Lab Exercises

Classes and Object (Assignment 1)

1. Create a class called Emp with data members empno, empname, designation, dept and salary and methods as readEmpData() (to read values to data members) and displayEmpData() (to display data members values to the screen) create an employee instance and display its information.

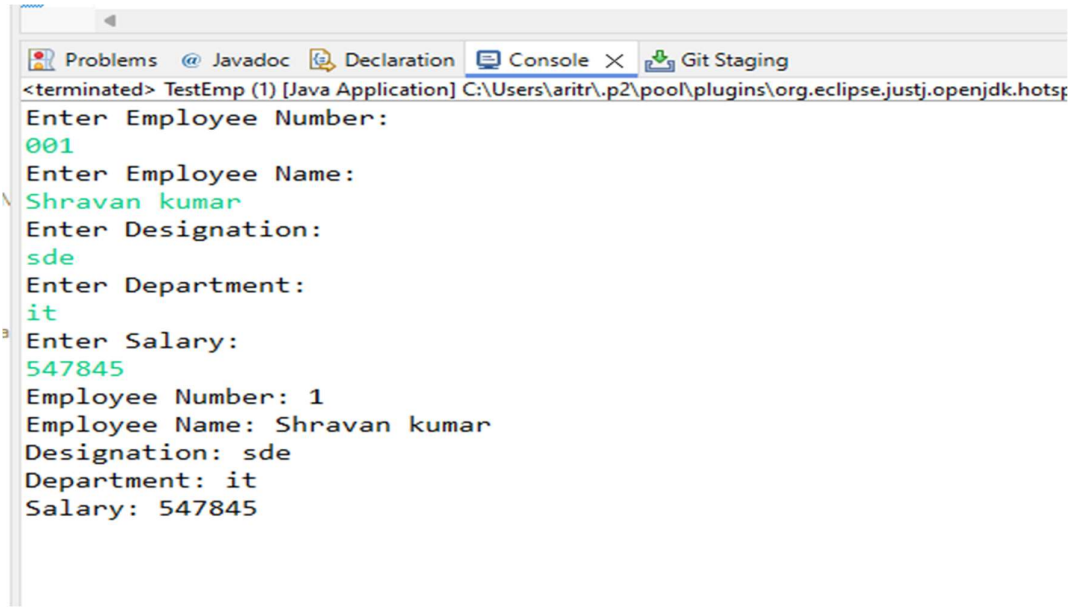
```java
import java.util.Scanner; // Import the Scanner class for taking user input

public class Emp {
    int empno; // Variable to store employee number
    String empname; // Variable to store employee name
    String designation; // Variable to store employee designation
    String dept; // Variable to store employee department
    int salary; // Variable to store employee salary

    // Method to read employee data from user input
    public void readEmpData() {
        Scanner sc = new Scanner(System.in); // Create a Scanner object for input
        System.out.println("Enter Employee Number: ");
        empno = sc.nextInt(); // Read integer input for employee number

        // Consume the newline character left by nextInt()
        sc.nextLine();

        System.out.println("Enter Employee Name: ");
        empname = sc.next(); // Read string input for employee name

        System.out.println("Enter Designation: ");
        designation = sc.next(); // Read string input for employee designation

        System.out.println("Enter Department: ");
        dept = sc.next(); // Read string input for employee department

        System.out.println("Enter Salary: ");
        salary = sc.nextInt(); // Read integer input for employee salary

        sc.close(); // Close the Scanner object to prevent resource leaks
    }

    // Method to display employee data
    public void displayEmpData() {
        System.out.println("Employee Number: " + empno); // Display employee number
        System.out.println("Employee Name: " + empname); // Display employee name
        System.out.println("Designation: " + designation); // Display employee designation
        System.out.println("Department: " + dept); // Display employee department
        System.out.println("Salary: " + salary); // Display employee salary
    }
```

```
}
```

```
    // Method to display employee data
    public void displayEmpData() {
        System.out.println("Employee Number: " + empno); // Display employee number
        System.out.println("Employee Name: " + empname); // Display employee name
        System.out.println("Designation: " + designation); // Display employee designation
        System.out.println("Department: " + dept); // Display employee department
        System.out.println("Salary: " + salary); // Display employee salary
    }
}
```

```
public class TestEmp {

    public static void main(String[] args) {
        // Create an instance of the Emp class
        Emp employee = new Emp();

        // Call the method to read employee data from user input
        employee.readEmpData();

        // Call the method to display the employee data
        employee.displayEmpData();
    }
}
```

Output



2. Create a class Electricity bill with data members as customer number, customer name, units consumed and methods as follows:

    1. readData() - to read the values of data members.
    2. showData - to display the customer details
    3. computeBill() - to calculate and return electricity charges to be paid. calculate the bill as specified below

number of units	charges
< = 100	Rs.1.20
for the next 200 units	Rs. 2.00
for the next 300 units	Rs. 3.00
for more	Rs. 5.00

ex: input = 320 units output = 100*1.20 +200*2.00+20*3.00 = Rs. 580

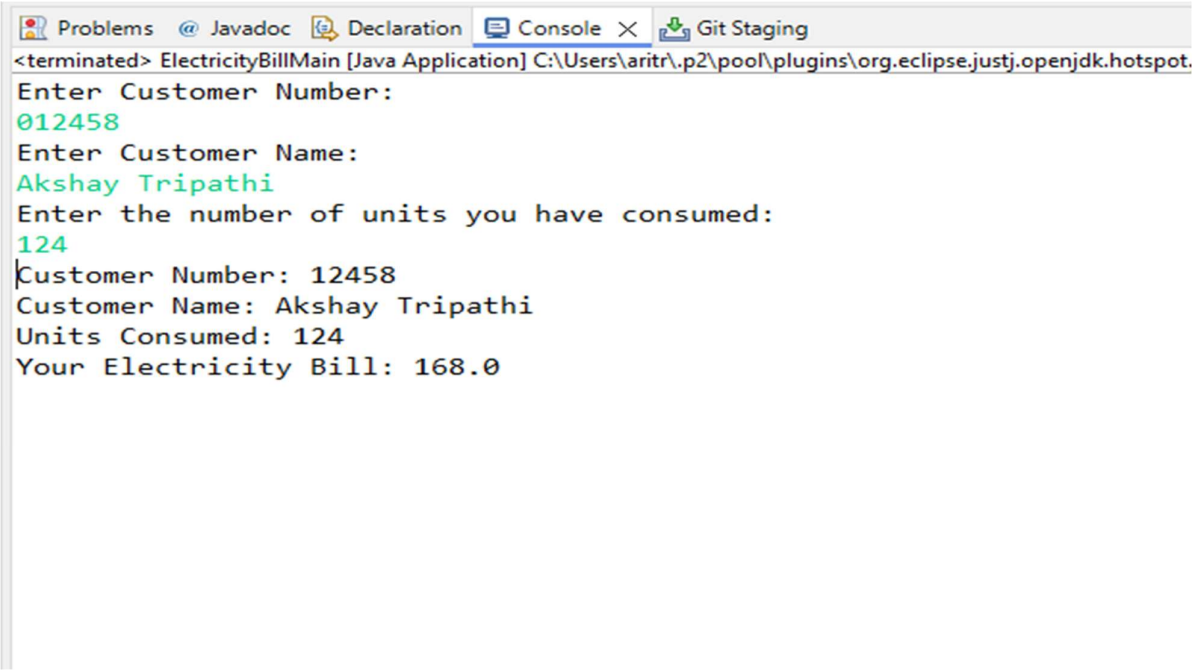
Read customer object values, calculate electricity bill and display the values.

```
import java.util.Scanner; // Import the Scanner class for taking user input

public class ElectricityBill {
    int customerNumber; // Variable to store customer number
    String customerName; // Variable to store customer name
    int unitsConsumed; // Variable to store the number of units consumed

    // Method to read customer data from user input
    public void readData() {
        Scanner s = new Scanner(System.in); // Create a Scanner object for input

        // Prompt and read customer number
        System.out.println("Enter Customer Number: ");
        customerNumber = s.nextInt();

        // Consume the newline character left by nextInt()
        s.nextLine();

        // Prompt and read customer name
        System.out.println("Enter Customer Name: ");
        customerName = s.nextLine();

        // Prompt and read the number of units consumed
        System.out.println("Enter the number of units you have consumed: ");
        unitsConsumed = s.nextInt();

        // Close the Scanner object to prevent resource leaks
        s.close();
    }

    // Method to display customer data
```

```
    public void showData() {
        // Display customer number
        System.out.println("Customer Number: " + customerNumber);

        // Display customer name
        System.out.println("Customer Name: " + customerName);

        // Display the number of units consumed
        System.out.println("Units Consumed: " + unitsConsumed);
    }

    // Method to compute the electricity bill based on units consumed
    public double computeBill() {
        double bill; // Variable to store the computed bill amount

        // Calculate the bill amount based on different slabs
        if (unitsConsumed <= 100) {
            bill = unitsConsumed * 1.20; // First 100 units at $1.20 per unit
        } else if (unitsConsumed <= 300) {
            bill = 100 * 1.20 + (unitsConsumed - 100) * 2.00; // Next 200 units at $2.00
per unit
        } else if (unitsConsumed <= 600) {
            bill = (100 * 1.20) + (200 * 2.00) + (unitsConsumed - 300) * 3.00; // Next 300
units at $3.00 per unit
        } else {
            bill = (100 * 1.20) + (200 * 2.00) + (300 * 3.00) + (unitsConsumed - 600) *
5.00; // Above 600 units at $5.00 per unit


        }

        // Return the computed bill amount
        return bill;
    }
}



public class ElectricityBillMain {

    public static void main(String[] args) {
        // Create an instance of the ElectricityBill class
        ElectricityBill eb = new ElectricityBill();

        // Call the method to read customer data from user input
        eb.readData();

        // Call the method to display customer data
        eb.showData();

        // Call the method to compute the electricity bill and store the result
        double bill = eb.computeBill();
```

```
        // Display the computed electricity bill
        System.out.println("Your Electricity Bill: " + bill);
    }
}
```

Output

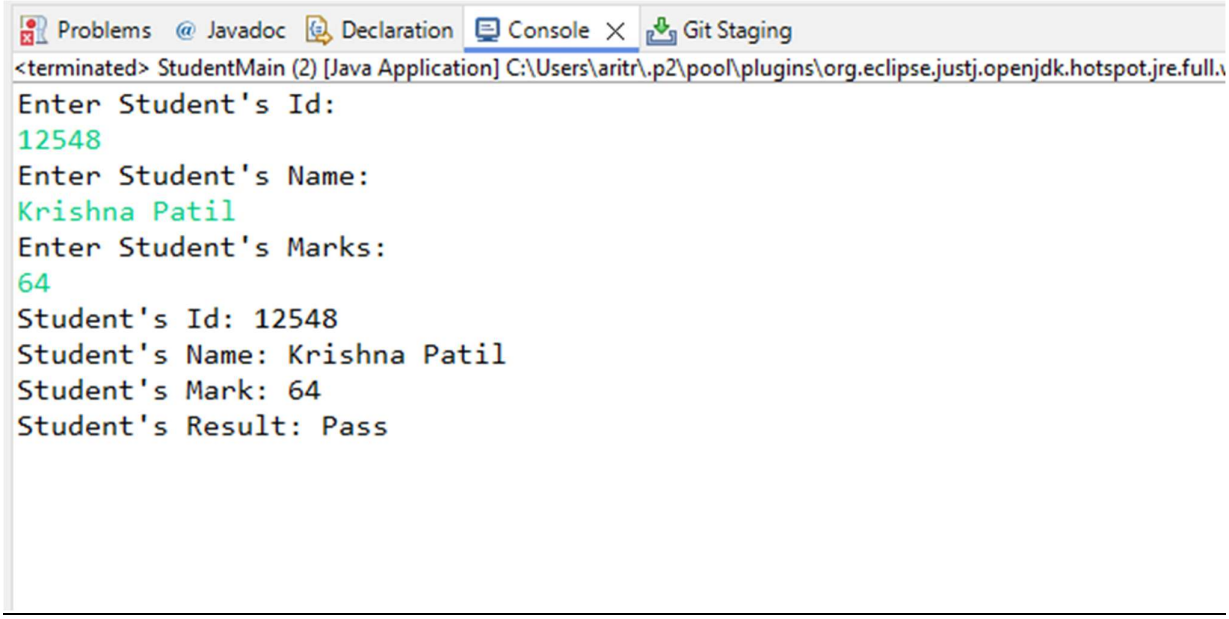


3. Write a Java program to create Student class with member variable as id, name, mark and result. Use method to initialize the value of name, id and marks. Write a member function to find the result and display the student details with result.

```
import java.util.Scanner;  // Import the Scanner class for taking user input

public class Student {
    static int id;  // Static variable to store student ID
    static String name;  // Static variable to store student name
    static int mark;  // Static variable to store student mark
    String result;  // Instance variable to store result (Pass/Fail)

    // Method to read student details from user input
    public void readDetails() {
        Scanner sc = new Scanner(System.in);  // Create a Scanner object for input

        // Prompt and read student ID
        System.out.println("Enter Student's Id: ");
        id = sc.nextInt();
```

```
        // Consume the newline character left by nextInt()
        sc.nextLine();

        // Prompt and read student name
        System.out.println("Enter Student's Name: ");
        name = sc.nextLine();

        // Prompt and read student marks
        System.out.println("Enter Student's Marks: ");
        mark = sc.nextInt();

        // Close the Scanner object to prevent resource leaks
        sc.close();
    }

    // Method to determine the result based on marks
    public void checkResult() {
        if (mark > 50) {
            result = "Pass";  // Assign "Pass" if marks are greater than 50
        } else {
            result = "Fail";  // Assign "Fail" if marks are 50 or less
        }
    }

    // Method to display student details and result
    public void dispDetails() {
        System.out.println("Student's Id: " + id);  // Display student ID
        System.out.println("Student's Name: " + name);  // Display student name
        System.out.println("Student's Mark: " + mark);  // Display student marks
        System.out.println("Student's Result: " + result);  // Display student result
    }
}

public class StudentMain {

    public static void main(String[] args) {
        // Create an instance of the Student class
        Student s1 = new Student();

        // Call the method to read student details from user input
        s1.readDetails();

        // Call the method to determine the student's result based on marks
        s1.checkResult();

        // Call the method to display student details and result
        s1.dispDetails();
    }
}
```

