University of Glasgow

Friday 27th April 2018, 2.00 pm - 3.30 pm
(1 hour 30 minutes)

DEGREES OF MSci, MEng, BEng, BSc, MA and MA (Social Sciences)

COMPUTING SCIENCE 2Y: OBJECT ORIENTED SOFTWARE ENGINEERING 2

Answer all 3 questions

This examination paper is worth a total of 80 marks.

The use of a calculator is not permitted in this examination.

INSTRUCTIONS TO INVIGILATORS: Please collect all exam question papers and exam answer scripts and retain for school to collect. Candidates must not remove exam question papers.

1. This question concerns software modelling. (20 marks total)

(a) Represent the following association specifications using UML class diagrams:

(i) A company has many employers, but the employer can only work for one company.[2]

(ii) Should an employee retire, then the company recruits another employee. [2]

(iii) Should an employee retire, then the company folds up. [2]

(iv) Should a company be liquidated, then all employees lose their jobs. [2]

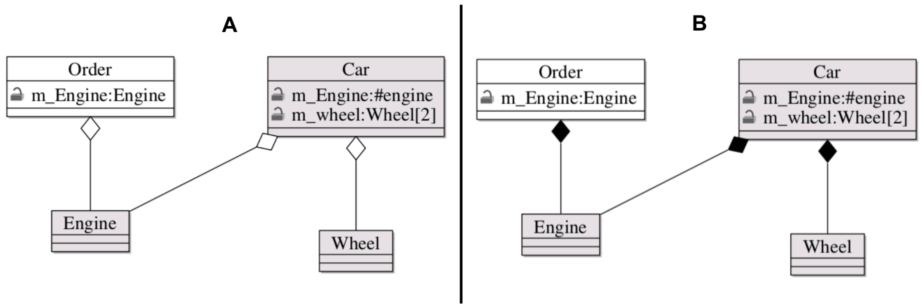
(v) Should a company be liquidated, then all employees move to another company.[2]

(b) Generate a class diagram for a Vehicle System using the following narrative:

- All Vehicles have common attributes (speed and colour) and common behaviour (turnLeft, turnRight).
- Bicycle and MotorVehicle are both kinds of Vehicle.
- MotorVehicles have engines and license plates, along with some behaviour that allows us to examine its attributes. Whereas, a Bicycle is able to ring a bell.
- MotorBike and Car are the two types of MotorVehicles.
- MotorBike has the capability to rev its engine to increase its speed. Whereas, a car has doors and air conditioner, along with the behaviour that allows us to examine these attributes.

[5]

(c) Which of the following two class diagrams showing the relationship between an order, an engine, a car and a wheel is correct. Briefly explain why? [5]



2. This question concerns software quality and testing. (30 marks total)

(a) The `main` program method below creates a `JButton` object and then adds an `ActionListener` to the object.

```
public static void main(String args[]) {
   JButton button1 = new JButton("PressME");
   button1.addActionListener(new ActionListener() {
   @Override
   public void actionPerformed(ActionEvent e) {
      int a = args.length;
      int b = 3;
      int c = 0;
      if(a>b) {
        c = b;
      }
      else {
        c = a;
      }
     }
   });
}
```

(i) Draw a Control Flow Graph representing `main` function. Marks will be awarded based on appropriate labelling and clear identification of control flows. [5]

(ii) What is the Cyclomatic Complexity of the `main` function? [2]

(iii) Based on your solution to (ii) above, briefly explain whether the `main` function is manageable or not manageable. [3]

(b) Table 1 below is the outcome of Chidamber and Kemerer metrics computed for three publicly available OSGi component based frameworks.

Table 1: Chidamber and Kemerer metrics for component based frameworks

Framework	**WMC**	**DIT**	**NOC**	**CBO**	**RFC**	**LCOM**
equinox	7.66	1.05	0.14	4.47	21.6	45.08
felix	9.13	1.18	0.08	4.44	26.95	89.31
knopflerfish	8.57	1.19	0.05	5.11	25.7	47.31

Assume you are a software architect tasked with the responsibility of recommending one of these three frameworks to build a distributed retail management system. Which framework will you recommend if the aim is to achieve the following quality objective? Clearly justify your reasons.

