Lab assessment for Software Design and Modelling

Week 6, Semester 2, 2024

- This assessment is to be done **individually**, making use of the tools you have used in the lab sessions so far.
- It is **open-book**: you may consult the course material and any online source you have found useful. However, you may **not** collaborate with one another or with anyone else, and you may **not** use any tool – including any AI-based tool – other than those specified in the questions.
- Please remember the **good scholarly practice** requirements of the University regarding work for credit. You can find guidance at the School page

  `https://web.inf.ed.ac.uk/infweb/admin/policies/academic-misconduct`

  which also has links to the relevant University pages. Please do not publish solutions to these exercises on the internet or elsewhere, until I tell you you may.
- The exercise has been designed to be done in **75 minutes**. Those given 25% extra time in exams may use 94 minutes, according to their usual arrangements.
- The assessment will be **marked out of 100**. It is in three parts.
  - Part A is worth 50 marks: all students should attempt it.
  - Part B is worth 30 marks: all students should attempt it.
  - Part C is worth 20 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. **It will not be marked unless your Parts A and B have already earned you at least 70 marks, i.e., a first-class mark.**

To submit

Each question tells you what to submit, giving a filename and format, e.g. A.pdf. Save your files locally. Then when you are ready to submit, make a zip file called `done.zip`, containing them all *at top level*, e.g. using DICE command:

```
zip done.zip A.pdf B1.pdf B2.pdf C.pdf
```

and upload it using the button in the question where you got this paper. *Use exactly the names and formats specified*, otherwise you may lose marks.

Part A

A class `WasherDryer` implements the controlling software for a washer-dryer: that is, for a machine which can automatically wash clothes, followed if the user desires by automatically drying them. A class `Programme` is concerned with the different washing and drying programmes that a user can select – delicates, cottons and linens, etc. You may assume that there is a navigable association from class `WasherDryer` to class `Programme`, with role-name `selected-programme`, and multiplicity 1, at the `Programme` end.

1. In LucidChart, using its UML shapes, draw a nested UML state diagram for a class `WasherDryer` showing:

   - Two high-level states `Washing` and `Drying`
   - That the initial state of an instance of `WasherDryer` is `Washing`
   - Within the state `Washing`, the washing process does first `Wash`, then `Rinse`, then `Spin`.
   - On completion of the washing process, if the `dry` attribute of the selected programme has value `true`, and the `weight` attribute of `WasherDryer` is less than 6, the machine should perform the action `start-dry` and enter the `Drying` state; otherwise, show that the object does no action and its life is finished. Use OCL for the guard.
   - If the instance receives the event `programme-finished`, from any state, show that the object's life is finished.

   (40 marks)

2. Add a Note to your UML state diagram. In it, write an OCL class invariant (in full, not just the OCL expression itself), to specify that for any valid instance of `WasherDryer`, the value of the attribute `weight` must be less than 10; and that, if the `dry` attribute of the selected programme is set to `true`, then `weight` must be less than 6. (10 marks)

Export your diagram as A.pdf

Part B

In LucidChart, using its UML shapes, draw two class diagrams to illustrate different early drafts of ways to implement `WasherDryer`:

1. **In a diagram which you export as B1.pdf** show the class `WasherDryer` being a specialisation both of a class `Washer`, and of a class `Dryer`. Show that the class `Washer` has a public method `spin` which takes one integer argument and returns no result, and a private integer attribute `weight`.

   (Note: in this part, for simplicity, we have deliberately left out `Programme`.)

2. **In a diagram which you export as B2.pdf** show that each object of class `WasherDryer` contains one object of class `Washer`, and one object of class `Dryer`. Use composition, with the appropriate notation. Add a class `Programme`, and show both:

   - that there is a navigable association from class `WasherDryer` to class `Programme`, with role-name `selected-programme`, and multiplicity 1, at the `Programme` end;
   - that a `WasherDryer` provides at least 4 `Programme`s.

(30 marks)

Part C

REMINDER: This part is intended for students in the upper reaches of the University Standard Marking Scale. It is aimed at students who have found Parts A and B easy and done them quickly. It will only be marked if your answers to Parts A and B have already secured you a first-class mark! So you are strongly advised not to attempt it until you are sure you have done as well as you can on Parts A and B.

You may write your answer in any text editor or word processor, and submit it *either* as a plain text file, C.txt, *or* as a PDF file, C.pdf, as you prefer. **Please do not submit in any other format.**

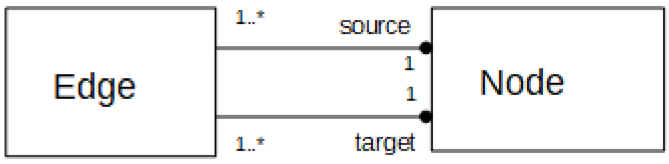
1. Consider the two class diagrams you drew in Part B. Comment briefly on the design decisions they incorporate. In particular, say whether you think specialisation, as in B1, or composition, as in B2, is the more promising approach, and why.

   Note: you may assume that `Programme` would be treated the same way in either case, even though – to avoid duplication of effort – you were only asked to include it in one of the diagrams in Part B.

   (5 marks)

2. This question concerns the formal definition of modelling languages, and involves a feature of UML that you have (probably) not considered before. You must answer it using the UML2.5.1 Specification (as a reminder, it can be found at `https://www.omg.org/spec/UML/2.5.1/About-UML`).

   Consider the following class diagram, which differs from most of those you have previously seen in having black blobs at some association ends.



   Study the UML specification to find out what the blobs mean.

   (a) **Give the page number** of the place in the specification where you found the relevant information. Note that the required page number is the one printed at the bottom of the page, opposite the text "Unified Modeling Language 2.5.1" – *not* the page number shown by your PDF reader.

   (b) Write a brief explanation of what the blobs mean. Use your own words, but also quote a relevant sentence (or more) from the UML specification.

   (c) Illustrate your answer by giving the simplest possible Java implementation of classes Edge and Node consistent with the diagram, and stating what, in your implementation, has been forced by the blobs.

   (d) Finally, state how the diagram above differs from the UML diagram you might more usually expect to see modelling your Java implementation. Comment on your answer – e.g., why do you imagine such blobs have not been prominent in the UML examples you have seen in this course?

   (15 marks)

Finally: submission

If you have done all parts of the lab assessment you will have files to submit as follows:

- A.pdf
- B1.pdf
- B2.pdf
- C.txt or C.pdf (if you did Part C)

Now create a single zip file called done.zip containing all the files you wish to submit. On DICE the command to do this is

```
zip done.zip A.pdf B1.pdf B2.pdf C.pdf
```

– adjust in the obvious way for files you have/have not created.

Upload it using the button in the question where you got this paper. *Use exactly the names and formats specified*, otherwise you may lose marks.

After you have submitted you may leave, quietly.