UNIVERSITY *of* LIMERICK

OLLSCOIL LUIMNIGH

FACULTY *of* SCIENCE *and* ENGINEERING

Department of Computer Science
and Information Systems

End-of-Semester Assessment Paper

Academic Year:	2008-2009	Semester:	I
Module Title:	Software Architecture	Module Code:	CS4135
Duration of Exam:	2.5 Hours	Percent of Total Marks:	50
Lecturer(s):	J.J. Collins	Paper marked out of :	100

Instructions to Candidates:

- **Answer Q1, and any two other questions**.
- It is at your discretion whether or not a diagram would be of value in illustrating your discussion, when one has **not** been explicitly requested in the question to reproduce one.

Q1 Answer ALL parts. Total marks awarded for this question: 40.

a) What are the characteristics of good software?

4 marks.

b) Use a diagram to illustrate Fowler's Pattern of Separated Interfaces and briefly discuss its benefits.

4 marks.

c) Discuss a few of the contributing factors to the failure of the software engineering profession to realise widespread REUSE?

4 marks.

d) Describe the features of a message broker, and briefly illustrate how a canonical data model can be used to solve the problem of architectural spaghetti.

4 marks.

e) Describe the N-tier Client Server architectural pattern, and briefly describe the benefits arising from the use of this pattern.

4 marks.

f) Describe the Process Coordinator pattern using quality attribute analysis. The following headings should be included: availability, failure handling, modifiability, performance, and scalability.

4 marks.

g) List the fields in a generic architecture documentation template.

4 marks.

h) Provide a definition of software architecture, and differentiate it from High-level design.

4 marks.

i) Briefly describe the Model Driven Architecture (MDA) making referring to

- Platform Independent Model (PIM)
- Platform Specific Model (PSM)
- Unified Modelling Language (UML)
- Common Warehouse MetaModel (CWM)
- Meta Object Facility (MOF)
- Synchronisation.

4 marks.

j) How does Aspect-Oriented Programming (AOP) overcome "the tyranny of dominant decomposition"?

4 marks.

Q2 Answer ALL parts. Total marks awarded for this question: 30.

a) Describe 2 process and 2 structural recommendations for the development of a "good" architecture.

6 marks.

b) Draw a diagram to illustrate the three architectural structures of a system, and briefly describe the Decomposition and Layered structures in Module.

6 marks.

c) Specify quality attribute scenarios for modifiability OR performance in tabular format.

6 marks.

d) Briefly outline modifiability OR performance tactics to achieve quality attributes.

6 marks.

e) Describe the principles of the Architecture Tradeoff Analysis Methodology (ATAM).

6 marks.

Q3 Answer ALL parts. Total marks awarded for this question: 30.

a) What is the intent of the Observer design pattern?
Draw a sequence diagram using the UML to illustrate the structure of the Observer design pattern.

6 marks.

b) Discuss the problems that are address by the Interceptor architectural pattern?

6 marks.

c) Describe the solution offered by the Interceptor architectural pattern, and illustrate your answer with a class diagram.

6 marks.

d) Use a sequence diagram to illustrate the dynamics of the Interceptor architectural pattern.

6 marks.

e) The Decorator design pattern is used to add responsibilities to classes. Compare and contrast the Decorator with the Interceptor architectural pattern with respect to the provision of additional responsibilities.

6 marks.

Q4 Answer ALL parts. Total marks awarded for this question: 30.

You have been requested to develop an application that will draw rectangles and circles with either of two drawing programs referred to as DP1 and DP2. The rectangle is defined as two pairs of opposite points, and you will be informed which drawing program to use after instantiating the rectangle.

The difference between drawing programs is summarised in table 1.

Table 1

	DP1	**DP2**
To draw a line	draw_a_line(x1, y1, x2, y2)	Drawline(x1, y1, x2, y2)
To draw a circle	draw_a_circle(x, y, r)	drawcircle(x, y, r)

Figure 1 depicts an initial design that was sketched to support these requirements.

