PS Programming Methodology

University of Innsbruck - Department of Computer Science

Blaas A., Ernst M., Frankford E., Huaman Quispe E., Hupfauf B., Kaltenbrunner L., Moosleitner M., Priller S., Simsek U.

universität innsbruck
Department of Computer Science

March 28th, 2022

Sheet 02 – Classes, Objects and Methods

Exercise 1 (Everything is an object) [3 Points]

The goal of this exercise is to get familiar with object orientation and classes. You have to design and implement a Java class that represents a computer. Our simplified version of a computer is composed by several attributes and components. A detailed list of the implementation requirements is provided below. A computer consists of:

- A model identifier and the year of manufacturing (only the year, e.g. 2020).
- A processor, which is composed by a name, processor-type (AMD or Intel), its number of cores and price in cents.
- A hard-disk, which has a name, type (SSD or HDD), a storage capacity (in GB), data-write-rate (in GB/s) and its price in cents.
- Two RAM-slots, where each slot contains information about the concrete model, the amount of memory (in GB), the clock rate (in MHz) and its price in cents.
- A GPU, which consists of a model identifier, the manufacturers' name and its price in cents.

Now elaborate on the following tasks:

a) Implement the computer-class according to the given requirements. Think about which representation is adequate to implement the required properties (objects, enums, primitive datatypes). Also choose the appropriate visibility of your variables and attributes in order to be compliant with the data-encapsulation principle in Java.

b) Implement adequate constructors to instantiate your computer-class. You may also need additional constructors to instantiate your computer-classes' components.

c) Implement the following methods:
   - A method `calculatePrice()` to calculate the price of a computer by summing up the prices of all its components.
   - A method `printInfo()` which prints out all the computers' attributes (including its components' attributes).
   - A method `estimateDataTransferDuration(dataToTransferInGB)` to calculate the duration required to write a given amount of data on the hard-disk.

   Choose an appropriate visibiliy, adequate parameter- and return-types for your methods.

d) Implement `Main.java` as a test-class, where you create at least two computer-instances and call each implemented method from c) once to demonstrate their functionallity.

Submit

- `at/ac/uibk/pm/gXX/zidUsername/s02/e01/Computer.java`
- `at/ac/uibk/pm/gXX/zidUsername/s02/e01/Main.java`
- `...`

Exercise 2 (Online-banking) [7 Points]

In this exercise you have to implement a trivial online-banking system in an object-oriented way. Please read carefully the following requirements and then proceed with the implementation. Your system is composed by the following entities:

- A customer (Person), which has a name, a surname and a credit rating (low, medium, high).
- A bank-account, which consists of an IBAN, its owner (customer) and the actual balance in cents.
- A transaction, which consists of a transaction-id, a source- and target-bank-account, a transaction-amount (in cents) and -status (failure, success).
- A banking-system, which holds a list of bank-accounts that are managed by the system. Additionally it contains a list that includes every executed transaction on the banking-system.

a) At first, implement the described entities using appropriate classes and enums. Think about adequate representations (classes, enums, primitive datatypes) of the entities' attributes. Choose appropriate visibility for your attributes and implement suitable constructors for object-instantiation.

Hint

Feel free to use the `java.util.ArrayList` data structure if you need a variable length array to store multiple objects of the same type (e.g. transactions). See the example provided in the file `ListExample.java` (OLAT).

b) Now implement for each entity the described functionality according to the provided requirements:
A `BankAccount` needs to provide functionality to `deposit` a certain amount of money on the account. A deposition means that the bank-accounts' balance is increased by the deposited amount. Similarly it needs to provide a method to `withdraw` money, where the accounts' balance is decreased by the withdrawed amount. Additionally a possibility to check if a withdraw can be performed, is required. It can only be performed if the following condition holds: The balance after the withdraw does not exceeds the permitted negative amount, which is determined by the bank-account owners' credit rating (i.e. the credit rating `low` allows at most -100 EUR, the credit rating `medium` at most -500 EUR and the rating `high` at most -1000 EUR).

The `BankingSystem` requires a method to `create` a new bank-account and add it to its list of managed bank-accounts. Additionally it needs to provide a `transfer`-method to transfer a certain amount of money from a source-banking-account (source-account) to a target-banking-account (target-account). During a transfer, the transfered amount of money is withdrawn from the source-account and deposited on the target-account. After each transfer, the transaction is added to the banking-systems' transaction-list. If the transaction succeeded, i.e. withdrawing money from the source-account was possible, the transaction-status is set to `success`. If withdrawing money from the source-account was not possible, the deposition to the target-account is not performed and transaction-status is set to `failure`. Finally, the banking-system requires a method to `print` every transaction performed on the banking-system (IBAN source-account, IBAN target-account, amount, transaction-status).

Choose an appropriate visibiliy, adequate parameter- and return-types for your methods.

Hint

Keep in mind to be compliant with the data-encapsulation- and DRY (i.e. don't repeat yourself) principles.

c) Implement a class `Main.java` and test your implementation by creating at least two different bank-accounts and calling each method described at b) at least once.

Submit

- `at/ac/uibk/pm/gXX/zidUsername/s02/e02/BankAccount.java`
- `at/ac/uibk/pm/gXX/zidUsername/s02/e02/BankingSystem.java`
- `at/ac/uibk/pm/gXX/zidUsername/s02/e02/Main.java`
- `...`

Important: Submit your solution to OLAT and mark your solved exercises with the provided checkboxes. The deadline ends at 6:00 pm (18:00) on the day before the discussion.