Output



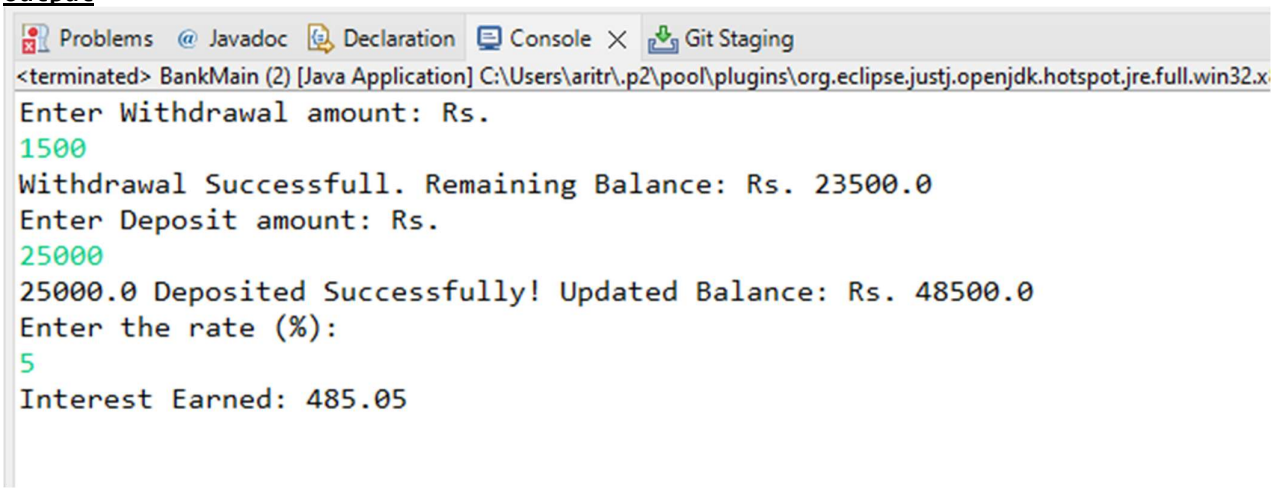
---

4. Write a Java program that creates a account class with instance variable accno, accname, amount and instance method withdraw, deposit, and interest. Create object of account class test all methods.

```
public class Account {
    int accNo;  // Variable to store account number
    String accName;  // Variable to store account holder's name
    double amount;  // Variable to store account balance

    // Constructor to initialize the account details
    public Account(int accNo, String accName, double amount) {
        this.accNo = accNo;  // Initialize account number
        this.accName = accName;  // Initialize account holder's name
        this.amount = amount;  // Initialize account balance
    }

    // Method to withdraw an amount from the account
    public void withdraw(double withdrawAmount) {
        // Check if the withdraw amount is valid and sufficient balance is available
        if (withdrawAmount > 0 && withdrawAmount <= amount) {
            amount -= withdrawAmount;  // Deduct the withdraw amount from the balance
            System.out.println("Withdrawal Successful. Remaining Balance: Rs. " + amount);
        } else {
            // Print an error message if the withdraw amount is invalid or insufficient
balance
            System.out.println("Withdrawal failed! Insufficient Balance.");
        }
    }

    // Method to deposit an amount into the account
    public void deposit(double depositAmount) {
```

```
        // Check if the deposit amount is valid
        if (depositAmount > 0) {
            amount += depositAmount;  // Add the deposit amount to the balance
            System.out.println(depositAmount + " Deposited Successfully! Updated Balance:
Rs. " + amount);
        } else {
            // Print an error message if the deposit amount is invalid
            System.out.println("Invalid Amount.");
        }
    }

    // Method to calculate the interest earned on the account balance
    public void calculateInterest(double rate) {
        double interest = (amount * rate) / 100;  // Calculate interest based on the rate
        System.out.println("Interest Earned: " + interest);  // Print the interest earned
    }
}

import java.util.Scanner;  // Import the Scanner class for taking user input

public class BankMain {

    public static void main(String[] args) {
        // Create a Scanner object for input
        Scanner s = new Scanner(System.in);

        // Creating an Account object with initial details
        Account a = new Account(154824, "Aritra", 25000);

        // Prompt and read the amount to withdraw
        System.out.println("Enter Withdrawal amount: Rs. ");
        double withdrawAmount = s.nextDouble();
        // Call the withdraw method to process the withdrawal
        a.withdraw(withdrawAmount);

        // Prompt and read the amount to deposit
        System.out.println("Enter Deposit amount: Rs. ");
        double depositAmount = s.nextDouble();
        // Call the deposit method to process the deposit
        a.deposit(depositAmount);

        // Prompt and read the interest rate
        System.out.println("Enter the rate (%): ");
        double rate = s.nextDouble();
        // Call the calculateInterest method to compute the interest
        a.calculateInterest(rate);

        // Close the Scanner object to prevent resource leaks
        s.close();
    }
}
```

Output



5. Write a Java program to create a class called player with name, age, country Name, total run as instance member. Create 5 player objects and write instance method to display the details of Player having more than 5000 as total run

```
import java.util.Scanner;  // Import the Scanner class for taking user input

public class Player {
    String name;  // Variable to store player's name
    int age;  // Variable to store player's age
    String countryName;  // Variable to store player's country name
    int totalRun;  // Variable to store player's total runs

    // Method to read player details from user input
    public void readDetails() {
        Scanner s = new Scanner(System.in);  // Create a Scanner object for input

        // Prompt and read player's name
        System.out.println("Enter Player's Name: ");
        name = s.nextLine();

        // Prompt and read player's age
        System.out.println("Enter Player's Age: ");
        age = s.nextInt();
        s.nextLine();  // Consume the newline character left by nextInt()

        // Prompt and read player's country name
        System.out.println("Enter Player's Country Name: ");
        countryName = s.nextLine();

        // Prompt and read player's total runs
        System.out.println("Enter Player's Total Run: ");
        totalRun = s.nextInt();
        s.nextLine();  // Consume the newline character left by nextInt()
    }
```

```
    // Method to check if the player's total runs exceed 5000
    public void checkRun() {
        if (totalRun > 5000) {
            dispPlayer();  // Call dispPlayer() method to display player details if total
runs exceed 5000
        }
    }

    // Method to display player details
    public void dispPlayer() {
        System.out.println("Player's Name: " + name);  // Display player's name
        System.out.println("Player's Age: " + age);  // Display player's age
        System.out.println("Player's Country Name: " + countryName);  // Display player's
country name
        System.out.println("Player's Total Run: " + totalRun);  // Display player's total
runs
    }
}

public class PlayerMain {

    public static void main(String[] args) {
        // Create instances of the Player class
        Player p1 = new Player();
        Player p2 = new Player();
        Player p3 = new Player();
        Player p4 = new Player();
        Player p5 = new Player();

        // Read details for each player
        p1.readDetails();
        p2.readDetails();
        p3.readDetails();
        p4.readDetails();
        p5.readDetails();

        // Check runs and display details if applicable for each player
        p1.checkRun();
        p2.checkRun();
        p3.checkRun();
        p4.checkRun();
        p5.checkRun();
    }
}
```

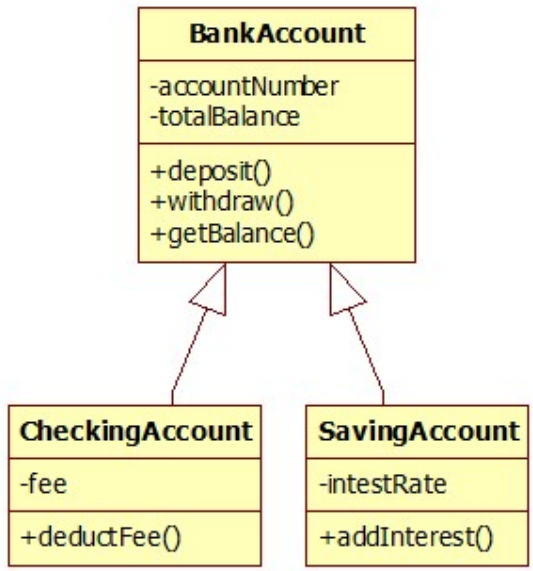
Output

```
Enter Player's Name:
Virat Kohli
Enter Player's Age:
35
Enter Player's Country Name:
```

```
India
Enter Player's Total Run:
8,848
Enter Player's Name:
Rohit Sharma
Enter Player's Age:
37
Enter Player's Country Name:
India
Enter Player's Total Run:
10,709
Enter Player's Name:
Chris Gayle
Enter Player's Age:
44
Enter Player's Country Name:
West Indies
Enter Player's Total Run:
10480
Enter Player's Name:
AB de Villiers
Enter Player's Age:
40
Enter Player's Country Name:
South Africa
Enter Player's Total Run:
9577
Enter Player's Name:
Shubman Gill
Enter Player's Age:
24
Enter Player's Country Name:
India
Enter Player's Total Run:
2271
Player's Name: Virat Kohli
Player's Age: 35
Player's Country Name: India
Enter Player's Total Run: 8848
Player's Name: Rohit Sharma
Player's Age: 37
Player's Country Name: India
Enter Player's Total Run: 10709
Player's Name: Chris Gayle
Player's Age: 44
Player's Country Name: West Indies
Enter Player's Total Run: 10480
Player's Name: AB de Villiers
Player's Age: 40
Player's Country Name: South Africa
Enter Player's Total Run: 9577
```

Inheritance and Polymorphism (Assignment 2)

1. Write java program to implement Inheritance with following example:




```
public class BankAccount {
    int accountNumber;  // Variable to store the account number
    double totalBalance;  // Variable to store the total balance

    // Constructor to initialize the account number and total balance
    public BankAccount(int accountNumber, double totalBalance) {
        this.accountNumber = accountNumber;  // Initialize account number
        this.totalBalance = totalBalance;  // Initialize total balance
    }

    // Method to deposit an amount into the account
    public double deposit(double depositAmount) {
        // Check if the deposit amount is greater than zero
        if (depositAmount > 0) {
            totalBalance += depositAmount;  // Add the deposit amount to the total balance
            System.out.println("Rs. " + depositAmount + " Deposited Successfully. Current
Balance Rs. " + totalBalance);
        }
        return totalBalance;  // Return the updated balance
    }

    // Method to withdraw an amount from the account
    public double withdraw(double withdrawAmount) {
        // Check if the withdraw amount is greater than zero and does not exceed the total
balance
        if (withdrawAmount > 0 && withdrawAmount <= totalBalance) {
            totalBalance -= withdrawAmount;  // Subtract the withdraw amount from the
total balance
```

```
            System.out.println(withdrawAmount + " Withdrawn Successfully. Current Balance:
" + totalBalance);
        } else {
            // Print an error message if the balance is insufficient
            System.out.println("Insufficient Balance.");
        }
        return totalBalance;  // Return the updated balance
    }

    // Method to get the current balance
    public void getBalance() {
        System.out.println("Your Current Balance is: " + totalBalance);  // Print the
current balance
    }
}



public class CheckingAccount extends BankAccount {
    double fee; // Variable to store the fee

    // Constructor to initialize the checking account with account number, balance, and
fee
    public CheckingAccount(int accountNumber, double totalBalance, double fee) {
        super(accountNumber, totalBalance); // Call the superclass constructor to
initialize account number and balance
        this.fee = fee; // Initialize the fee
    }

    // Method to deduct the fee from the account balance
    public void deductFee(double fee) {
        totalBalance -= fee; // Subtract the fee from the total balance
        System.out.println("Total Balance after fees deduction: " + totalBalance); //
Display the updated balance
    }
}


public class SavingsAccount extends BankAccount {
    double interestRate; // Variable to store the interest rate

    // Constructor to initialize the savings account with account number, balance, and
interest rate
    public SavingsAccount(int accountNumber, double totalBalance, double interestRate) {
        super(accountNumber, totalBalance); // Call the superclass constructor to
initialize account number and balance
        this.interestRate = interestRate; // Initialize the interest rate
    }

    // Method to add interest to the account balance
    public double addInterest(double interestRate) {
        // Calculate the total interest earned
```

```
        double totalInterest = (totalBalance * interestRate * 1) / 100;
        System.out.println("Interest earned: Rs. " + totalInterest); // Display the
interest earned

        // Add the earned interest to the total balance
        totalBalance += totalInterest;
        // Return the updated balance
        return totalBalance;
    }
}

import java.util.Scanner; // Import the Scanner class for taking user input

public class BankMain {

    public static void main(String[] args) {
        int accNo = 0; // Variable to store account number
        double totalBalance; // Variable to store total balance
        double fees = 0; // Variable to store fees
        double interestRate = 0; // Variable to store interest rate

        Scanner s = new Scanner(System.in); // Create a Scanner object for input

        // Prompt and read account details
        System.out.println("Enter the Account Details Below: ");
        System.out.println("Enter Account Number: ");
        accNo = s.nextInt();

        System.out.println("Enter Total Balance: ");
        totalBalance = s.nextDouble();

        // Create a BankAccount object with the provided account number and balance
        BankAccount ba = new BankAccount(accNo, totalBalance);

        // Prompt and read deposit amount, then deposit it into the account
        System.out.println("Enter the amount you want to Deposit: ");
        double depositAmount = s.nextDouble();
        ba.deposit(depositAmount);

        // Prompt and read withdrawal amount, then withdraw it from the account
        System.out.println("Enter the amount you want to Withdraw: ");
        double withdrawAmount = s.nextDouble();
        ba.withdraw(withdrawAmount);

        // Prompt and read fees, then create a CheckingAccount object and deduct the fee
        System.out.println("Enter Fees: ");
        fees = s.nextDouble();
        CheckingAccount ca = new CheckingAccount(accNo, totalBalance, fees);
        ca.deductFee(fees);

        // Prompt and read interest rate, then create a SavingsAccount object and add
interest
```

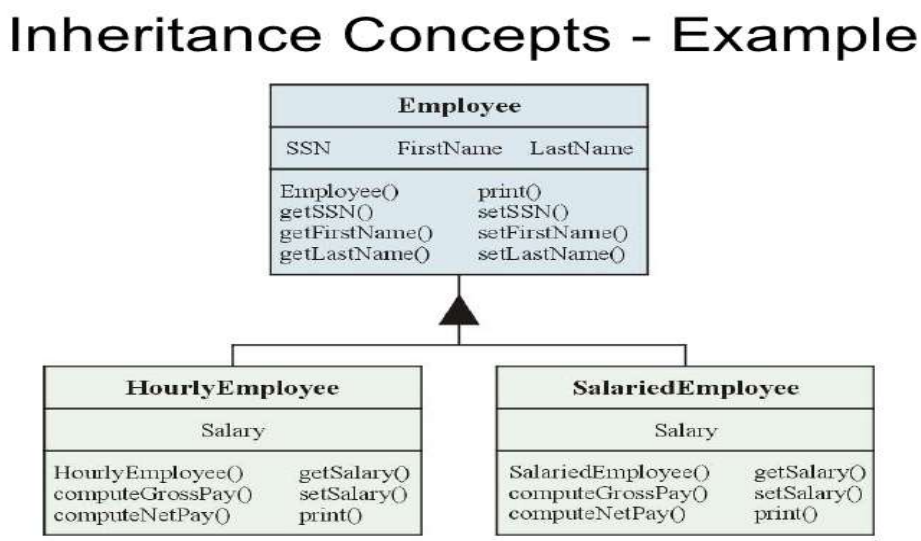
```
        System.out.println("Enter Interest Rate: ");
        interestRate = s.nextDouble();
        SavingsAccount sa = new SavingsAccount(accNo, totalBalance, interestRate);
        double interest = sa.addInterest(interestRate);

        // Calculate the total balance after all transactions
        totalBalance = totalBalance + depositAmount - withdrawAmount - fees + interest;

        // Display the current balance
        ba.getBalance();

        s.close(); // Close the Scanner object
    }
}
```

Output

```
Problems  Javadoc  Declaration  Console  Git Staging
<terminated> BankMain (3) [Java Application] C:\Users\aritr\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.2.v2024
Enter the Account Details Below:
Enter Account Number:
123456
Enter Total Balance:
123654
Enter the amount you want to Deposit:
25000
Rs. 25000.0 Deposited Successfully. Current Balance Rs. 148654.0
Enter the amount you want to Withdraw:
10000
10000.0Withdrawn Successfuylly. Current Balance: 138654.0
Enter Fees:
150
Total Balance after fees deduction: 123504.0
Enter Interest Rate:
5
Interest earned: Rs. 6182.7
Your Current Balance is: 138654.0
```