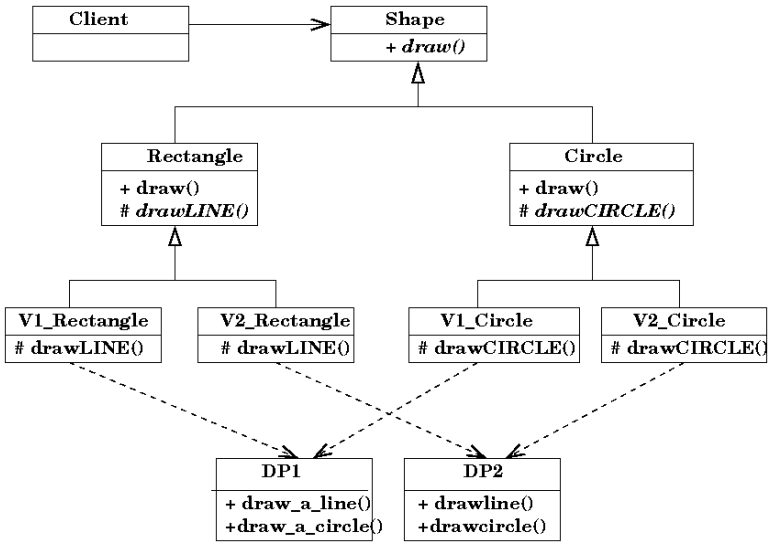

Figure 1

Figure 2 overleaf depicts a coding fragment to illustrate the implementation of a rectangle.

```
abstract public class Rectangle extends Shape{
  private double _x1, _y1, _x2, _y2;
  public Rectangle (double x1, double y1, double x2, double y2) {
    _x1 = x1; _y1=y1; _x2=x2; _y2=y2;
  }
  public void draw() {
    drawLINE(_x1, _y1, _x2, _y1);
    drawLINE(_x2, _y1, _x2, _y2);
    drawLINE(_x2, _y2, _x1, _y2);
    drawLINE(_x1, _y2, _x1, _y1);
  }

  abstract protected void drawLINE(double x1, double y1, double
x2, double y2);
}
```

Figure 2

One of the problems with this design is the explosion in concrete shapes if requested to support additional shapes or drawing programs.

a) Briefly critique the diagram in figure 1.

3 marks.

b) What is the intent of the Bridge Design pattern?
Draw a structural diagram that captures intent of the Bridge design pattern.

6 marks.

c) Sketch a design for the application that illustrates how the bridge design pattern can help to decouple the abstraction (Shape) from its implementation (Drawing). Justify the design decisions implicit in the new diagram.

9 marks.

d) Produce coding fragment or pseudocode to illustrate an implementation of the application whose design was sketched in part (c).

6 marks.

e) One of the principles advocated by the Gang of Four is "Program to interfaces, not implementation". Draw a diagram to illustrate this principle, and discuss the benefits arising from application of this principle.

6 marks.

Q5 Answer ALL parts. Total marks awarded for this question: 30.

a) Discuss the creational problem embodied in the coding sample in figure 3. Draw a simple class diagram to capture the impact of applying the Replace Constructors with Creation Methods refactoring technique to the code. Briefly outline the rationale underlying the diagram. Please note that one is not required to demonstrate knowledge of the domain specific intent of the constructors in figure 3.

9 marks.

b) The coding fragments in Appendix A are taken from chapter 1 in the texts "Refactoring: Improving the Design of Existing Code" by Martin Fowler. Fowler proceeds to critique the code by stating that "it is not object-oriented". Summarise the critique, and state the reasons for agreeing or disagreeing with the critique.

6 marks.

c) Suggest an alternative design incorporating a design pattern that might better resolve the criticisms directed at the original codebase in Appendix A. Briefly discuss how the new design resolves the problems identified in (b).

