**Java OOPS**

**Task Description:**

**Solve the below problems using OOPs concept using Java.**

1. Create a class called "Person" with attributes "name" and "age". Also create a constructor

and getter methods for the attributes.

| **package** trainingtaskcompletion;  **import** java.util.Scanner;  **public** **class** Person\_constructor {  **private** String name;  **private** **int** age;  // Constructor  **public** Person\_constructor(String name, **int** age) {  **this**.name = name;  **this**.age = age;  }  // Getter method for name  **public** String getName() {  **return** name;  }  // Getter method for age  **public** **int** getAge() {  **return** age;  }  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***); // Create a Scanner object  // Prompt the user to enter name  System.***out***.print("Enter the name of the person: ");  String name = scanner.nextLine();  // Prompt the user to enter age  System.***out***.print("Enter the age of the person: ");  **int** age = scanner.nextInt();  // Create a Person\_constructor object  Person\_constructor person = **new** Person\_constructor(name, age);  scanner.close(); // Close the Scanner object  // Access attributes using getter methods  System.***out***.println("Name: " + person.getName());  System.***out***.println("Age: " + person.getAge());  }  } |
| --- |

**Output:-**

Enter the name of the person: Mcenroe

Enter the age of the person: 16

Name: Mcenroe

Age: 16

2. Employee class with the attributes id, name, and salary, and includes a method called raiseSalary(percent) that updates the salary by a specified percentage

| **package** trainingtaskcompletion;  /\*\*  \* The Employee class represents an employee with basic information such as ID,  \* first name, last name, and salary.  \*/  **public** **class** Employee {  // Private instance variables to store employee information  **private** **int** id; // Employee ID  **private** String firstName; // First name of the employee  **private** String lastName; // Last name of the employee  **private** **int** salary; // Salary of the employee  /\*\*  \* Constructs an Employee object with the specified ID, first name, last name,  \* and salary.  \*  \* **@param** id the employee ID  \* **@param** firstName the first name of the employee  \* **@param** lastName the last name of the employee  \* **@param** salary the salary of the employee  \*/  **public** Employee(**int** id, String firstName, String lastName, **int** salary) {  **this**.id = id;  **this**.firstName = firstName;  **this**.lastName = lastName;  **this**.salary = salary;  }  /\*\*  \* Returns the employee ID.  \*  \* **@return** the employee ID  \*/  **public** **int** getId() {  **return** id;  }  /\*\*  \* Returns the first name of the employee.  \*  \* **@return** the first name of the employee  \*/  **public** String getFirstName() {  **return** firstName;  }  /\*\*  \* Returns the last name of the employee.  \*  \* **@return** the last name of the employee  \*/  **public** String getLastName() {  **return** lastName;  }  /\*\*  \* Returns the full name of the employee.  \*  \* **@return** the full name of the employee  \*/  **public** String getName() {  **return** firstName + " " + lastName;  }  /\*\*  \* Returns the salary of the employee.  \*  \* **@return** the salary of the employee  \*/  **public** **int** getSalary() {  **return** salary;  }  /\*\*  \* Sets the salary of the employee.  \*  \* **@param** salary the new salary to set  \*/  **public** **void** setSalary(**int** salary) {  **this**.salary = salary;  }  /\*\*  \* Returns the annual salary of the employee.  \*  \* **@return** the annual salary of the employee  \*/  **public** **int** getAnnualSalary() {  **return** salary \* 12;  }  /\*\*  \* Increases the salary of the employee by the specified percentage.  \*  \* **@param** percent the percentage by which to increase the salary  \* **@return** the new salary after the increase  \*/  **public** **int** raiseSalary(**int** percent) {  salary += salary \* percent / 100;  **return** salary;  }  /\*\*  \* Returns a string representation of the Employee object.  \*  \* **@return** a string representation of the Employee object  \*/  @Override  **public** String toString() {  **return** "Employee[id=" + id + ", name=" + getName() + ", salary=" + salary + "]";  }  }  **create an instance of the Employee class in another class & call its methods using the instance.**  **package** trainingtaskcompletion;  **public** **class** EmployeeTest {  **public** **static** **void** main(String[] args) {  // Create an instance of the Employee class  Employee employee = **new** Employee(123, "John", "Doe", 50000);  // Access and print employee information using getter methods  System.***out***.println("Employee ID: " + employee.getId());  System.***out***.println("Employee First Name: " + employee.getFirstName());  System.***out***.println("Employee Last Name: " + employee.getLastName());  System.***out***.println("Employee Full Name: " + employee.getName());  System.***out***.println("Employee Salary: " + employee.getSalary());  // Update employee salary using setter method  employee.setSalary(55000);  System.***out***.println("Updated Employee Salary: " + employee.getSalary());  // Calculate and print annual salary  System.***out***.println("Annual Salary: " + employee.getAnnualSalary());  // Increase employee salary by 10%  employee.raiseSalary(10);  System.***out***.println("Increased Salary: " + employee.getSalary());  // Print employee information using toString() method  System.***out***.println(employee);  }  } |
| --- |