---

2. Write java program to implement Inheritance with following example:



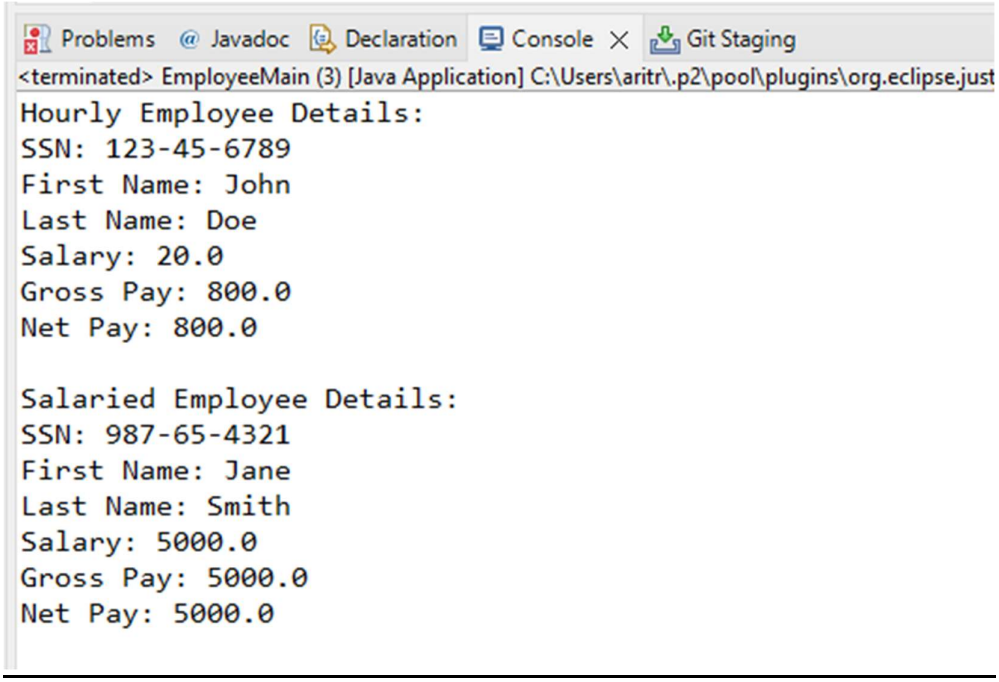
```java
// Base class: Employee
public class Employee {
    // Private fields to store the employee's SSN, first name, and last name
    private String ssn;
    private String firstName;
    private String lastName;

    // Constructor to initialize the employee's SSN, first name, and last name
    public Employee(String ssn, String firstName, String lastName) {
        this.ssn = ssn;
        this.firstName = firstName;
        this.lastName = lastName;
    }

    // Getter method for SSN
    public String getSSN() {
        return ssn;
    }

    // Setter method for SSN
    public void setSSN(String ssn) {
        this.ssn = ssn;
    }

    // Getter method for first name
    public String getFirstName() {
        return firstName;
    }

    // Setter method for first name
    public void setFirstName(String firstName) {
        this.firstName = firstName;
    }

    // Getter method for last name
    public String getLastName() {
        return lastName;
    }

    // Setter method for last name
    public void setLastName(String lastName) {
        this.lastName = lastName;
    }

    // Method to print the employee's information
    public void print() {
        // Print SSN
        System.out.println("SSN: " + ssn);
        // Print first name
        System.out.println("First Name: " + firstName);
```

```
        // Print last name
        System.out.println("Last Name: " + lastName);
    }
}

// Subclass: HourlyEmployee extends Employee
public class HourlyEmployee extends Employee {
    // Private field to store the hourly salary
    private double salary;

    // Constructor to initialize HourlyEmployee with ssn, firstName, lastName, and salary
    public HourlyEmployee(String ssn, String firstName, String lastName, double salary) {
        // Call the constructor of the superclass Employee
        super(ssn, firstName, lastName);
        // Initialize the salary field
        this.salary = salary;
    }

    // Getter method for salary
    public double getSalary() {
        return salary;
    }

    // Setter method for salary
    public void setSalary(double salary) {
        this.salary = salary;
    }

    // Method to compute the gross pay based on hours worked
    public double computeGrossPay(int hoursWorked) {
        return salary * hoursWorked;
    }

    // Method to compute the net pay (in this example, it's the same as gross pay)
    public double computeNetPay(int hoursWorked) {
        return computeGrossPay(hoursWorked);
    }

    // Override the print method from the Employee class to include salary
    @Override
    public void print() {
        // Call the print method of the superclass to print SSN, first name, and last name
        super.print();
        // Print the salary
        System.out.println("Salary: " + salary);
    }
}
```

```
// Subclass: SalariedEmployee extends Employee
public class SalariedEmployee extends Employee {
    // Private field to store the fixed salary
    private double salary;

    // Constructor to initialize SalariedEmployee with ssn, firstName, lastName, and
salary
    public SalariedEmployee(String ssn, String firstName, String lastName, double salary)
{
        // Call the constructor of the superclass Employee
        super(ssn, firstName, lastName);
        // Initialize the salary field
        this.salary = salary;
    }

    // Getter method for salary
    public double getSalary() {
        return salary;
    }

    // Setter method for salary
    public void setSalary(double salary) {
        this.salary = salary;
    }

    // Method to compute gross pay (for simplicity, assuming fixed salary)
    public double computeGrossPay() {
        return salary;
    }

    // Method to compute net pay (in this example, it's the same as gross pay)
    public double computeNetPay() {
        return computeGrossPay();
    }

    // Override the print method from the Employee class to include salary
    @Override
    public void print() {
        // Call the print method of the superclass to print SSN, first name, and last name
        super.print();
        // Print the salary
        System.out.println("Salary: " + salary);
    }
}



public class EmployeeMain {
    public static void main(String[] args) {
        // Create an instance of HourlyEmployee with SSN, first name, last name, and
hourly salary
        HourlyEmployee hourlyEmployee = new HourlyEmployee("12345", "John", "Doe", 20.0);
```

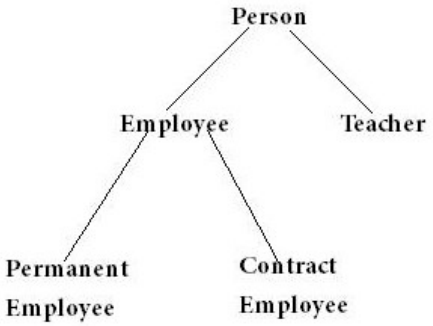
```
        // Create an instance of SalariedEmployee with SSN, first name, last name, and
fixed salary
        SalariedEmployee salariedEmployee = new SalariedEmployee("98765", "Jane", "Smith",
5000.0);

        // Display details of the HourlyEmployee
        System.out.println("Hourly Employee Details:");
        hourlyEmployee.print(); // Call print method to display employee details
        System.out.println("Gross Pay: " + hourlyEmployee.computeGrossPay(40)); // Compute
and display gross pay for 40 hours
        System.out.println("Net Pay: " + hourlyEmployee.computeNetPay(40)); // Compute and
display net pay for 40 hours

        // Display details of the SalariedEmployee
        System.out.println("\nSalaried Employee Details:");
        salariedEmployee.print(); // Call print method to display employee details
        System.out.println("Gross Pay: " + salariedEmployee.computeGrossPay()); // Compute
and display gross pay
        System.out.println("Net Pay: " + salariedEmployee.computeNetPay()); // Compute and
display net pay
    }
}
```

Output



3. Write java program to implement Inheritance with following example:

Person will have name and age as data members. Teacher and employee will inherit data members in the super class and create its own method myProfession() to display their profession. Then create objects of Teacher, Permanent, and Contract employee to display their profession..

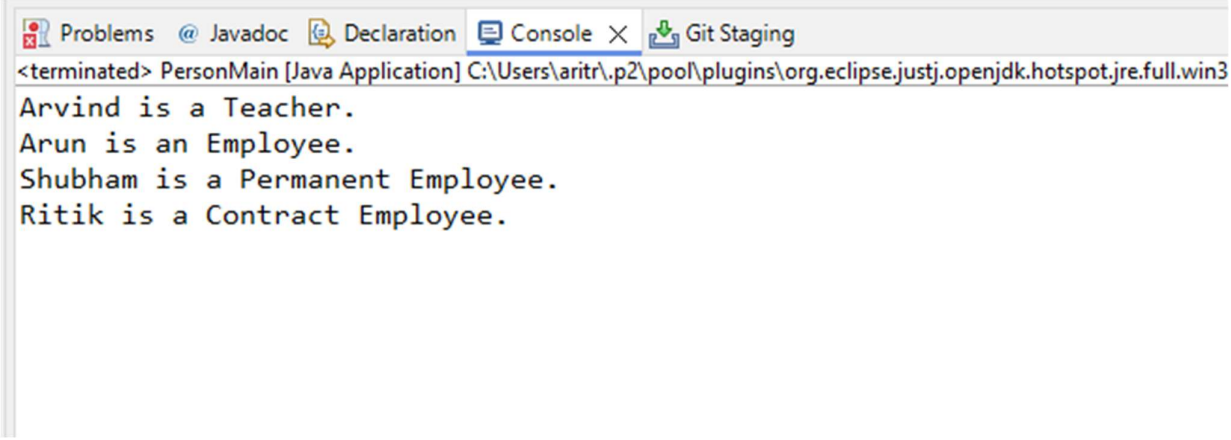



```java
public class Person {
    // Attributes of the Person class
    String name;
    int age;

    // Constructor to initialize the Person object
    public Person(String name, int age) {
        this.name = name; // Set the name of the person
        this.age = age;   // Set the age of the person
    }

    // Method to display the details of the Person
    public void display() {
        System.out.println("Name Of Person: " + name); // Print the person's name
        System.out.println("Age Of Person: " + age);   // Print the person's age
    }
}



//Extending the Person class.
public class Employeee extends Person {

    //Constructor for Employeee objects.
    public Employeee(String name, int age) {
        super(name, age);
    }


    //Prints the profession of the employee.
    public void myProfession() {
        System.out.println(name + " is an Employee.");
    }
}
```

```
//Extending the Person class.
public class Teacher extends Person {

    // Constructor for Teacher objects.
    public Teacher(String name, int age) {
        super(name, age);
    }

    //Prints the profession of the teacher.
    public void myProfession() {
        System.out.println(name + " is a Teacher.");
    }

}

//Extending the Employeee class.
public class PermanentEmployee extends Employeee {


    //Constructor for PermanentEmployee objects.
    public PermanentEmployee(String name, int age) {
        super(name, age);
    }


    //Prints the profession of the permanent employee.
    public void myProfession() {
        System.out.println(name + " is a Permanent Employee.");
    }

}

//Extending the Employeee class.
public class ContractEmployee extends Employeee {


    //Constructor for ContractEmployee objects.
    public ContractEmployee(String name, int age) {
        super(name, age);
    }


    //Prints the profession of the contract employee.
    public void myProfession() {
        System.out.println(name + " is a Contract Employee.");
    }

}

public class PersonMain {

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

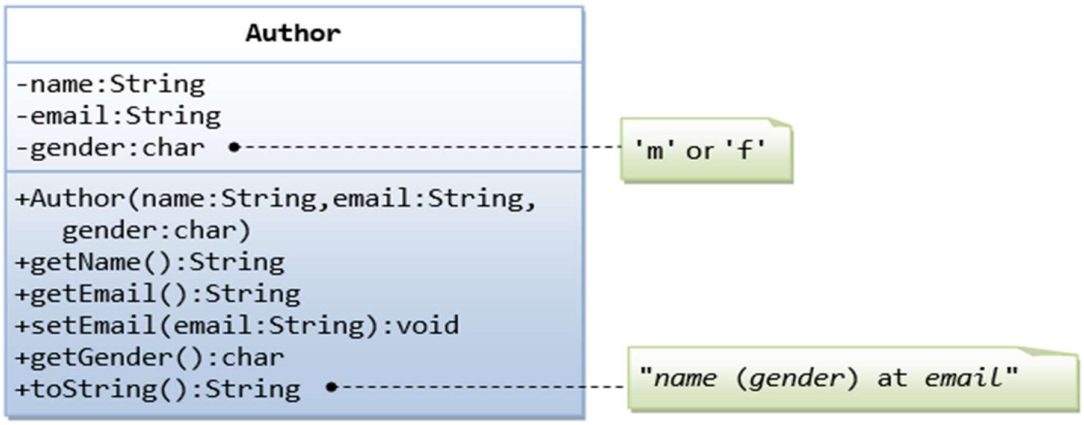
```
            // Creating Objects
            Teacher teacher = new Teacher("Arvind", 35);
            Employeee emp = new Employeee("Arun", 37);
            PermanentEmployee permanentEmployee = new
PermanentEmployee("Shubham", 45);
            ContractEmployee contractEmployee = new ContractEmployee("Ritik",
34);

            // Calling the methods
            teacher.myProfession();
            emp.myProfession();
            permanentEmployee.myProfession();
            contractEmployee.myProfession();

      }

}
```

Output



---

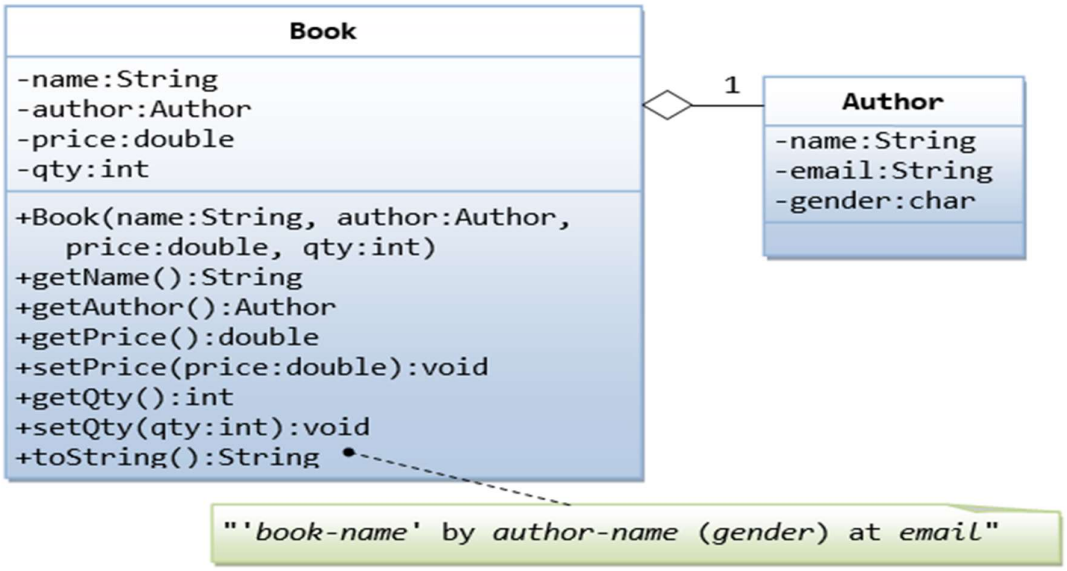
4. Inheritance Using has a Relationship



A class called `Author` is designed as shown in the class diagram. It contains:

- Three `private` member variables: `name` (`String`), `email` (`String`), and `gender` (`char` of either `'m'` or `'f'` - you might also use a `boolean` variable called `isMale` having value of `true` or `false`).
- A constructor to initialize the `name`, `email` and `gender` with the given values. (There is no *default constructor*, as there is no default value for `name`, `email` and `gender`.)
- Public getters/setters: `getName()`, `getEmail()`, `setEmail()`, and `getGender()`. (There are no setters for `name` and `gender`, as these properties are not designed to be changed.)

A `Book` is written by one `Author` - Using an "Object" Member Variable



Let's design a `Book` class. Assume that a book is written by one (and exactly one) author. The Book class (as shown in the class diagram) contains the following members:

- Four `private` member variables: `name` (`String`), `author` (an *instance* of the `Author` class we have just created, assuming that each book has exactly one author), `price` (`double`), and `qty` (`int`).
- The `public` getters and setters: `getName()`, `getAuthor()`, `getPrice()`, `setPrice()`, `getQty()`, `setQty()`.

