IMPERIAL COLLEGE OF SCIENCE, TECHNOLOGY AND MEDICINE

EXAMINATIONS 2019

BEng Honours Degree in Computing Part II
MEng Honours Degrees in Computing Part II
BEng Honours Degree in Mathematics and Computer Science Part II
MEng Honours Degree in Mathematics and Computer Science Part II
for Internal Students of the Imperial College of Science, Technology and Medicine

This paper is also taken for the relevant examinations for the Associateship of the City and Guilds of London Institute

PAPER C220=MC220

SOFTWARE ENGINEERING DESIGN

Wednesday 1st May 2019, 14:00
Duration: 180 minutes

Answer ALL TWO questions

Paper contains 2 questions
Calculators not required

1 Working with Legacy Code

For this question, look at the code provided to you in the directory Q1. This code represents a fragment from a mobile phone billing system. If you look at the class PhoneCallTest, it shows how a call is created, started, ended, and charged.

a Identify two sources of coupling or dependency in the PhoneCall class.

i) Explain the first source of coupling.

ii) Explain the second source of coupling.

iii) Describe the problems that these can cause.

b The PhoneCallTest is not really a test, it is just an example of usage. Carefully refactor the code given to allow you to write some better tests for the system. Do not change the signature of any of the existing public methods.

Your tests should:

Show that peak time calls are charged correctly.

Show that off-peak calls are charged correctly.

Not take any real time to run - i.e. running your unit tests should take milliseconds, rather than seconds or minutes.

c Give a brief description (a few sentences) of the approach you took in part b).

d The phone company decides to change their pricing model, so that any call that either starts, or ends, within the peak period is charged at the peak rate. Update your tests and implementation to match this requirement.

The four parts carry, respectively, 20%, 50%, 10% and 20% of the marks.

2 **Interactive Applications**

For this question, look at the code provided to you in the directory Q2. This code implements a very small GUI statistics app. As you click buttons on the UI, aggregate statistics about the numbers entered are presented.

a Give two benefits of using a Model-View-Controller architecture for a GUI application.

   i) Explain the first benefit.

   ii) Explain the second benefit.

   iii) Name one alternative architectural style that you might use in a GUI application.

b Refactor the code in the Q2 directory to follow the Model-View-Controller architecture. Do not worry too much about totally separating view from controller, most important is the separation of the model.

c Write some unit tests for the model in your refactored code.
Write at least two tests.

d Refactor the code further to allow views to be *observers* of the model.

e Add a unit test to check that updates to the state of the model are correctly propagated to the view.

The five parts carry, respectively, 15%, 30%, 15%, 20% and 20% of the marks.

Appendix of Code Samples.

These will be given to the candidates on their computer - not needed as part of the printed examination paper.

Q1

```
package Q1;

import static java.time.temporal.ChronoUnit.MINUTES;

import java.time.LocalTime;

public class PhoneCall {

  private static final long PEAK_RATE = 25;
  private static final long OFF_PEAK_RATE = 10;

  private final String caller;
  private final String callee;

  private LocalTime startTime;
  private LocalTime endTime;

  public PhoneCall(String caller, String callee) {
    this.caller = caller;
    this.callee = callee;
  }

  public void start() {
    startTime = LocalTime.now();
  }

  public void end() {
    endTime = LocalTime.now();
  }

  public void charge() {
    BillingSystem.getInstance().addBillItem(caller, callee, priceInPence());
  }

  private long priceInPence() {
    if (startTime.isAfter(LocalTime.of(9, 00)) && endTime.isBefore(LocalTime.of(18, 00))) {
      return duration() * PEAK_RATE;
    } else {
      return duration() * OFF_PEAK_RATE;
    }
  }

  private long duration() {
    return MINUTES.between(startTime, endTime) + 1;
  }

}
```

```
package Q1;

import org.junit.Test;

public class PhoneCallTest {

  @Test
  public void exampleOfHowToUsePhoneCall() throws Exception {

    PhoneCall call = new PhoneCall("+447770123456", "+4479341554433");

    call.start();

    waitForSeconds(150);

    call.end();
    call.charge();
  }

  private void waitForSeconds(int n) throws Exception {
    Thread.sleep(n * 1000);
  }

}
```

