

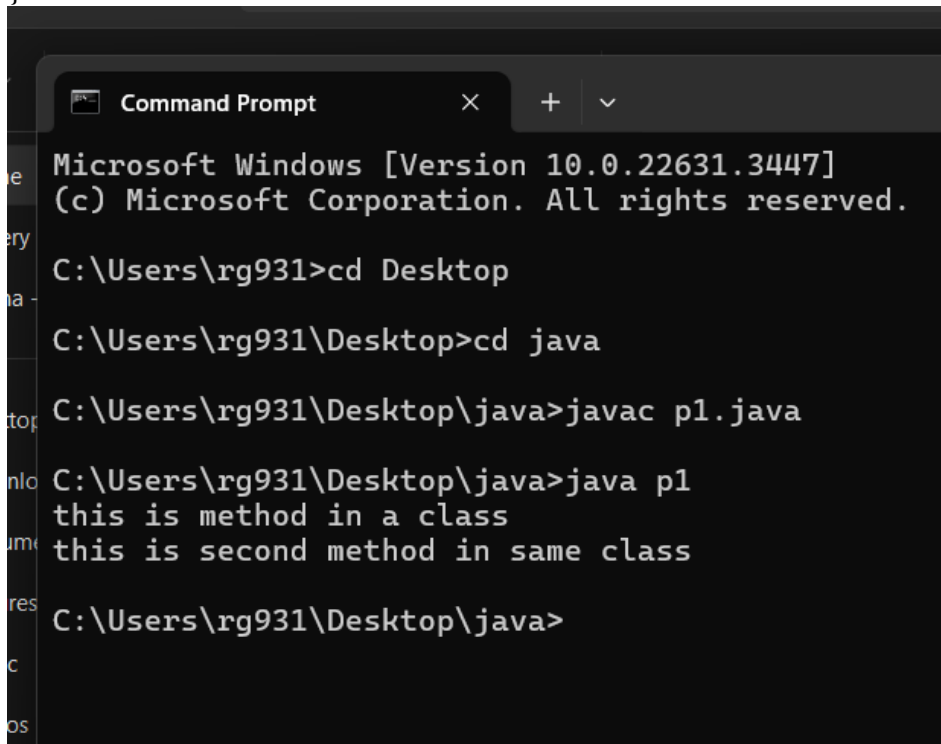
PRACTICAL-01

To write a JAVA program to implement class mechanism. - Create a class, methods and invoke them inside main method.

Program-

```
class subclass{
    void method1(){
        System.out.println("this is method in a class");
    }
    void method2(){
        System.out.println("this is second method in same class");
    }
}

public class p1{
    public static void main(String[] args) {
        subclass s1=new subclass();
        s1.method1();
        s1.method2();
    }
}
```



The screenshot shows a Windows Command Prompt window with the following text:

```
Microsoft Windows [Version 10.0.22631.3447]
(c) Microsoft Corporation. All rights reserved.

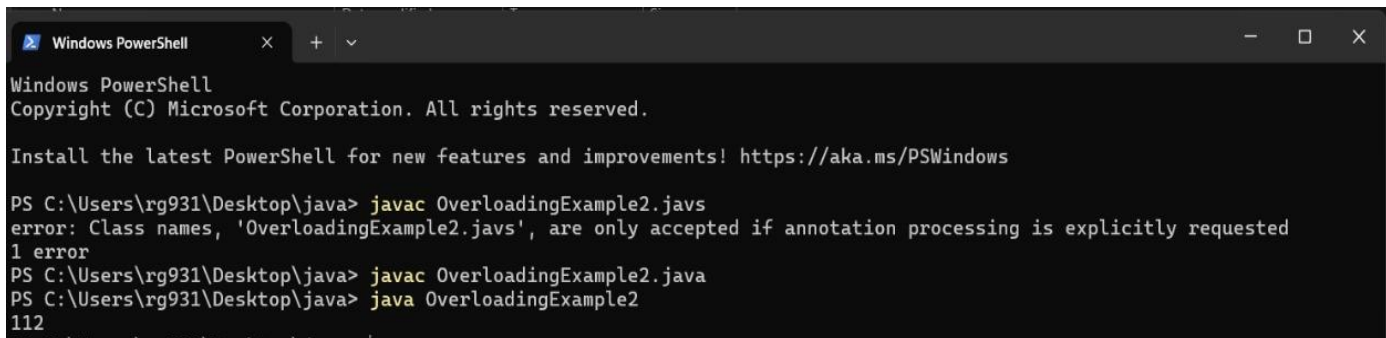
C:\Users\rg931>cd Desktop
C:\Users\rg931\Desktop>cd java
C:\Users\rg931\Desktop\java>javac p1.java
C:\Users\rg931\Desktop\java>java p1
this is method in a class
this is second method in same class
C:\Users\rg931\Desktop\java>
```

