SEOC: summary, revision, exam

Perdita Stevens

School of Informatics
University of Edinburgh

Learning objectives

1. Design simple object-oriented systems, making appropriate use of available components;
2. Design simple software components, making sensible API decisions;
3. Evaluate and evolve object-oriented software designs, making use of common design patterns if appropriate;
4. Create, read and modify UML diagrams documenting designs;
5. Discuss the use of modelling in software development, e.g. why and how models of software can have varying degrees of formality.

Using SEOC

- Please remember what you learned here when you develop software in future – e.g., in your project. This doesn't mean "draw UML diagrams for everything" (think about whether they're useful in the circumstances!) but it does mean "make your designs SOLID"!
- Please do well in your exams...

SEOC exams

Same format as in past years: three questions, of which you choose two.

Aim to have a balance between:

- bookwork
- straightforward application of knowledge
- more challenging problem-solving.

Single-semester visiting undergraduates examined this semester, everyone else in the summer.

Example exam question types

Be prepared to answer exam qustions *such as*

- You are to design a system to do .... Draw a ... diagram to illustrate ....
- Explain what ... means in this ... diagram.
- Draft an API for ...
- Write an OCL expression to ... / What does the following OCL mean... ?
- Imagine you are ... Suggest how you should use modelling to help you work.
- To make the following design decision, what information would you need and why?
- What is the ... design principle/pattern? Apply it/explain whether it is useful in this situation ...

Notes on doing SEOC exams

Write clearly and concisely. Use precisely correct UML.

Many questions lend themselves to addendums like "Comment briefly on [missing or unclear requirements, design alternatives, problems in a design]" – this is your chance to show that you can think like a designer! Don't woffle. Do show understanding. (The mark scheme will typically say something like "1 mark each for any two reasonable points, e.g ...")

Assume your papers are being read by a reasonable – but picky! – human being.

If you are confused about what a question wants you to assume, briefly say why, say what you are assuming, and do what seems best to you.

Writing implements

If you pay close attention to detail, you will notice that the script books and a page on the university site says you have to do exams in pen, not pencil.

This has caused concern for exams with diagrams in – are you really supposed to draw them in pen?

Two things are true:

1. It looks as though the University really does say Yes. So do.
2. But personally I can't imagine any Board of Examiners in Informatics agreeing to dock marks from someone who wrote a correct answer in pencil!

Do not waste time redrawing a whole diagram if you make an error. If you leave your script in any state where I can tell that you meant the correct thing, you will get the marks.

Examinable material

Basically, everything covered in

- lectures
- required readings
- tutorial sheets
- videos, including both the ones I made and the Bloch API one

unless otherwise stated.

Revision resources

The final tutorial is for revision, based around past exam questions.

Feel free to post any questions which you think might be of general interest to the mailing list.

Use the tutorial sheets, including the extension ones, especially if you didn't do those as you went through.

Past exam papers (on university past paper site): useful but NB syllabus shift just before 2012/13. E.g. many past papers ask you to draw use case diagrams; I wouldn't. And some ask you to draw a "CRC card game" – I might as you about CRC cards, but wouldn't ask you to represent their use in a scenario.

Please fill in the questionnaire!

Would very much appreciate feedback on your experience of this course.

Some questions I'd particularly like input on this year:

- Next year, there will be a minor change in the specification of this course so that students will be assumed competent in Java. This will allow me to reinforce the design material with programming exercises. Is there any particular aspect of the course where you think a programming exercise would have helped you understand the design issues?
- To what extent were the videos and MCQs useful? Would you like more? On what?
- Have you read all the required reading? Which items did you like most/least? Would more self-assessment questions on the reading be useful?

Finally, I am always interested to hear from past students now working in software development. Please drop me an email some day!