1 continued…

```
package Q1;

public class BillingSystem {

  private static final BillingSystem INSTANCE = new BillingSystem();

  public static BillingSystem getInstance() {
    return INSTANCE;
  }

  public void addBillItem(String caller, String callee, long callCostInPence) {

    // Imagine lots more code here that really does payment processing - we
    // did not implement it all for the purposes of the exam.

    System.out.println(
        String.format("Bill item added: %s => %s [ cost: %d ]", caller, callee, callCostInPence));
  }

}
```

Q2

```
package Q2;

import java.awt.Panel;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.util.ArrayList;
import java.util.List;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JTextField;

public class SimpleStats {

  private final List<Integer> numbers = new ArrayList<>();
  private int max;
  private double mean;

  private void display() {

    JFrame frame = new JFrame("Simple Stats");
    frame.setSize(250, 350);

    Panel panel = new Panel();

    JTextField currentMax = new JTextField(11);
    JTextField currentMean = new JTextField(11);

    panel.add(new JLabel("Max: value "));
    panel.add(currentMax);
    panel.add(new JLabel("Mean: value "));
    panel.add(currentMean);

    for (int i = 1; i <= 12; i++) {
      final int n = i;
      JButton button = new JButton(String.valueOf(i));
      button.addActionListener(new ActionListener() {
        @Override
        public void actionPerformed(ActionEvent e) {
          numbers.add(n);
          max = Math.max(max, n);
          mean = numbers.stream().mapToInt(val -> val).average().orElse(0.0);
          currentMax.setText(String.valueOf(max));
          currentMean.setText(String.valueOf(mean));
        }
      });
      panel.add(button);
    }

    frame.getContentPane().add(panel);

    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
  }

  public static void main(String[] args) {
    new SimpleStats().display();
  }

}
```

General Information and Computer Instructions

When the exam starts, log in to the Linux computer in front of you, which is in Lexis mode. Use your College username as both your username and password to log in to Lexis. Don't worry if it is slow to start.

AnswerBook

All your answers should be submitted electronically by accessing the *AnswerBook* website at `https://co220.doc.ic.ac.uk/exam` using a standard web browser from the Linux environment. All other access to the network is intentionally blocked. Log in to this website using your normal College username and password (not username twice as with Lexis). You should not use a paper answer booklet.

Refer to the exam paper for the full version of the exam questions.

Skeleton Code

On the computer you will find a directory `/exam/co220-exam` containing code related to each of the questions in the exam. Work on this code on the computer using IntelliJ IDEA.

Using IntelliJ IDEA

You can start IntelliJ from the menu, or by typing `idea &` in the terminal. The skeleton files are set up with an IntelliJ project configuration, so: **click *Open* (not *Import Project*)**, navigate to the `/exam/co575-exam` directory, and click OK.

The lib folder contains all the testing libraries that we used during the course. These should be added into your IDE automatically if you open the project as above. If no JDK is set, select JDK 10. Please ask an invigilator if you have problems with opening the project.

There is no `build.sh` or suite of automated tests/checks for you to pass, but do aim to use good code style and clear formatting in your solutions.

You do not have access to GitLab, but you can use a Git repository locally on your machine if you find it helpful.

When you are happy with your work, paste your code into the relevant boxes on the *AnswerBook* site to answer the questions. Click the button in the top corner to save your changes. Make sure all changes are saved before the end of the exam.

At the end of the exam, just log out from your machine.

Time

There is plenty of time for this exam - just in case of any technical problems. The exam is not designed to take three hours, and you may well finish early. If you have finished, you may then leave the exam room, as long as you do so calmly and silently. However, regulations state you may not leave during the last 15 minutes of the exam. After leaving, you may not re-enter the exam room..