**Lab 3: Encapsulation – Classes – Objects – Constructors**

**Objectives of Lab 5:**

In this lab, you will apply the concepts covered in Lecture 5. By the end of this lab, you will:

* Understand the concept of encapsulation
* Understand the concept of classes and objects in Java.
* Use constructors to initialize object states.
* Implement and use setter and getter methods.
* Explore memory allocation for multiple objects from the same class.

**Exercise 1: Simple Encapsulation Example**

**Task:**  
Create a class Person with the following private attributes:

* name (String)
* age (int)

Write setter and getter methods to access and modify these attributes. Ensure that the age cannot be set to a negative number using validation logic in the setter method.

In the main method, create a Person object, set the name and age using the setter methods, and then print the values using the getter methods.

**Expected Output:**  
The details of the Person object, with validation ensuring the age is non-negative.

**Exercise 2: Creating and Using Classes**

**Task:**  
Create a Car class with the following attributes:

* model (String)
* color (String)
* speed (int)

In the main method, create three objects of the Car class representing different car models. Use the constructor to initialize the attributes. Print the details of each car.

**Expected Output:**  
Details of each car (model, color, and speed).

**Exercise 3: Implementing Setters and Getters**

**Task:**  
Modify the Car class from Exercise 1 to use private attributes and provide setter and getter methods for each. Use the setter method to validate that the speed cannot be negative. Print the car details using the getter methods.

**Expected Output:**  
Car details printed using getter methods, with validation ensuring speed is non-negative.

**Exercise 4: Constructor Overloading**

**Task:**  
Create a class BankAccount with the following private attributes:

* accountNumber (String)
* balance (double)
* accountHolder (String)

Implement two constructors:

* A constructor with all three attributes as parameters.
* A constructor that only takes accountNumber and sets the default balance to 0 and accountHolder to "Unknown".

Create objects using both constructors and print the account details.

**Expected Output:**  
Account details printed for objects created with both constructors.

**Exercise 5: Using ‘this’ Keyword**

**Task:**  
Modify the BankAccount class from Exercise 3 to include setter methods. Use the this keyword within the setters to assign values to the attributes. Add a condition to check that the balance is not negative when setting the value. Print the account details using getter methods.

**Expected Output:**  
Account details printed with valid values using getter methods, with this keyword used in setters.

**Exercise 6: Memory Concepts and Multiple Objects**

**Task:**  
Write a Java program that creates multiple Student objects from the Student class. Each object should have unique attributes such as name, studentID, and GPA. Demonstrate that each object occupies a unique place in memory by printing the memory address of each object.

**Expected Output:**  
Details of each student object, along with their unique memory addresses.

**Exercise 7: Account Management Application**

**Task:**  
Extend the BankAccount class from the previous exercises by adding methods for depositing and withdrawing funds:

* deposit(double amount)
* withdraw(double amount)

Ensure the withdraw method checks if there are sufficient funds before making a withdrawal. Print a message if the withdrawal is successful or fails due to insufficient funds.

**Expected Output:**  
Account balance after deposits and withdrawals, with a message if funds are insufficient.

**Exercise 8: Constructor Chaining**

**Task:**  
In the BankAccount class, implement constructor chaining. If only the accountNumber is provided, call the constructor that sets the default balance and accountHolder. Create objects using different constructors to demonstrate this feature.

**Expected Output:**  
Account details printed, showing how constructors are called through chaining.

**Exercise 9: VIP Customer Class**

**Task:**  
Create a VIPCustomer class with the following attributes:

* name (String)
* creditLimit (double)
* email (String)

Implement three overloaded constructors:

1. A constructor that takes all three attributes.
2. A constructor that takes two attributes (name and creditLimit) and sets a default email.
3. A constructor that takes only the name and sets default values for the other two attributes.

Create objects using each constructor and print their details.

**Expected Output:**  
Details of VIP customers, demonstrating how different constructors initialize the objects.

**Exercise 10: Bank Application with Constructors**

**Task:**  
Create a Bank application where the user can create a BankAccount object using the constructor. The program should automatically generate an account number (use a random number generator for this), and the user should input the account holder's name and an initial deposit. Print the account details once the account is created.

**Expected Output:**  
New bank account details with auto-generated account number and initial deposit.

**Exercise 11: Implementing a Simple Class for Employee**

**Task:**  
Create an Employee class with attributes like name, ID, and salary. Implement getter and setter methods for each attribute and a method increaseSalary that increases the employee's salary by a certain percentage. Create an object of the Employee class and call the increaseSalary method. Print the employee details before and after the salary increase.

**Expected Output:**  
Employee details before and after salary increase.

**Exercise 12: Handling Multiple Objects in a Loop**

**Task:**  
Create an array of Student objects in the main method. Use a loop to initialize the objects with different names and grades, then print their details.

**Expected Output:**  
Details of all students, initialized and printed using a loop.

**Exercise 13: Creating a Method to Calculate GPA**

**Task:**  
Create a method calculateGPA in the Student class that takes an array of marks (integers) as input and calculates the GPA based on a given formula. Call this method from the main method to calculate and print the GPA for a student.

**Expected Output:**  
The GPA for a student based on their marks.

**Exercise 14: Validating Data with Setters**

**Task:**  
Modify the BankAccount class to validate the input data for balance. Ensure that the balance cannot be negative using the setter method. Print appropriate messages if invalid data is entered.

**Expected Output:**  
Validation messages if invalid data is provided, with valid account details printed.

**Exercise 15: Constructor Validation**

**Task:**  
Modify the VIPCustomer class to validate the creditLimit in the constructor. If the credit limit is negative, set it to 0 and print a warning message. Create objects with different credit limits and demonstrate this validation.

**Expected Output:**  
Credit limit validation messages, with details of the VIPCustomer objects.

**Exercise 16: Final Bank Application with All Features**

**Task:**  
Combine all features from the previous exercises into a single Bank application. The program should:

1. Allow the user to create a BankAccount object with an auto-generated account number.
2. Use constructors to initialize the account.
3. Implement methods to deposit and withdraw funds.
4. Validate the data entered for balance and withdrawals.

**Expected Output:**  
A fully functional bank application with user interactions, account creation, deposits, withdrawals, and validation.