**Assignment 1. Java Programming Language, CSE3040 & AIE3052**

Student Name: 이예준

Student ID: 20212022

**Q1. Vehicle management system.**

**Task Requirements:**

1. Create a base class named Vehicle. This class should have private fields for common vehicle attributes: brand, model, and year.

* Use encapsulation to control access to these fields by providing appropriate getter and setter methods.
* The constructor should take the brand, model, and year as parameters and initialize the fields.
* Override the toString() method to print the vehicle's details in a readable format.

1. Create two subclasses: Car and Motorcycle, which both inherit from the Vehicle class.

* The Car class should have an additional field seats (number of seats). Provide getter and setter methods for this field.
* The Motorcycle class should have a field hasSidecar (whether the motorcycle has a sidecar). Provide getter and setter methods for this field.

1. Implement a custom exception class named InvalidVehicleDetailException to handle invalid vehicle details.

* For example, throw this exception if the year is earlier than 1886, or if the seats number is less than or equal to zero.

1. Create a class named VehicleManager that allows adding, removing, and searching for vehicles.

* Use a list to manage multiple vehicles.
* Throw a custom exception DuplicateVehicleException when attempting to add a vehicle that already exists in the list.
* Throw a custom exception VehicleNotFoundException if a vehicle is searched for but does not exist in the list.

Vehicle Class

public class Vehicle {  
 private String brand;  
 private String model;  
 private int year;  
  
 public Vehicle(String brand, String model, int year) throws InvalidVehicleDetailException {  
 // Fill in this line  
 // Answer  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(year < 1886) throw new InvalidVehicleDetailException("Year must be 1886 or later");  
 this.brand = brand;  
 this.model = model;  
 this.year = year;  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public String getBrand() {  
 return brand;  
 }  
  
 public String getModel() {  
 return model;  
 }  
  
 public int getYear() {  
 return year;  
 }  
  
 public void setYear(int year) throws InvalidVehicleDetailException {  
 // Fill in the if statement and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(year < 1886) throw new InvalidVehicleDetailException("Year must be 1886 or later");  
 this.year = year;  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 @Override  
 public String toString() {  
 // Fill in return statement  
 // Answer  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 return String.format("%s %s (%d)", brand, model, year);

///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }

}

Car Class

public class Car extends Vehicle{  
 private int seats;  
  
 public Car(String brand, String model, int year, int seats) throws InvalidVehicleDetailException {  
 super(brand, model, year);  
 // Fill in this line  
 // Answer  
 //////////////////////////////////////////////////////////////////////////////////////////////  
 if(seats <= 0) throw new InvalidVehicleDetailException("Number of seats must be positive");  
 this.seats = seats;  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
  
 }  
  
 public int getSeats() {  
 return seats;  
 }  
  
 public void setSeats(int seats) throws InvalidVehicleDetailException {  
 // Fill in the if statement and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(seats <= 0) throw new InvalidVehicleDetailException("Number of seats must be positive");  
 this.seats = seats;  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 @Override  
 public String toString() {  
 // Fill in return statement  
 // Answer  
  
 //////////////////////////////////////////////////////////////////////////////////////////////  
 return String.format("Car: %s, Seats: %d", super.toString(), seats);  
  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
}

Motorcycle Class

public class Motorcycle extends Vehicle {  
 private boolean hasSidecar;  
  
 public Motorcycle(String brand, String model, int year, boolean hasSidecar) throws InvalidVehicleDetailException {  
 super(brand, model, year);  
 this.hasSidecar = hasSidecar;  
 }  
  
 public boolean isHasSidecar() {  
 return hasSidecar;  
 }  
  
 public void setHasSidecar(boolean hasSidecar) {  
 this.hasSidecar = hasSidecar;  
 }  
  
 @Override  
 public String toString() {  
 // Fill in return statement  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 return String.format("Motorcycle: %s, Has Sidecar: %s", super.toString(), hasSidecar ? "Yes" : "No");  
  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
}

Custom Exception Classes

public class InvalidVehicleDetailException extends Exception {

public InvalidVehicleDetailException(String message) {

super(message);

}

}

public class DuplicateVehicleException extends Exception {

public DuplicateVehicleException(String message) {

super(message);

}

}

public class VehicleNotFoundException extends Exception {

public VehicleNotFoundException(String message) {

super(message);

}

}

VehicleManager Class

import java.util.ArrayList;  
import java.util.List;  
  
public class VehicleManager {  
 private List<Vehicle> vehicles = new ArrayList<>();  
  
 public void addVehicle(Vehicle vehicle) throws DuplicateVehicleException {  
 // Fill in the duplicate check and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(vehicles.contains(vehicle)) throw new DuplicateVehicleException("Vehicle already exists");  
 vehicles.add(vehicle);

///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public Vehicle searchVehicle(String brand, String model) throws VehicleNotFoundException {  
 // Fill in the search logic and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 for(Vehicle v : vehicles){  
 if(v.getBrand().equals(brand) && v.getModel().equals(model)) return v;  
 }  
 throw new VehicleNotFoundException("Vehicle not found");  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public void removeVehicle(Vehicle vehicle) throws VehicleNotFoundException {  
 // Fill in the remove logic and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(!vehicles.remove(vehicle)) throw new VehicleNotFoundException("Vehicle not found");  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public void printAllVehicles() {  
 // Fill in the print logic  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 for(Vehicle v : vehicles){  
 System.outprintln(v.toString());  
 }  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
}

**Q2. Bank account management system**

**Task Requirements:**

1. Create a base class named BankAccount. This class should have private fields for accountNumber and balance.

* The constructor should take the account number and an initial balance as parameters to initialize the fields.
* Implement methods deposit() and withdraw() to perform deposit and withdrawal operations. If a withdrawal amount exceeds the available balance, throw a custom exception InsufficientBalanceException.