9 marks.

d) Describe three of the following bad code smells:

1. Long Method
2. Feature Envy.
3. Primitive Obsession
4. Lazy Class
5. Temporary Field
6. Shotgun Surgery

6 marks.

```
// Adapted from Kierevsky, J. Refactoring to Patterns.
// Addison-Wesley. 2005.

public class Loan ….

  public Loan(double commitment, int riskRating, Date maturity) {
    this(commitment, 0.00, riskRating, maturity, null};
  }

  public Loan(double commitment, int riskRating, Date maturity,
              Date enquiry) {
    this(commitment, 0.00, riskRating, maturity, expiry};
  }

  public Loan(double commitment, double outstanding,
           int riskRating, Date maturity, Date expiry) {
    this(null,commitment,outstanding,riskRating,maturity,expiry};
  }

  public Loan(CapitalStrategy capitalStrategy, double commitment,
          int riskRating, Date maturity, Date expiry) {
    this(capitalStrategy,commitment,0.00,riskRating,maturity, expiry};
  }

  public Loan(CapitalStrategy capitalStrategy, double commitment,
              int riskRating, Date maturity, Date expiry) {
    this.commitment = commitment;
    this.outstanding = outstanding;
    this.riskRating = riskRating;
    this.maturity = maturity;
    this.expiry = expiry;
    this.capitalStrategy = capitalSTrategy;

    if(capitalStrategy == null) {
      if(expiry==null)
        this.capitalStrategy = new CapitalStrategyTermLoan()
      else if (maturity == null)
        this.capitalStrategy = new CapitalStrategyRevolver()
      else
        this.capitalStrategy = new CapitalStrategyRCTL()
    }
}
```

Figure 3

Appendix A

```
// From book: 'Refactoring' by Martin Fowler
// This is the original code before refactoring begins



public class Movie {

   public static final int CHILDRENS = 2;
   public static final int NEW_RELEASE = 1;
   public static final int REGULAR = 0;

   private String _title;
   private int _priceCode;

   public Movie(String title, int priceCode) {
      _title = title;
      _priceCode = priceCode;
   }

   public int getPriceCode() {
      return _priceCode;
   }

   public void setPriceCode(int arg) {
      _priceCode = arg;
   }
   public String getTitle() {
      return _title;
   }


}

/**
 * The rental class represents a customer renting a movie.
 */
public class Rental {

   private Movie _movie;
   private int _daysRented;

   public Rental(Movie movie, int daysRented) {
      _movie = movie;
      _daysRented = daysRented;
   }

   public int getDaysRented() {
      return _daysRented;
   }

   public Movie getMovie() {
      return _movie;
   }



}
```

```
import java.util.Enumeration;
import java.util.Vector;

public class Customer {

   private String _name;
   private Vector _rentals = new Vector();

   public Customer(String name) {
      _name = name;
   }
   public void addRental(Rental arg) {
      _rentals.addElement(arg);
   }
   public String getName() {
      return _name;
   }

   public String statement() {
      double totalAmount = 0;
      int frequentRenterPoints = 0;
      Enumeration rentals = _rentals.elements();
      String result = "Rental Record for " + getName() + "\n";

      while (rentals.hasMoreElements()) {
         double thisAmount = 0;
         Rental each = (Rental) rentals.nextElement();

         //determine amounts for each line
         switch (each.getMovie().getPriceCode()) {
         case Movie.REGULAR:
            thisAmount += 2;
            if (each.getDaysRented() > 2)
               thisAmount += (each.getDaysRented() - 2) * 1.5;
            break;
         case Movie.NEW_RELEASE:
            thisAmount += each.getDaysRented() * 3;
            break;
         case Movie.CHILDRENS:
            thisAmount += 1.5;
            if (each.getDaysRented() > 3)
               thisAmount += (each.getDaysRented() - 3) * 1.5;
            break;
         }

         // add frequent renter points
         frequentRenterPoints++;
         // add bonus for a two day new release rental
         if ((each.getMovie().getPriceCode() == Movie.NEW_RELEASE) &&
              each.getDaysRented() > 1)
            frequentRenterPoints++;
         // show figures for this rental
         result += "\t" + each.getMovie().getTitle() + "\t" +
                     String.valueOf(thisAmount) + "\n";
         totalAmount += thisAmount;
      }
      // add footer lines
      result += "Amount owed is " + String.valueOf(totalAmount) + "\n";
      result += "You earned " + String.valueOf(frequentRenterPoints) +
                  " frequent renter points";

      return result;
   }
}
```