```
// Author class representing an author of a book
public class Author {
    // Private member variables to store author's information
    private String name;
    private String email;
    private char gender;

    // Constructor to initialize the name, email, and gender of the author
    public Author(String name, String email, char gender) {
        this.name = name;
        this.email = email;
        this.gender = gender;
    }

    // Getter method to retrieve the name of the author
    public String getName() {
        return name;
    }
```

```
    // Getter method to retrieve the email of the author
    public String getEmail() {
        return email;
    }

    // Getter method to retrieve the gender of the author
    public char getGender() {
        return gender;
    }
}


// Book class representing a book written by an author
public class Book {
    // Private member variables to store book information
    private String name;
    private Author author;
    private double price;
    private int qty;

    // Constructor to initialize the name, author, price, and quantity of the book
    public Book(String name, Author author, double price, int qty) {
        this.name = name;
        this.author = author;
        this.price = price;
        this.qty = qty;
    }

    // Getter method to retrieve the name of the book
    public String getName() {
        return name;
    }

    // Getter method to retrieve the author of the book
    public Author getAuthor() {
        return author;
    }

    // Getter method to retrieve the price of the book
    public double getPrice() {
        return price;
    }

    // Setter method to set the price of the book
    public void setPrice(double price) {
        this.price = price;
    }

    // Getter method to retrieve the quantity of the book
    public int getQty() {
        return qty;
```

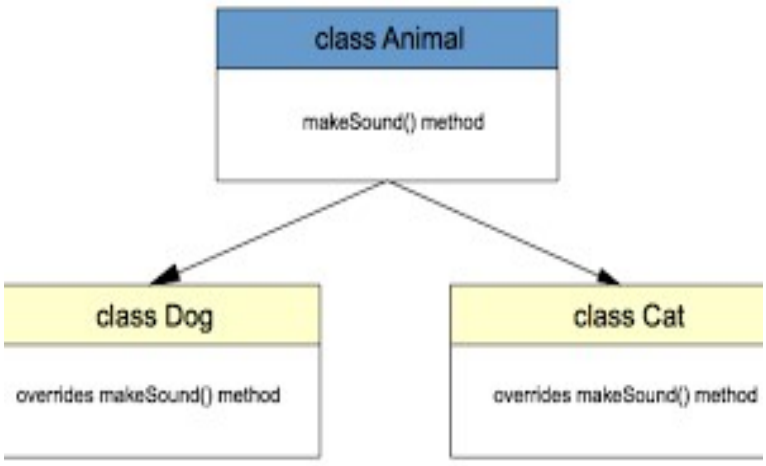
```
    }

    // Setter method to set the quantity of the book
    public void setQty(int qty) {
        this.qty = qty;
    }
}



public class AuthorMain {
    public static void main(String[] args) {
        // Create an author
        Author author = new Author("John Doe", "john@example.com", 'm');

        // Create a book
        Book book = new Book("Java Programming", author, 560, 100);

        // Display book information
        System.out.println("Book Name: " + book.getName());
        System.out.println("Author Name: " + book.getAuthor().getName());
        System.out.println("Author Email: " + book.getAuthor().getEmail());
        System.out.println("Author Gender: " + book.getAuthor().getGender());
        System.out.println("Price: Rs" + book.getPrice());
        System.out.println("Quantity Available: " + book.getQty());

        // Update book price and quantity
        book.setPrice(630);
        book.setQty(50);

        // Display updated book information
        System.out.println("\nUpdated Price: Rs" + book.getPrice());
        System.out.println("Updated Quantity Available: " + book.getQty());
    }
}
```

Output

```
Problems  Javadoc  Declaration  Console  Git Staging
<terminated> AuthorMain [Java Application] C:\Users\aritr\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.
Book Name: Java Programming
Author Name: John Doe
Author Email: john@example.com
Author Gender: m
Price: Rs560.0
Quantity Available: 100

Updated Price: Rs630.0
Updated Quantity Available: 50
```

5. Write java program to implement Polymorphism with following example

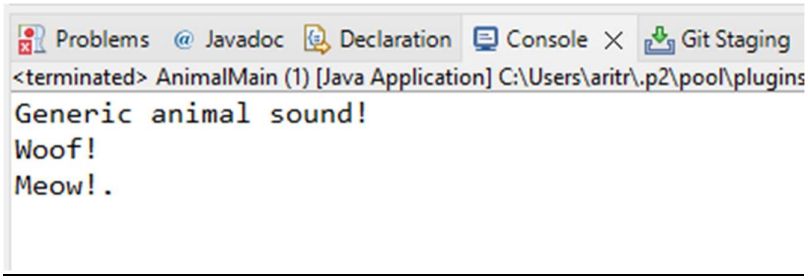
a)



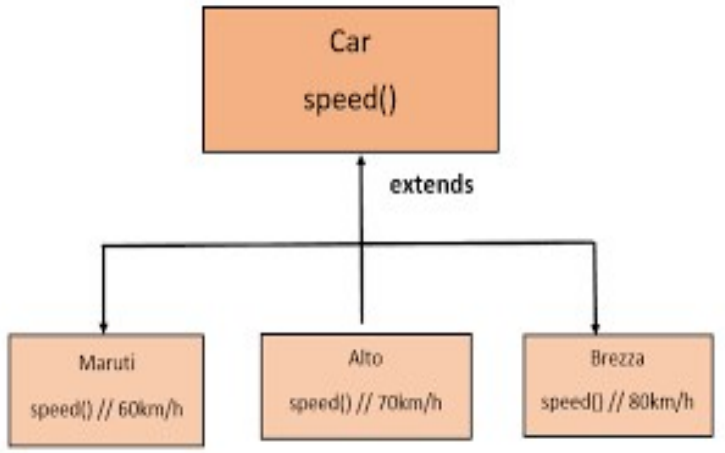

```java
// Animal class representing a generic animal
public class Animal {

    // Method to make a generic animal sound
    public void makeSound() {
        System.out.println("Generic animal sound!");
    }
}

// Dog class representing a specific type of animal - a dog
public class Dog extends Animal {
```

```
    // Override the makeSound() method from the Animal class
    @Override
    public void makeSound() {
        System.out.println("Woof!");
    }
}

public class AnimalMain {

    public static void main(String[] args) {
        // Create an instance of Animal
        Animal animals = new Animal();

        // Create instances of Dog and Cat
        Dog d = new Dog();
        Cat c = new Cat();

        // Make sounds of the animals
        System.out.println("Sound of generic animal:");
        animals.makeSound(); // Invoke makeSound() of Animal class
        System.out.println("\nSound of a dog:");
        d.makeSound(); // Invoke overridden makeSound() of Dog class
        System.out.println("\nSound of a cat:");
        c.makeSound(); // Invoke overridden makeSound() of Cat class
    }
}
```

Output



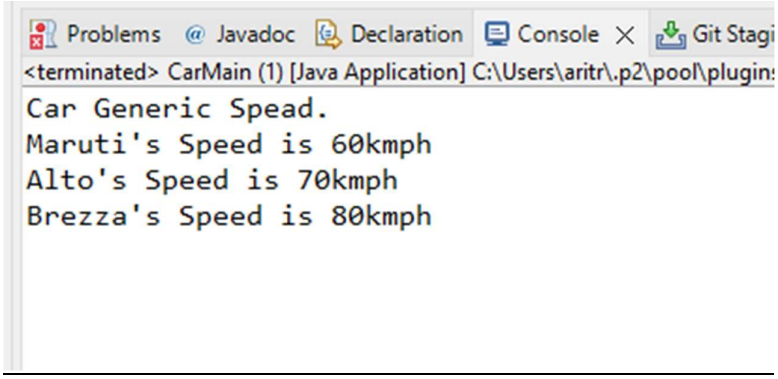
b)




```java
// Car class representing a generic car
public class Car {

    // Method to represent a generic speed of a car
    public void speed() {
        System.out.println("Car Generic Speed.");
    }
}

// Maruti class representing a specific type of car - a Maruti
public class Maruti extends Car {

    // Override the speed() method from the Car class
    @Override
    public void speed() {
        System.out.println("Maruti's Speed is 60 km/h.");
    }
}

// Alto class representing a specific type of car - an Alto
public class Alto extends Car {

    // Override the speed() method from the Car class
    @Override
    public void speed() {
        System.out.println("Alto's Speed is 70 km/h.");
    }
}

// Brezza class representing a specific type of car - a Brezza
public class Brezza extends Car {

    // Override the speed() method from the Car class
    @Override
    public void speed() {
        System.out.println("Brezza's Speed is 80 km/h.");
    }
```

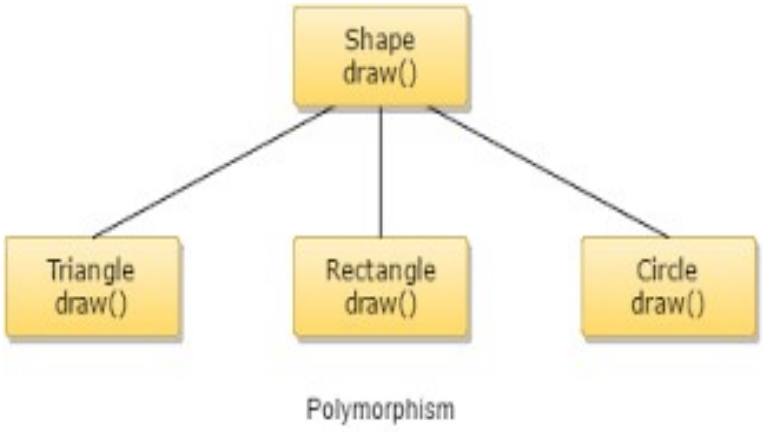
```
}

// Main class to test polymorphism with Car, Maruti, Alto, and Brezza classes
public class CarMain {

    public static void main(String[] args) {
        // Create an instance of Car
        Car c = new Car();

        // Create instances of Maruti, Alto, and Brezza
        Maruti m = new Maruti();
        Alto a = new Alto();
        Brezza b = new Brezza();

        // Invoke the speed() method of each object
        System.out.println("Generic Car Speed:");
        c.speed(); // Invoke speed() of Car class
        System.out.println("\nMaruti's Speed:");
        m.speed(); // Invoke overridden speed() of Maruti class
        System.out.println("\nAlto's Speed:");
        a.speed(); // Invoke overridden speed() of Alto class
        System.out.println("\nBrezza's Speed:");
        b.speed(); // Invoke overridden speed() of Brezza class
    }

}
```

Output



---

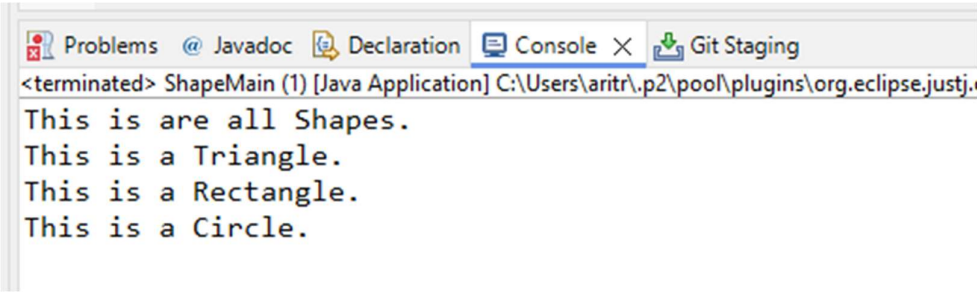
c)



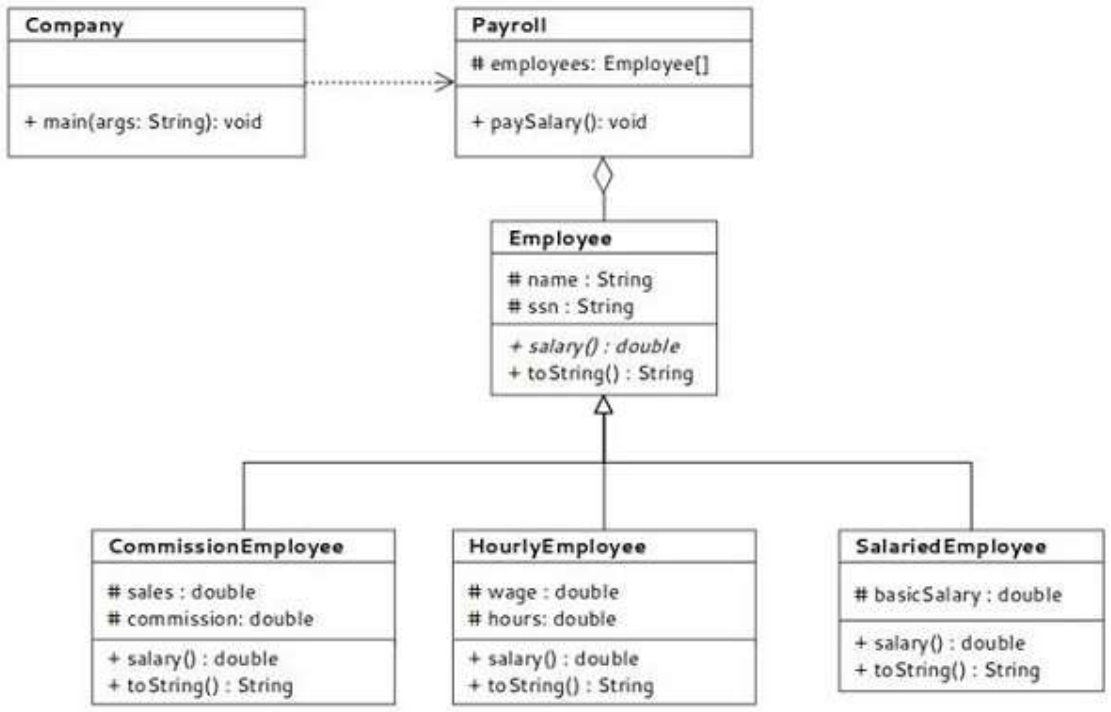
Polymorphism

```
// Shape class representing a generic shape
public class Shape {

    // Method to represent drawing a generic shape
    public void draw() {
        System.out.println("This is a generic shape.");
    }
}

// Triangle class representing a specific type of shape - a triangle
public class Triangle extends Shape {

    // Override the draw() method from the Shape class
    @Override
    public void draw() {
        System.out.println("This is a triangle.");
    }
}

// Rectangle class representing a specific type of shape - a rectangle
public class Rectangle extends Shape {

    // Override the draw() method from the Shape class
    @Override
    public void draw() {
        System.out.println("This is a rectangle.");
    }
}

// Circle class representing a specific type of shape - a circle
public class Circle extends Shape {

    // Override the draw() method from the Shape class
    @Override
    public void draw() {
        System.out.println("This is a circle.");
    }
}
```

```java
// Main class to test polymorphism with Shape, Triangle, Rectangle, and Circle classes
public class ShapeMain extends Shape {

    public static void main(String[] args) {
        // Create instances of Shape, Triangle, Rectangle, and Circle
        Shape s = new Shape();
        Triangle t = new Triangle();
        Rectangle r = new Rectangle();
        Circle c = new Circle();

        // Invoke the draw() method of each object
        System.out.println("Drawing a generic shape:");
        s.draw(); // Invoke draw() of Shape class
        System.out.println("\nDrawing a triangle:");
        t.draw(); // Invoke overridden draw() of Triangle class
        System.out.println("\nDrawing a rectangle:");
        r.draw(); // Invoke overridden draw() of Rectangle class
        System.out.println("\nDrawing a circle:");
        c.draw(); // Invoke overridden draw() of Circle class
    }

}
```



d)



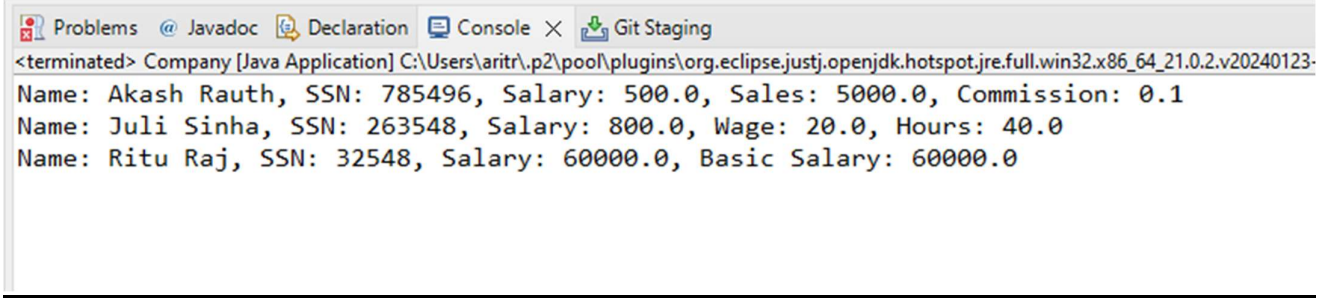

```
public class Employee1 {
    // Protected attributes for name and SSN, accessible within the class and
    // subclasses
    protected String name;
    protected String ssn;

    // Constructor to initialize the Employee1 object
    public Employee1(String name, String ssn) {
        this.name = name; // Set the name of the employee
        this.ssn = ssn; // Set the SSN of the employee
    }

    // Method to return the salary of the employee
    public double salary() {
        return 20000.00; // Default salary
    }

    // Override the toString method to provide a string representation of the
    // Employee1 object
    @Override
    public String toString() {
        return "Name: " + name + ", SSN: " + ssn + ", Salary: " + salary();
    }
}

public class CommissionEmployee extends Employee1 {
    // Private attributes for sales and commission rate
    private double sales;
```

```
    private double commission;

    // Constructor to initialize the CommissionEmployee object
    public CommissionEmployee(String name, String ssn, double sales, double commission) {
        super(name, ssn); // Call the constructor of the superclass Employee1
        this.sales = sales; // Set the sales amount
        this.commission = commission; // Set the commission rate
    }

    // Override the salary method to calculate the salary based on sales and
    // commission
    @Override
    public double salary() {
        return sales * commission; // Calculate salary as sales multiplied by commission
rate
    }

    // Override the toString method to provide a string representation of the
    // CommissionEmployee object
    @Override
    public String toString() {
        return super.toString() + ", Sales: " + sales + ", Commission: " + commission;
    }
}

public class HourlyEmployee1 extends Employee1 {
    // Private attributes for wage and hours worked
    private double wage;
    private double hours;