PRACTICAL-02

To write a JAVA program to implement constructor overloading

```
public class OverloadingExample2{
    int rollNum;
    OverloadingExample2() {
        rollNum =100; }
    OverloadingExample2(int rnum) {
        this();
        rollNum = rollNum+ rnum; }
    public int getRollNum() {
        return rollNum; }

    public static void main(String args[]) {
        OverloadingExample2 obj = new OverloadingExample2(12);
        System.out.println(obj.getRollNum()); } }
```



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\rg931\Desktop\java> javac OverloadingExample2.java
error: Class names, 'OverloadingExample2.java', are only accepted if annotation processing is explicitly requested
1 error
PS C:\Users\rg931\Desktop\java> javac OverloadingExample2.java
PS C:\Users\rg931\Desktop\java> java OverloadingExample2
112
```

PRACTICAL-03

Program-Develop a Java application to generate Electricity bill. Create a class with the following members: Consumer no., consumer name, previous month reading, current month reading, type of EB connection (i.e. domestic or commercial). Commute the bill amount using the following tariff.

If the type of the EB connection is domestic, calculate the amount to be paid as follows:

First 100 units - Rs. 1 per unit; 101-200 units - Rs. 2.50 per unit; 201-500 units - Rs. 4 per unit; 501 units - Rs. 6 per unit

If the type of the EB connection is commercial, calculate the amount to be paid as follows: First 100 units - Rs. 2 per unit; 101-200 units - Rs. 4.50 per unit; 201-500 units - Rs. 6 per unit; 501 units - Rs. 7 per unit.

Code-

```
import java.util.Scanner;

class ElectricityBill {
    int consumerNo;
    String consumerName;
    int prevMonthReading;
    int currMonthReading;
    String connectionType;

    ElectricityBill(int consumerNo, String consumerName, int prevMonthReading, int currMonthReading, String
connectionType) {
        this.consumerNo = consumerNo;
        this.consumerName = consumerName;
        this.prevMonthReading = prevMonthReading;
        this.currMonthReading = currMonthReading;
        this.connectionType = connectionType;
    }

    double calculateBillAmount() {
        int unitsConsumed = currMonthReading - prevMonthReading;
        double billAmount = 0;

        if (connectionType.equalsIgnoreCase("domestic")) {
            if (unitsConsumed <= 100) {
                billAmount = unitsConsumed * 1;
            } else if (unitsConsumed <= 200) {
                billAmount = 100 * 1 + (unitsConsumed - 100) * 2.50;
            } else if (unitsConsumed <= 500) {
                billAmount = 100 * 1 + 100 * 2.50 + (unitsConsumed - 200) * 4;
            } else {
                billAmount = 100 * 1 + 100 * 2.50 + 300 * 4 + (unitsConsumed - 500) * 6;
            }
        } else if (connectionType.equalsIgnoreCase("commercial")) {
            if (unitsConsumed <= 100) {
                billAmount = unitsConsumed * 2;
            } else if (unitsConsumed <= 200) {
                billAmount = 100 * 2 + (unitsConsumed - 100) * 4.50;
            } else if (unitsConsumed <= 500) {
                billAmount = 100 * 2 + 100 * 4.50 + (unitsConsumed - 200) * 6;
            } else {
                billAmount = 100 * 2 + 100 * 4.50 + 300 * 6 + (unitsConsumed - 500) * 7;
            }
        }

        return billAmount;
    }

    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);
        System.out.println("Enter consumer number:");
        int consumerNo = scanner.nextInt();
    }
}
```

