UNIVERSITY OF LONDON

BSc EXAMINATION 2022

For Internal Students of
Royal Holloway

DO NOT TURN OVER UNTIL TOLD TO BEGIN

CS2800:Software Engineering
CS2800R: Software Engineering – for FIRSTSIT/RESIT CANDIDATES

Time Allowed: **TWO hours**

Please answer **ALL** questions

1. For each of the following pairs of related software engineering concepts you must:

   - Briefly *describe* each of the two concepts and why they are important in software engineering. This should be enough to introduce the concept to a new student on CS2800.

     Each description could be about six lines of text.

   - Show that you understand how the two concepts are *connected*.

     For example, they have the same or contrasting goals, or they may be techniques that rely on each other to work. This needs careful thought.

     A good answer could be about four lines of text.

   (a) *Control Flow Graph* and *Code Complexity*. [10 marks]

   (b) *Aggregation Relationship* and *Composition Relationship*. [8 marks]

   (c) *Good Code* and *Test Driven Development (TDD)*. [12 marks]

   (d) *Reintegration Merge* and *Sync Merge*. [10 marks]

2. (a) Draw the control flow graph for the following Java method. [6 marks]

```java
public void foo(int a, int b) {
 if (a > b) {
  foobar();
 }
 while (a < b) {
   if (a > 3) {
    foobar();
   }
 }
}
```

(b) This question is about the primitive obsession smell.

i. Give a careful description of the primitive obsession smell, showing how it can occur for a single field or a group of related fields. [2 marks]

ii. Give a simple example to show how refactoring primitive obsession can increase coherence. [2 marks]

(c) For each of the following kinds of system test, *briefly explain why* we might do these tests and *how* we do this kind of testing.

i. Configuration testing. [3 marks]

ii. Compliance testing. [3 marks]

iii. Availability testing. [3 marks]

(d) For the following applications *explain why* would you consider *each of* configuration testing, compliance testing and availability testing to be *necessary*, *useful* or *unimportant*.

i. A government system for managing the tax details of citizens. [2 marks]

ii. An ambulance system for responding to emergency calls. [2 marks]

iii. A new web browser plugin for highlighting telephone numbers on pages. [2 marks]

3. (a) Maven is a Software Engineering tool for managing Java based projects. Give three benefits of using Maven that you have found while doing the coursework for CS2800. [3 marks]

   (b) Explain carefully what Maven does when you run:

      i. A lifecycle like `maven clean`. [2 marks]

      ii. A phase like `maven test`. [2 marks]

      iii. A goal like `maven javafx:run`. [2 marks]

   (c) The following is an excerpt from the build section of a Maven pom.xml file:

```
 1  <plugin>
 2    <groupId>org.apache.maven.plugins</groupId>
 3    <artifactId>maven-checkstyle-plugin</artifactId>
 4    <version>3.1.1</version>
 5   <configuration>
 6     <configLocation>/src/main/resources/gPl.xml</configLocation>
 7     <failsOnError>true</failsOnError>
 8    <violationSeverity>warning</violationSeverity>
 9   </configuration>
10   <executions>
11    <execution>
12     <id>validate</id>
13      <phase>validate</phase>
14     <goals>
15      <goal>check</goal>
16     </goals>
17    </execution>
18   </executions>
19  </plugin>
```

   Explain the purpose of each of the highlighted lines 2, 3, 4, 6, 7 and 13. [6 marks]

   (d) We are working in a development branch of a project using the SVN source code control system. Our task is to introduce design patterns to reduce coupling.

      i. How does `svn update` help us to work together with multiple engineers working on this branch. [2 marks]

      ii. When working on such an important refactoring, under what circumstances should we reintegrate and/or sync merge? [2 marks]

      iii. What is it about doing such a major refactor that might cause tree conflicts when merging? [2 marks]

4. The flyweight pattern saves memory by refactoring the shareable parts of memory heavy objects into `Flyweight` objects. The memory heavy (parent) object then contains a reference to a (shared) `Flyweight` object. Each method of the `Flyweight` object is passed a reference to its parent.

   For example we might have only a few possible sprites for game images, but we may have very, very many game objects. We share the sprites between game objects using the flyweight pattern.

   The implementation uses an `ImageObjectFactory` (the Factory pattern) that has a list of `FlyweightSprite` objects. The factory method is passed a filename, and only creates a new `FlyweightSprite` object (from that file) if it has not seen the filename before - otherwise it returns an existing (shared) `FlyweightSprite`.

   `FlyweightSprite` has two methods - `draw` and `move` - which are defined by an interface `GObject`.

   A `GameObject` has the fields: `FlyweightSprite image`, `Position position`, `float speed`, and `Vector direction`. The `GameObject` also implements `GObject`.

   (a) Sketch a UML class diagram for this project, showing `FlyweightSprite`, `GObject`, `GameObject`, and the `ImageObjectFactory`.
   You should draw in associations, multiplicities and stereotypes for patterns. You only need to show the attributes and methods that are given in the above description. [6 marks]

   (b) About the choice of design patterns.

      i. Why is it sensible to create flyweight instances using a factory? [2 marks]

      ii. Why is the ImageFactory best created as a Singleton? [2 marks]

   (c) What modifications would you need to make to the `GameObject` class to make its `position` observable by another object that implements the following interface: [4 marks]

```
public interface Observer {
  void update(Position updatedPosition);
}
```

END

DAC