    // Constructor to initialize the HourlyEmployee1 object
    public HourlyEmployee1(String name, String ssn, double wage, double hours) {
        super(name, ssn); // Call the constructor of the superclass Employee1
        this.wage = wage; // Set the hourly wage
        this.hours = hours; // Set the number of hours worked
    }

    // Override the salary method to calculate the salary based on wage and hours worked
    @Override
    public double salary() {
        return wage * hours; // Calculate salary as wage multiplied by hours worked
    }

    // Override the toString method to provide a string representation of the
HourlyEmployee1 object
    @Override
    public String toString() {
        return super.toString() + ", Wage: " + wage + ", Hours: " + hours;
    }
}

public class SalariedEmployee1 extends Employee1 {
```

```
    // Private attribute for basic salary
    private double basicSalary;

    // Constructor to initialize the SalariedEmployee1 object
    public SalariedEmployee1(String name, String ssn, double basicSalary) {
        super(name, ssn); // Call the constructor of the superclass Employee1
        this.basicSalary = basicSalary; // Set the basic salary
    }

    // Override the salary method to return the basic salary
    @Override
    public double salary() {
        return basicSalary; // Return the basic salary
    }

    // Override the toString method to provide a string representation of the
SalariedEmployee1 object
    @Override
    public String toString() {
        return super.toString() + ", Basic Salary: " + basicSalary;
    }
}

import java.util.ArrayList;

public class Payroll {
    // Private attribute to store a list of employees
    private ArrayList<Employee1> employees;

    // Constructor to initialize the Payroll object
    public Payroll() {
        this.employees = new ArrayList<>(); // Initialize the ArrayList of employees
    }

    // Method to add an employee to the payroll
    public void addEmployee(Employee1 employee) {
        employees.add(employee); // Add the provided employee to the list
    }

    // Method to pay salary to all employees and print their details
    public void paySalary() {
        for (Employee1 employee : employees) { // Iterate through each employee in the
list
            System.out.println(employee); // Print the details of the employee, including
the computed salary
        }
    }
}

public class Company {
    public static void main(String[] args) {
        // Create a Payroll object to manage employee salaries
```

```
        Payroll payroll = new Payroll();

        // Add different types of employees to the payroll
        payroll.addEmployee(new CommissionEmployee("Akash Rauth", "785496", 5000, 0.10));
        payroll.addEmployee(new HourlyEmployee1("Juli Sinha", "263548", 20, 40));
        payroll.addEmployee(new SalariedEmployee1("Ritu Raj", "32548", 60000));

        // Pay salary to all employees and print their details
        payroll.paySalary();
    }
}
```

Output



---

6. Write java program to implement interface with following example

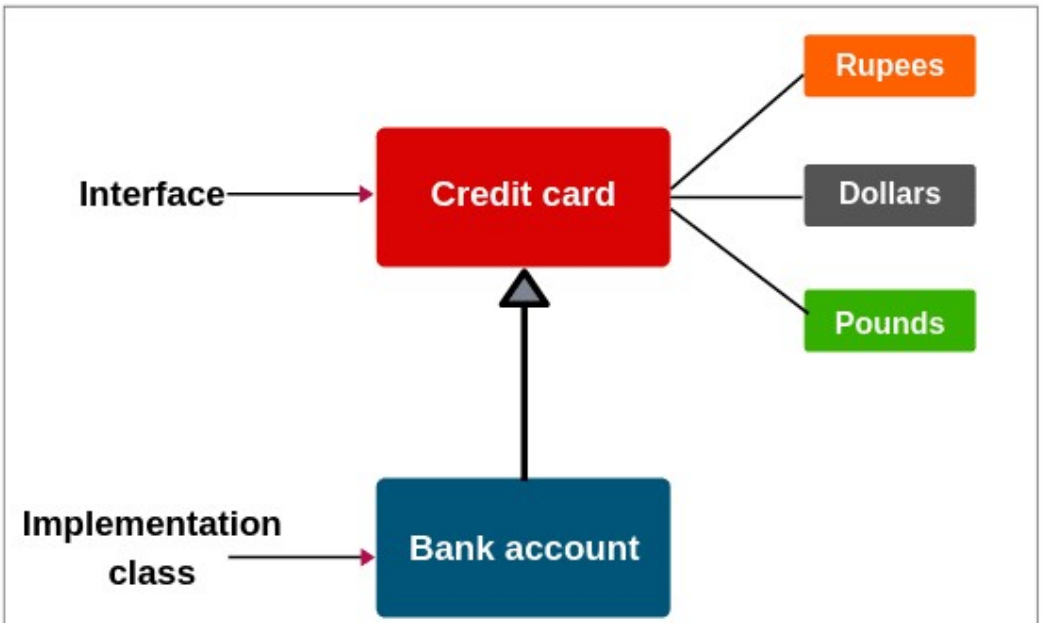
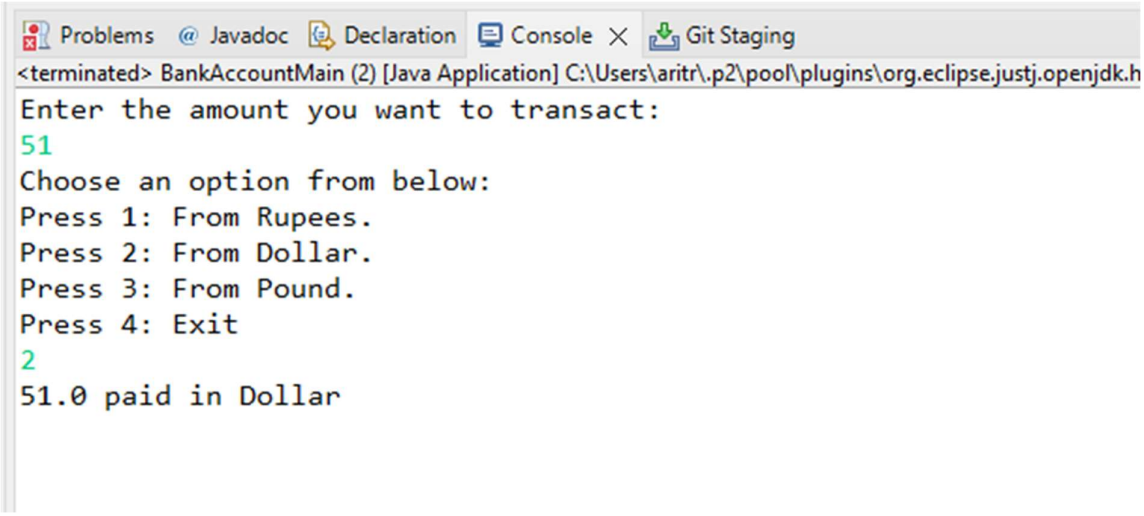


Fig: Interface and Implementation class

```java
// Interface representing a credit card with methods for making payments in different currencies
public interface CreditCard {
    // Method to make payment in Rupees
    void rupees(float amount);

    // Method to make payment in Dollars
    void dollar(float amount);

    // Method to make payment in Pounds
    void pound(float amount);
}

// Class implementing the CreditCard interface
class Bank implements CreditCard {

    // Method implementation for making payment in Rupees
    @Override
    public void rupees(float amount) {
        System.out.println(amount + " paid in Rupees");
    }

    // Method implementation for making payment in Dollars
    @Override
    public void dollar(float amount) {
        System.out.println(amount + " paid in Dollar");
    }

    // Method implementation for making payment in Pounds
    @Override
    public void pound(float amount) {
        System.out.println(amount + " paid in Pound");
    }
}

import java.util.Scanner;

// Main class to facilitate transactions using Bank class
public class BankAccountMain {

    public static void main(String[] args) {
        Scanner sc = new Scanner(System.in);

        // Prompt user to enter the amount to transact
        System.out.println("Enter the amount you want to transact: ");
        float amount = sc.nextFloat();

        // Display options for currency selection
        System.out.println("Choose an option from below: ");
        System.out.println("Press 1: From Rupees. ");
        System.out.println("Press 2: From Dollar. ");
        System.out.println("Press 3: From Pound. ");
```

```
        System.out.println("Press 4: Exit");

        // Create a reference variable of type CreditCard
        CreditCard c;

        // Read user's choice
        int choice = sc.nextInt();

        switch (choice) {
            case 1:
                c = new Bank();
                c.rupees(amount); // Make payment in Rupees
                break;
            case 2:
                c = new Bank();
                c.dollar(amount); // Make payment in Dollars
                break;
            case 3:
                c = new Bank();
                c.pound(amount); // Make payment in Pounds
                break;
            case 4:
                System.out.println("Thanks for using.");
                break;
            default:
                System.out.println("Invalid Option! Try Again");
        }

        sc.close(); // Close the scanner
    }

}
```

Output



Packages (Assignment 3)

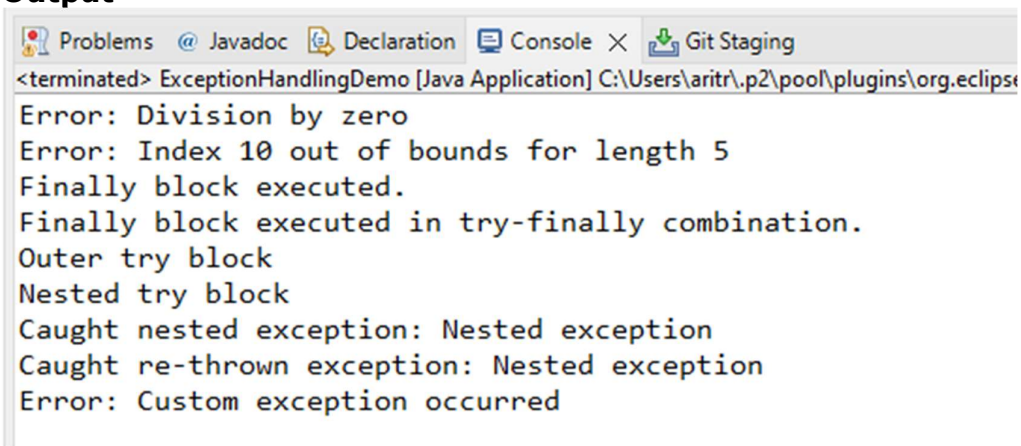
Exception

1. Write a program to demonstrate the use of try, catch, finally throw and throws keywords and demonstrate the following points in the program.
   a) Multiple catch blocks.
   b) try-catch-finally combination.
   c) try-finally combination.
   d) Nested try blocks

```
public class ExceptionHandlingDemo {
    public static void main(String[] args) {
        // (a) Multiple catch blocks
        try {
            int result = divide(10, 0);
            System.out.println("Result: " + result);
        } catch (ArithmeticException e) {
            System.out.println("Error: " + e.getMessage());
        } catch (Exception e) {
            System.out.println("General Exception: " + e.getMessage());
        }

        // (b) try-catch-finally combination
        try {
            int[] arr = new int[5];
            System.out.println("Accessing out of bounds element: " + arr[10]);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println("Error: " + e.getMessage());
        } finally {
            System.out.println("Finally block executed.");
        }

        // (c) try-finally combination
        int result;
        try {
            result = divide(15, 3);
        } finally {
            System.out.println("Finally block executed in try-finally combination.");
        }

        // (d) Nested try blocks
        try {
            System.out.println("Outer try block");
            try {
                System.out.println("Nested try block");
                throw new RuntimeException("Nested exception");
            } catch (RuntimeException e) {
                System.out.println("Caught nested exception: " + e.getMessage());
```

```
                throw e; // Re-throwing the exception
            }
        } catch (Exception e) {
            System.out.println("Caught re-thrown exception: " + e.getMessage());
        }

        // Demonstrating throws keyword
        try {
            performOperation();
        } catch (CustomException e) {
            System.out.println("Error: " + e.getMessage());
        }
    }

    private static int divide(int a, int b) {
        if (b == 0) {
            throw new ArithmeticException("Division by zero");
        }
        return a / b;
    }

    private static void performOperation() throws CustomException {
        throw new CustomException("Custom exception occurred");
    }
}

class CustomException extends Exception {
    public CustomException(String message) {
        super(message);
    }
}
```

Output

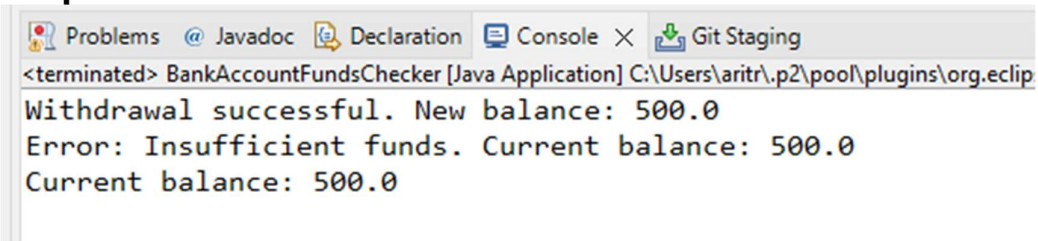


2. Write a java application for checking the bank account funds using InsufficientFundsException using user defined exception using throw keword and create a class for CheckingAccount with methods withdraw() and getBalance() Among these withdraw() should throw an InsufficientFundsException.

```
// Custom exception class for handling insufficient funds
class InsufficientFundsException extends Exception {
    public InsufficientFundsException(String message) {
        super(message);
    }
}

// CheckingAccount class representing a bank account
class CheckingAccount {
    private double balance; // The current balance of the account

    // Constructor to initialize the account with an initial balance
    public CheckingAccount(double initialBalance) {
        this.balance = initialBalance;
    }

    // Method to deposit funds into the account
    public void deposit(double amount) {
        balance += amount;
    }

    // Method to get the current balance of the account
    public double getBalance() {
        return balance;
    }

    // Method to withdraw funds from the account
    public void withdraw(double amount) throws InsufficientFundsException {
        // Check if the withdrawal amount exceeds the balance
        if (amount > balance) {
            // Throw an InsufficientFundsException if there are not enough funds
            throw new InsufficientFundsException("Insufficient funds. Current balance: " +
balance);
        }
        // Deduct the amount from the balance if sufficient funds are available
        balance -= amount;
        System.out.println("Withdrawal successful. New balance: " + balance);
    }
}

public class BankAccountFundsChecker {
    public static void main(String[] args) {
        CheckingAccount account = new CheckingAccount(1000.0); // Create a checking
account with an initial balance of $1000

        try {
```

```
            // Attempt to withdraw $500
            account.withdraw(500.0);
            // Attempt to withdraw $700 (insufficient funds)
            account.withdraw(700.0);
        } catch (InsufficientFundsException e) {
            // Catch and handle the InsufficientFundsException
            System.out.println("Error: " + e.getMessage());
        }

        // Print the current balance after withdrawal attempts
        System.out.println("Current balance: " + account.getBalance());
    }
}
```

Output



Collection

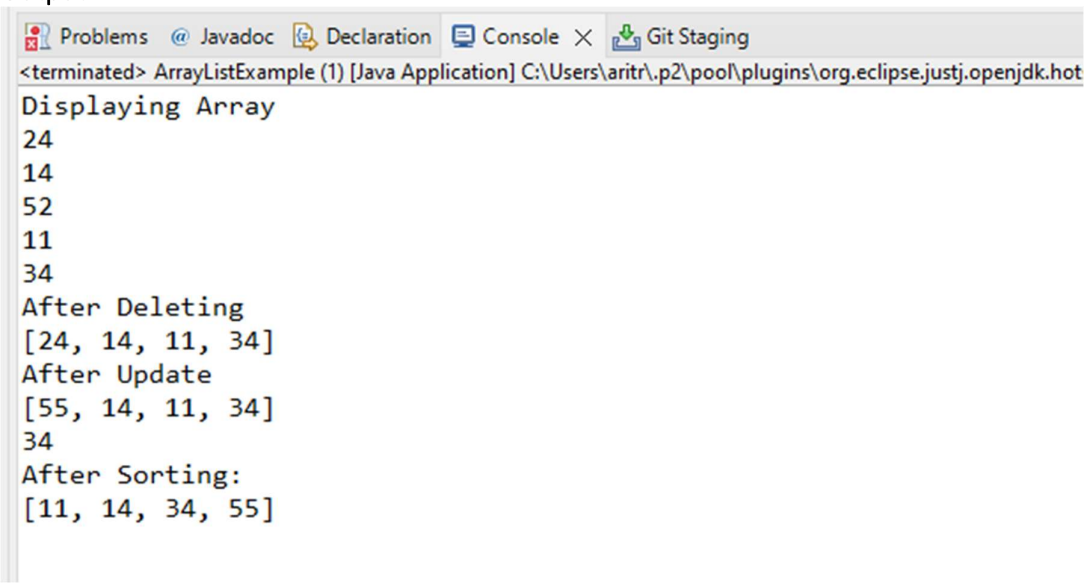
1. Write a java program to do following operations on Array List(Integer)

    a) Add element
    b) Remove a particular element
    c) Modify
    d) View All elements (Use Iterator)
    e) View a Particular element (get())
    f) Sort (Collections.sort)

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Iterator;

public class ArrayListExample {
    public static void main(String[] args) {
        // Create an ArrayList of Integer type
        ArrayList<Integer> al = new ArrayList<>();

        // Add elements to the ArrayList
        al.add(24);
        al.add(14);
        al.add(52);
        al.add(11);
        al.add(34);

        // Display the ArrayList using an iterator
```

```
        System.out.println("Displaying Array");
        Iterator<Integer> it = al.iterator();
        while (it.hasNext()) {
            System.out.println(it.next());
        }

        // Remove an element from the ArrayList
        System.out.println("After Deleting");
        int pos = al.indexOf(52); // Find the position of the element to be removed
        al.remove(pos); // Remove the element at the found position
        System.out.println(al);

        // Update the first element of the ArrayList
        System.out.println("After Update");
        al.set(0, 55);
        System.out.println(al);

        // Retrieve an element from the ArrayList
        int a = al.get(3); // Get the element at index 3
        System.out.println(a);

        // Sort the ArrayList
        System.out.println("After Sorting: ");
        Collections.sort(al); // Sort the ArrayList in ascending order
        System.out.println(al);
    }
}
```

