

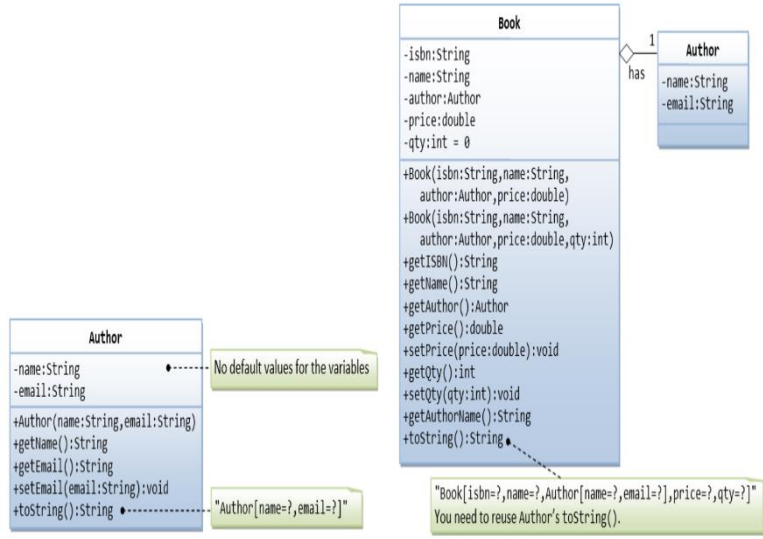


**Practice No. : 6**

**Topic : OOPs- Class, Object, Constructor, Encapsulation**

**Date : 14-05-2024**

## Solve the following problems

Q. No.	Question Detail	Level
1	Print the average of three numbers entered by user by creating a class named 'Average' having a method to calculate and print the average.	Easy
2	<p>a) A class called Author, which models an author of a book, is designed as shown in the class diagram.</p> <p>b) A class called Book, which models a book written by ONE author and composes an instance of Author as its instance variable, is designed as shown in the class diagram. Test the methods of Book and Author Class by creating instance of Book class.</p> 	Easy
3	<p>You are developing a simple program to manage product information for an online store. To represent each product, you want to create a Java class with default constructors that initialize the product details to default values.</p> <p>Define a Java class named <b>Product</b> with the following specifications:</p>	Easy

***Sometimes later becomes never. DO IT NOW!***



## SELF PRACTICE

### SDE Readiness Training

	<ol style="list-style-type: none"> <li>1. Private instance variables to store the product ID, name, price, and quantity.</li> <li>2. Implement a default constructor that initializes the product details as follows: <ul style="list-style-type: none"> <li>• Product ID: 0</li> <li>• Product name: "Unknown"</li> <li>• Price: 0.0</li> <li>• Quantity: 0</li> </ul> </li> <li>3. Implement another constructor that takes parameters for product ID, name, price, and quantity, and initializes the corresponding instance variables with the provided values.</li> </ol> <p>Write the Java class <b>Product</b> with the given requirements and demonstrate its usage in the <b>Main</b> class by creating instances of the <b>Product</b> class using both default and parameterized constructors.</p>	
<b>4</b>	Write a Java class called crop and help the farmer to separate the vegetables in his farm. The farm contains vegetables like Carrot, Brinjal, Potato. Separate the vegetables and print the result in following format using constructor overloading C 15 P 25 B 30.	Easy
<b>5</b>	<p>Write a program by creating an 'Employee' class having the following methods and print the final salary.</p> <ol style="list-style-type: none"> <li>i. 'getInfo()' which takes the salary, number of hours of work per day of employee as parameter</li> <li>ii. 'AddSal()' which adds \$10 to salary of the employee if it is less than \$500.</li> <li>iii. 'AddWork()' which adds \$5 to salary of employee if the number of hours of work per day is more than 6 hours.</li> </ol>	Easy
<b>6</b>	<p>The machine has two main components: A built-in cash register and several dispensers to hold and release the products.</p> <p>Define class cashRegister in C# with the following descriptions :</p> <ol style="list-style-type: none"> <li>i. Private Members: cashOnHand of type integer</li> <li>ii. Public Members: A default constructor cashRegister() sets the cash in the register to 100.</li> <li>iii. A constructor cashRegister(int) sets the cash in the register to a specific amount.</li> </ol>	Medium

