KU LEUVEN

Object Oriented Programming 2021-2022
Professor: S. Desmet and K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Lab Session Week 5 and 6
Communication Between Objects

Theory
Theory that should be covered before this lab you will have seen during the lecture of week 2: interacting with objects

Goal
In the past labs, we were the ones creating objects and calling methods. Of course, this is not how other programs work. A simple click in a program can create thousands of objects which call methods on each other. In this lab, we will learn how to create objects through objects. In addition, we will call more complex methods on these objects to change their state or we will use the return values to change the state of other objects. We have seen a sneak peek of object creation and communication between objects in the test classes of the lab session from week 4 where we received a parameter of class Barrel to transfer fluid between objects.

Format
Each lab session is divided into three different parts: starting exercises, lab exercises and home exercises. **Starting exercises** are exercises that we expect you to solve at home, before the lab session. In this way you become familiar with the material and you check whether you have properly understood and processed the theory of the lectures. **Lab exercises** are exercises that are covered during the lab sessions. **Home exercises** are exercises that delve deeper into the matter. They are optional for those who work faster, or as extra exercise material when studying/practicing at home. Contact your lab assistant for feedback on those exercises.

Learning Objectives
After this session, you should be able to understand and implement the following concepts:

- Creating objects and call their methods in objects from other classes
- Understand nullpointer exceptions
- Understand method visibility
- Use simple control structures and loops to manipulate objects

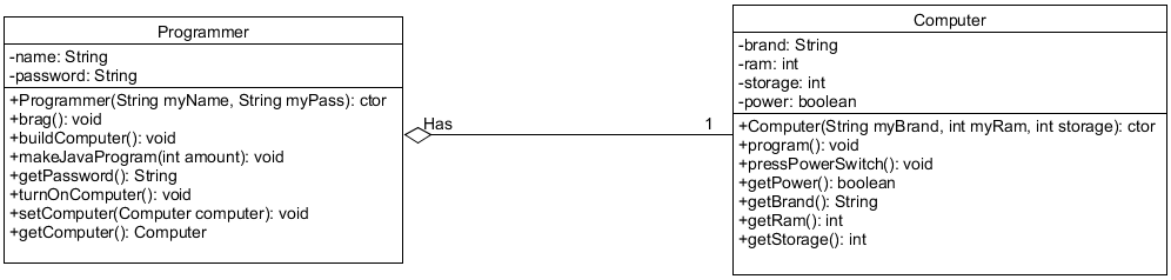
Object Oriented Programming 2021-2022

Professor: S. Desmet and K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

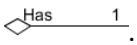
Starter Exercises

Exercise 1: The Fellowship of the Computer

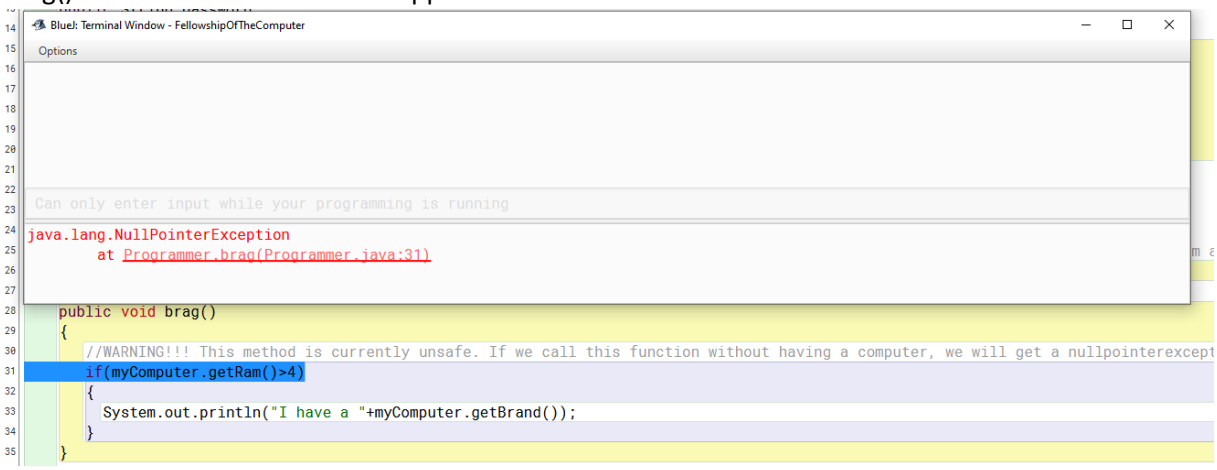
Open the FellowShipOfTheComputer Project on Toledo. Inspect the code by first looking at the Computer class and afterwards the Programmer Class. The UML is given below.