Output



2. Write a Java menu driven program to perform the following operations-using Array List Create a class called Player with the following instance variable

Player name
total run
no of wicket
no of matches
country
no-of-century
Category = null

* Create 5 Players add into the Array List
* Display the player details who has scored maximum no of runs
* Sort the players by name
* Remove the player who has scored less than 100 runs
* Update Category 'A' if player has got 10 century else update Grade 'B'

```
public class Player {
    private String playerName; // Name of the player
    private int totalRuns; // Total runs scored by the player
    private int noOfWickets; // Number of wickets taken by the player
    private int noOfMatches; // Number of matches played by the player
    private String country; // Country the player represents
    private int noOfCenturies; // Number of centuries scored by the player
    private String category; // Category of the player (A or B)

    // Constructor to initialize a Player object
    public Player(String playerName, int totalRuns, int noOfWickets, int
noOfMatches, String country, int noOfCenturies) {
        this.playerName = playerName;
        this.totalRuns = totalRuns;
        this.noOfWickets = noOfWickets;
        this.noOfMatches = noOfMatches;
        this.country = country;
        this.noOfCenturies = noOfCenturies;
        this.category = null; // Category is initially null
    }

    // Getter method to retrieve the player's name
    public String getPlayerName() {
        return playerName;
    }

    // Getter method to retrieve the total runs scored by the player
    public int getTotalRuns() {
        return totalRuns;
    }
```

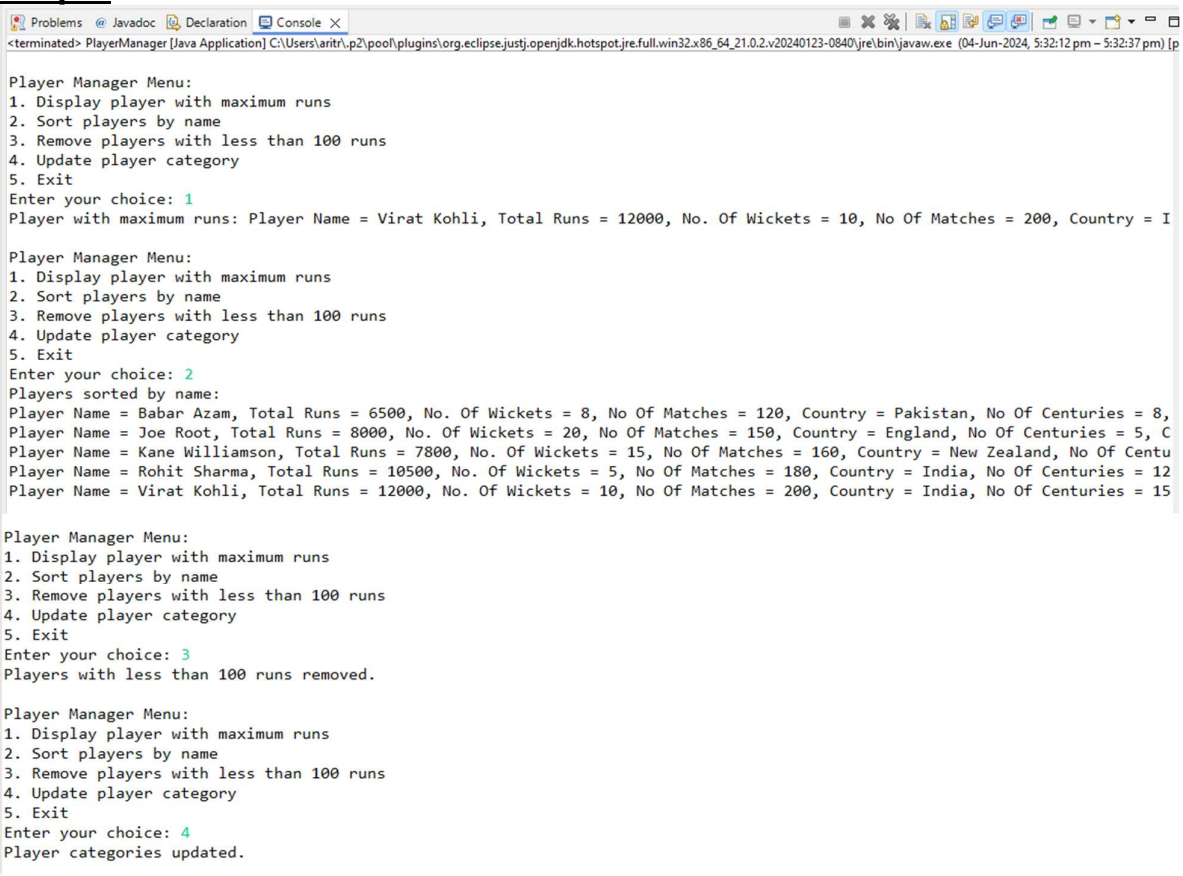
```
    // Method to update the category of the player based on the number of
centuries
    public void updateCategory() {
        if (noOfCenturies >= 10) {
            category = "A"; // Category A for players with 10 or more centuries
        } else {
            category = "B"; // Category B for players with less than 10
centuries
        }
    }

    // Override the toString method to provide a string representation of the
player
    @Override
    public String toString() {
        return "Player Name = " + playerName + ", Total Runs = " + totalRuns +
", No. Of Wickets = " + noOfWickets
                + ", No Of Matches = " + noOfMatches + ", Country = " + country
+ ", No Of Centuries = " + noOfCenturies
                + ", Category = " + category;
    }
}
```

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;
import java.util.Scanner;

public class PlayerManager {
    private static ArrayList<Player> players = new ArrayList<>(); // List to store players
    private static Scanner scanner = new Scanner(System.in); // Scanner for user input

    public static void main(String[] args) {
        createPlayers(); // Create initial list of players
        int choice;
        do {
            displayMenu(); // Display the menu
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character
            handleChoice(choice); // Handle user choice
        } while (choice != 5); // Exit if choice is 5
    }

    // Method to create initial list of players
    private static void createPlayers() {
        Player player1 = new Player("Virat Kohli", 12000, 10, 200, "India", 15);
        Player player2 = new Player("Rohit Sharma", 10500, 5, 180, "India", 12);
        Player player3 = new Player("Joe Root", 8000, 20, 150, "England", 5);
        Player player4 = new Player("Babar Azam", 6500, 8, 120, "Pakistan", 8);
        Player player5 = new Player("Kane Williamson", 7800, 15, 160, "New Zealand", 9);
```

```
        players.add(player1);
        players.add(player2);
        players.add(player3);
        players.add(player4);
        players.add(player5);
    }

    // Method to display the menu
    private static void displayMenu() {
        System.out.println("\nPlayer Manager Menu:");
        System.out.println("1. Display player with maximum runs");
        System.out.println("2. Sort players by name");
        System.out.println("3. Remove players with less than 100 runs");
        System.out.println("4. Update player category");
        System.out.println("5. Exit");
        System.out.print("Enter your choice: ");
    }

    // Method to handle user choices
    private static void handleChoice(int choice) {
        switch (choice) {
            case 1:
                displayPlayerWithMaximumRuns(); // Display player with max runs
                break;
            case 2:
                sortPlayersByName(); // Sort players by name
                break;
            case 3:
                removePlayersWithLessThan100Runs(); // Remove players with < 100 runs
                break;
            case 4:
                updatePlayerCategory(); // Update player categories
                break;
            case 5:
                System.out.println("Exiting program...");
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }

    // Method to display the player with maximum runs
    private static void displayPlayerWithMaximumRuns() {
        Player playerWithMaxRuns = players.stream()
                .max(Comparator.comparingInt(Player::getTotalRuns))
                .orElse(null);

        if (playerWithMaxRuns != null) {
            System.out.println("Player with maximum runs: " + playerWithMaxRuns);
        } else {
            System.out.println("No players found.");
        }
```

```
    }

    // Method to sort players by name
    private static void sortPlayersByName() {
        players.sort(Comparator.comparing(Player::getPlayerName));
        System.out.println("Players sorted by name:");
        for (Player player : players) {
            System.out.println(player);
        }
    }

    // Method to remove players with less than 100 runs
    private static void removePlayersWithLessThan100Runs() {
        players.removeIf(player -> player.getTotalRuns() < 100);
        System.out.println("Players with less than 100 runs removed.");
    }

    // Method to update player categories
    private static void updatePlayerCategory() {
        for (Player player : players) {
            player.updateCategory();
        }
        System.out.println("Player categories updated.");
    }
}
```

Output



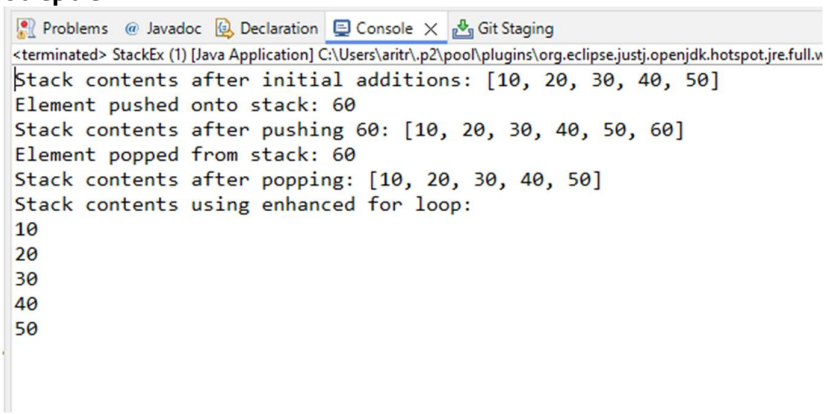
```
Problems  Javadoc  Declaration  Console
<terminated> PlayerManager [Java Application] C:\Users\aritr\.p2\pool\plugins\org.eclipse.justj.openjdk.hotspot.jre.full.win32.x86_64_21.0.2.v20240123-0840\jre\bin\javaw.exe (04-Jun-2024, 5:32:12 pm – 5:32:37 pm) [p

Player Manager Menu:
1. Display player with maximum runs
2. Sort players by name
3. Remove players with less than 100 runs
4. Update player category
5. Exit
Enter your choice: 1
Player with maximum runs: Player Name = Virat Kohli, Total Runs = 12000, No. Of Wickets = 10, No Of Matches = 200, Country = I

Player Manager Menu:
1. Display player with maximum runs
2. Sort players by name
3. Remove players with less than 100 runs
4. Update player category
5. Exit
Enter your choice: 2
Players sorted by name:
Player Name = Babar Azam, Total Runs = 6500, No. Of Wickets = 8, No Of Matches = 120, Country = Pakistan, No Of Centuries = 8,
Player Name = Joe Root, Total Runs = 8000, No. Of Wickets = 20, No Of Matches = 150, Country = England, No Of Centuries = 5, C
Player Name = Kane Williamson, Total Runs = 7800, No. Of Wickets = 15, No Of Matches = 160, Country = New Zealand, No Of Centu
Player Name = Rohit Sharma, Total Runs = 10500, No. Of Wickets = 5, No Of Matches = 180, Country = India, No Of Centuries = 12
Player Name = Virat Kohli, Total Runs = 12000, No. Of Wickets = 10, No Of Matches = 200, Country = India, No Of Centuries = 15

Player Manager Menu:
1. Display player with maximum runs
2. Sort players by name
3. Remove players with less than 100 runs
4. Update player category
5. Exit
Enter your choice: 3
Players with less than 100 runs removed.

Player Manager Menu:
1. Display player with maximum runs
2. Sort players by name
3. Remove players with less than 100 runs
4. Update player category
5. Exit
Enter your choice: 4
Player categories updated.
```

```
Player Manager Menu:
1. Display player with maximum runs
2. Sort players by name
3. Remove players with less than 100 runs
4. Update player category
5. Exit
Enter your choice: 5
Exiting program...
```

3. Write a java program to do push, pop, display objects on stacks(integer).

```java
import java.util.Stack;

public class StackEx {
    public static void main(String[] args) {
        // Create a new Stack instance of Integer type
        Stack<Integer> s = new Stack<>();

        // Add elements to the stack
        s.add(10);
        s.add(20);
        s.add(30);
        s.add(40);
        s.add(50);

        // Print the stack contents
        System.out.println("Stack contents after initial additions: " + s);

        // Push an element onto the stack and retrieve the element
        Integer ele = s.push(60);
        System.out.println("Element pushed onto stack: " + ele);
        System.out.println("Stack contents after pushing 60: " + s);

        // Pop an element from the stack and retrieve the element
        Integer ele1 = s.pop();
        System.out.println("Element popped from stack: " + ele1);
        System.out.println("Stack contents after popping: " + s);

        // Iterate through the stack using enhanced for loop
        System.out.println("Stack contents using enhanced for loop:");
        for (Integer n : s) {
            System.out.println(n);
        }
    }
}
```

Output



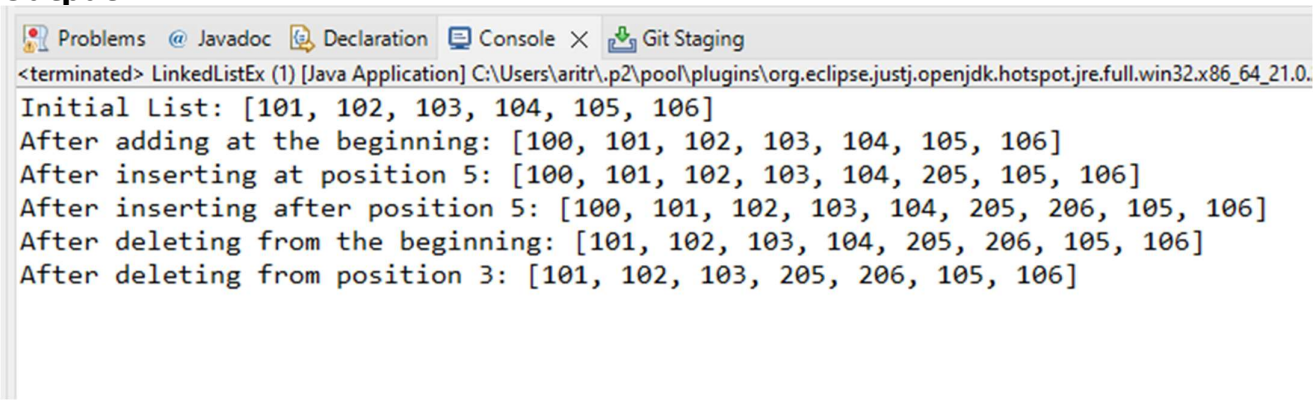
4. Implement single linked list(Integer) with following options:

    a. Insert at the beginning.
    b. Insert at given position.
    c. Insert after a given node.
    d. Delete at the beginning.
    e. Delete at given position.
    f. Delete after a given node.
    g. Reverse the linked list.