***Sometimes later becomes never. DO IT NOW!***



## SELF PRACTICE

### SDE Readiness Training

	<ul style="list-style-type: none"><li>iv. A function <code>getCurrentBalance()</code> which returns value of <code>cashOnHand</code></li><li>v. A function <code>acceptAmount(int)</code> to receive the amount deposited by the customer and update the amount in the register.</li></ul>	
<b>7</b>	<p>You are developing a simple program to manage a library's collection of books. Each book in the library has a unique identification number assigned to it. Additionally, you want to define a constant variable to represent the maximum number of books that the library can hold. Define a Java class named <b>Library</b> with the following specifications:</p> <ol style="list-style-type: none"><li>1. Implement a static variable named <b>totalBooks</b> to keep track of the total number of books in the library.</li><li>2. Define a constant variable named <b>MAX_CAPACITY</b> to represent the maximum number of books that the library can hold. Set its value to 1000.</li><li>3. Implement a method named <b>addBook()</b> that increments the <b>totalBooks</b> count each time a new book is added to the library.</li><li>4. Implement a method named <b>getTotalBooks()</b> that returns the current count of total books in the library.</li><li>5. Ensure that the <b>MAX_CAPACITY</b> variable cannot be modified after initialization.</li></ol> <p>Write the Java class <b>Library</b> with the given requirements and demonstrate its usage in the <b>Main</b> class by adding books to the library and retrieving the total count.</p>	Medium
<b>8</b>	<p>You are developing a simple banking system where you need to calculate the compound interest earned on a savings account. To accomplish this, you want to define a Java class with static methods for calculating compound interest.</p> <p>Define a Java class named <b>InterestCalculator</b> with the following specifications:</p> <ol style="list-style-type: none"><li>1. Implement a static method named <code>calculateCompoundInterest()</code> that takes three parameters: the principal amount (initial investment), the annual interest rate (in percentage), and the number of years for which the interest is compounded. This method should return the total amount</li></ol>	Medium

***Sometimes later becomes never. DO IT NOW!***



## SELF PRACTICE

### SDE Readiness Training

	<p>including the compound interest, calculated using the formula: <math>\text{Total Amount} = \text{Principal} \times (1 + \frac{\text{Annual Interest Rate}}{100})^{\text{Number of Years}}</math></p> <p><math>\text{Total Amount} = \text{Principal} \times (1 + \frac{100 \times \text{Annual Interest Rate}}{100})^{\text{Number of Years}}</math></p> <p>2. Implement another static method named <code>calculateInterestOnly()</code> that takes the same parameters as <code>calculateCompoundInterest()</code>, but it only calculates the compound interest earned over the given number of years. This method should return the compound interest amount, calculated as: <math>\text{Compound Interest} = \text{Principal} \times ((1 + \frac{\text{Annual Interest Rate}}{100})^{\text{Number of Years}} - 1)</math></p> <p>Write the Java class <code>InterestCalculator</code> with the given requirements and demonstrate its usage in the <code>Main</code> class by calculating compound interest and compound interest only for different scenarios.</p>	
9	<p>You are tasked with developing a simple application to manage employee information for a company. To maintain data integrity and ensure security, you need to encapsulate the employee details within a Java class and provide controlled access using getter and setter methods.</p> <p>Define a Java class named <code>Employee</code> with the following specifications:</p> <ul style="list-style-type: none"><li>Private instance variables to store the employee's ID, name, age, and salary.</li><li>Public getter and setter methods for each instance variable to provide controlled access to the employee details. Ensure that the setter methods validate the input before assigning it to the instance variables:<ul style="list-style-type: none"><li>i. The employee ID must be a positive integer.</li><li>ii. The employee's name cannot be empty.</li><li>iii. The employee's age must be a positive integer.</li><li>iv. The employee's salary must be a non-negative value.</li></ul></li></ul> <p>Implement a method named <code>raiseSalary()</code> that takes a percentage increase as input and raises the employee's salary accordingly.</p> <p>Implement a method named <code>displayInfo()</code> that displays all the employee details.</p> <p>Write the Java class <code>Employee</code> with the given requirements and demonstrate its usage in the <code>Main</code> class by creating an employee, updating their details, giving them a salary raise, and displaying their information.</p>	Medium

