TP07

Java Class and Objects (part 2)

Remark

1. Overloading methods – Are methods which has the same name but different arguments. Example:

```
public class Compare {

   int max(int a, int b) {
        return (a > b) ? a : b;
    }

    double max(double a, double b) {
        return (a > b) ? a : b;
    }

    String max(String a, String b) {
      return (a.compareTo(b) > 0) ? a : b;
   }

}
```

Method `max()` is called overloading method, because there are 2 methods with the same name `max`.

```
public class ShapeAreaCalculator {
   public double calculateArea(double radius) {
        return Math.PI * radius * radius;
    }

    public double calculateArea(double width, double height) {
        return width * height;
    }

    public double calculateArea(double side1, double side2, double side3) {
      double s = (side1 + side2 + side3) / 2;
        // Calculate the area using Heron's formula
        double area = Math.sqrt(s * (s - side1) * (s - side2) * (s - side3));
        return area;
    }
}
```

Method `calculateArea()` is called overloading method.

2. Class Variable – a variable is common to all instance of this class. It means that all object instance of this class share this variable only one address memory. In Java, we use static keyword place in front of variable that we want it to become class variable.

Example:

```java
public class User {
    static int user_count = 0;
    String username;
    public User(String username){
        this.username = username;
        user_count++;
    }
}
```

Every time, we create a User object, the user_count variable is increased by 1. The variable user_count is

```java
public class UserTest {
    public static void main(String[] args) {
        System.out.println("user_count = "+User.user_count);
        User u1 = new User("Dara");
        System.out.println("user_count = "+User.user_count);
        User u2 = new User("Sotha");
        System.out.println("user_count = "+User.user_count);
        User u3 = new User("Sothea");
        System.out.println("user_count = "+User.user_count);
        User u5 = new User("Nida");
        System.out.println("user_count = "+User.user_count);
    }
}
```

created and initialized only one time. Output of program above:

```
user_count = 0
user_count = 1
user_count = 2
user_count = 3
user_count = 4
```

3. Class Method – method that can be called without instantiating object of that class. In Java, we use static keyword in method declaration. Example: Math.abs(int), Math.sqrt(double)
4. Constants – variable that is not changeable at runtime. Example: Math.PI
5. Object Destruction – a special method named `finalize()` will be called automatically when an object is removed from memory.

TP07.1. Rectangle

We have a class Rectangle as below:

```java
public class Rectangle {
    int width;
    int height;
    public Rectangle(int width, int height) {
        this.width = width;
        this.height = height;
    }
    public int calculatePerimeter(){
        return (width + height) * 2;
    }
    public int calculateSurface(){
        return width * height;
    }
}
```

Create a class RectangleTest that is a Java application that test:

1. Create an instance of class Rectangle (call its constructor)
2. Display perimeter of it (the new created rectangle object)
3. Display surface of it (the new created rectangle object)

TP07.2. Bank Account

Create a class named BankAccount with class variable totalAccounts (static) and instance variables accountHolder and balance.

- Implement a constructor to initialize the object variables and increment the totalAccounts count.
- Implement methods displayAccountInfo(), deposit(), and withdraw() to display account information, deposit money, and withdraw money, respectively.
- Implement a static method displayTotalAccounts() to display the total number of bank accounts.

In the Main class, create two bank account objects, display their information, and perform deposit and withdrawal transactions.

TP07.3. SMS Encrypt

Create a Java class represents SMS. SMS class contains:

- Attributes: Subject, From phone number, Receiver number, Type (Text, MMS), content, and status (new, read)
- Constructor (s): suggests 3 constructors

Then create a class named SMSList, that represents list of SMS and manage SMS in and out:

- Attributes: ArrayList<SMS>, and static field max_characters_per_sms
- Operations/Methods: suggests 3 overloaded methods

Then write a program that will display a menu:

1. Send new SMS with Encrypted content using password method
2. View SMS detail
3. List SMSes
4. Remove SMSes by index
5. Quit

TP07.4. Shape Hierarchy

Create an abstract class Shape with attributes like name and abstract methods like calculateArea(), calculatePerimeter() and displayShape().

Create concrete classes Circle, Rectangle, and Triangle that inherit from the Shape class. Implement the abstract methods with specific formulas for each shape.
Create instances of Circle, Rectangle, and Triangle. Calculate the area, perimeter and display details for each shape.

TP07.5. ATM Simulator

Create an ATM simulator program using Java with the implementation of a **BankAccount** class. The program should allow users to log in using a card number and PIN, and perform operations such as viewing the balance, depositing, and withdrawing money.

Requirements:

- **BankAccount Class:**

  Implement a **BankAccount** class with the following attributes:

  - **cardNumber** (String)

- **cardHolder** (String)
- **pin** (String)
- **balance** (double)

Implement the following methods in the **BankAccount** class:

- **displayAccountInfo()**: Display the card holder's name, card number, and balance.
- **deposit(double amount)**: Deposit the specified amount into the account.
- **withdraw(double amount)**: Withdraw the specified amount from the account, if sufficient funds are available.
- **getBalance()**: Return the current balance.

- **ATM Simulator:**
  - In the **main** method, create an ArrayList to store multiple **BankAccount** objects.
  - Create at least two **BankAccount** objects and add them to the ArrayList.
  - Implement a menu-driven ATM simulator with the following options:

    Login:

    - Prompt the user to enter a card number and PIN.
    - Verify the entered card number and PIN against the **BankAccount** objects in the ArrayList.
    - If the login is successful, allow the user to perform ATM operations.

    ATM Operations Menu (available after login):

    - View Balance
    - Deposit
    - Withdraw
    - Logout

    Operations:

    - View Balance: Display the account information (card holder, card number, and balance).
    - Deposit: Prompt the user to enter the deposit amount and update the account balance.
    - Withdraw: Prompt the user to enter the withdrawal amount and update the account balance, if sufficient funds are available.
    - Logout: Exit the ATM operations.

    Exit:

    - Allow the user to exit the ATM simulator.