Informatics I – EProg HS15

Exercise 8

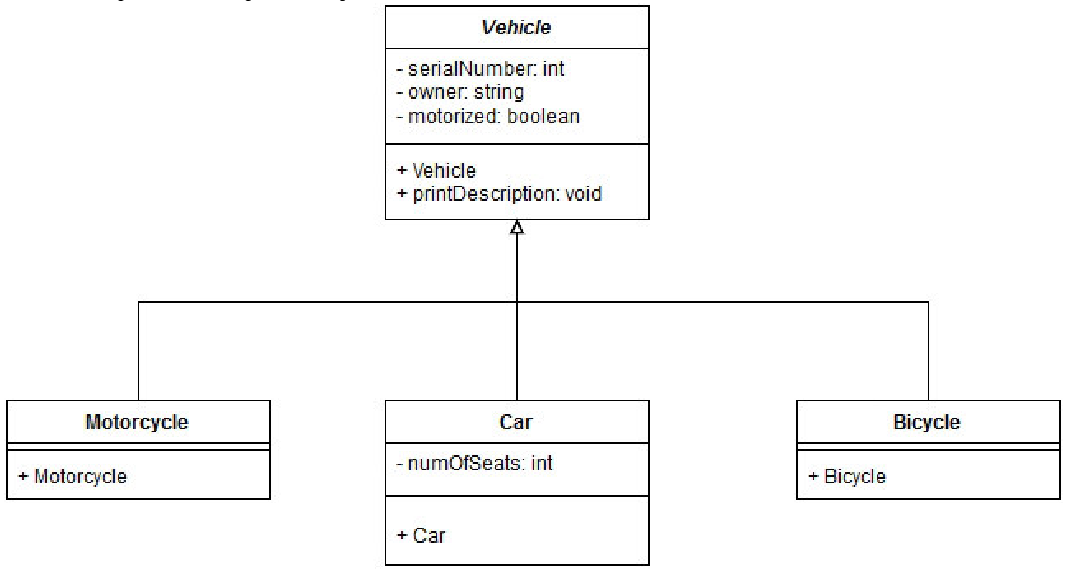
1 Task: Inheritance

1.1 Learning Objectives

1. You are able to read UML diagrams and write the appropriate code.
2. You understand the principles of inheritance and polymorphism.

1.2 Assignment

The following UML diagram is given:



a) UML Notation

Remarks about the notation in the above class diagram:

- Abstract classes/methods are written in italic (see `Vehicle`)
- Arrows with a white summit show the inheritance hierarchy. `Motorcycle`, `Car` and `Bicycle` are therefore subclasses of `Vehicle`.
- In the topmost section of a box the class name is written. The second part contains the attributes and the last one specifies the methods of the class.
- Attributes are illustrated according to the following notation:

  ```
  <+/-> <name> : <type>
  ```

  where a `+` stands for the visibility modifier `public` and a `-` for `private`.
- Methods are illustrated according to the following notation:

  ```
  <+/-> <name> : <return-type>
  ```

  with the same meaning of `+` and `-`.

b) Implementation

1. Implement the classes `Vehicle`, `Motorcycle`, `Car`, and `Bicycle` according to the UML class diagram. Create accessor methods for the private fields in `Person`. The method `printDescription` should print a short description like the following:

   This vehicle's owner's name is John Doe, it's serial number is 184382 and it is motorized.

2. In the class `Bicycle` override the method `setSerialNumber(int serialNumber)` such that it prints a message to the console if one tries to set a serial number with more than 5 digits (we assume that serial numbers of bicycles consist of fewer than 6 digits).
3. Create a `TestDriver` with a `main()`-method that creates various objects of the implemented classes and put them into an array. Afterwards use a loop to iterate over the array and execute `printDescription` for all the objects in the array.
4. Add an additional method to the class `Motorcycle` that does not appear in all the other classes (e.g. a method `burnout()`). This method should print a simple statement to the console (e.g. 'My tires are melting!'). Use again the TestDriver from 3 to execute the new method. Write down the error that occurs and try to explain why it occurs.

2 Task: Mealy State Machine

2.1 Learning Objectives

1. Practice drawing mealy state machine graphs.
2. Implement a mealy state machine with the *State Design Pattern*.

2.2 Assignment

Strategy games often involve diplomacy between different factions. Depending on the current relationship between two factions, certain actions may lead to different results and can influence their relationship. You want to design a simplified model using a mealy state machine.

a) Modeling

Draw a state automation graph (Automatengraph) according to the following description: The relationship between two factions can be either `FRIENDLY`, `NEUTRAL`, or `HOSTILE`. If in a `FRIENDLY` relationship, a provocation will lead to output `o5` and deteriorate the relationship to `NEUTRAL`. Offering a gift can reestablish a `FRIENDLY` relationship and causes the output `o4`. Independently of the current relationship, an attack action will always lead to a `HOSTILE` relationship. However, the output differs depending on the previous state and is either `o1` when in `FRIENDLY` state before or `o2` when in `NEUTRAL` state before. A `HOSTILE` relationship can be neutralized by offering a gift which leads to output `o3`. Finally, simple chatting won't affect the relationship but leads to output `o6` when in `FRIENDLY` relationship, `o7` when in `HOSTILE` relationship, and `o8` when in `NEUTRAL` relationship.
You only have to model explicitly described state transitions.

