Lab assessment for Software Design and Modelling

Week 6, Semester 1, 2019

Introduction

This assessment is to be done individually on your DICE machines during the lab session, making use of the tools you have used in the lab sessions so far.

In order to be able to give students with schedules of adjustments extra time, the exercise has been designed to be done in 75 minutes. Those given extra time in exams may use the remaining time, according to their usual arrangements.

The assessment will be marked out of 20. It is in three parts.

- Part A is worth 10 marks, and is intended to take no more than 30 minutes, provided you understand the work and the tools well.
- Part B is worth 6 marks. You are advised not to work on it until you are confident that you have done Part A well. It is intended to take no more than 30 minutes for those who understand the work and the tools very well, but might take up to 45 minutes with a few false steps.
- Part C is worth 4 marks. It is intended to challenge those students who find parts A and B easy and do them fast. Don't worry if you don't get to it: as you see, a first-class mark can be obtained without attempting it. You are advised not to attempt it *unless and until* you feel you have done Parts A and B very well.

To submit

Some questions ask you to submit specific files. To do this, use a terminal, navigate to the directory containing the file, and use examsubmit to submit it, e.g.

```
examsubmit Foo.java
```

Others ask you to submit your entire Eclipse project. To do this, in Eclipse, use menu File → Export → General → Archive file. (NB do *not* use a Papyrus-specific export menu!) Use the Select All button to select all your files. (It does not matter if some of them are irrelevant.) Use the Browse button to choose where to save your archive; be sure to give it the required name. Then use examsubmit to submit it, as above, e.g.

```
examsubmit A1.zip
```

Part A

1. Create a Papyrus project called A1 containing a UML class diagram. **Use "basic primitive types" (not ecore types):** leave the corresponding box ticked when you create your Papyrus project. Show the following:

   (a) An interface `MainCourse`, with operation `getPrice` taking no argument and returning an integer;

   (b) A class `Topping` with private attribute `description` of type `String`

   (c) A class `Pizza`;

   (d) An aggregation between `Pizza` and `Topping`, showing that a `Pizza` may contain some `Topping`s;

   (e) Multiplicities indicating that a `Pizza` may be linked to any number of `Topping`s, while a `Topping` is linked to exactly one `Pizza`;

   (f) That the class `Pizza` realizes the interface `MainCourse` (using an appropriate kind of line, not a ball/lollipop).

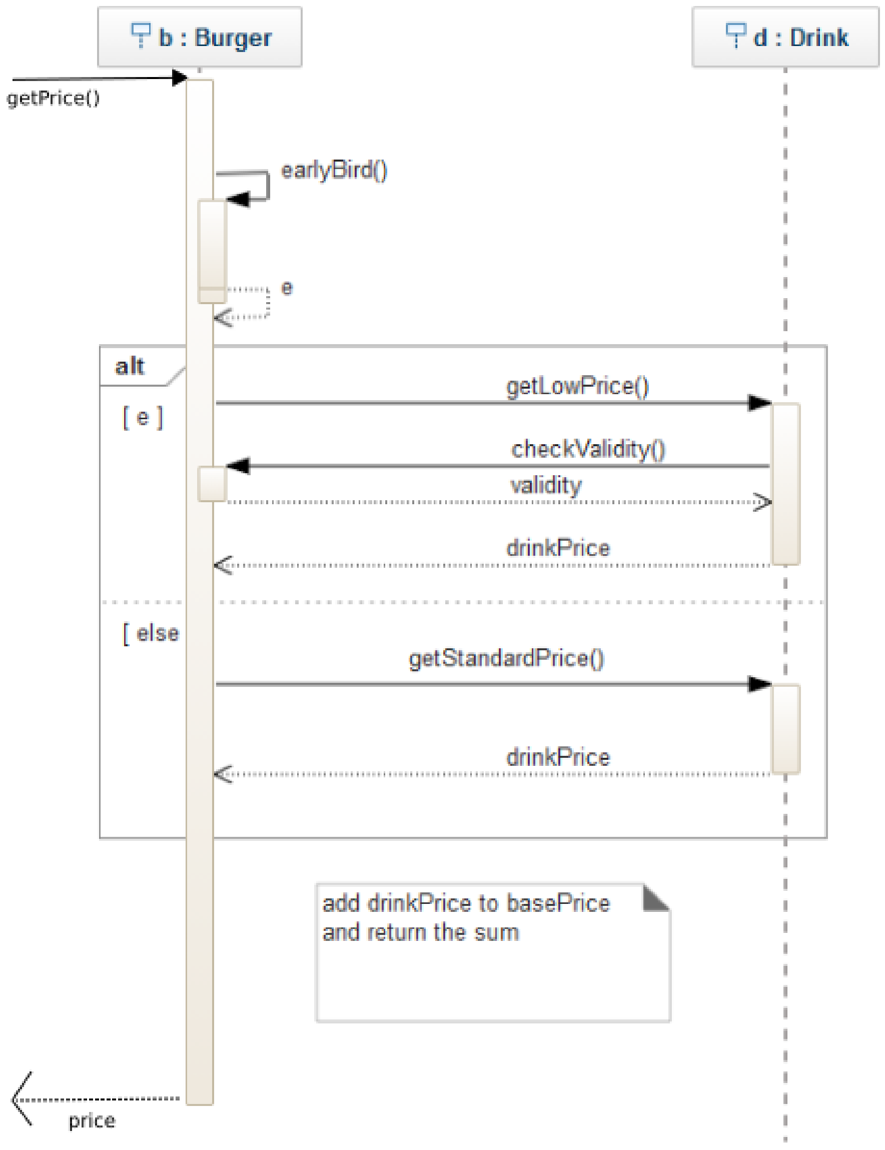
   For full marks, ensure that the contents of your model are apparent in your diagram: for example, make sure that the operation in `MainCourse` is displayed.

   Export your project as A1.zip as described above. **Submit A1.zip**.

   [6 marks]

2. Consider the following sequence diagram, concerning the implementation of `getPrice()` in a class `Burger` which also implements `MainCourse`. Add code to the provided file `Burger.java` to make it consistent with the sequence diagram.

   Note: because interface MainCourse and class Drink are not provided, your class Burger will have errors if you compile it. Do not worry about this. Try to make your own Java code correct, however.



   Submit Burger.java.

[4 marks]

Part B

Using Papyrus, and calling your project B, draw a UML activity diagram representing the pizza chef's process:

1. make the pizza base;
2. add each topping in turn, until there are no more toppings to add;
3. cook the pizza.

Use opaque actions, and use opaque expressions in natural language (not in OCL). Include initial and final nodes.

[6 marks]

Export your project as B.zip as described above. **Submit B.zip**.

Part C

Your answers to Part C should be typed into a text file called C.txt.

1. Write, in the context of class `Pizza`, an OCL constraint on pizza `p` expressing that `p` is a mushroom pizza.

   (Assume that the `description` attribute of `Topping` expresses what the topping is. Assume that cheese and tomato sauce are not represented as toppings, so that any `Topping` on a mushroom pizza should have "mushroom" as the value of `description`.)

   [2 marks]

2. Considering the pizza class diagram as the conceptual model for an online pizza ordering system, what are the most significant problems you see with it (apart from its incompleteness)? Can you suggest improvements? Write no more than 50 words. [2 marks]

Submit file C.txt.

Summary

If you did every part of this assessment, the files you should have submitted are:

- A1.zip
- Burger.java
- B.zip
- C.txt

Later submissions override earlier ones, so if in doubt, resubmit.