Pay attention to the following:

1) Programmer has an instance variable called Computer myComputer. But this instance variable isn't explicitly present in the instance variables in the UML. This is represented in the UML with the aggregation 

. Which indicates that Programmer has a reference of 1 Computer object. This means that objects of the Programmer class have a reference to an object of the computer Class and can use the **public** methods of that object.
2) References to objects are one of the main source of errors while programming. If we call methods on objects that are null (*nullpointer*, they don't have a reference), we get a nullpointerexception (sometimes marked as "NPE"). Create an object of Programmer. As we can see in the constructor, the myComputer instance variable is null by default. Inspect the brag() method and call it. What happens?



3) The terminal window tells us that there was a NullPointerException. It is even so kind to tell us where the problem is: line 31. The line is also indicated in blue in the BlueJ editor. Look at the method buildComputer(). Execute this method, then execute brag(). Are there still errors?
4) There are other ways to link objects. Make a Computer and a Programmer object. Then link them both using the setComputer() method of the Programmer instance. Execute brag() to check that you don't get any errors.

Exercise 2: The Two Programmers

Up until now, all of our methods have been public and all instance variables have been private. Public methods and variables can be called by any object, also outside the class. Private methods and variables can only be used by the class itself. Take a look at the Programmer Testclass. It is logical that we do not want our password to be accessible by anyone. Set the access modifier of the getPassword() method to private, compile, and try to rerun the test. Try to make an object of Programmer and try to execute the method yourself. Both should not be possible. Setting methods to private will allow us to "block" users or other objects from using methods that are only for internal use.

Exercise 3: Return of the Value

Now we know how to use references to use other objects, we can make more use of return values. In the previous labs, return values looked like an inconvenient way of displaying information. Take a look at the brag() method. We can see that in line 31 that we use an if statement which first executes getRam() of the myComputer object, retrieves that value, and checks whether it is bigger than 4. The turnOnComputer() method equally uses get methods to know whether the computer is already on or not. This is a simple example of how we can use return values to implement logic in objects, something you will need to use in the coming lab sessions.

KU LEUVEN

Object Oriented Programming 2021-2022

Professor: S. Desmet and K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Lab Excercises

Exercise 1: Improving The FellowShip Of The Computer

Open the FellowShipOfTheComputer project again and implement the following methodality:

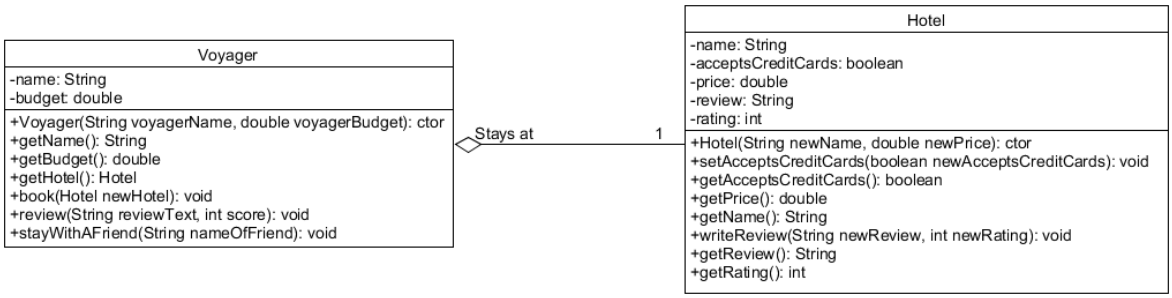
1) Make the brag() method nullpointerexception safe. Print out "I don't have a computer yet." If there is no computer set.
2) Make a method turnOffComputer() which turns off the computer if it is turned on. Verify your functionality by looking whether the Boolean in the computer object changes.

Exercise 2: Traveler

This is an exercise on test-driven design. You start from a class diagram, some extra requirements, and a test class that you will find in VoyagerStart on Toledo. It is up to you to implement the classes Voyager and Hotel. You can not change anything in the diagram, not even the names. You can however add as many variables and methods as you want. Work in a structured manner: read test 1, understand what you have to do, implement the code, and run the test. After test 1, you continue with test 2 and so on.

