FOR INTERNAL SCRUTINY (date of this version: 8/9/2016)

UNIVERSITY OF EDINBURGH

COLLEGE OF SCIENCE AND ENGINEERING

SCHOOL OF INFORMATICS

Tuesday 1st April 2014

00:00 to 00:00

INSTRUCTIONS TO CANDIDATES

Answer QUESTION 1 and ONE other question.

Question 1 is COMPULSORY.

All questions carry equal weight.

CALCULATORS MAY NOT BE USED IN THIS EXAMINATION

THIS EXAMINATION WILL BE MARKED ANONYMOUSLY

FOR INTERNAL SCRUTINY (date of this version: 8/9/2016)

1. THIS QUESTION IS COMPULSORY.

   (a) In software engineering, what is encapsulation? [*2 marks*]

   (b) How can careful use of encapsulation reduce the cost of software maintenance? [*2 marks*]

   (c) Given an example of a language feature in Java which supports encapsulation. [*1 mark*]

   (d) Recall that SRP is one of the SOLID design principles. What does SRP stand for? Briefly explain the principle. [*3 marks*]

   (e) Draw a fragment of UML to show that a class named A implements an interface named B. [*2 marks*]

   (f) Consider the following Java class. Which of the method calls in method `doIt` are permitted under the Law of Demeter, and which are not? Assume the code compiles (so `SomeOtherClass` exists and contains appropriate methods).

```
public class SomeClass {
  private SomeOtherClass x;
  // constructors and other methods omitted...
  public void doIt (SomeOtherClass a) {
    x.foo();
    a.foo();
    SomeOtherClass b = new SomeOtherClass();
    b.foo();
    a.foo().toString();
    super.toString();
  }
}
```

   [*3 marks*]

   (g) Explain briefly why the Law of Demeter is worth knowing, and why, nevertheless, well-designed code does not always obey it. [*4 marks*]

   (h) Assume that metamodel `Class` contains a metaclass `DataType`, and metamodel `Relational` contains a metaclass `Type`. `DataType` and `Type` each have an attribute called `name`, of type `String`. Write an ATL rule that transforms each `DataType` into a `Type` with the same `name`. [*4 marks*]

   (i) What does it mean to say that a bidirectional transformation is *correct*? [*2 marks*]

   (j) Briefly explain why somebody might not wish all their bidirectional transformations to be *hippocratic*. [*2 marks*]

ANSWER EITHER QUESTION 2 OR QUESTION 3

2. Write an essay on the *difficulties* that companies may encounter in using UML models in software engineering, drawing on anything you learned in this course including in these two relevant pieces of required reading:

   - *Five Reasons Developers Don't Use UML and Six Reasons to Use It*, blog post by Paulo Merson;
   - *Empirical assessment of MDE in industry*, paper by Hutchinson et al.

   Your essay should explain: what companies might be aiming to gain by using UML; why these gains may not be straightforward to achieve; which difficulties you see as easy to overcome, and how; which seem to you to be harder to overcome; what steps a company experiencing difficulty might consider taking, either to achieve successful use of UML or to find an alternative means to the benefits they seek. [*25 marks*]

3. (a) What is a design pattern? [*2 marks*]

   (b) Suggest three ways in which studying design patterns might help someone to become a better designer. [*3 marks*]

   (c) In the course, we discussed the fact that several design patterns aim to address design problems that arise from the use of `new` in object-oriented software.

   A web page titled *New Considered Harmful*, on a well-known site `http://c2.com/cgi/wiki`, begins by reminding the reader of the basic responsibilities of `new`, namely: "to return an object name; to create a new object; to obtain memory from the memory system; to call a 'constructor' method."

   It goes on to make some statements about properties of `new` that it says may be "cause for concern", including:

   - "The 'constructor' is often fixed. If so, `new` specializes the code in which it is described for a particular behavior definition. AbstractFactory or AbstractConstructor can work around this."
   - "The 'constructor' often contains other constructors internally (sometimes implicitly, when dealing with member attributes). If so, then constructors may become specialized to an object's position in the overall object-graph. DependencyInjection is aimed to work around this."

   i. Explain carefully why each of these points may be considered "cause for concern". You might mention, for example, what maintenance problems each gives rise to. [*10 marks*]

   ii. Explain how the Abstract Factory pattern, and Dependency Injection, may help to mitigate the concern. In your answer, you should briefly explain what Abstract Factory and Dependency Injection are; you may use diagrams if you wish, but this is not essential. Make sure you also explain their relevance in this context. [*10 marks*]