

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

Student Name: 원대호

Student ID: 20212021

## **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.
2. 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.
3. 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.
4. 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 {  
        if (year < 2000) {  
            throw new InvalidVehicleDetailException("Invalid YEAR");  
        }  
        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 {  
        if (year < 2000) {  
            throw new InvalidVehicleDetailException("Not Valid YEAR");  
        }  
        this.year = year; → 37[3] ↓ Invalid YEAR  
    }  
  
    @Override  
    public String toString() {  
        return "Vehicle [1. Brand = " + brand + ", 2. Model = " + model + ", 3. Year = " + 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);  
  
        if (seats < 1) {  
            throw new InvalidVehicleDetailException("Seats must be greater than zero.");  
        }  
        this.seats = seats;  
    }  
  
    public int getSeats() {  
        return seats;  
    }  
  
    public void setSeats(int seats) throws InvalidVehicleDetailException {  
        if (seats < 1) {  
            throw new InvalidVehicleDetailException("Seats must be greater than zero.");  
        }  
        this.seats = seats  
    }  
  
    @Override  
    public String toString() {  
        return "Car [1. Brand = " + getBrand() + ", 2. Model = " + getModel() + ", 3. Year = " + getYear() + ", 4. Seats = " + seats +  
    "]";  
    }  
}
```

Error:

Error:

Error:

## 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() {  
        return "Motorcycle [Brand = " + getBrand() + ", Model = " + getModel() + ", Year = " + getYear() + ", Has Sidecar = " +  
hasSidecar + "]";  
    }  
}
```

1.      2.      3.      4.

## 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 {
        for (int i = 0; i < vehicles.size(); i++) {
            Vehicle V = vehicles.get(i);
            if (V.getBrand().equals(vehicle.getBrand()) && V.getModel().equals(vehicle.getModel()) && V.getYear() == vehicle.getYear()) {
                throw new DuplicateVehicleException("Error: Vehicle already exists in the list.");
            }
        }
        vehicles.add(vehicle);
    }

    public Vehicle searchVehicle(String brand, String model) throws VehicleNotFoundException {
        for (int i = 0; i < vehicles.size(); i++) {
            Vehicle v = vehicles.get(i);
            if (v.getBrand().equals(brand) && v.getModel().equals(model)) {
                return v;
            }
        }
        throw new VehicleNotFoundException("Vehicle [Brand = " + brand + ", Model = " + model + "] not found.");
    }

    public void removeVehicle(Vehicle vehicle) throws VehicleNotFoundException {
        if (!vehicles.remove(vehicle)) {
            throw new VehicleNotFoundException("Error : Vehicle not found in the list.");
        }
    }

    public void printAllVehicles() {
        for (int i = 0; i < vehicles.size(); i++) {
            Vehicle V = vehicles.get(i);
            System.out.println(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.
2. 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.
3. 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) {  
        balance += amount;  
    }  
  
    public void withdraw(double amount) throws InsufficientBalanceException {  
        if (amount > balance) {  
            throw new InsufficientBalanceException("Error : Insufficient balance");  
        }  
        balance -= amount;  
    }  
}
```

## SavingAccount Class

```
public class SavingsAccount extends BankAccount {  
    private double interestRate;  
  
    public SavingsAccount(String accountNumber, double initialBalance, double interestRate) {  
        super(accountNumber, initialBalance);  
        this.interestRate = interestRate;  
    }  
  
    public void applyInterest() {  
        double interest = getBalance() * interestRate;  
        deposit(interest);  
    }  
}
```

## 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 {  
        if (amount > getBalance() + overdraftLimit) {  
            throw new InsufficientBalanceException("Withdrawal denied: Exceeds overdraft limit.");  
        }  
        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 {
        if (accounts.containsKey(account.getAccountNumber())) {
            throw new DuplicateAccountException("Error : Account already exists.");
        }
        accounts.put(account.getAccountNumber(), account);
    } add

    public BankAccount findAccount(String accountNumber) throws AccountNotFoundException {
        if (!accounts.containsKey(accountNumber)) {
            throw new AccountNotFoundException("Error : Account not found.");
        }
        return accounts.get(accountNumber);
    } return

    public void deposit(String accountNumber, double amount) throws AccountNotFoundException {
        BankAccount account = findAccount(accountNumber);
        account.deposit(amount);
    }

    public void withdraw(String accountNumber, double amount) throws AccountNotFoundException, InsufficientBalanceException {
        BankAccount account = findAccount(accountNumber);
        account.withdraw(amount);
    }

    public void printAllAccounts() {
        for (BankAccount account : accounts.values()) {
            System.out.println("1. Account Number : " + account.getAccountNumber() + " 2. Balance : " + account.getBalance());
        }
    }
}
```