***Sometimes later becomes never. DO IT NOW!***



11	<p>Write the Java implementation for a class named '<b>TaxOnSalary</b>' to calculate tax on salary. The class <b>TaxOnSalary</b> is described as follows:</p> <p><b>Attributes:</b></p> <ul style="list-style-type: none"><li>(i) salary: double // salary to calculate tax</li><li>(ii) isPANsubmitted:boolean // PAN submission status</li></ul> <p><b>Methods:</b></p> <p>The class supplies the operation(s) as per the following specification:</p> <ul style="list-style-type: none"><li>a) A TaxOnSalary instance can be created either by supplying the value for the instance field isPANsubmitted OR without supplying value for any field. If TaxOnSalary instance is created by providing the value for isPANsubmitted then the value for salary is initialized with 1000.00.However it can be reinitialized through the method inputSalary() [which is described below] . If TaxOnSalary instance is created without supplying value for any field, then value for salary and isPANsubmitted is by default initialized to 0.0 and false respectively.</li><li>b) Accessor methods(s) are provided for every instance field.</li><li>c) A method for computing the tax based on salary [caculateTax() : double] is supplied.</li></ul> <p>The tax is calculated as per the rules shown below:</p> <ul style="list-style-type: none"><li>a. if salary &lt; 180000 and isPANsubmitted = true, then tax payable is zero</li><li>b. if salary &lt; 180000 and isPANsubmitted = false, then tax payable is 5% of the salary</li><li>c. if 180000 &lt; salary &lt; 500000, then tax payable is 10% of the salary</li><li>d. if 500000 &lt; salary &lt; 1000000, then tax payable is 20% of the salary</li><li>e. if 1000000 &lt; salary, then tax payable is 30% of the salary</li></ul> <p>A method named inputSalary() is supplied to read the value for the salary as an input from the user [consider reading this value from keyboard] and to assign the value to the corresponding instance variable salary. <b>Write a Test class named TestTax.java which</b></p>	Medium
----	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	--------

***Sometimes later becomes never. DO IT NOW!***



	<p>a) Creates two instances of tax1 and tax2 of the class TaxOnSalary with different initializations [see point (a) in the description of Methods].</p> <p>b) Takes salary as an input from the user [using keyboard] for both the instances tax1 and tax2.</p> <p>c) Calculate and display tax for both the instance tax1 and tax2</p>	
<b>12</b>	<p>Design a Java program to model bank accounts, including standard accounts, savings accounts, and current accounts, using inheritance. The program should allow users to perform basic banking operations like deposit, withdrawal, and display account information. The savings accounts should have an additional feature of earning interest on the account balance, and the current accounts should have an overdraft limit to prevent overdrawing.</p> <ol style="list-style-type: none"> <li>Create a class called "Account" with the following attributes and methods: <b>Attributes:</b> <ul style="list-style-type: none"> <li>accountNumber (String)</li> <li>accountName(String)</li> <li>balance (double)</li> </ul> <b>Methods:</b> <ul style="list-style-type: none"> <li>Account constructor: Accepts an account number and initial balance as parameters and initializes the account.</li> <li>deposit(double amount): Adds the given amount to the account balance.</li> <li>withdraw(double amount): Subtracts the given amount from the account balance, ensuring the balance does not go negative.</li> <li>getAccountNumber(): Returns the account number.</li> <li>getAccountName(): Returns the account holder name.</li> <li>getBalance(): Returns the current account balance.</li> <li>displayInfo(): Displays the account number and current balance.</li> </ul> </li> <li>Create a class called "<b>SavingsAccount</b>" that inherits from the "Account" class. The "<b>SavingsAccount</b>" should have an additional attribute and method:</li> </ol>	Medium