```

scanner.nextLine();
System.out.println("Enter consumer name:");
String consumerName = scanner.nextLine();
System.out.println("Enter previous month reading:");
int prevMonthReading = scanner.nextInt();
System.out.println("Enter current month reading:");
int currMonthReading = scanner.nextInt();
scanner.nextLine();
System.out.println("Enter type of EB connection (domestic or commercial):");
String connectionType = scanner.nextLine();

ElectricityBill bill = new ElectricityBill(consumerNo, consumerName, prevMonthReading, currMonthReading,
connectionType);
double billAmount = bill.calculateBillAmount();
System.out.println("Electricity bill amount: Rs. " + billAmount);

scanner.close();
}

```

```

C:\Users\admin\Desktop>javac ElectricityBill.java

C:\Users\admin\Desktop>java ElectricityBill
Enter consumer number:
1234
Enter consumer name:
radha
Enter previous month reading:
105
Enter current month reading:
301
Enter type of EB connection (domestic or commercial):
domestic
Electricity bill amount: Rs. 340.0

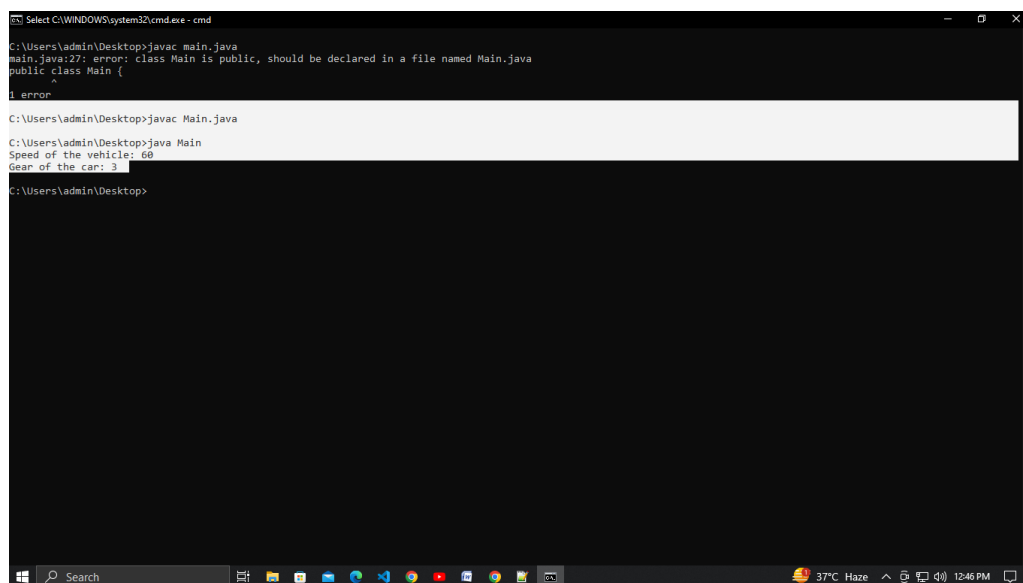
C:\Users\admin\Desktop>

```

PRACTICAL-04

Write a JAVA program give example for "super" keyword

```
class Vehicle {
    int speed;
    Vehicle(int speed) {
        this.speed = speed;
    }
    void displaySpeed() {
        System.out.println("Speed of the vehicle: " + speed);
    }
}
class Car extends Vehicle {
    int gear;
    Car(int speed, int gear) {
        super(speed); // Calling superclass constructor using super keyword
        this.gear = gear;
    }
    void display() {
        super.displaySpeed(); // Calling superclass method using super keyword
        System.out.println("Gear of the car: " + gear);
    }
}
public class Main {
    public static void main(String[] args) {
        Car car = new Car(60, 3);
        car.display();
    }
}
```



