EXAMINATION QUESTION PAPER: **Semester A, 2007/8**

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

Date:	**24 January 2008**
Day / evening:	**Evening**
Start time:	**18:00**
Duration:	**2 Hours**

Exam type:	**Unseen. Closed**
Materials supplied:	**None**
Materials permitted:	**None**
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 (6)questions.** **Each question carries 25 marks.** **No credit will be given for attempting further questions.** **DO NOT TURN PAGE OVER UNTIL INSTRUCTED**

QUESTION 1

Consider the development of a software system *to help the counter staff allocating* vacant PCs for use in a public library to its readers on the basis of first-come-first-serve. Each session is initially allocated to a reader for 30 minutes, which can be extended only once for another 30 minutes at the reader request. After having used up the extended 1-hour session the reader can request for another new session provided there is a vacant PC and s/he is at the top of the queue for PCs at that time. All the reader's requests must be made in person at the counter. Please note that the user of the system in question is the library counter staff, not the library's reader.

(a) Determine and draw a use case diagram of the described system.

(4 marks)

(b) List the relevant operations for each of the use cases.

(6 marks)

(c) Identify classes and draw a UML class diagram of the classes and their relationships. No class member is required in the diagram.

(4 marks)

(d) Draw CRC cards for the classes from your answer to Question 1(c), indicating class responsibilities and collaborators.

(6 marks)

(e) Draw a sequence diagram for processing a user request to extend a session.

(5 marks)

QUESTION 2

(a) In the context of object-oriented design

(i) Name three types of relationship among classes

(3 marks)

(ii) For each type the relationship from your answer to sub-question 2a(i) provide a definition, an example and its standard UML notation in a UML class diagram.

(6 marks)

(b) Inspect the code of a simple calculator in Java provided in Appendix A at the end of this paper.

(i) Draw a UML class diagram of the classes, explaining the relationship between them.

(4 marks)

(ii) Re-write the code to minimise the dependency between the classes.

(8 marks)

(iii) Explain why your re-written code has less class dependency than the original version.

(4 marks)

QUESTION 3

(a) What is *Information hiding* in object oriented programming? Explain how *Information hiding* can be achieved through *access qualifiers*: *public, protected* and *private* in Java?

(8 marks)

(b) Inspect the code of classes `Semester` and `TestSemester`, and the screenshot of the `TestSemester` output at runtime provided in Appendix B at the end of this paper. Explain, at the code level, why the dates of the semester start and end are the same as displayed.

(8 marks)

(c) Modify the implementation of the `Semester` class, but not the `TestSemester` class, in order for the output of the `TestSemester` class to display the correct start and end dates of the Semester object with ID 1.

(9 marks)

QUESTION 4

(a) What is "*event-driven programming*"?

(5 marks)

(b) Concider the Java event-handling model

(i) What is the "*event source*" in the model? Provide examples of two concrete Java Swing components that could be event sources.

(5 marks)

(ii) What is the role of the *event* class in event handling? Provide four examples of *Java event classes*.

(5 marks)

(iii) Describe how an event can be 'captured' by an *event listener*?

(5 marks)

(c) In Java, distinguish between the *Listener* interface and the *Adapter* class in implementing an event handler.

(5 marks)

QUESTION 5

(a) Distinguish between static and dynamic data structures. Provide an example of a static data structure, and one of a dynamic data structure.

(8 marks)

(b) Describe, in any suitable fashion, the linked list data structure and its constituents?

(9 marks)

(c) Explain, in any suitable fashion, why the size of a linked list is changed in terms of the occupied memory, whenever nodes are added or removed?

(8 marks)

QUESTION 6

(a) Explain why "*class design with loose coupling*" is beneficial in object-oriented programming. Provide a concrete example to illustrate 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 the code in the class Timetable violates the law of Demeter.

```
public class Timetable {
   public String showModuleLeader(Module myModule)
   {   Module aModule= myModule;
       return aModule.getModuleLeader().getStaffName();     }
  // there are other methods ….
}

class Module {
   Staff moduleLeader;
   public void setModuleLeader(Staff aStaff)
   {   moduleLeader = aStaff;      }

   public Staff getModuleLeader()
   {       return moduleLeader;      }
}//~class Module...

class Staff {
   String name;
   public Staff(String StaffName)
   {       name = StaffName;    }

   public String getStaffName()
   {       return name;    }
}//~class Staff...
```

(8 marks)

(ii) Re-write the code of classes Timetable and Module to conform to the law of Demeter, preserving the same effect of the showModuleLeader() method.

(9 marks)

Appendix A

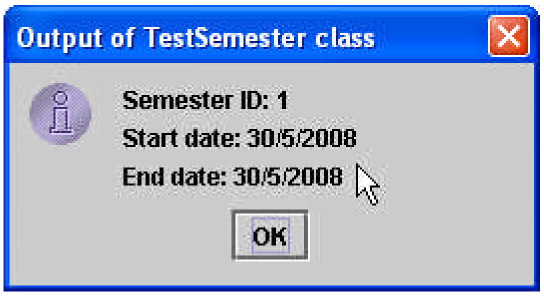
```
import javax.swing.JOptionPane;
/** simple calculator */
class Calculator {
    public static void main(String arg[])
    {        AddOperation performAddition = new AddOperation();
        int sum = performAddition.getSum();
        System.out.println("The result is: " + sum);
    }
}

/** a class to get user input */
class Keypad {
    public int getInput()
    {   String input = JOptionPane.showInputDialog("Please enter an
integer?");
        if (input != null) return Integer.parseInt(input);
        else return 0;
    }
}

/** an operation to calculate the sum of 2 integers */
class AddOperation {
    public int getSum()
    {   Keypad myKeypad = new Keypad();
        int num1=0, num2=0;
        num1 = myKeypad.getInput();
        num2 = myKeypad.getInput();
        return num1 + num2;
    }
}//~class Add...
```

Appendix B

```
/** TestSemester Class */
import java.util.Date;
import javax.swing.JOptionPane;

public class TestSemester
{
    public static void main (String args[])    {
        Semester myApt = new Semester(1);

        // set semester start date
            Date tempDate = new Date(2007, 9, 20);
        myApt.setStartDate(tempDate);

        // set semester end date
        tempDate.setTime((new Date(2008, 5, 30)).getTime());
        myApt.setEndDate(tempDate);

        //display semester details
            String outputString = myApt.toString() + "\n";
        JOptionPane.showMessageDialog(null, outputString,
            "Output of TestSemester class",
JOptionPane.INFORMATION_MESSAGE);
        System.exit(0);
    }//~public static void main ...
}//~public class TestSemesterC...
```

```
/** Semester Class */
class Semester
{
    private int semesterID;
    private Date start, end;

    Semester (int eventID)
    {   semesterID = eventID;    }

    public void setStartDate(Date startDate)
    {   start= startDate;    }

    public void setEndDate(Date endDate)
    {   end= endDate;    }

    public String toString()    {
        return "Semester ID: " + semesterID + "\nStart date: " +
start.getDate() + "/" + start.getMonth() + "/"  + start.getYear() +
"\nEnd date: " + end.getDate() + "/" + end.getMonth() +"/"  +
end.getYear() ;
    }
}//~class Semester...

//end of code
```

The program output screenshot



```
Semester ID: 1
Start date: 30/5/2008
End date: 30/5/2008
```

End of the paper