```
import java.util.LinkedList;

public class LinkedListEx {
    public static void main(String[] args) {
        // Create a LinkedList of Integer type
        LinkedList<Integer> ll = new LinkedList<>();

        // Add elements to the linked list
        ll.add(101);
        ll.add(102);
        ll.add(103);
        ll.add(104);
        ll.add(105);
        ll.add(106);

        // Display the initial linked list
        System.out.println("Initial List: " + ll);

        // Insert an element at the beginning of the linked list
        ll.addFirst(100);
        System.out.println("After adding at the beginning: " + ll);

        // Insert an element at a specific position in the linked list
        ll.add(5, 205);
        System.out.println("After inserting at position 5: " + ll);
```

```
        // Insert an element after a given node in the linked list
        ll.add(6, 206);
        System.out.println("After inserting after position 5: " + ll);

        // Remove the first element from the linked list
        ll.removeFirst();
        System.out.println("After deleting from the beginning: " + ll);

        // Remove an element at a specific position from the linked list
        ll.remove(3);
        System.out.println("After deleting from position 3: " + ll);
    }
}
```

Output



5. Write a Java program menu driven to perform the following operations using Map

* Create a class called Book with the following instance variable

    name
    price
    author name
    isbn no;
    publication

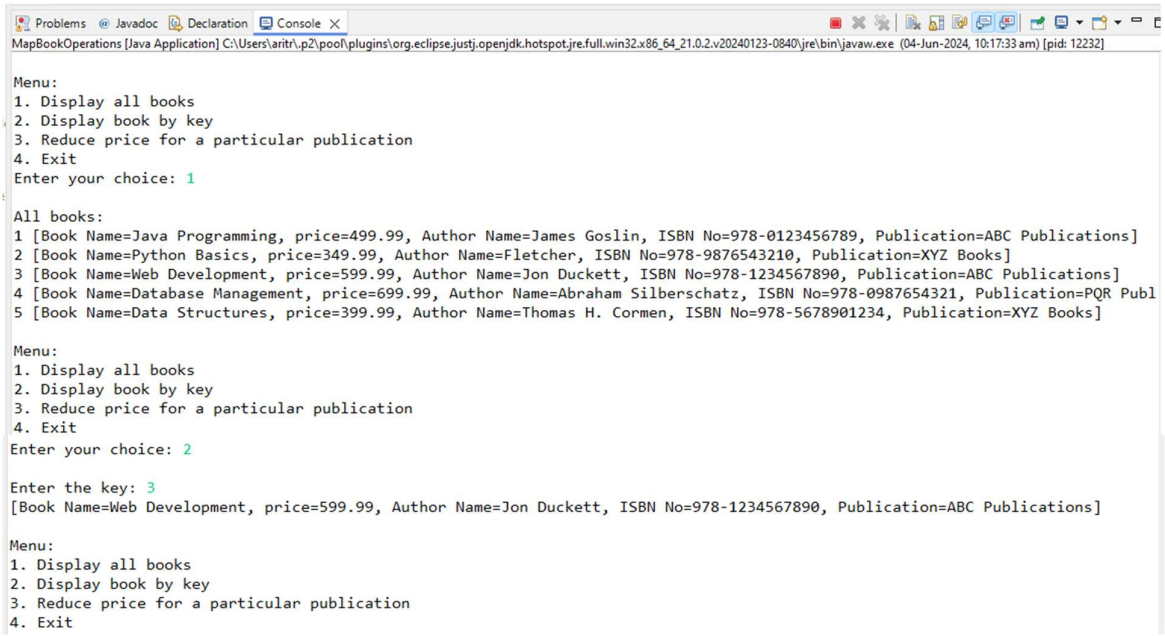
* Create 5 Objects of book class
* Add all the object in to the Map using key as integer(book id) and value as Book Object
* Display all the map object using for each
* Read any key from the user and display the particular book object based on key
* Reduce the price by 10% for a particular publication books and display all object with reduced price.

```
public class Book {
    String name; // Name of the book
    double price; // Price of the book
    String authorName; // Name of the author
    String isbnNo; // ISBN number of the book
    String publication; // Publication information

    // Constructor to initialize a Book object
    public Book(String name, double price, String authorName, String isbnNo, String
publication) {
        this.name = name;
        this.price = price;
        this.authorName = authorName;
        this.isbnNo = isbnNo;
        this.publication = publication;
    }

    // Overriding the toString method to provide a string representation of a Book object
    @Override
    public String toString() {
        return "[Book Name=" + name + ", price=" + price + ", Author Name=" + authorName +
", ISBN No=" + isbnNo
                + ", Publication=" + publication + "]";
    }
}

import java.util.HashMap;
import java.util.Map;
import java.util.Scanner;

public class MapBookOperations {
    public static void main(String[] args) {
        Map<Integer, Book> bookMap = new HashMap<>(); // Map to store books
with integer keys
        Scanner scanner = new Scanner(System.in);

        // Create 5 objects of the Book class
        Book book1 = new Book("Java Programming", 499.99, "James Goslin", "978-
0123456789", "ABC Publications");
        Book book2 = new Book("Python Basics", 349.99, "Fletcher", "978-
9876543210", "XYZ Books");
        Book book3 = new Book("Web Development", 599.99, "Jon Duckett", "978-
1234567890", "ABC Publications");
        Book book4 = new Book("Database Management", 699.99, "Abraham
Silberschatz", "978-0987654321", "PQR Publishing");
        Book book5 = new Book("Data Structures", 399.99, "Thomas H. Cormen",
"978-5678901234", "XYZ Books");

        // Add all the objects to the Map with integer keys
        bookMap.put(1, book1);
        bookMap.put(2, book2);
        bookMap.put(3, book3);
```

```
        bookMap.put(4, book4);
        bookMap.put(5, book5);

        int choice;
        do {
            displayMenu();
            choice = scanner.nextInt();
            switch (choice) {
                case 1:
                    displayAllBooks(bookMap);
                    break;
                case 2:
                    displayBookByKey(bookMap, scanner);
                    break;
                case 3:
                    reducePrice(bookMap, scanner);
                    break;
                case 4:
                    System.out.println("Exiting...");
                    break;
                default:
                    System.out.println("Invalid choice. Please try again.");
            }
        } while (choice != 4);

        scanner.close();
    }

    // Display the menu options
    public static void displayMenu() {
        System.out.println("\nMenu:");
        System.out.println("1. Display all books");
        System.out.println("2. Display book by key");
        System.out.println("3. Reduce price for a particular publication");
        System.out.println("4. Exit");
        System.out.print("Enter your choice: ");
    }

    // Display all books in the map
    public static void displayAllBooks(Map<Integer, Book> bookMap) {
        System.out.println("\nAll books:");
        for (Map.Entry<Integer, Book> entry : bookMap.entrySet()) {
            System.out.println(entry.getKey() + " "+ entry.getValue());
        }
    }

    // Display a book based on user input key
    public static void displayBookByKey(Map<Integer, Book> bookMap, Scanner
scanner) {
        System.out.print("\nEnter the key: ");
        int key = scanner.nextInt();
        if (bookMap.containsKey(key)) {
            System.out.println(bookMap.get(key));
```

```
        } else {
            System.out.println("Book with key " + key + " not found.");
        }
    }

    // Reduce the price for books from a particular publication
    public static void reducePrice(Map<Integer, Book> bookMap, Scanner scanner)
{
        System.out.print("\nEnter the publication: ");
        scanner.nextLine(); // Consume newline character
        String publication = scanner.nextLine();
        double discount = 0.9; // 10% discount

        System.out.println("\nBooks with reduced price:");
        for (Map.Entry<Integer, Book> entry : bookMap.entrySet()) {
            Book book = entry.getValue();
            if (book.publication.equalsIgnoreCase(publication)) {
                book.price *= discount; // Apply the discount
                System.out.println("Key: " + entry.getKey() + ", Value: " +
book);
            }
        }
    }
}
```

Output



```
Enter your choice: 3

Enter the publication: XYZ Books

Books with reduced price:
Key: 2, Value: [Book Name=Python Basics, price=314.99100000000004, Author Name=Fletcher, ISBN No=978-9876543210, Publication=X
Key: 5, Value: [Book Name=Data Structures, price=359.99100000000004, Author Name=Thomas H. Cormen, ISBN No=978-5678901234, Pub

Menu:
1. Display all books
2. Display book by key
3. Reduce price for a particular publication
4. Exit
Enter your choice: 4
Exiting...
```

6. Write a Java program to simulate an online shopping cart.

A shopping cart is a collection of items selected by a customer for purchase. A user can add items to the cart, remove them, empty the cart, view the items in the cart, view the items in the shop, and end shopping and proceed to checkout. (using arraylist)

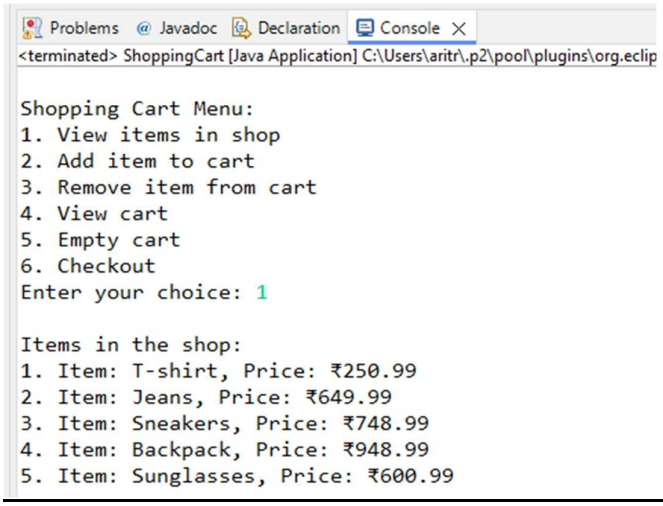
```
public class Item {
    String name; // Name of the item
    double price; // Price of the item

   // Constructor to initialize an Item object with a name and price
   public Item(String name, double price) {
       this.name = name;
       this.price = price;
   }

   // Getter method to retrieve the name of the item
   public String getName() {
       return name;
   }

   // Getter method to retrieve the price of the item
   public double getPrice() {
       return price;
   }

   // Override the toString method to provide a string representation of the item
   @Override
   public String toString() {
       return "Item: " + name + ", Price: ₹" + price;
   }
}

import java.util.ArrayList;
import java.util.Scanner;

public class ShoppingCart {
    ArrayList<Item> cart; // List to store items in the cart
    ArrayList<Item> shop; // List to store items in the shop
    Scanner scanner; // Scanner object for user input
```

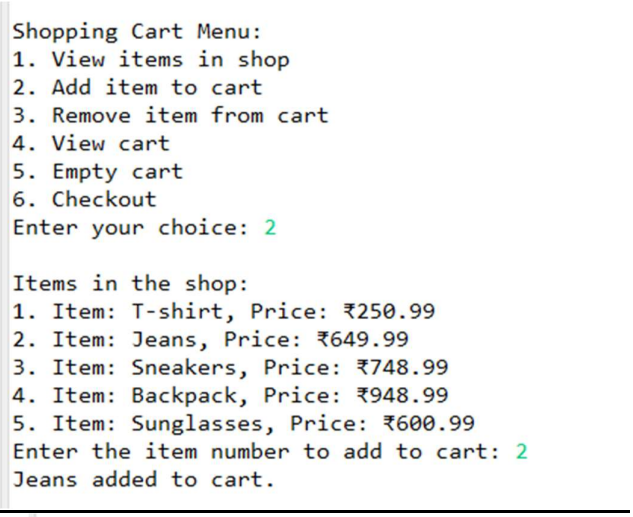
```
    // Constructor to initialize the shopping cart and shop
    public ShoppingCart() {
        cart = new ArrayList<>();
        shop = new ArrayList<>();
        scanner = new Scanner(System.in);
        initializeShop();
    }

    // Method to initialize the shop with some items
    public void initializeShop() {
        shop.add(new Item("T-shirt", 250.99));
        shop.add(new Item("Jeans", 649.99));
        shop.add(new Item("Sneakers", 748.99));
        shop.add(new Item("Backpack", 948.99));
        shop.add(new Item("Sunglasses", 600.99));
    }

    // Method to start the shopping cart system
    public void start() {
        int choice;
        do {
            displayMenu();
            choice = scanner.nextInt();
            scanner.nextLine(); // Consume the newline character
            handleChoice(choice);
        } while (choice != 6);
        scanner.close(); // Close the scanner after use
    }

    // Method to display the shopping cart menu
    public void displayMenu() {
        System.out.println("\nShopping Cart Menu:");
        System.out.println("1. View items in shop");
        System.out.println("2. Add item to cart");
        System.out.println("3. Remove item from cart");
        System.out.println("4. View cart");
        System.out.println("5. Empty cart");
        System.out.println("6. Checkout");
        System.out.print("Enter your choice: ");
    }

    // Method to handle user choices in the shopping cart menu
    public void handleChoice(int choice) {
        switch (choice) {
            case 1:
                viewShop();
                break;
            case 2:
                addToCart();
                break;
            case 3:
```

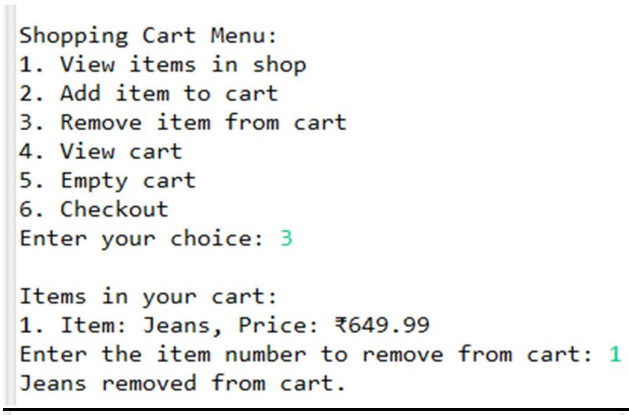
```
                removeFromCart();
                break;
            case 4:
                viewCart();
                break;
            case 5:
                emptyCart();
                break;
            case 6:
                checkout();
                break;
            default:
                System.out.println("Invalid choice. Please try again.");
        }
    }

    // Method to display items in the shop
    public void viewShop() {
        System.out.println("\nItems in the shop:");
        for (int i = 0; i < shop.size(); i++) {
            System.out.println((i + 1) + ". " + shop.get(i));
        }
    }

    // Method to add an item to the cart
    public void addToCart() {
        viewShop();
        System.out.print("Enter the item number to add to cart: ");
        int itemNumber = scanner.nextInt();
        scanner.nextLine(); // Consume the newline character

        if (itemNumber > 0 && itemNumber <= shop.size()) {
            Item item = shop.get(itemNumber - 1);
            cart.add(item);
            System.out.println(item.getName() + " added to cart.");
        } else {
            System.out.println("Invalid item number.");
        }
    }

    // Method to remove an item from the cart
    public void removeFromCart() {
        viewCart();
        System.out.print("Enter the item number to remove from cart: ");
        int itemNumber = scanner.nextInt();
        scanner.nextLine(); // Consume the newline character

        if (itemNumber > 0 && itemNumber <= cart.size()) {
            Item item = cart.remove(itemNumber - 1);
            System.out.println(item.getName() + " removed from cart.");
        } else {
            System.out.println("Invalid item number.");
```

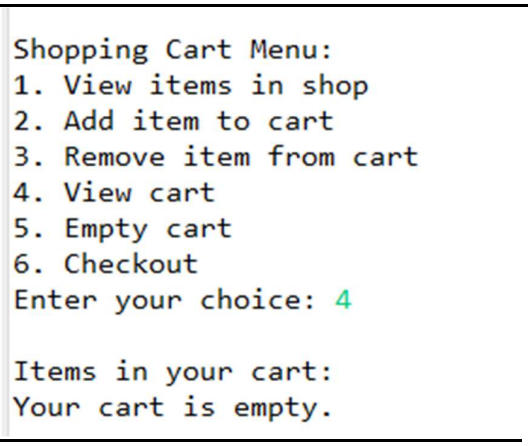
```
        }
    }

    // Method to display items in the cart
    public void viewCart() {
        System.out.println("\nItems in your cart:");
        if (cart.isEmpty()) {
            System.out.println("Your cart is empty.");
        } else {
            for (int i = 0; i < cart.size(); i++) {
                System.out.println((i + 1) + ". " + cart.get(i));
            }
        }
    }

