IMPERIAL COLLEGE LONDON

TIMED REMOTE ASSESSMENTS 2020-2021

MSc Computing
for Internal Students of the Imperial College of Science, Technology and Medicine

PAPER COMP70055=COMP97121

SOFTWARE ENGINEERING DESIGN (MSC)

Tuesday 4 May 2021, 10:00
Duration: 120 minutes
Includes 0 minutes for access and submission

Answer ALL TWO questions
Open book assessment

Paper contains 2 questions

General Information

This is a remote assessment involving programming. To do it you will need:

- a computer with internet connection
- a Java development environment installed (IntelliJ IDEA is recommended)

AnswerBook

All your answers should be submitted electronically by accessing the AnswerBook website using a standard web browser. Log in to AnswerBook using your standard College username and password.

All work that you want marked should be entered into AnswerBook. Paste your code into the boxes on AnswerBook. Do not try to upload PDFs or images. You should not upload anything to CATE or push anything to GitLab.

Skeleton Code

Download https://www.doc.ic.ac.uk/~rbc/2021-exams/SED-exam.zip which contains code related to the exam questions. Unzip this file and work on this code on your computer using your IDE.

Using IntelliJ IDEA

If you open the downloaded folder as a project in IntelliJ IDEA, you should see two directories, Q1 and Q2, each of which should contain a blue src directory and a green test directory.

The project should already be configured with all the testing libraries that we used during the course, in case you need to use them in answering the questions.

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. The code will not be auto-tested, it will be read by a human.

Answer the questions on the following pages using the downloaded code. When you are happy with your answer for each part, paste your code into the relevant boxes on the AnswerBook site. Use the button top-right to save your work. You can update your answer and save again as often as you want. We recommend saving after you have answered each part.

When You Have Finished

At the end of the exam, or when you have finished, check that you are happy with your work and that all your answers are saved in AnswerBook.

1 **Question 1**

Look at the code under the folder Q1, which implements part of a simple retail application. The class RetailExample shows some examples of how orders can be created and processed.

Make changes to the provided code to address the following. Show the relevant sections of your code for each part.

a There is quite a lot of code that is duplicated between SmallOrder and BulkOrder. Introduce a template method pattern to reduce this duplication.

b Constructing different types of order requires passing a lot of parameters into the constructors all at once. Apply an appropriate design pattern to improve this.

Design your code to allow for the following:

It should be possible to add items to an order one at a time.
If an order has more than 3 items in it, this should be treated as a bulk order.
An order with 3 items or fewer is treated as a small order.
It should be possible to specify different addresses for billing and shipping, but if no shipping address is specified, orders should ship to the billing address.
Small orders can be gift wrapped, but bulk orders cannot.
Bulk orders can have an additional discount applied, but small orders cannot.

c i) What design pattern is being used in the CreditCardProcessor class?

ii) What problem does this typically cause?

iii) Apply the dependency inversion principle and hence write a test that verifies that different types of orders are charged correctly. You may change the signature of the process() method if you wish.

The three parts carry, respectively, 30%, 40% and 30% of the marks.

2 **Question 2**

Look at the code under the folder Q2, which implements a simple GUI application for scoring a game of tennis. You should be able to run it by running the main() method and click the buttons to update the scores. It's not necessary to understand the rules of tennis in order to answer this question, but if you want to check something, the rules and scoring system are explained on Wikipedia here: *https://en.wikipedia.org/wiki/Tennis#Scoring*. Here we are only concerned with a single game, not with sets or matches.

This question involves introducing the MVC architecture to the given code.

a Starting with the given code for TennisScorer, refactor the code to extract a model from the rest of the application. Do not worry about separating the view or controller in this part. Show the relevant parts of your code.

b Add some unit tests to test some of the core behaviour that is now encapsulated in your model. Write at least three tests showing different behaviours.

c Introduce the observer pattern, so that the view becomes an observer of your model. Show the relevant parts of your code.

d Write a mock object unit test to show that the view is correctly updated each time that a point is scored during the game. You can either add a new test, or convert one of your existing tests.

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

APPENDIX Provided Code

In the exam this will be provided as a download for candidates to work on. These pages should not appear in the final exam paper, they are for examiners' reference.