```
C:\Users\admin\Desktop>javac main.java
main.java:27: error: class Main is public, should be declared in a file named Main.java
public class Main {
      ^
1 error

C:\Users\admin\Desktop>javac Main.java

C:\Users\admin\Desktop>java Main
Speed of the vehicle: 60
Gear of the car: 3

C:\Users\admin\Desktop>
```

PRACTICAL-05

Create a base class Fruit which has name, taste and size as its attributes. A method called eat() is created which describes the name of the fruit and its taste. Inherit the same in 2 other class Apple and Orange and override the eat() method to represent each fruit taste

```
class Fruit {
    // Attributes
    String name;
    String taste;
    String size;
    public Fruit(String name, String taste, String size) {
        this.name = name;
        this.taste = taste;
        this.size = size;
    }
    public void eat() {
        System.out.println("Eating " + name + ". It tastes " + taste + ".");
    }
}
class Apple extends Fruit {
    public Apple(String name, String taste, String size) {
        super(name, taste, size);
    }
    @Override
    public void eat() {
        System.out.println("Biting into the " + name + ". It tastes crisp and " + taste + ".");
    }
}
class Orange extends Fruit {
    public Orange(String name, String taste, String size) {
        super(name, taste, size);
    }
    @Override
    public void eat() {
        System.out.println("Peeling the " + name + ". It tastes tangy and " + taste + ".");
    }
}

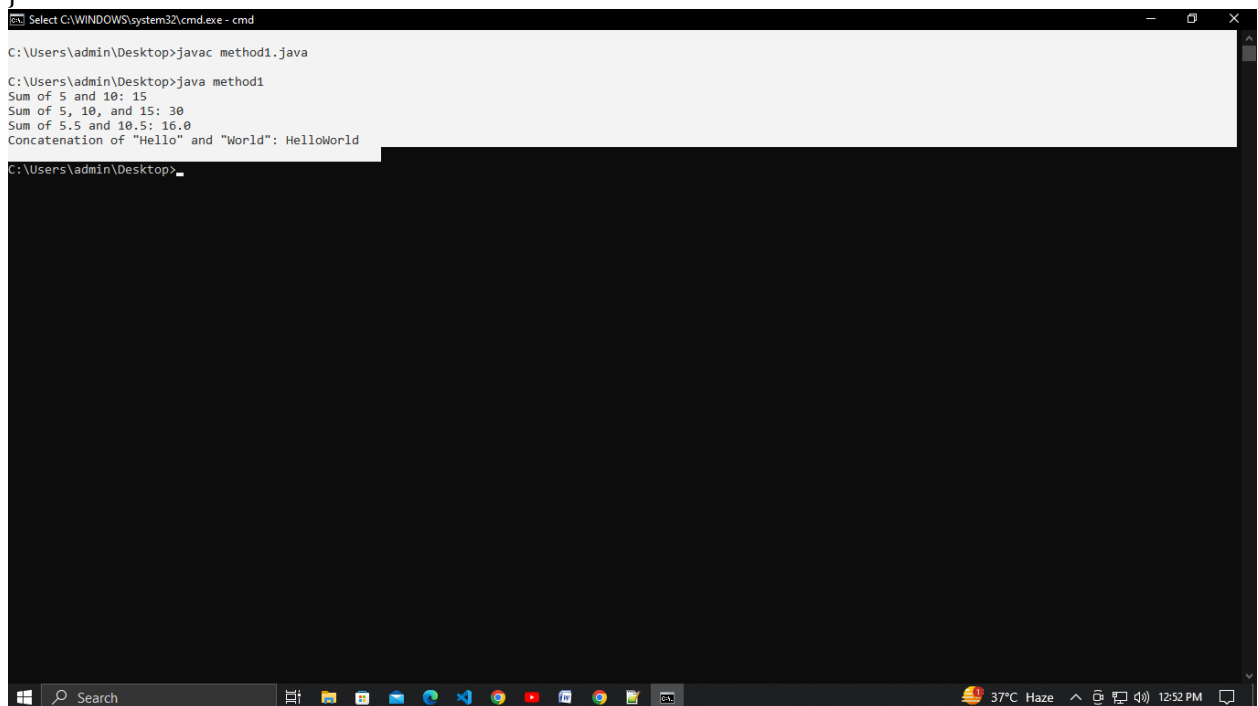
public class Main {
    public static void main(String[] args) {
        Apple apple = new Apple("Apple", "sweet", "medium");
        Orange orange = new Orange("Orange", "juicy", "large");
        apple.eat();
        orange.eat();
    }
}
```

PRACTICAL-06

Write a java program to illustrate the concept of class with method overloading.

