

CSSE2003

Software Engineering Studio

Tutorial Week 11

Semester 2, 2009

School of Information Technology and Electrical Engineering
The University of Queensland

Goal of this tutorial:

1. Exam preparation. This tutorial is a sample examination question.

Introduction

To get the most from this tutorial it is recommended that you review this exercise before attending the class.

Question 1 [30 marks] (30 minutes)

The code for this question implements a linked list (with values in non-descending order) in an object-oriented way. Class *Node* defines the common behaviour of classes *StartNode*, *TailNode*, and *InternalNode*. Class *Value* encapsulates the integer values stored in a list.

Note that it is in your interest to present answers clearly.

- (a) **[15 marks]** Draw a UML class diagram for the code. Show classes, attributes and methods. Do not include the main class *LinkedList*. Do not show accessibilities of class members. Do not show method parameter and return types. Show class relationships (inheritance, association, non-structural). Show multiplicities at both ends of an association.
- (b) **[15 marks]** Draw a sequence diagram for the interaction generated by execution of *main()* in class *LinkedList*. Make sure that the diagram faithfully captures the creation of objects and sequencing of messages that is involved in the interaction.

```

public class LinkedList {
    public static void main(String[] args) {
        new StartNode().insert(new Value(1)).insert(new Value(0))
    }
}

class Value {
    private int value;
    Value(int v) {value = v;}
    boolean lessThan(Value v) {
        return (value < v.value);
    }
}

abstract class Node {
    abstract Node insert(Value v);
}

class StartNode extends Node {

    private Node firstNode;

    StartNode() {
        firstNode = new TailNode();
    }

    Node insert(Value v) {
        firstNode = firstNode.insert(v);
        return this;
    }
}

class InternalNode extends Node {

    private Value value;
    private Node nextNode;

    InternalNode(Value v, Node next) {
        value = v;
        nextNode = next;
    }

    Node insert(Value v) {
        if (value.lessThan(v)) {
            nextNode = nextNode.insert(v);
            return this;
        } else {
            return new InternalNode(v, this);
        }
    }
}

class TailNode extends Node {

    Node insert(Value v) {
        return new InternalNode(v, this);
    }
}

```

Question 2 [30 marks (30 minutes)]

A variant on the above question follows. (Evidently these two questions would not appear in the same examination.) The code for this question implements a linked list (with values in non-descending order) in an object-oriented way. Class *Node* defines the common behaviour of classes *StartNode*, *TailNode*, and *InternalNode*. Class *Value* encapsulates the integer values stored in a list.

Note that it is in your interest to present answers clearly.

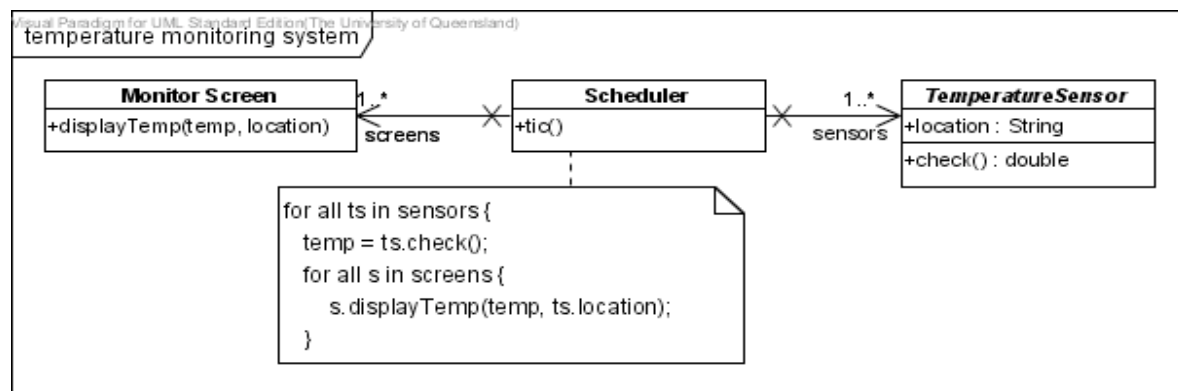
- (c) [15 marks] Draw a UML class diagram for the code. Show classes, attributes and methods. Do not include the main class *LinkedList*. Do not show accessibilities of class members. Do not show method parameter and return types. Show class relationships (inheritance, association, non-structural). Show multiplicities at both ends of an association.
- (d) [15 marks] Draw a sequence diagram for the interaction generated by execution of *main()* in class *LinkedList*. Make sure that the diagram faithfully captures the creation of objects and sequencing of messages that is involved in the interaction.

```
public class LinkedList {
    public static void main(String[] args) {
        new StartNode().insert(0).insert(1)
    }
}
class Value {
    private int value;
    Value(int v) {value = v;}
    int getValue() {return value;}
}
abstract class Node {
    abstract Node insert(int v);
}
class StartNode extends Node {
    private Node firstNode;
    StartNode() {
        firstNode = new TailNode();
    }
    Node insert(int v) {
        firstNode = firstNode.insert(v);
        return this;
    }
}
class InternalNode extends Node {
    private Value value;
    private Node nextNode;

    InternalNode(int v, Node next) {
        value = new Value(v);
        nextNode = next;
    }
    Node insert(int v) {
        if (value.getValue() < v) {
            nextNode = nextNode.insert(v);
            return this;
        } else {
            return new InternalNode(v, this);
        }
    }
}
class TailNode extends Node {
    Node insert(int v) {
        return new InternalNode(v, this);
    }
}
```

Question 1. [30 marks total] (From the csse2003 2007 final exam)

OO Design, UML and Design patterns. A temperature monitoring system monitors the temperature of a city using multiple temperature sensors installed around the city. It also displays the temperatures on multiple monitor screens installed around the city. The monitoring system has a scheduler that tells temperature sensors regularly when to check the temperature and passes the reported temperatures to the monitor screens to display (the `tic` method is responsible for this). An initial design diagram is presented below.



The current design couples the Scheduler to both temperature sensors and monitor screens (high-coupling). The Scheduler has low cohesion because it combines unrelated responsibilities. As new screens are added, they need to be added to the Scheduler.

- (a) **[15 marks]** Identify one or more design patterns that decouple the monitor screens from the Scheduler so that the Scheduler does not need to know about the monitoring screens and focuses solely on telling the sensors when to check the temperature. Your design must not couple the temperature sensors directly to the monitor screens so that changes to the monitor screens require changes to the sensors. Give a UML class diagram for your design and explain how it relates to the design pattern(s). Identify suitable attributes and operations (with parameters) and include these in your class diagram.
- (b) **[10 marks]** Give a UML sequence diagram for your design developed in part (a).
- (c) **[5 marks]** Justify how your design resolves the issues identified with the original design.