Programming 1

Ali Osman TOPAL (aliosman.topal@uni.lu)

Lab 7 – Polymorphism

Submission (Preliminary: 14/11, Final: 21/11)
The mandatory exercises for **Code Submission** are **1, 4** (marked with an ❗). All other exercises on this sheet are optional but still highly recommended! The **Explainer Video** 🎥 for this sheet must be realized on **Exercise 1 ("Mission Mars")**. A Flipgrid invitation link will be posted on Moodle.

Exercise 1 – ❗ 🎥 Mission Mars #AbstractType #Polymorphism #Inheritance

Yeahh! We are going Mars!
Not too fast, we should work on a prototype first. As a developer, you work on a prototype robot and if all goes well, the real robot will soon fly to Mars and collect rock samples and photos.
Your task is to create a software that controls a robot with the help of a **Controller**. The controller can send some ***Action*** commands to the robot to be executed; go left, go right, go forward, go back, take a photo, and pick a rock. You will test the robot on a maze, as shown in Figure 1, which consists of 20x20 tiles. So when you send a "go action", the robot moves only 20 cm in the corresponding direction. Each action has a cost in terms of battery life. Our robot has a battery life of 3000 seconds. Using go actions takes 10 seconds, taking a high resolution photo takes 60 seconds, low resolution photo 40 seconds, and picking up a rock takes 150 seconds, consuming the robot's battery life.

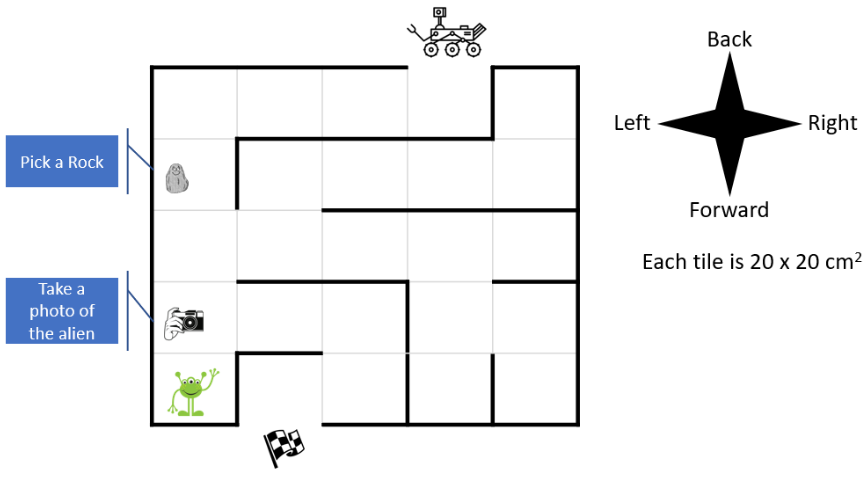


Figure 1: Test drive: Maze

Design a hierarchy with set of specified actions and make sure the hierarchy can be arbitrarily extended with other actions.

In the end, you should implement a "Controller" so that the user can create a sequence of actions (forward, forward, pick, left, photo, left, ...) and then send them to the robot. The robot executes them in order. For this test drive, the number of actions is limited to 10 in a set. After each execution of an action, the robot should report what it did and how much battery time was used. Finally, when all the actions in the set have been performed, it would be useful to see how many actions were performed and the total remaining battery life of the robot.

Write a main method for your robot to find the exit of the maze in Figure 1 while completing side quest, such as picking up a rock and taking a picture of an alien.

Exercise 2 – Bank Accounts

A normal bank account has an attribute `balance` and 2 methods `deposit` and `withdraw`. One can not withdraw more money than the account contains. An overdraft account has an additional attribute `allowedOverdraft` which represents the limit below zero that the balance of the account is allowed to reach, meaning that the balance of an overdraft account can go below zero when withdrawing money but not more than the allowed overdraft.

Design and implement a corresponding class hierarchy.

Exercise 3 – Payments

In this exercise, developed a class architecture for realizing payments.

A payment is done for a given amount. A payment type requires the two operations of checking whether a payment can be done and of finally executing it. The latter operation returns the change, if any. There are two types of payments in this application, either cash or by card.