Q1

```
package retail;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Collections;
import java.util.List;

public class BulkOrder {

  private final List<Product> items;
  private final CreditCardDetails creditCardDetails;
  private final Address billingAddress;
  private final Address shippingAddress;
  private final Courier courier;
  private final BigDecimal discount;

  public BulkOrder(
      List<Product> items,
      CreditCardDetails creditCardDetails,
      Address billingAddress,
      Address shippingAddress,
      Courier courier,
      BigDecimal discount) {
    this.items = Collections.unmodifiableList(items);
    this.creditCardDetails = creditCardDetails;
    this.billingAddress = billingAddress;
    this.shippingAddress = shippingAddress;
    this.courier = courier;
    this.discount = discount;
  }

  public void process() {

    BigDecimal total = new BigDecimal(0);

    for (Product item : items) {
      total = total.add(item.unitPrice());
    }

    if (items.size() > 10) {
      total = total.multiply(BigDecimal.valueOf(0.8));
    } else if (items.size() > 5) {
      total = total.multiply(BigDecimal.valueOf(0.9));
    }

    total = total.subtract(discount);

    CreditCardProcessor.getInstance().charge(round(total), creditCardDetails, billing-
Address);

    courier.send(new Parcel(items), shippingAddress);
  }

  private BigDecimal round(BigDecimal amount) {
    return amount.setScale(2, RoundingMode.CEILING);
  }
}
```

```java
package retail;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Collections;
import java.util.List;

public class SmallOrder {

  private static final BigDecimal GIFT_WRAP_CHARGE = new BigDecimal(3);

  private final List<Product> items;
  private final CreditCardDetails creditCardDetails;
  private final Address billingAddress;
  private final Address shippingAddress;
  private final Courier courier;
  private final boolean giftWrap;

  public SmallOrder(
      List<Product> items,
      CreditCardDetails creditCardDetails,
      Address billingAddress,
      Address shippingAddress,
      Courier courier,
      boolean giftWrap) {
    this.items = Collections.unmodifiableList(items);
    this.creditCardDetails = creditCardDetails;
    this.billingAddress = billingAddress;
    this.shippingAddress = shippingAddress;
    this.courier = courier;
    this.giftWrap = giftWrap;
  }

  public void process() {

    BigDecimal total = new BigDecimal(0);

    for (Product item : items) {
      total = total.add(item.unitPrice());
    }

    total = total.add(courier.deliveryCharge());

    if (giftWrap) {
      total = total.add(GIFT_WRAP_CHARGE);
    }

    CreditCardProcessor.getInstance()
                       .charge(round(total), creditCardDetails, billingAddress);

    if (giftWrap) {
      courier.send(new GiftBox(items), shippingAddress);
    } else {
      courier.send(new Parcel(items), shippingAddress);
    }
  }

  private BigDecimal round(BigDecimal amount) {
    return amount.setScale(2, RoundingMode.CEILING);
  }
}
```

```java
package retail;

import java.math.BigDecimal;

public interface Courier {
  void send(Parcel shipment, Address shippingAddress);

  BigDecimal deliveryCharge();
}
```

```java
package retail;

import java.math.BigDecimal;

public class CreditCardProcessor {

  private static final CreditCardProcessor INSTANCE = new CreditCardProcessor();

  private CreditCardProcessor() {}

  public static CreditCardProcessor getInstance() {
    return INSTANCE;
  }

  public void charge(BigDecimal amount, CreditCardDetails account, Address billingAd-
dress) {

    System.out.println("Credit card charged: " + account + " amount: " + amount);

    // further implementation omitted for exam question
  }
}
```

```java
import retail.*;

import java.math.BigDecimal;
import java.util.List;

public class RetailExample {

  public static void main(String[] args) {

    Courier royalMail = new RoyalMail();
    Courier fedex = new Fedex();

    BulkOrder bigOrder =
        new BulkOrder(
            List.of(
                new Product("One Book", new BigDecimal("10.00")),
                new Product("One Book", new BigDecimal("10.00")),
                new Product("One Book", new BigDecimal("10.00")),
                new Product("One Book", new BigDecimal("10.00")),
                new Product("One Book", new BigDecimal("10.00")),
                new Product("One Book", new BigDecimal("10.00"))),
            new CreditCardDetails("1234123412341234", 111),
            new Address("180 Queens Gate, London, SW7 2AZ"),
            new Address("180 Queens Gate, London, SW7 2AZ"),
            fedex,
            BigDecimal.ZERO);

    bigOrder.process();

    SmallOrder smallOrder =
        new SmallOrder(
            List.of(
                new Product("One Book", new BigDecimal("10.00"))
            ),
            new CreditCardDetails("1234123412341234", 111),
            new Address("180 Queens Gate, London, SW7 2AZ"),
            new Address("180 Queens Gate, London, SW7 2AZ"),
            royalMail,
            false);

    smallOrder.process();
  }
}
```

Q2

```
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JPanel;
import javax.swing.JTextField;
import javax.swing.WindowConstants;

public class TennisScorer {

  private int playerOneScore = 0;
  private int playerTwoScore = 0;
  private final String[] scoreNames = {"Love", "15", "30", "40"};

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

  private void display() {
    JFrame window = new JFrame("Tennis");
    window.setSize(400, 150);

    JButton playerOneScores = new JButton("Player One Scores");
    JButton playerTwoScores = new JButton("Player Two Scores");

    JTextField scoreDisplay = new JTextField(20);
    scoreDisplay.setHorizontalAlignment(JTextField.CENTER);
    scoreDisplay.setEditable(false);

    playerOneScores.addActionListener(
        e -> {
          playerOneWinsPoint();
          scoreDisplay.setText(score());
          if (gameHasEnded()) {
            playerOneScores.setEnabled(false);
            playerTwoScores.setEnabled(false);
          }
        });

    playerTwoScores.addActionListener(
        e -> {
          playerTwoWinsPoint();
          scoreDisplay.setText(score());
          if (gameHasEnded()) {
            playerOneScores.setEnabled(false);
            playerTwoScores.setEnabled(false);
          }
        });

    JPanel panel = new JPanel();
    panel.add(playerOneScores);
    panel.add(playerTwoScores);
    panel.add(scoreDisplay);

    window.add(panel);

    window.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE);
    window.setVisible(true);
  }

  private String score() {
    if (playerOneScore > 2 && playerTwoScore > 2) {
      int difference = playerOneScore - playerTwoScore;
      switch (difference) {
        case 0:
          return "Deuce";
        case 1:
          return "Advantage Player 1";
        case -1:
          return "Advantage Player 2";
        case 2:
          return "Game Player 1";
        case -2:
          return "Game Player 2";
      }
    }

    if (playerOneScore > 3) {
      return "Game Player 1";
    }

    if (playerTwoScore > 3) {
      return "Game Player 2";
    }

    if (playerOneScore == playerTwoScore) {
      return scoreNames[playerOneScore] + " all";
    }

    return scoreNames[playerOneScore] + " - " + scoreNames[playerTwoScore];
  }

  private void playerOneWinsPoint() {
    playerOneScore++;
  }

  private void playerTwoWinsPoint() {
    playerTwoScore++;
  }

  private boolean gameHasEnded() {
    return score().contains("Game");
  }
}
```