(i) Minimise complexity [3]

(ii) Maximise re-usability [3]

(iii) Maximise modularity [4]

(c) Assume that a GregorianCalendar class extends Calendar, and contains the following constructor and methods:

```
public GregorianCalendar(int year, int month, int dayOfMonth){}
public void add(int field, int amount){}//field=year,month,dayOfMonth
public int get(int field){}
```

The class DateTest below contains two unit tests to check that no Calendar object can ever get into an invalid state.

```
  public class DateTest {
    @Test
    public void test1(){
        Calendar cal = new
           GregorianCalendar(2050,Calendar.FEBRUARY,15);
       cal.add(Calendar.DATE, 4);

       assertEquals(cal.get(Calendar.YEAR), 2050);
       assertEquals(cal.get(Calendar.MONTH), Calendar.FEBRUARY);
       assertEquals(cal.get(Calendar.DAY_OF_MONTH), 19);
    }

    @Test
    public void test2(){
       Calendar cal = new
          GregorianCalendar(2050,Calendar.FEBRUARY,15);
       cal.add(Calendar.DATE, 14);

       assertEquals(cal.get(Calendar.YEAR), 2050);
       assertEquals(cal.get(Calendar.MONTH), Calendar.MARCH);
       assertEquals(cal.get(Calendar.DAY_OF_MONTH), 1);
    }
}
```

Identify five problems in DateTest. (two marks each). [10]

3. This question concerns software design patterns (30 marks total)

(a) The program code for `Rectangle` and `Square` as shown below demonstrates a relationship between a base and a derived class.

```
public class Rectangle {
  private double width, height;

  public double area() {
    return width * height;
  }
  public void setWidth(double w) {
    width = w;
  }
  public void setHeight(double h) {
    height = h;
  }
  public double getWidth() {
    return width;
  }
  public double getHeight() {
    return height;
  }
}

public class Square extends Rectangle {
  public void setWidth(double w) {
    super.setWidth(w);
    super.setHeight(w);
  }

  public void setHeight(double h) {
    super.setWidth(h);
    super.setHeight(h);
  }
}
```

(i) Infer the invariant satisfied by `setWidth()` and `setHeight()` operations in the `Rectangle` class. [2]

(ii) Infer the postconditions for `setWidth()` in `Rectangle` and `Square` classes. [3]

(iii) Give a short explanation on whether the derived class satisfies the Liskov Substitution Principle. [3]

(iv) Present a brief argument on whether inheritance is a suitable approach to modelling the relationship between `Square` and `Rectangle` classes. [2]

(b) Assume the State Machine representation for a Gumball system in Figure 1.

(i) List four disadvantages of implementing the state machine using a set of state constants. [4]

(ii) Based on the state design pattern, draw a UML class diagram showing how the state machine can be implemented as a set of state objects. [6]

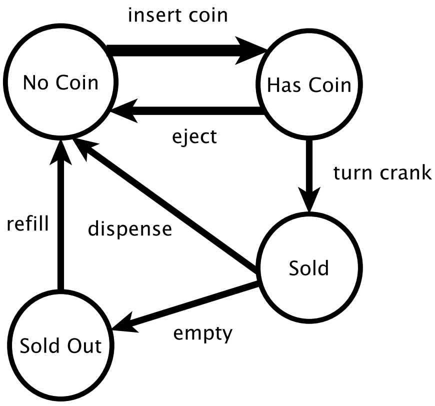


Figure 1: Gumball State Machine

(c) The program snippet below contains `Component` and `Visitor` abstract base classes. `ComponentX` and `VisitorY` are placeholders for one of many possible derived concrete classes of `Component` and `Visitor` respectively.

```
Component c = new ComponentX();
c.ls();
Visitor v = new VisitorY();
c.accept(v);
```

(i) Use the program snippet to briefly describe the concept of single-dispatch during program execution. [5]

(ii) Use the program snippet to briefly describe the concept of double-dispatch during program execution. [5]