UECS2344 Software Design: Practical 11

Design Patterns Problems

2015 Q2 (b) (15 marks)

A software designer would like to create a class hierarchy that can model classes representing single units as well as a collection of units in a way that allows objects from either of these classes to be treated in a uniform way. An example of the code in the `main()` program that would use such a hierarchy is shown in Figure 2.1, and the corresponding output from that program is shown in Figure 2.2.

Identify the design pattern used here. Write the code for the classes in the class hierarchy based on this pattern that will allow the code statements shown in Figure 2.1 to produce the output in Figure 2.2.

```
Item harddisk = new Part("Hard Disk", 300);
Item intelcpu = new Part("Intel CPU", 100);
Item amdcpu = new Part("AMD CPU", 150);
Item monitor = new Part("Monitor", 500);
Item wiring = new Part("Lab Wiring", 800);

Item computer1 = new Assembly("Computer 1");
computer1.addItem(harddisk);
computer1.addItem(intelcpu);
computer1.addItem(monitor);

Item computer2 = new Assembly("Computer 2");
computer2.addItem(amdcpu);
computer2.addItem(monitor);

Item computerlab = new Assembly("Computer Lab");
computerlab.addItem(computer1);
computerlab.addItem(computer2);
computerlab.addItem(wiring);

System.out.println(harddisk);
System.out.println(intelcpu);
System.out.println(computer1);
System.out.println(computerlab);
```

Figure 2.1

```
Hard Disk (cost 300)

Intel CPU (cost 100)

Computer 1 is a collection. The components in it are:
Hard Disk (cost 300)
Intel CPU (cost 100)
Monitor (cost 500)
Overall cost is 900

Computer Lab is a collection. The components in it are:
Computer 1 is a collection. The components in it are:
Hard Disk (cost 300)
Intel CPU (cost 100)
Monitor (cost 500)
Overall cost is 900
```

```
Computer 2 is a collection. The components in it are:
AMD CPU (cost 150)
Monitor (cost 500)
Overall cost is 650

Lab wiring (cost 800)
Overall cost is 2350
```

Figure 2.2

2016 Q2 (a) (12 marks)

Suppose you are a software designer working on a project and have come across the following design situations. Recommend a design pattern that can be used to solve the issues. State the name of the design pattern and briefly explain how the pattern can be applied.

(i) You need to create one object (or instance) of a database connection class and use this one object in all methods that require it. You need to make sure that the same object is used for all the methods. (6 marks)

(ii) When a new application was being developed, you realise that you can reuse a class named `Pricing` from an existing application. This existing class contains a method named `ComputePrice` but the method required in the class for the new application is to be called `DeterminePrice`. You cannot change anything in the existing class `Pricing`. You also cannot change the method name in the new application. (6 marks)

2016 Q5 (a) (20 marks)

An application contains a class that represents a part with a description and a cost. A part may itself be made up of an *assembly* of other parts, that is, a part may consist of subparts. The cost of an *assembly* is the total cost of all its subparts. The class should include methods that return the description and cost of the part.

(i) Suggest a design pattern that can be used and draw a class diagram to illustrate how it is to be applied in this context. (7 marks)

(ii) Write Java code to implement the classes in the class diagram you have drawn for part (a) (i) above. (13 marks)

2017 Q4 (a) (16 marks)

Analyse the Java code below.

(i) Identify the design pattern that has been applied. (5 marks)

(ii) What problem does this design pattern generally address? (4 marks)