**Output:-**

Employee ID: 123

Employee First Name: John

Employee Last Name: Doe

Employee Full Name: John Doe

Employee Salary: 50000

Updated Employee Salary: 55000

Annual Salary: 660000

Increased Salary: 60500

Employee[id=123, name=John Doe, salary=60500]

3. Create a class circle with radius as data member. Create two constructors (no

argument, and two arguments) and a method to calculate Circumference.

| **package** trainingtaskcompletion;  **import** java.util.Scanner;  **public** **class** CircleConstructor {  **private** **double** radius;  // No-argument constructor  **public** CircleConstructor() {  Scanner scanner = **new** Scanner(System.***in***); // Create a Scanner object  System.***out***.print("Enter the radius of the circle: ");  **this**.radius = scanner.nextDouble(); // Read radius input from the user  scanner.close(); // Close the Scanner object  }  // Constructor with two argument to get input from the user  **public** CircleConstructor(**boolean** inputFromUser,**double** Inputradius) {  **if** (inputFromUser) {  Scanner scanner = **new** Scanner(System.***in***); // Create a Scanner object  **try** {  System.***out***.println("Enter the radius of the circle: " +Inputradius);  **this**.radius = scanner.nextDouble(); // Read radius input from the user  } **catch** (java.util.NoSuchElementException e) {  **this**.radius = Inputradius;  } **finally** {  scanner.close(); // Close the Scanner object  }  } **else** {  System.***out***.println("Enter the radius of the circle: " +Inputradius);  **this**.radius = Inputradius; // Default radius value  }  }  // Two-argument constructor with input from the user  **public** CircleConstructor(**double** radius, **boolean** inputFromUser) {  **if** (inputFromUser) {  Scanner scanner = **new** Scanner(System.***in***); // Create a Scanner object  System.***out***.print("Enter the radius of the circle: ");  **this**.radius = scanner.nextDouble(); // Read radius input from the user  scanner.close(); // Close the Scanner object  } **else** {  **this**.radius = radius;  }  }  // Method to calculate circumference  **public** **double** calculateCircumference() {  **return** 2 \* Math.***PI*** \* radius;  }  **public** **static** **void** main(String[] args) {  // Create a circle object with radius entered by the user using no-argument  // constructor  CircleConstructor circle1 = **new** CircleConstructor();  System.***out***.println(  "Circle 1: Radius = " + circle1.radius + ", Circumference = " + circle1.calculateCircumference());  // Create a circle object with radius entered by the user using one-argument  // constructor  CircleConstructor circle2 = **new** CircleConstructor(**false**,10.0);  System.***out***.println(  "Circle 2: Radius = " + circle2.radius + ", Circumference = " + circle2.calculateCircumference());  // Create a circle object with radius 5 using two-argument constructor  CircleConstructor circle3 = **new** CircleConstructor(**false**,15.0);  System.***out***.println(  "Circle 3: Radius = " + circle3.radius + ", Circumference = " + circle3.calculateCircumference());  }  } |
| --- |

**Output:-**

Enter the radius of the circle: 5

Circle 1: Radius = 5.0, Circumference = 31.41592653589793

Enter the radius of the circle: 10.0

Circle 2: Radius = 10.0, Circumference = 62.83185307179586

Enter the radius of the circle: 15.0

Circle 3: Radius = 15.0, Circumference = 94.24777960769379

4. Create a class Account class with balance as data member. Create two constructors (no

argument, and two arguments) and methods to withdraw and deposit balance.

| **package** trainingtaskcompletion;  **import** java.util.Scanner;  **public** **class** Account {  **private** **double** balance;  // No-argument constructor  **public** Account() {  **this**.balance = 0.0; // Set initial balance to 0  }  // Constructor with two arguments  **public** Account(**double** initialBalance) {  **this**.balance = initialBalance;  }  // Method to withdraw balance  **public** **void** withdraw(**double** amount) {  **if** (amount > 0 && amount <= balance) {  balance -= amount;  System.***out***.println("Withdrawal successful. Current balance: " + balance);  } **else** {  System.***out***.println("Invalid withdrawal amount or insufficient balance.");  }  }  // Method to deposit balance  **public** **void** deposit(**double** amount) {  **if** (amount > 0) {  balance += amount;  System.***out***.println("Deposit successful. Current balance: " + balance);  } **else** {  System.***out***.println("Invalid deposit amount.");  }  }  **public** **static** **void** main(String[] args) {  Scanner scanner = **new** Scanner(System.***in***);  // Create an account object with no-argument constructor  Account account1 = **new** Account();  System.***out***.println("Account 1 Balance: " + account1.balance);  // Prompt the user to enter initial balance  System.***out***.print("Enter the initial balance for Account 2: ");  **double** initialBalance = scanner.nextDouble();  // Create an account object with two-argument constructor  Account account2 = **new** Account(initialBalance);  System.***out***.println("Account 2 Balance: " + account2.balance);  // Test withdrawal and deposit methods  account1.deposit(100.0);  account1.withdraw(50.0);  scanner.close();  }  } |
| --- |