1. Create two subclasses: SavingsAccount and CheckingAccount, which both inherit from BankAccount.

* SavingsAccount should have an additional field interestRate. Implement a method applyInterest() that adds interest to the account's balance.
* CheckingAccount should have an additional field overdraftLimit. Modify the withdraw() method so that the account can overdraw up to the overdraft limit.

1. Implement a BankManager class to manage multiple bank accounts.

* When adding a new account, throw a custom exception DuplicateAccountException if an account with the same account number already exists.
* Implement methods to search for an account by account number and perform deposit and withdrawal operations. If an account is not found, throw an AccountNotFoundException.
* Ensure that the balance can only be modified through deposit() and withdraw() methods to maintain encapsulation.

BankAccount Class

public class BankAccount {  
 private String accountNumber;  
 private double balance;  
  
 public BankAccount(String accountNumber, double initialBalance) {  
 this.accountNumber = accountNumber;  
 this.balance = initialBalance;  
 }  
  
 public String getAccountNumber() {  
 return accountNumber;  
 }  
  
 public double getBalance() {  
 return balance;  
 }  
  
 public void deposit(double amount) {  
 // Fill in deposit logic  
 // Answer: balance += amount;  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 balance += amount;  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
  
 }  
  
 public void withdraw(double amount) throws InsufficientBalanceException {  
 // Fill in the withdraw logic and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(balance < amount) throw new InsufficientBalanceException("Insufficient balance");  
 balance -= amount;  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
}

SavingAccount Class

public class SavingAccount extends BankAccount {  
 private double interestRate;  
  
 public SavingAccount(String accountNumber, double initialBalance, double interestRate) {  
 super(accountNumber, initialBalance);  
 this.interestRate = interestRate;  
 }  
  
 public void applyInterest() {  
 // Fill in the interest calculation and deposit logic  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 deposit(getBalance() \* interestRate);  
  
  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
}

CheckingAccount Class

public class CheckingAccount extends BankAccount {  
 private double overdraftLimit;  
  
 public CheckingAccount(String accountNumber, double initialBalance, double overdraftLimit) {  
 super(accountNumber, initialBalance);  
 this.overdraftLimit = overdraftLimit;  
 }  
  
 @Override  
 public void withdraw(double amount) throws InsufficientBalanceException {  
 // Fill in the overdraft check and withdraw logic  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(getBalance() + overdraftLimit < amount) throw new InsufficientBalanceException("Exceeds overdraft limit");  
 super.deposit(-amount);  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
}

Custom Exception Class

public class InsufficientBalanceException extends Exception {

public InsufficientBalanceException(String message) {

super(message);

}

}

public class DuplicateAccountException extends Exception {

public DuplicateAccountException(String message) {

super(message);

}

}

public class AccountNotFoundException extends Exception {

public AccountNotFoundException(String message) {

super(message);

}

}

BankManager Class

import java.util.HashMap;  
import java.util.Map;  
public class BankManager {  
 private Map<String, BankAccount> accounts = new HashMap<>();  
  
 public void addAccount(BankAccount account) throws DuplicateAccountException {  
 // Fill in the duplicate check logic and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 if(accounts.containsKey(account.getAccountNumber()))  
 throw new DuplicateAccountException("Account already exists");  
 accounts.put(account.getAccountNumber(), account);  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public BankAccount findAccount(String accountNumber) throws AccountNotFoundException {  
 // Fill in the search logic and throw exception if necessary  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 BankAccount account = accounts.get(accountNumber);  
 if(account == null) throw new AccountNotFoundException("Account not found");  
 return account;  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public void deposit(String accountNumber, double amount) throws AccountNotFoundException {  
 // Fill in deposit logic  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 findAccount(accountNumber).deposit(amount);  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public void withdraw(String accountNumber, double amount) throws AccountNotFoundException, InsufficientBalanceException {  
 // Fill in withdraw logic  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 findAccount(accountNumber).withdraw(amount);  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
  
 public void printAllAccounts() {  
 // Fill in print logic  
 // Answer:  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 for (BankAccount account : accounts.values()) {  
 System.out.printf("[Account Type: %s] Account Number: %s, Balance: %.2f%n", account.getClass().getSimpleName(), account.getAccountNumber(), account.getBalance());  
 }  
 ///////////////////////////////////////////////////////////////////////////////////////////////////////  
 }  
}