EXAMINATION QUESTION PAPER - SEMESTER 2, 2006/7

Module code:	**CSP032N**
Module title:	**Object - Oriented Software Design and Development**
Module leader:	**Dr Quan Dang**

Date:	**Today**
Day / evening:	**Day**
Duration:	**2 hours**

Materials supplied:	**None**
Materials permitted:	**Pen; Blank paper**
Warning:	**Candidates are warned that possession of unauthorised materials in an examination is a serious assessment offence.**

Instructions to candidates:	**Answer any Four(4) of the SIX questions. Each question carries 25 marks. No credit will be given for attempting further questions.**

DO NOT TURN PAGE OVER UNTIL INSTRUCTED

QUESTION 1

(a) What are the "*use case*" and its purpose in software design?

(5 marks)

(b) Provide an example of a use case from a software development scenario of your own choosing, listing its sequence of actions, and variations.

(7 marks)

(c) Explain in any suitable manner, what is a CRC card, indicating the purpose of each part of the card.

(5 marks)

(d) *"CRC cards can be used to negotiate the use case's operations with class' responsibility in object-oriented software design"*. Identify at least two classes from the scenario in your answer to sub question (b), then draw CRC card for the two classes to explain the statement.

(8 marks)

QUESTION 2

(a) Explain in any suitable manner, what "*class design with loose coupling*" means, providing a concrete example in your answer.

(8 marks)

(b) In the context of OO programming the law of Demeter states that a method should only use objects that are: instance fields of its class, parameters, and objects that it constructs with new methods.

(i) Inspect the code below and explain why it violates the law of Demeter.

(8 marks)

```
public class StudentRegistry
{
   public void showModuleTaken(Student aStudent)
   {
      Module ModuleTaken[ ] = aStudent.getModuleList();
      for(int i=0; i< ModuleTaken.length; i++)
        this.printToScreen(ModuleTaken[i].moduleTitle());
   }
  // NB: there are other methods in this class
}
```

(ii) There three classes present in the StudentRegistry code above. Re-write the classes i.e. StudentRegistry, Student and Module for it to be allowed by the law of Demeter, preserving the same effect of the showModuleTaken() method.

(9 marks)

QUESTION 3

(a) In the context of Object-oriented programming, concisely explain the concepts of Encapsulation and Information Hiding, and why these features make the class more robust.

(8 marks)

(b) Consider the code in Appendix A. Is class Rectangle mutable? Justify your answer.

(7 marks)

(c) Applying the principle of Information hiding, re-write the code given in Appendix A, including the RectangleTester class, to provide NO direct access to class Rectangle's data members, retaining the same effect of the given program. Explain why your re-written code would make the class to be more robust (if any).

(10 marks)

QUESTION 4

(a) In Java programming, explain in any suitable manner, what an instance data field of a class is, and what a class (static) data field is, providing an example for each of the concepts.

(4 marks)

(b) Consider a BankAccount class' code below.

```
public class BankAccount
{
   private double balance;
   public void deposit(double amount) { balance += amount;  }
   public void withdraw(double amount){ balance -= amount;  }
   public double getBalance() { return balance; }
}
```

(i) Write additional code to the class definition in order for the bank: (1) to keep track of, and to retrieve, the number of all existing back account objects of the class, and (2) to assign an account number to each account at its creation time and to retrieve it afterward. Justify your solution.

(8 marks)

(ii) A first-step account is a bank account, except there is a service charge of £0.50 for a withdrawal. Define a subclass FirstStepAccount from the BankAccount class, with a constructor FirstStepAccount(double initialBalance). Redefine the withdraw() method to deduct transaction fees.

(9 marks)

(c) Is the type of polymorphism in the code of your answer to sub-question (b) of the type (i) polymorphism via inheritance or (ii) polymorphism through interface type? Justify your answer.

(4 marks)

QUESTION 5

(a) Explain, in any suitable manner, what an *abstract class* is. Can an abstract class include a method with implementation, providing an example in your answer?

(7 marks)

(b) Explain in any suitable manner what a Java *interface* is and its purpose, comparing the interface to the abstract class.

(8 marks)

(c) Inspect the code of a Java program in Appendix B. Write your code to add a JButton to the given program so that when the button is clicked at runtime the content of the textfield MyTextField will be reset (empty).

(10 marks)

QUESTION 6

(a) Explain, in any suitable manner, the meaning of the following statement: *"In object-oriented programming, a subclass not only inherits what has been built in its parent class(es) but can also be further extended with new functionality"*.

(8 marks)

(b) Draw a class hierarchy tree that shows the inheritance relationship of classes that represent Employee, Full-time-Employee, Part-time-Employee, and Full-time-Manager in a company's payroll system, using UML class diagram's notation.

(7 marks)

(c) Referring to the classes from sub question (b), assume the Employee class specifies a public method calculateMonthlyPay( ), but different calculation formula is applied for different sub classes of Employee, i.e. Full-time-Employee, Part-time-Employee, and Full-time-Manager. Describe in any suitable fashion how you would implement the Java.util.Comparable *interface* in your code in order for any two objects of the Employee class to be comparable by their monthly pays.

(10 marks)

P.T.O

APPENDIX A

```
public class RectangleTester
{
    public static void main(String[] args)
    {
        Rectangle aRectangle = new Rectangle();
        aRectangle.width = 15;
        aRectangle.height = 10;
        System.out.println("The area is: " + aRectangle.area());
    }
}//~public class RectangleT...

class Rectangle
{
    public double width = 0;
    public double height = 0;

    public double area()
    {
        return (width * height);
    }
}//~class Rectangle...
```

APPENDIX B

```
//Start of code
import java.awt.*;
import java.awt.event.*;
import javax.swing.*;
public class TextFieldTest extends JFrame
{
    private JTextField MyTextField = new JTextField("Hello World!");
    // set up GUI
    public TextFieldTest()
    {
        Container contentPane = getContentPane();
        contentPane.setLayout(new FlowLayout());
        contentPane.add(MyTextField);

        setSize( 250, 100 );
        setVisible( true );
    }//~public TextFieldTest()...

    // execute application
    public static void main( String args[] )
    {
        TextFieldTest application = new TextFieldTest();
    }
}
```

End of the paper