b) Implementation

1. Familiarize yourself with the state design pattern by reading this article (german only).
2. Write a class `Faction` that has a field relationship of type `IRelationship` (don't bother, we will create this type in a moment) and create accessor (getter and setter) methods for it.
3. Create a Java interface called `IRelationship` with the `void` methods `provoke`, `gift`, `attack`, and `chat` representing the actions of the mealy automation. Add a method `name` that is designated to return a `String` with name of the relationship (e.g. `FRIENDLY`).
4. Implement the mealy automaton modeled in task a) by writing a class that implements the `IRelationship` interface for each relationship (i.e. state). Each concrete relationship should have a field of type `Faction` and provide a constructor that initializes this field based on a parameter. You might simulate the output by printing the following statements to the console:

```
o1 An act of treasure from out best friends!
o2 Your neutrality was fairly suspicious, we are prepared!
o3 We interpret your concessions as a symbol of good faith, let's have peace.
o4 We are very pleased by your generosity, let's become allied.
o5 This provocation is intolerable, our alliance is broken.
o6 Welcome, we are always pleased to meet you.
o7 Grrr!
o8 Go on. What do you have to say?
```

5. Finish the implementation of `Faction.java` by creating methods for each action available. These methods should delegate "the real work" to the current relationship (i.e. state). Also ensure that each new faction will be initialized with a neutral relationship.
6. Implement a command line interface (CLI) called `FactionRelationshipCLI` in order to test our relationship model. The application should allow to enter the first character of an available action (i.e. `'a'` for `attack`, `'c'` for `chat`, `'g'` for `gift`, `'p'` for `provoke`) to trigger the corresponding action. Invalid characters should be ignored and `'q'` (i.e. quit) should immediately terminate the application. It should be also possible to enter an entire sequence of actions at once. An example usage might be as follows[1]:

   ```
   $ java FactionRelationshipCLITestDriver
   c
   o8 Go on. What do you have to say?
   NEUTRAL
   gca
   o4 We are very pleased by your generosity, let's become allied.
   FRIENDLY
   o6 Welcome, we are always pleased to meet you.
   FRIENDLY
   o1 An act of treasure from out best friends!
   HOSTILE
   stuff
   q
   Goodbye!
   $
   ```

   Hint1: Use the `Scanner` class from the Java-API.
   Hint2: The constructor of `FactionRelationshipCLI` should take a `Faction` instance as parameter. Therefore you have to write a test driver `FactionRelationshipCLITestDriver` which initializes the CLI and starts it e.g. via a `run()` method.
7. In our contemporary model, hostilities can be simply ceased by offering a gift. In order to improve this model, implement a strategy wherein only $1/4$ of the "bribe" attempts are successful (remember the Java `Random` API). Unsuccessful attempts should be ignored.
8. Continuous provocations should also affect the neutral relationship state. Implement a strategy wherein within the `NEUTRAL` relationship the first two provocations are answered with `o9` (e.g. Stop now.) and the third one is answered with `o10` (e.g. That was enough!). In the latter case, the relationship changes from `NEUTRAL` to `HOSTILE`.

[1]The $ sign represents the command line prompt after which the user can enter his commands.

3 Task: An Address Book Application - Part 2

3.1 Learning Objectives

1. Extend the existing fully working address book application from last week.
2. Implement a command line interface (CLI).

3.2 Assignment

Last week, in exercise 7 you have implemented some basic functionality of a classical address book application. In this task, you will modify your code in order to extend and optimize the application. Use *your own code* from last week. Before you start, it is particularly important that you compare your code with the provided solution (as you should always do) to avoid consequential errors.

a) User Interface: Command Line Interface (CLI)

In this week, a user should be able to interact with your application via a simple text-based CLI. Next week, we will implement a more advanced graphical user interface. Therefore, it is important that the user interface knows (i.e. has a reference) the address book but not vice-versa. If you follow this rule, you can extend your application with a graphical interface without modifying your existing address book application code.

1. Write a class `CommandLineInterface` that expects an instance of the address book in its constructor. Naturally, this instance should be stored within an instance variable.
2. The class should have a method `void run()` that reads user input from the command line via a `java.util.Scanner` object.
3. Each time when `run()` is called, the CLI should present a menu to the user (see example below) and store its choice.

   ```
   Make your choice:
   1. Add new contact.
   2. Select existing contact.
   3. Delete current selection.
   4. List all contacts.
   5. Quit the application.
   ```

4. If the user enters `'1'`, the CLI should ask for the firstname and subsequently for the lastname. A corresponding contact should be created via calling the `addContact(String firstName, String lastName)` method of the address book.
5. If the user enters `'2'`, the CLI should show a numbered list of all contacts. Use the unique number of each contact for this purpose. Subsequently, the user can enter a number and may delete the contact out of the address book with `'3'` *Delete the current selection*. Example:

   ```
   Make your choice:
   1. Add new contact.
   2. Select existing contact.
   3. Delete current selection.
   4. List all contacts.
   5. Quit the application.
   $ 2
   ```

```
Select one of the following contacts by number:
---
1414072125401224000. Michael Wuersch
---
1414072134408538000. Emanuel Giger
---
Which contact do you want to select?
$ 1414072125401224000
Make your choice:
1. Add new contact.
2. Select existing contact.
3. Delete current selection.
4. List all contacts.
5. Quit the application.
$ 3
Deleted 'Michael Wuersch'.
Make your choice:
1. Add new contact.
2. Select existing contact.
3. Delete current selection.
4. List all contacts.
5. Quit the application.
```

6. The choice `'4'` and `'5'` should be self-explaining: `'4'` prints all contacts to the command line and `'5'` terminates the address book CLI.