(iii) Draw a class diagram (showing only class names and relationships) to represent the code and identify the role the classes play with respect to the design pattern. (4 marks)

```
public class Dimension {

	private double measurement;

	public Dimension() {
	}

	public double getMeasurement() {
		return measurement;
	}

	public void setMeasurement(double measurement) {
		this.measurement = measurement;
	}
}

public interface DimensionAny {

	public double getSquareFeet();

	public void setSquareFeet(doublemeasurement);

	public double getSquareMeters();

	public void setSquareMeters(double measurement);
}

public class DimensionBoth implements DimensionAny {

	private Dimension dimension;

	public DimensionBoth() {
			dimension = new Dimension();
	}

	public double getSquareFeet() {
		return dimension.getMeasurement();
	}

	public void setSquareFeet(double measurement) {
		dimension.setMeasurement(measurement);
	}

	public double getSquareMeters() {
		return dimension.getMeasurement()*0.092903;
	}

	public void setSquareMeters(double measurement) {
		dimension.setMeasurement(measurement/0.092903);
	}
}

public class DimensionDemo {

	public static void main(String[] args) {

		 DimensionAnydimension = new DimensionBoth();

		 dimension.setSquareFeet(1000.0);
		 System.out.println("In square feet:" +
                                        dimension.getSquareFeet());
```

```
                System.out.println("In square meters:" +
                                                dimension.getSquareMeters());

                dimension.setSquareMeters(92.903);
                System.out.println("In square feet:" +
                                                dimension.getSquareFeet());
                System.out.println("In square meters:" +
                                                dimension.getSquareMeters());
        }
}
```

2017 Q5 (a) (13 marks)

Analyse the Java code below.

(i) Identify the design pattern that has been applied. (5 marks)

(ii) Explain the circumstances under which this design pattern is applicable. (3 marks)

(iii) Draw a class diagram (showing only class names and relationships) to represent the code and identify the role the classes play with respect to the design pattern. (5 marks)

```
public interface Booking {
     public abstract void bookSeat();
}

public class PersonalBooking implements Booking {

     public PersonalBooking() {
             . . .
     }

     public void bookSeat() {
             . . .
     }
}

public class GroupBookingimplements Booking {

      private ArrayList<PersonalBooking>bookings;

      public GroupBooking() {
             bookings = new ArrayList<PersonalBooking>();
      }

      public void bookSeat() {
             for(PersonalBookingaBooking : bookings) {
                    aBooking.bookSeat();
             }
      }

      public void add(IndividualBooking booking){
             bookings.add(booking);
      }

      public void remove(IndividualBooking booking){
              bookings.remove(booking);
      }
}
```

```
public class BookingDemo {

       public static void main(String[] args) {
            GroupBooking group = new GroupBooking();

            IndividualBooking booking1 = new IndividualBooking();
            group.add(booking1);

            IndividualBooking booking2 = new IndividualBooking();
            group.add(booking2);

            IndividualBooking booking1 = new IndividualBooking();
            group.add(booking3);

            group.bookSeat();
        }
}
```

2018 Q4 (a) (15 marks)

Analyse the Java code and corresponding output (Figure 4.1) below.

(i) Identify the design pattern that has been applied. (5 marks)

(ii) What situation does this design pattern generally address? (4 marks)

(iii) Draw a class diagram (showing only class names and relationships) to represent the code and identify the role the classes play with respect to the design pattern. (6 marks)

```
public interface Element {

     public abstract int getSize();

     public abstract void addElement(Element elment);

}

public class SingleElement implements Element {

      private int size;

      public SingleElement(int size) {
            this.size = size;
      }

      public int getSize() {
            return size;
      }

      public void addElement(Element element) {

      }
}

public class MultipleElement implements Element {
```

```
      private ArrayList<Element> elements;

      public MultipleElement() {
            elements = new ArrayList<Element>();
      }

      public void addElement(Element element) {
            elements.add(element);
      }

      public int getSize() {
            int totalSize = 0;

            for (6n ti=0; i< elements.size(); i++) {
                 Element element = (Element)elements.get(i);
                 totalSize += element.getSize();
            }

            return totalSize;
      }
}

public class ElementTest {

      public static void main(String[] args) {

            Element one = new SingleElement(20);
            Element two = new SingleElement(30);

            Element multiple1 = new MultipleElement();

            multiple1.addElement(one);
            multiple1.addElement(two);

            System.out.println("Size of multiple1 is "
                              + multiple1.getSize());

            Element three = new SingleElement(5);
            Element four = new SingleElement(10);

            Element multiple2 = new MultipleElement();

            multiple2.addElement(three);
            multiple2.addElement(four);
            multiple2.addElement(multiple1);;

            System.out.println("Size of multiple2 is "
                               + multiple2.getSize());
      }
}
```

Output

```
Size of multiple1 is 50
Size of multiple2 is 65
```