Problem Description

For this project you will work with two classes: Voyager (traveller) and Hotel. The idea is that a traveler makes a journey and is able to book a hotel. After creating a Voyager object there is no hotel object linked to it yet. We will assume that the traveller is on a weekend trip and will book the hotel for one night. The traveller has a budget that they can't exceed. This means you will only be able to link a hotel to a traveller if it fits within the budget. There is one exception; if a hotel accepts credit cards any traveller, regardless of their budget, is able to book that hotel. By default a hotel does not accept credit cards. The traveller can also submit a review of the hotel. In our primitive version we won't keep a "history" of past reviews; the last traveller decides the score of the hotel. De review will consist of a written text (String) and a score (int between 0 and 5). A review with a score outside the range will be ignored. The last functionality you will add allows the traveller to "cheat"; the traveller is able to stay with an acquaintance instead of at a hotel. If this option is chosen the traveller will create a new "hotel" with the name of their acquaintance. The price for this stay is always €5 (to buy a small gift).



KU LEUVEN

Object Oriented Programming 2021-2022

Professor: S. Desmet and K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Exercise 3: Weather Station

There is no starting project for this exercise. Create one yourself and test the functionality yourself. Ask your lab assistant for further clarification if needed.

Problem description:

A weather station contains 2 measuring devices: a thermometer and a barometer. Temperature is recorded in °C and atmospheric pressure in hectoPascal. The thermometer and barometer not only record the present value, but also remember the maximum and minimum of the recorded values. Define the class(es) and object diagram of your solution.

When you create a measurement device (thermometer/barometer), all values are initialized to meaningful values. Provide the weather station with a method to record new measurements (one method that records temperature and pressure at the same time). In reality this value is given by the sensor, we will simulate this by providing the values. It is also possible to "reset" the weather station. At that moment all extremes are forgotten, so after providing a first measurement, both values, min and max should be the same.

Also provide a method to get an overview of the extremes.

This example on object communication is a situation where the "system" (i.e. the weather station) is responsible for creating all the parts. So the constructor of the "system" creates instances of the individual elements.

[Challenge] Print the values together with their units (e.g °C and hPa). Make it possible to do this for any unit and avoid code duplication.

Exercise 4: RFID Tags

Start from the RFIDTag project on Toledo. There is a test class available to test your functionality. Note that two tests are empty. Implement these following the description in their comments.

Problem Description

We want to simulate the payment of a shopping trolley with the use of RFID tags. Every product in the trolley has a RFID tag attached. Every tag has an ID, a price and a payment status (payed or not). When a RFIDTag passes through the scanner, the scanner first checks if it is able to process this tag. To check this the scanner has a range of valid ID's ([minimum, maximum]). Be sure that when you create a scanner, the minimum is smaller than the maximum. If this is not the case, swap both values. When the scanned tag is within the range, the scanner increments its number of scanned products and updates the total price to pay. The RFIDScanner should also be able to generate an overview showing the amount of scanned tags and their total price. Both classes of course have the necessary constructors, setters and getters.

In this example, you have a situation where 1 object (the scanner) has to be able to consecutively communicate with different objects of type Tag. So you should be able to change the link that RFIDScanner has to RFIDTag over time.

KU LEUVEN

Object Oriented Programming 2021-2022
Professor: S. Desmet and K. Gerling
Labo-assistenten: M. Aufheimer, L. Bruynseels, S. Langendries, K. Pelsmaekers, Z. Sun, R. Vanhoof, N. Vinckier, G. Vranken en J. Wauters.

Home Exercises

Exercise 1: Car

In this exercise you are going to integrate solutions from previous labs, so reuse existing code as much as possible. There is no pre-existing project, create one yourself.

Problem Description

We want to program a drive with a car. Our car is equipped with a fuel tank of a certain capacity and has a given fuel consumption expressed as a number of liters per 100km. You should define the necessary class requirements of Car yourself. Use the Barrel from session 3-4 as the fuel tank. You can also reuse the Car class from session 2.

If you simulate a drive, the content of the fuel tank is updated and the kilometer counter increased. Additionally, the x-coordinate of your car changes. Of course, if there is not enough fuel, your car may stop earlier (or even not drive at all). A negative distance shifts your representation to the left on the x-axis.