    // Method to empty the cart
    public void emptyCart() {
        cart.clear();
        System.out.println("Cart emptied.");
    }

    // Method to proceed to checkout
    public void checkout() {
        if (cart.isEmpty()) {
            System.out.println("Your cart is empty. Nothing to checkout.");
        } else {
            double total = 0;
            System.out.println("\nItems in your cart:");
            for (Item item : cart) {
                System.out.println(item);
                total += item.getPrice();
            }
            System.out.println("Total: ₹" + total);
            System.out.println("Thank you for your purchase!");
            cart.clear(); // Clear the cart after checkout
        }
    }

    // Main method to create an instance of ShoppingCart and start the shopping cart
system
    public static void main(String[] args) {
        ShoppingCart shoppingCart = new ShoppingCart();
        shoppingCart.start();
    }
}
```

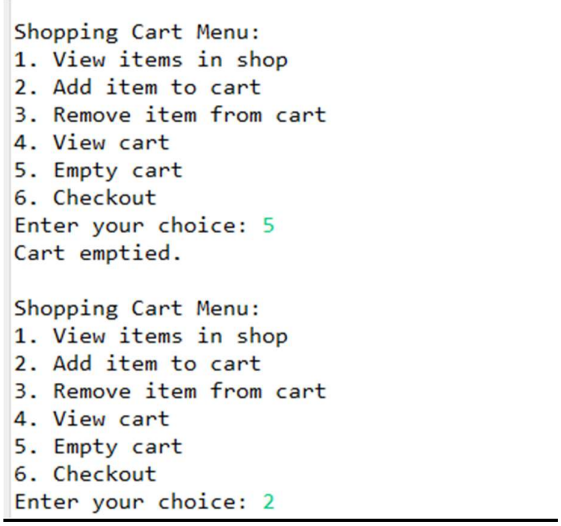
Output

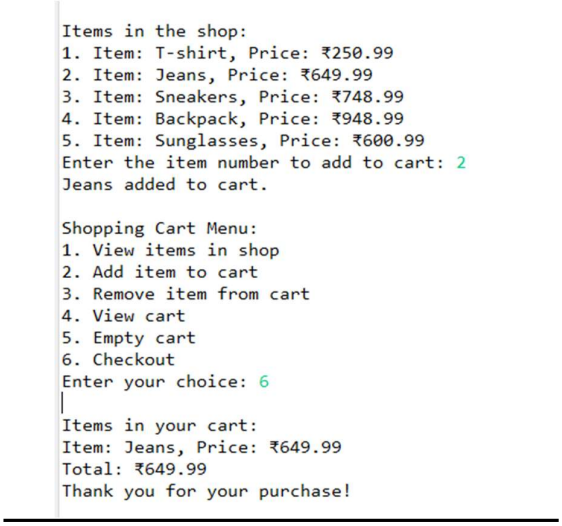

Problems
Javadoc
Declaration
Console
<terminated> ShoppingCart [Java Application] C:\Users\aritr\.p2\pool\plugins\org.eclip
Shopping Cart Menu:
1. View items in shop
2. Add item to cart
3. Remove item from cart
4. View cart
5. Empty cart
6. Checkout
Enter your choice: 1
Items in the shop:
1. Item: T-shirt, Price: ₹250.99
2. Item: Jeans, Price: ₹649.99
3. Item: Sneakers, Price: ₹748.99
4. Item: Backpack, Price: ₹948.99
5. Item: Sunglasses, Price: ₹600.99


Shopping Cart Menu:
1. View items in shop
2. Add item to cart
3. Remove item from cart
4. View cart
5. Empty cart
6. Checkout
Enter your choice: 2
Items in the shop:
1. Item: T-shirt, Price: ₹250.99
2. Item: Jeans, Price: ₹649.99
3. Item: Sneakers, Price: ₹748.99
4. Item: Backpack, Price: ₹948.99
5. Item: Sunglasses, Price: ₹600.99
Enter the item number to add to cart: 2
Jeans added to cart.


Shopping Cart Menu:
1. View items in shop
2. Add item to cart
3. Remove item from cart
4. View cart
5. Empty cart
6. Checkout
Enter your choice: 3
Items in your cart:
1. Item: Jeans, Price: ₹649.99
Enter the item number to remove from cart: 1
Jeans removed from cart.


Shopping Cart Menu:
1. View items in shop
2. Add item to cart
3. Remove item from cart
4. View cart
5. Empty cart
6. Checkout
Enter your choice: 4
Items in your cart:
Your cart is empty.


Shopping Cart Menu:
1. View items in shop
2. Add item to cart
3. Remove item from cart
4. View cart
5. Empty cart
6. Checkout
Enter your choice: 5
Cart emptied.
Shopping Cart Menu:
1. View items in shop
2. Add item to cart
3. Remove item from cart
4. View cart
5. Empty cart
6. Checkout
Enter your choice: 2


Items in the shop:
1. Item: T-shirt, Price: ₹250.99
2. Item: Jeans, Price: ₹649.99
3. Item: Sneakers, Price: ₹748.99
4. Item: Backpack, Price: ₹948.99
5. Item: Sunglasses, Price: ₹600.99
Enter the item number to add to cart: 2
Jeans added to cart.
Shopping Cart Menu:
1. View items in shop
2. Add item to cart
3. Remove item from cart
4. View cart
5. Empty cart
6. Checkout
Enter your choice: 6
Items in your cart:
Item: Jeans, Price: ₹649.99
Total: ₹649.99
Thank you for your purchase!

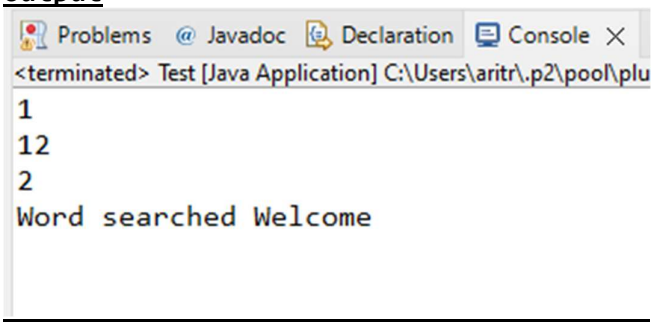
File Handling

1. Program to count the number of character, no of word, no of lines in a given file.

```
import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Test {
    public static void main(String[] args) throws IOException {
        // Create a FileWriter object to write to the file
        FileWriter fw = new FileWriter("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\d.txt");
        String str1 = "Welcome Here";
        fw.write(str1); // Write the string to the file
        fw.close(); // Close the FileWriter to save the file

        try {
            // Create a FileReader and BufferedReader to read the file
            FileReader f = new FileReader("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\d.txt");
            BufferedReader b = new BufferedReader(f);
            int nLine = 0; // Initialize line count
            int nword = 0; // Initialize word count
            int nChars = 0; // Initialize character count
            String line;

            // Loop through each line in the file
            while ((line = b.readLine()) != null) {
                nLine++; // Increment line count
                nChars += line.length(); // Increment character count by the length of the
line
                String[] words = line.split("\\s+"); // Split the line into words
                nword += words.length; // Increment word count by the number of words in
the line
            }
            b.close(); // Close the BufferedReader

            // Print the counts
            System.out.println("Number of lines: " + nLine);
            System.out.println("Number of characters: " + nChars);
            System.out.println("Number of words: " + nword);
        } catch (Exception e) {
            System.out.println("File not found or cannot be read");
            e.printStackTrace(); // Print the stack trace for debugging
        }
```

```
        // Search for a specific word in the file
        String wordToSearch = "Welcome";

        try {
            // Create a FileReader and BufferedReader to read the file
            FileReader f = new FileReader("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\d.txt");
            BufferedReader b = new BufferedReader(f);
            String line;
            int lineNumber = 0;

            // Loop through each line in the file
            while ((line = b.readLine()) != null) {
                lineNumber++; // Increment line number
                // Check if the line contains the word to search
                if (line.contains(wordToSearch)) {
                    System.out.println("Word \"" + wordToSearch + "\" found at line " +
lineNumber);
                    break; // Exit the loop once the word is found
                }
            }
            b.close(); // Close the BufferedReader
        } catch (IOException e) {
            System.err.println("Error reading the file: " + e.getMessage());
        }
    }
}
```

Output

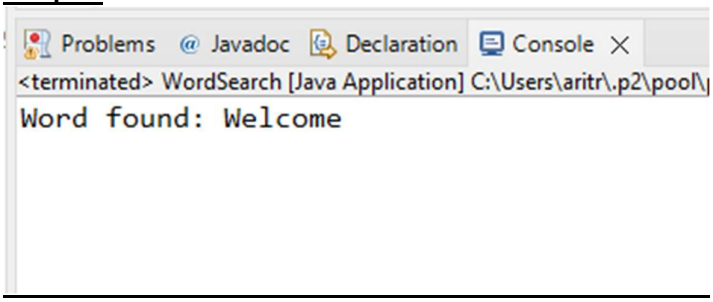


2. Java Program to Search for a given word in a File ?

```
import java.io.File;
import java.io.FileNotFoundException;
import java.util.Scanner;

public class WordSearch {
    public static void main(String[] args) {
```

```
        // The word to search for
        String wordToFind = "Welcome";
        // The file to search in
        File file = new File("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\d.txt");

        try {
            // Create a Scanner object to read the file
            Scanner scanner = new Scanner(file);
            boolean found = false; // Flag to track if the word is found

            // Loop through each line in the file
            while (scanner.hasNextLine()) {
                String line = scanner.nextLine(); // Read the next line

                // Check if the line contains the word to search for
                if (line.contains(wordToFind)) {
                    System.out.println("Word found: " + wordToFind);
                    found = true; // Set the flag to true if the word is found
                    break; // Exit the loop once the word is found
                }
            }

            // If the word is not found in the entire file, print a message
            if (!found) {
                System.out.println("Word not found: " + wordToFind);
            }

            // Close the Scanner
            scanner.close();
        } catch (FileNotFoundException e) {
            System.out.println("File not found: " + e.getMessage());
        }
    }
}
```

Output

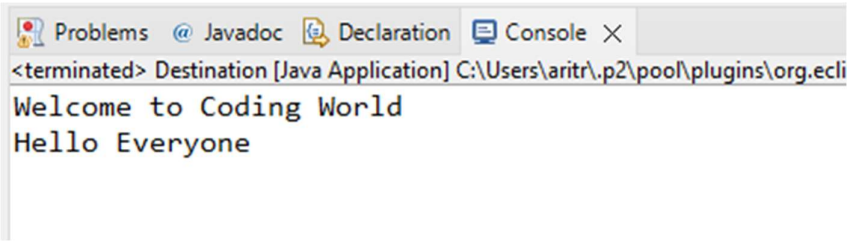


3. Read source and destination file from the user as an argument. Copy the contents of the source into another destination file in a faster, efficient way.

```
import java.io.FileWriter;
import java.io.IOException;

public class Source {
    public static void main(String[] args) throws IOException {
        // Create a FileWriter object to write to the file
        FileWriter fw = new FileWriter("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\src.txt");

        // Strings to be written to the file
        String str1 = "Welcome to Coding World";
        String str2 = "Hello Everyone";

        // Write the strings to the file, each followed by a newline character
        fw.write(str1 + '\n');
        fw.write(str2 + '\n');

        // Close the FileWriter to save the file
        fw.close();
    }
}

import java.io.BufferedReader;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

public class Destination {
    public static void main(String[] args) throws IOException {
        // Write initial content to des.txt
        try {
            FileWriter fw = new FileWriter("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\des.txt");
            String str1 = "Good Evening";
            String str2 = "Nice to meet you all";
            fw.write(str1 + '\n');
            fw.write(str2 + '\n');
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Read from src.txt and write to des.txt
        try {
            // Read from src.txt
            FileReader fr = new FileReader("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\src.txt");
```

```
            BufferedReader br = new BufferedReader(fr);
            String st = br.readLine();
            String st2 = br.readLine();

            // Write to des.txt (overwriting previous content)
            FileWriter fw = new FileWriter("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\des.txt");
            fw.write(st + '\n');
            fw.write(st2 + '\n');

            // Print the content read from src.txt
            System.out.println(st);
            System.out.println(st2);

            // Close readers and writers
            br.close();
            fw.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output



4. Write a program to serialize the student object(id,name,marks,subject) *Write a student object (minimum five) into the file
   * Read the same from the file,
   * Find out result and display the student details along with result.
   * Find out how many no of students passed and failed in a particular Subject.
   *Sort the students object based on marks.

```java
import java.io.Serializable;

// The Student class implements Serializable to allow objects of this class to be
serialized
public class Student implements Serializable {
    // Fields for the Student class
    int id;
    String name;
    int marks;
    String subject;
    String result;

    // Constructor to initialize the Student object
    public Student(int id, String name, int marks, String subject) {
        super();
        this.id = id;
        this.name = name;
        this.marks = marks;
        this.subject = subject;
    }

    // Getter and setter methods for id
    public int getId() {
        return id;
    }

    public void setId(int id) {
        this.id = id;
    }

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

    public void setName(String name) {
        this.name = name;
    }

    // Getter and setter methods for marks
    public int getMarks() {
        return marks;
    }

    public void setMarks(int marks) {
        this.marks = marks;
    }

    // Getter and setter methods for subject
    public String getSubject() {
        return subject;
```

```
    }

    public void setSubject(String subject) {
        this.subject = subject;
    }

    // Getter and setter methods for result
    public String getResult() {
        return result;
    }

    public void setResult(String result) {
        this.result = result;
    }

    // Override the toString() method to return a string representation of the Student
object
    @Override
    public String toString() {
        return "Student [id=" + id + ", name=" + name + ", marks=" + marks + ", subject="
+ subject + ", result="
                + result + "]";
    }
}


import java.io.FileInputStream;
import java.io.FileOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

public class StudentManagement {
    public static void main(String[] args) throws IOException, ClassNotFoundException {
        // Create an ArrayList of Student objects
        ArrayList<Student> students = new ArrayList<>();
        students.add(new Student(1001, "Shan", 90, "OS"));
        students.add(new Student(1002, "Raj", 40, "Java"));
        students.add(new Student(1003, "Kumar", 50, "Python"));
        students.add(new Student(1004, "Kiran", 90, "OS"));
        students.add(new Student(1005, "Kaveri", 100, "C"));

        // Write the list of Student objects to a file
        try {
            FileOutputStream fileOut = new FileOutputStream("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\Demo2.txt");
            ObjectOutputStream objectOut = new ObjectOutputStream(fileOut);
            objectOut.writeObject(students);
```

```
            // Close the file and object output streams
            fileOut.close();
            objectOut.close();
        } catch (IOException e) {
            e.printStackTrace();
        }

        // Read the list of Student objects from the file
        try {
            FileInputStream fileIn = new FileInputStream("D:\\CDAC2024\\Core
Java\\240350120028_AritraDas\\Assignment3\\src\\Demo2.txt");
            ObjectInputStream objectIn = new ObjectInputStream(fileIn);

            // Deserialize the ArrayList of Student objects
            ArrayList<Student> readStudents = (ArrayList<Student>) objectIn.readObject();

            // Initialize counters for pass and fail counts
            int passCount = 0;
            int failCount = 0;

            // Iterate through the list of students to determine their results
            for (Student student : readStudents) {
                if (student.getMarks() > 50) {
                    student.setResult("Pass");
                    passCount++;
                } else {
                    student.setResult("Fail");
                    failCount++;
                }
                // Print the student details
                System.out.println(student.toString());
            }

            // Sort the students by marks in ascending order
            Collections.sort(readStudents, Comparator.comparingDouble(Student::getMarks));
            System.out.println("After Sorting:");
            for (Student student : readStudents) {
                // Print the sorted student details
                System.out.println(student.toString());
            }

            // Determine the number of students who passed or failed in a particular
subject
            String subject = "OS";
            int subjectPassed = 0;
            int subjectFailed = 0;
            for (Student student : readStudents) {
                if (student.getSubject().equalsIgnoreCase(subject)) {
                    if (student.getMarks() > 56) {
                        subjectPassed++;
                    } else {
                        subjectFailed++;
```

```
                    }
                }
            }
            // Print the number of students passed and failed in the specific subject
            System.out.println("Number of Students passed in " + subject + ": " +
subjectPassed);
            System.out.println("Number of Students failed in " + subject + ": " +
subjectFailed);

            // Close the file and object input streams
            fileIn.close();
            objectIn.close();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Output