```
class Calculator {
    int add(int a, int b) {
        return a + b;
    }
    int add(int a, int b, int c) {
        return a + b + c;
    }
    double add(double a, double b) {
        return a + b;
    }
    String add(String a, String b) {
        return a + b;
    }
}

public class method1 {
    public static void main(String[] args) {
        Calculator calculator = new Calculator();
        System.out.println("Sum of 5 and 10: " + calculator.add(5, 10));
        System.out.println("Sum of 5, 10, and 15: " + calculator.add(5, 10, 15));
        System.out.println("Sum of 5.5 and 10.5: " + calculator.add(5.5, 10.5));
        System.out.println("Concatenation of \"Hello\" and \"World\": " +
calculator.add("Hello", "World"));
    }
}
```



```
Select C:\WINDOWS\system32\cmd.exe - cmd

C:\Users\admin\Desktop>javac method1.java

C:\Users\admin\Desktop>java method1
Sum of 5 and 10: 15
Sum of 5, 10, and 15: 30
Sum of 5.5 and 10.5: 16.0
Concatenation of "Hello" and "World": HelloWorld

C:\Users\admin\Desktop>
```

PRACTICAL-07

Write Java program(s) on use of inheritance, preventing inheritance using final, abstract classes.

```
class Vehicle {
    void display() {
        System.out.println("This is a vehicle.");
    }
}

class Car extends Vehicle {
    void displayCar() {
        System.out.println("This is a car.");
    }
}

public class Main {
    public static void main(String[] args) {
        Car car = new Car();
        car.display();    // Calls the display method of the Vehicle class
        car.displayCar(); // Calls the displayCar method of the Car class
    }
}
```


PRACTICAL-08

Develop a java application to implement currency converter (Dollar to INR, EURO to INR, Yen) using Interfaces

```
interface CurrencyConverter {
    double convertToINR(double amount);}
class DollarConverter implements CurrencyConverter {
    public double convertToINR(double amount) {
        // Assuming 1 Dollar = 74.52 INR
        return amount * 74.52;
    }
}
class EuroConverter implements CurrencyConverter {
    public double convertToINR(double amount) {
        // Assuming 1 Euro = 87.62 INR
        return amount * 87.62;
    }
}
class YenConverter implements CurrencyConverter {
    public double convertToINR(double amount) {
        // Assuming 1 Yen = 0.68 INR
        return amount * 0.68;
    }
}
public class Main {
    public static void main(String[] args) {
        CurrencyConverter dollarConverter = new DollarConverter();
        double dollarsToINR = dollarConverter.convertToINR(100);
        System.out.println("100 Dollars in INR: " + dollarsToINR);

        CurrencyConverter euroConverter = new EuroConverter();
        double eurosToINR = euroConverter.convertToINR(100);
        System.out.println("100 Euros in INR: " + eurosToINR);

        CurrencyConverter yenConverter = new YenConverter();
        double yenToINR = yenConverter.convertToINR(100);
        System.out.println("100 Yen in INR: " + yenToINR);
    }
}
```

PRACTICAL-09

To write a JAVA program that implements Runtime polymorphism

```
class Animal {
    void sound() {
        System.out.println("Animal makes a sound");
    }
}
class Dog extends Animal {
    void sound() {
        System.out.println("Dog barks");
    }
}
class Cat extends Animal {
    void sound() {
        System.out.println("Cat meows");
    }
}
public class Main {
    public static void main(String[] args) {
        Animal animal1 = new Dog();
        Animal animal2 = new Cat();
        animal1.sound();
        animal2.sound();
    }
}
```

PRACTICAL-10

To write a JAVA program that describes exception handling mechanism

