2nd Part: Problems (5 points out of 10)

Duration: 120 minutes
Maximum grade: 5 points
Date: 3rd of June, 2013

---

- Books, written notes, texts, or switched on electronic devices are not allowed in the examination room. Candidates contravening this may be required to withdraw from the examination room.
- Fill in your personal data at the beginning of this sheet before starting the exam.
- Use the empty boxed spaces after each question to write your answers.

**PROBLEM 1 (2.5 points)**

Let Message be a class that represents a message in a messaging system (the dots bellow substitute already coded methods, which you can use in your answer, but whose details are not shown here for brevity):

```java
public abstract class Message {
    private String text;
    private Person from;
    private Person to;
    private int priority;

    public Message(String text, Person from, Person to, int priority) {...}
    public String getText() {...}
    public Person getFrom() {...}
    public Person getTo() {...}
    public int getPriority() {...}
    public void dispatch() {...} // sends the message
    public void archive() {...} // stores the message
    public abstract String format(); // formats the message as a String
}
```

There are no more methods or attributes in this class and you are not allowed to modify them.
Continue reading on the next page.
Question 1 (0.75 points)

Write the code of the class Answer, with all its needed methods and attributes. This class inherits from Message and represents an answer to a message. Be mindful of:

- The arguments of its constructor must be: the text of the answer and the original message it is responding to. The attributes from and to will be the taken from the to and from of the original message. The priority of the answer must be the same as the priority of the original message.
- The format method must return the concatenation of: (1) the format’s return value of the original message, between square brackets; (2) a blank space character; (3) the text of the answer.

```java
public class Answer extends Message {
    private Message original;

    public Answer(String text, Message original) {
        super(text, original.getTo(), original.getFrom(), original.getPriority());
        this.original = original;
    }

    public String format() {
        return "[" + original.format() + "] " + getText();
    }
}
```
Question 2 (1 point)

Write the code for the class `Question`, with all its needed methods and attributes. This class inherits from `Message` and represents a message with a question. Be mindful of:

- It must have a constructor with the same arguments as the `Message`’s constructor.
- The `format` method will just return the text of the message.
- You are not supposed to store `Questions` without an answer. Therefore, you should override the `archive` method so it won’t do anything.
- It has an `Answer reply(String text)` method that instantiate an answer message for the current question, using its argument as the answer’s text. This method must also store the current question by calling the `archive` method inherited from the `Message` class (This is, don’t use the overridden method in the `Question` class, but the one on `Message`).

```java
public class Question extends Message {
    public Question(String text, Person from, Person to, int priority) {
        super(text, from, to, priority);
    }

    public void archive() {
        // questions cannot be archived without answering them
    }

    public Answer reply(String text) {
        Answer answer = new Answer(text, this);
        super.archive();
        return answer;
    }

    public String format() {
        return getText();
    }
}
```
Question 3 (0.75 points)

There will be a simple graphical user interface for filling in answers to questions. Given a question to user, it will record her answer and create an Answer object.

The GUI will show a text box and two buttons (labeled as “Accept” and “Cancel”). Pressing the accept button will create the Answer object using the text from the text box and send it.

Whatever the button pressed, the text box must be emptied.

The GUI will be coded as a single class, which will work as the main window and its own event listener from the buttons.

Fill in the underscored blanks below to answer this question.

```java
public class GraphicalInterface extends JFrame implements ActionListener {

    private Message original;
    private JTextArea textArea;

    public GraphicalInterface(Message original) {
        this.original = original;
        getContentPane().setLayout(new FlowLayout());
        JButton acceptButton = new JButton("Accept");
        JButton cancelButton = new JButton("Cancel");
        textArea = new JTextArea(10, 40);

        getContentPane().add(textArea);
        getContentPane().add(acceptButton);
        getContentPane().add(cancelButton);
        acceptButton.addActionListener(this);
        cancelButton.addActionListener(this);
        pack();
        setVisible(true);
    }

    public void actionPerformed(ActionEvent e) {
        String text = textArea.getText();
        textArea.setText(
"");
        if (e.getActionCommand().equals("Accept")) {
            Message answer = new Answer(text, original);
            answer.send();
        }
    }
}
```
PROBLEM 2 (2.5 points)

Let Queue be a queue implemented as a linked list (LinkedList), as the one shown below:

```
public class Node {
    public Object info;
    public Node next;
    public Node() {...}
    public Node(Object info, Node next) {...}
}

public class LinkedList {
    public Node first;
    public int size = 0;
    public LinkedList() {...}
}
```

Continue reading on the next page.
Question 1 (0.75 points)

Write the code for the following method of the LinkedList class:

- public void insertAt(Object data, int position)

It will insert a datum (data) in the position just after position, updating size.

**NOTES:** Positions in the list start at 1. If position is less or equal than 0 the insertion must be made at the beginning of the list. If position is greater or equal than size the insertion will be made at the end of the list.

```java
/*
 * NOTE: in the exam it was clarified that getters and setters could be used for the Node class.
 */

public void insertAt(Object data, int position) {
    if ((first == null) || position <= 0) {
        Node tmp = new Node(data, first);
        first = tmp;
    } else {
        Node aux = first;
        for (int i = 1; i < position && aux.getNext() != null; i++) {
            aux = aux.getNext();
        }
        Node tmp = new Node(data, aux.getNext());
        aux.setNext(tmp);
    }
    size++;
}
```
Question 2 (0.75 points)

Modify the LinkedList declaration so it implements the Queue interface defined here:

```java
public interface Queue {
    public void enqueue(Object o);
    public boolean isEmpty();
    public int size();
}
```

You must write the code for these 3 methods.

**NOTES:** Here we are ignoring the dequeue method to simplify the problem, you must do the same. You can reuse the insertAt method from the previous question to implement the enqueue method, whether you answer that question or not.

```java
public class LinkedList implements Queue {

    (...)

    public void enqueue(Object info) {
        insertAt(info, size);
    }

    public boolean isEmpty() {
        return (first == null);
    }

    public int size() {
        return size;
    }
}
```
Question 3 (1 point)

Write the code for the new method public Queue invert() of the LinkedList class, which will return a new queue instance, with all the elements in the linked list, in reverse order.

NOTES: You can use the following stack class as a help and suppose all its methods are already implemented:

```
public Queue invert() {
    Stack stack = new Stack();
    Queue queue = new LinkedList();
    Node aux = first;
    while (aux != null) {
        stack.push(aux.getInfo());
        aux = aux.getNext();
    }
    while (!stack.isEmpty()) {
        queue.enqueue(stack.pop());
    }
    return queue;
}
```