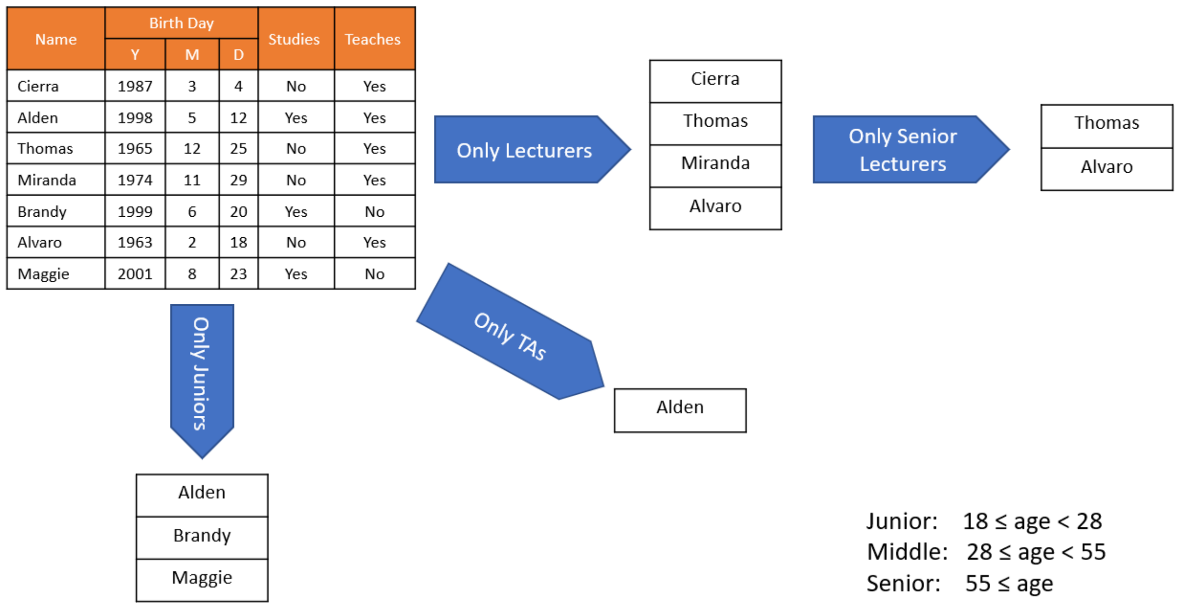
A cash payment holds the information of the given money.

A card payment is related to a card. A card again is related to a bank account (see Exercise 2). There are either debit cards or credit cards. Make sure that credit cards are related to a bank account allowing an overdraft. When executing a card payment, the amount is obviously withdrawn from the related bank account.

Be sure to apply polymorphism correctly for the different payment types. Feel free to add any additional ones, e.g., wire transfer, PayPal, Apple Pay, ... Also, write a main program to test your implementation.

Exercise 4 – ❗ Filters

Implement different filters which can be used to filter people meeting a certain criterion out of a given set of people. The example in Figure 2 points out the required functionality.



Name	Birth Day			Studies	Teaches
	Y	M	D		
Cierra	1987	3	4	No	Yes
Alden	1998	5	12	Yes	Yes
Thomas	1965	12	25	No	Yes
Miranda	1974	11	29	No	Yes
Brandy	1999	6	20	Yes	No
Alvaro	1963	2	18	No	Yes
Maggie	2001	8	23	Yes	No

Figure 2: Example of different filters being applied to a set of people

It should be possible to apply a filter to a given set of people in order to filter out a certain subset. Moreover the sequential composition of filters needs to be possible. In the example the *Lecturer Filter* filters out all lecturers of the set and afterwards the *Senior Filter* filters all seniors out of the resulting subset. Applying the *Junior Filter* to the initial set filters out all juniors.

The following types of filters should be implemented:

– **Lecturer Filter** Filters out all people who only teaches.
– **Student Filter** Filters out all people who only studies.
– **TA filter** Filters out all people who studies and teaches (poor TAs!).
– **Junior Filter** Filters out all people whose age is between 18 and 28.
– **Middle Filter** Filters out all people whose age is between 28 and 55.
– **Senior Filter** Filters out all people whose age is greater than 55.

Think carefully about a good design of a class hierarchy that fulfills the requirements mentioned above. Some questions that need to be addressed when designing an appropriate hierarchy are:

– Which class do you actually need?
– Which classes can be abstract?
– How can the set of people be realized?
– Should a filter modify an existing people set or create a new one?
– How big is the new filtered set (array)? Does it have empty slots? If so, how do you get rid of them? (No ArrayList || its friends)
– Where to implement a method to apply a given filter to a given set?

  – Should a filter be tied to a specific set or should sets and filters be more loosely coupled?

Make sure that your hierarchy allows the seamless replacement of filters and that the extension by additional filter types is easily possible.

After carefully designing a class hierarchy implement your approach. Think about what are the advantages and disadvantages of your solution. It is even recommended to implement different approaches since this could be pretty helpful in comparing the different designs. Moreover implement a main function to test your classes.