**Output:-**

Account 1 Balance: 0.0

Enter the initial balance for Account 2: 500

Account 2 Balance: 500.0

Deposit successful. Current balance: 100.0

Withdrawal successful. Current balance: 50.0

5. Create a Tea class in Java that includes the following methods:

a)prepareTea() - a method that prepares a basic tea with hot water and tea leaves.

b)addMilkO - a method that adds milk to the basic tea.

c)addSugarQ - a method that adds sugar to the basic tea.

| **package** trainingtaskcompletion;  **public** **class** CustomeTea {  // Method to prepare basic tea  **public** **void** prepareTea() {  System.***out***.println("Prepare basic tea with hot water and tea leaves.");  }  // Method to add milk to the tea  **public** **void** addMilk() {  System.***out***.println("Add milk to the tea.");  }  // Method to add sugar to the tea  **public** **void** addSugar() {  System.***out***.println("Add sugar to the tea.");  }  **public** **static** **void** main(String[] args) {  // Create an instance of the class  CustomeTea tea = **new** CustomeTea();  // Call the methods  tea.prepareTea();  tea.addMilk();  tea.addSugar();  }  } |
| --- |

**Output:-**

Prepare basic tea with hot water and tea leaves.

Add milk to the tea.

Add sugar to the tea.

6. Create three subclasses of the Tea class: BlackTea, GreenTea, and HerbalTea Each

subclass should override the prepareTeaQ) method to prepare the specific type of tea

(black tea, green tea, or herbal tea) with appropriate ingredients and brewing times.

| **package** trainingtaskcompletion;  **public** **class** TeaClsses {  // Method to prepare basic tea  **public** **void** prepareBasicTea() {  System.***out***.println("Prepare basic tea with hot water and tea leaves.");  }  }  **class** BlackTea **extends** TeaClsses {  // Override method to prepare black tea  @Override  **public** **void** prepareBasicTea() {  System.***out***.println("Prepare black tea with hot water and black tea leaves.");  }  }  **class** GreenTea **extends** TeaClsses {  // Override method to prepare green tea  @Override  **public** **void** prepareBasicTea() {  System.***out***.println("Prepare green tea with hot water and green tea leaves.");  }  }  **class** HerbalTea **extends** TeaClsses {  // Override method to prepare herbal tea  @Override  **public** **void** prepareBasicTea() {  System.***out***.println("Prepare herbal tea with hot water and herbal tea leaves.");  }  }  **Calling Class:-**  package trainingtaskcompletion;  public class TeaCalling {  public static void main(String[] args) {  // Create instances of each subclass  BlackTea blackTea = new BlackTea();  GreenTea greenTea = new GreenTea();  HerbalTea herbalTea = new HerbalTea();  // Call the overridden method for each subclass  blackTea.prepareBasicTea();  greenTea.prepareBasicTea();  herbalTea.prepareBasicTea();  }  } |
| --- |

**Output:-**

Prepare black tea with hot water and black tea leaves.

Prepare green tea with hot water and green tea leaves.

Prepare herbal tea with hot water and herbal tea leaves.

7. Implement polymorphism in your program by creating an array of Tea objects that

includes instances of the Tea class and its subclasses.

| **package** trainingtaskcompletion;  //Base class Tea  **class** Tea {  **public** **void** brew() {  System.***out***.println("Brewing tea...");  }  }  //Subclass BlackTea  **class** BlackTea **extends** Tea {  @Override  **public** **void** brew() {  System.***out***.println("Brewing black tea...");  }  }  //Subclass GreenTea  **class** GreenTea **extends** Tea {  @Override  **public** **void** brew() {  System.***out***.println("Brewing green tea...");  }  }  **public** **class** PolyTea {  **public** **static** **void** main(String[] args) {  // Create an array of Tea objects  Tea[] teas = **new** Tea[3];  // Populate the array with instances of Tea and its subclasses  teas[0] = **new** Tea(); // Tea object  teas[1] = **new** BlackTea(); // BlackTea object  teas[2] = **new** GreenTea(); // GreenTea object  // Brew each tea in the array using polymorphism  **for** (Tea tea : teas) {  tea.brew(); // Polymorphic method call  }  }  } |
| --- |

**Output:-**

Brewing tea...

Brewing black tea...

Brewing green tea...