***Sometimes later becomes never. DO IT NOW!***



	<p><b>Additional Attribute:</b></p> <ul style="list-style-type: none"> <li>interestRate (double)</li> </ul> <p><b>Additional Method:</b></p> <ul style="list-style-type: none"> <li>addInterest(): Adds interest to the account balance based on the interest rate.</li> </ul> <p>3. Create a class called <b>"CurrentAccount"</b> that inherits from the "Account" class. The <b>"CurrentAccount"</b> should have an additional attribute and method:</p> <p><b>Additional Attribute:</b></p> <ul style="list-style-type: none"> <li>overdraftLimit (double)</li> </ul> <p><b>Additional Method:</b></p> <ul style="list-style-type: none"> <li>getOverdraftLimit(): Returns the overdraft limit for the current account.</li> </ul> <p>4. In the "SavingsAccount" class, override the "displayInfo()" method from the base class to include the interest rate in the account information display.</p> <p>5. In the "CurrentAccount" class, override the "displayInfo()" method from the base class to include the overdraft limit in the account information display.</p> <p>6. In a separate class, create a main method to demonstrate the functionality of the "Account", "SavingsAccount", and "CurrentAccount" classes:</p> <ul style="list-style-type: none"> <li>Create a standard account, a savings account, and a current account with initial balances and specific interest rates, and overdraft limits.</li> <li>Perform deposit and withdrawal operations on all accounts.</li> <li>Display the account information after each transaction, including the interest rate for the savings account and the overdraft limit for the current account.</li> <li>For the savings account, add interest and display the updated balance.</li> <li>Ensure that the program correctly handles withdrawals that exceed the account balance and overdraft limit.</li> </ul>	
<b>13</b>	<p>You are tasked with implementing method overloading in the <b>Employee</b> class, which models an employee in an organization. The</p>	

***Sometimes later becomes never. DO IT NOW!***



	<p><b>Employee</b> class already has attributes such as <b>ID</b>, <b>name</b>, and <b>salary</b>. You need to implement the following methods in the <b>Employee</b> class using method overloading:</p> <ol style="list-style-type: none"><li>1. <b>calculateYearlySalary</b>: This method should calculate and return the employee's yearly salary based on their monthly salary. The method should accept the monthly salary as a parameter.</li><li>2. <b>calculateYearlySalary</b>: This method should calculate and return the employee's yearly salary based on their daily salary. The method should accept the daily salary and number of days in a year as a parameter.</li><li>3. <b>calculateYearlySalary</b>: This method should calculate and return the employee's yearly salary based on their hourly salary and the number of hours worked per day. The method should accept the hourly salary and hours worked per day as parameters.</li></ol> <p><b>toString</b>: This method should print the employee's ID, name, and salary</p>	
<b>14</b>	<p>You have been hired as a software developer to create a system for managing different types of vehicles. The system should utilize the concepts of polymorphism and interfaces in Java. Your task is to design and implement the following:</p> <ol style="list-style-type: none"><li>1. Create an interface called <b>Vehicle</b> that declares the following methods:<ul style="list-style-type: none"><li>• <b>void start ()</b>: This method should start the vehicle.</li><li>• <b>void stop ()</b>: This method should stop the vehicle.</li></ul></li><li>2. Create three classes that implement the <b>Vehicle</b> interface:<ul style="list-style-type: none"><li>• <b>Car</b>: This class represents a car and should provide an implementation for the <b>start()</b> and <b>stop()</b> methods specific to a car.</li><li>• <b>Motorcycle</b>: This class represents a motorcycle and should provide an implementation for the <b>start ()</b> and <b>stop()</b> methods specific to a motorcycle.</li><li>• <b>Truck</b>: This class represents a truck and should provide an implementation for the <b>start()</b> and <b>stop()</b> methods specific to a truck.</li></ul></li></ol>	

***Sometimes later becomes never. DO IT NOW!***





3. In the **main** method of your program, create an array of **Vehicle** objects. Include instances of cars, motorcycles, and trucks in the array.
4. Iterate over the array and call the **start()** method for each vehicle to start it.
5. Iterate over the array again and call the **stop()** method for each vehicle to stop it.

Your program should demonstrate the concept of polymorphism by treating each vehicle object as an instance of the **Vehicle** interface, allowing the same method calls (**start()** and **stop()**) to be made on different types of vehicles.

Ensure that your program is logically structured, follows naming conventions, and includes appropriate comments to explain the purpose of each class and method.

Note: Focus on implementing the polymorphism concept and interface methods. You do not need to implement additional functionality such as vehicle details or specific actions in the **start()** and **stop()** methods.

***Sometimes later becomes never. DO IT NOW!***