Exercise 10

Software Development 2018
Department of Computer Science
University of Copenhagen

Thomas Veje Christensen <veje@di.ku.dk>,
Mathias Graae Larsen <cala@di.ku.dk>

Version 1.0: Maj 11th

Due: Maj 25th, **15:00**
No resubmission

Abstract

For this exercise you have both an individual part and a group part. The individual part ***must*** be done alone. For the group part you will be continue working on the *SpaceTaxi Game*. You must submit your code from the group part on **GitHub** and include the link to the repository in your report. The individual report and the groups technical report must be submitted on Absalon.

Contents

1 Individual Part **2**
1.1 Reflection on your approach to development in block 3 . . . 2
1.2 Testing using doubles 2

2 Group Part **3**
2.1 Requirements 3
2.2 Specifications 3
2.2.1 Customers 3
2.3 Technical Report 4

3 Submission **4**

4 Cleaning up your code **4**

1 Individual Part

1.1 Reflection on your approach to development in block 3

For this task you should describe, how you and your group approached the exercises of block 3 with respect to the software development process.

Each exercise consisted of some requirements you had to meet. How did you examine/disambiguate the requirements? (They might not always have been clear-cut). What was your strategy for meeting the requirements? Can your general work process be divided into specific phases? How did you collaborate on the task at hand? Which type of software development process describes your software development process in block 3 best?

Once you have reflected on and answered the questions, describe what worked well and what caused you problems or issues.

From the lecture on *Agile Development*, which methods, principles, and practices could you imagine utilizing in block 4, and how would they improve your work?

1.2 Testing using doubles

In the handed out solution you will find:

TestingWithDoubles/Bookkeeper - This is the class we wish to test.

TestingWithDoubles/ITaxCalculator - An interface that is used by the *Bookkeeper* class.

TestingWithDoubles/TaxCalculator - An unfinished implementation of *ITaxCalculator*. You should assume someone else will implement this part of the program at some later date.

Tests/BookkeeperTests - Here you should implement your tests.

Your assignment is to test the *Bookkeeper* class without depending on the implementation of *ITaxCalculator* (*TaxCalculator*) being finished.

Hint: "Chapter 5: Testing" in AC

Deliverable: A report presenting the following:

1.1. Your answer to 1.1 Reflection on your approach to development in block 3. Max. 1½ pages.

1.2. Your solution code for 1.2 Testing using doubles. Please provide a brief explanation, if it is non-trivial.

2 Group Part

2.1 Requirements

2.1. Spawning of customers as per the Customer field in the level files (for instance: `"Customer: Alice 10 1 ^J 10 100"`)

2.2. Upon *collision* with a customer, the customer should disappear, meaning it has been picked up.

2.3. Number of points for correct pick up and drop off of customers should be counted and displayed.

2.4. Flying through the portal takes you to the next level.

2.5. Unit tests for all new implementations. You should include a test plan in your technical report.

2.2 Specifications

2.2.1 Customers

Customer fields specification (example reference: `"Customer: Bob 10 J r 10 100"`):

2.1. Customer fields begin the string `"Customer "`

2.2. Its first property is a customer name. In our example `"Bob"`.

2.3. Following the customers name is a number (`"10"` in our example). This is the number of seconds that should pass in the level, before the customer appears (is spawned).

2.4. The next field is a `char` (`"J"` in our example), determining on which platform the customer should be spawned.

2.5. The next field is the destination platform of the customer (`"r"` in our example). It can be

  2.5.1. a single `char x` (excluding `^`), the customer should be dropped off on platform `x` in the *current* level.

  2.5.2. `^` the customer should be dropped off on *any* platform in the next level.

  2.5.3. `^x` the customer should be dropped off on platform `x` in the next level.

2.6. the next field is a number (`"10"` in our example), which is the number of seconds you have to drop off the customer at the correct platform. You decide what happens if this duration is exceeded. For instance, no points, less points or SUDDEN DEATH!

2.7. The last field is also a number (`"100"` in our example), which is the number of points a correct drop off of the customer is worth.

Extras (not mandatory):

2.1. Once a customer has been spawned, on the platform specified in the level file, it should walk back and forth between the left and right edge of the platform.

2.2. Having dropped off a passenger on the correct platform, it should walk to one of its edges and disappear.

2.3 Technical Report

Write a technical report:
Discuss your implementation of `SpaceTaxi` at the point of submission. Your report should be a PDF document written in LaTeX, and use the su18.sty package.

Your report must at least include:

- Give an overview of the requirements for `SpaceTaxi`.
- Give an overview of your `SpaceTaxi` implementation.
- Discuss your design decisions and how well you followed the SOLID principles.
- Discuss how you designed and implemented your tests.
- Discuss the non-trivial parts of your implementation.
- Disambiguate any ambiguities in the exercise sets.

You must read and follow our guide on writing technical reports. `https://github.com/diku-dk/su18-guides/blob/master/files/techReport.pdf`

You can also look at our example report for the `TicTacToe` program if you need some inspiration. `https://github.com/diku-dk/su18-guides/blob/master/files/su17-tic-tac-toe.pdf`

3 Submission

For the submission you must hand in:

- One *per person*, individual report.
- One *per group*, technical report with the git url in the report.
- One *per group* group.txt file containing your group information.

4 Cleaning up your code

To keep your TA happy, and receive more valuable feedback, you should:

- Remove commented out code.
- Make sure that your files and folders have the correct names.
- Make sure the code compiles without errors and warnings.
- Make the code comprehensible (perform adequate renaming, separate long methods into several methods, add comments where appropriate).
- Make sure the code follows our style guide.