```
public class ExceptionHandlingExample {  
    public static void main(String[] args) {  
        try {  
            int result = divide(10, 0);  
            System.out.println("Result: " + result);  
        } catch (ArithmeticException e) {  
            System.out.println("Error: Division by zero");  
        }  
    }  
    static int divide(int num1, int num2) {  
        return num1 / num2;  
    }  
}
```

PRACTICAL-11

To write a JAVA program Illustrating Multiple catch clause

```
public class MultipleCatchExample {  
    public static void main(String[] args) {  
        try {  
            int result = divide(10, 0);  
            System.out.println("Result: " + result);  
        } catch (ArithmeticException e) {  
            System.out.println("Error: Division by zero");  
        } catch (NullPointerException e) {  
            System.out.println("Error: NullPointerException occurred");  
        } catch (Exception e) {  
            System.out.println("Error: An exception occurred");  
        }  
    }  
    static int divide(int num1, int num2) {  
        return num1 / num2;  
    }  
}
```

PRACTICAL-12

To write a JAVA program for creation of Illustrating throw

```
public class ThrowExample {
    public static void main(String[] args) {
        try {
            int result = divide(10, 0);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero");
        }
    }
    static int divide(int num1, int num2) {
        if (num2 == 0) {
            throw new ArithmeticException("Division by zero");
        }
        return num1 / num2;
    }
}
```

PRACTICAL-13

To write a JAVA program for creation of Java Built-in Exceptions

```
public class BuiltInExceptionsExample {
    public static void main(String[] args) {
        try {
            int[] arr = {1, 2, 3};
            System.out.println(arr[3]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: Array index out of bounds");
        }
        try {
            String str = null;
            System.out.println(str.length());
        } catch (NullPointerException e) {
            System.out.println("Error: Null pointer exception");
        }
        try {
            int result = 10 / 0;
        } catch (ArithmeticException e) {
            System.out.println("Error: Division by zero");
        }
    }
}

// Custom exception class
class InvalidAgeException extends Exception {
    // Constructor to initialize the exception message
    public InvalidAgeException() {
        super("Invalid age entered");
    }
}
```

PRACTICAL-14

To write a JAVA program for creation of Java user defined Exceptions

```
// Class representing a person
class Person {
    private String name;
    private int age;

    // Constructor
    public Person(String name, int age) throws InvalidAgeException {
        if (age < 0 || age > 120) {
            // If age is invalid, throw InvalidAgeException
            throw new InvalidAgeException();
        }
        this.name = name;
        this.age = age;
    }

    // Getter methods
    public String getName() {
        return name;
    }

    public int getAge() {
        return age;
    }
}

// Main class to test the user-defined exception
public class UserDefinedExceptionExample {
    public static void main(String[] args) {
        try {
            // Creating a person with invalid age
            Person person = new Person("John", 150);
            System.out.println("Name: " + person.getName() + ", Age: " + person.getAge());
        } catch (InvalidAgeException e) {
            // Catch block to handle InvalidAgeException
            System.out.println("Error: " + e.getMessage());
        }
    }
}
```

PRACTICAL-15

Write a Java program for multi-thread implementation

```
class MyThread extends Thread {
    public void run() {
        // Code inside the run method represents the task to be executed by the thread
        for (int i = 1; i <= 5; i++) {
            System.out.println("Thread: " + Thread.currentThread().getId() + " Count: " + i);
            try {
                Thread.sleep(1000); // Pause the thread execution for 1 second
            } catch (InterruptedException e) {
                System.out.println(e);
            }
        }
    }
}

public class MultiThreadExample {
    public static void main(String[] args) {
        MyThread thread1 = new MyThread(); // Creating first thread
        MyThread thread2 = new MyThread(); // Creating second thread
        MyThread thread3 = new MyThread(); // Creating third thread

        thread1.start(); // Starting first thread
        thread2.start(); // Starting second thread
        thread3.start(); // Starting third thread
    }
}
```