pack.MyClass;  public class Main {  public static void main(String[] args) {  MyClass myClass = new MyClass();  myClass.sayHello();  } } |

Remember:

* Ensure that the my\_pack package is accessible to your current project. This usually involves including it in your project's build path or classpath.
* Classes you want to use from other packages must be declared as public. Otherwise, they won't be accessible outside their package.

|  |
| --- |
| package my\_pack\_2;  public class Compare {    int num1;  int num2;    public Compare(int n, int m) {  num1 = n;  num2 = m;  }    public void getMax() {  if(num1>num2) {  System.out.println("Max: "+num1);  }  else if (num1<num2){  System.out.println("Max: "+num2);  }  else {  System.out.println("Both numbers are equal");  }  }                public static void main(String[] args) {  Compare obj = new Compare(27,29);  obj.getMax();      } } |

Another package:

|  |
| --- |
| package my\_pack;  import my\_pack\_2.Compare;  public class Test {   public static void main(String[] args) {  Compare obj = new Compare(12,4);  obj.getMax();     }  } |

**Exception Handling**

**Task1**

Write a program that takes 10 positive integers from user and prints the sum. If any negative value is entered, the program should catch it as an exception and display “Input positive integer only”. The program must continue taking input until it gets 10 positive integers.

|  |
| --- |
| package error\_handling;  import java.util.Scanner; public class task1 {   public static void main(String[] args) {    Scanner scanner = new Scanner(System.in);  int sum = 0, count = 0;   for (int i = 0; count < 10; i++) {  System.out.print("Enter positive integer " + (count + 1) + ": ");  try {  int num = scanner.nextInt();  if (num > 0) {  sum += num;  count++;  }  else {  int j = num/0;  }    } catch (Exception e) {  System.out.println("Please enter a positive integer only.");        }  }     System.out.println("The sum of the 10 positive integers is: " + sum);        }     } |

**Task2**

Write a program that creates an integer array of size 100 and initialize it with random values:

int a = (int) (Math.random() \* 10000);

The program then takes an integer from user, use it as an index and tries to print the corresponding element of that array. If index is out of array size, the program should catch it and display appropriate message.

|  |
| --- |
| package error\_handling; import java.util.Scanner; import java.util.Random; public class task2 {  public static void main(String[] args) {  int[] array = new int[100];  int random;    // Initialize array with random values  for (int i = 0; i < array.length; i++) {  random = (int) (Math.random() \* 10000);  }   Scanner scanner = new Scanner(System.in);   try {  System.out.print("Enter an index (0-99): ");  int index = scanner.nextInt();     int element = array[index];  System.out.println("The element at index " + index + " is: " + element);    } catch (ArrayIndexOutOfBoundsException e) {  System.out.println("Index out of bounds!");  }  } } |

**Task3**

Create a Triangle class. Now create IllegalTriangleException class that extends Exception. If the sum of any two sides is not greater than the third side, the Triangle class should throw IllegalTriangleException.

|  |
| --- |
| package error\_handling;  class IllegalTriangleExceptionAgeException extends Exception{  public IllegalTriangleExceptionAgeException(String s) {  super(s);  } }  public class Triangle {   public void Check(double first, double second, double third) throws IllegalTriangleExceptionAgeException{  if((first+second<third)||(first+third<second)||(second+third<first)) {  throw new IllegalTriangleExceptionAgeException("Invalid side lengths");  }  else {  System.out.println("Okay");  }    }    public static void main(String[] args) {  Triangle obj = new Triangle();  try{  obj.Check(2.2,2.2,100);  }  catch(IllegalTriangleExceptionAgeException e) {  System.out.println("Exception occured");  System.out.println(e.getMessage());  }  }  } |