COMPSCI 230 Tutorial 12 Review

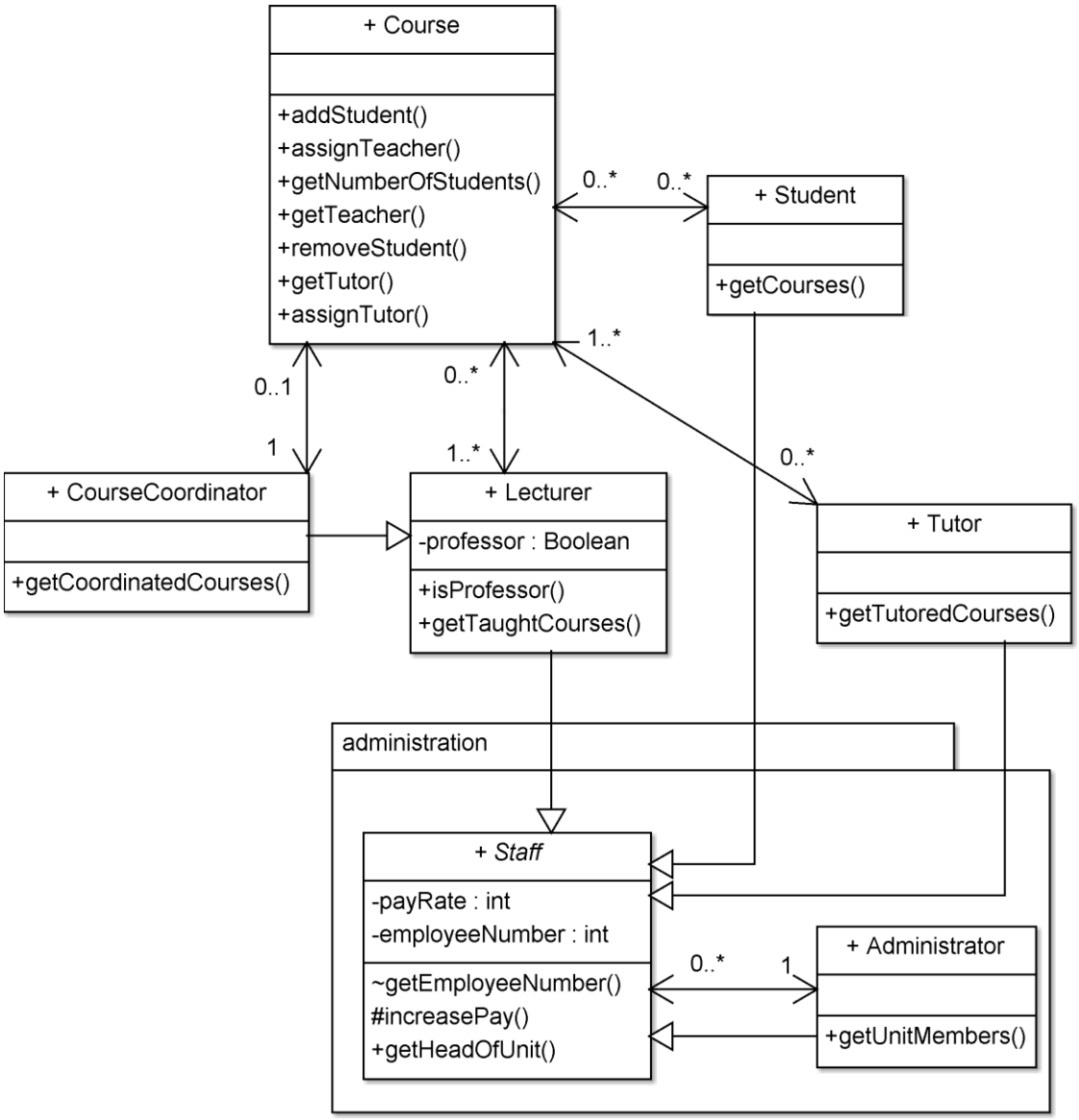
Theme A: the object-oriented programming paradigm

1) Remind ourselves of what the following keywords for OO in java mean:

public	
private	
protected	
static	
(No Modifier)	

2) Bob is a fisherman, he has his own boat! When he is fishing, he does three things: drive the boat, go fishing, and eat his lunch.

Draw a use-case diagram to represent Bob's fishing. Use «include» to provide a little more detail for the go-fishing part.



3) Refer to the above class diagram. List the instance variables of the Lecturer class.

4) Describe the relationships between Course, Lecturer and CourseCoordinator.

Hint: See Appendix for Class Diagram summary sheet.

Theme B: frameworks

5) Write an event listener so that the Tutor gets a $1.00 pay rise every time the button is clicked. You should "program by example": you'll find a suitable example in the appendix.

```
Tutor tut = new Tutor();
JButton b = new JButton("Moar Money!");
```

6) What does separation of model and view mean? Name one advantage of this.

Theme C: software quality

8) Write 3 test cases for the increasePay() method in the Administrator class. You should assume that the payRate field of Staff is private. You should also assume the Staff class has a getPayRate() method which returns an int (representing the pay in cents); this method is package-private. You should also assume that increasePay() takes two arguments: a reference variable of type Staff (to indicate who is getting a payrise), and a double (to indicate the magnitude of the payrise, in dollars). Assume that the pay rate starts a $0 / hr.

```
Tutor tut = new Tutor();
Administrator admin = new Administrator();
```

9) Minimum wage in New Zealand is $14.75 / hr. Write a test to ensure that the increasePay method can never cause the pay to go below 14.75.

```
Tutor tut = new Tutor();
Administrator admin = new Administrator();
admin.increasePay(tut, 14.75);
```

10) What is the key difference between black box testing and white box testing

Theme D: application-level concurrent programming

Questions 19, 20, and 21 from the s2 2013 final exam (on the next page). Note that question 22 would not be suitable for this year's examination unless additional information was provided on the invokeLater() and invokeAndWait() methods because these were not covered in this year's lectures or assignments.

19. Consider the following Java code fragment.

```
private int x = 0;
synchronized int incX() {
  return(x++);
}
```

Which of the following is **true**?

a. If two threads attempt to invoke `incX()` simultaneously, both will eventually succeed and the value of x may be increased by either 1 or 2.
b. If two threads attempt to invoke `incX()` simultaneously, both will eventually succeed and the value of x will be increased by 2.
c. If two threads attempt to invoke `incX()` simultaneously, both will eventually succeed and the value of x will be increased by 1.
d. If two threads attempt to invoke `incX()` simultaneously, a `SynchronizationException` will be thrown.

(1.5 marks)

20. Which of the following could be observed in a deadlocked Swing application with two workers?

a. The Event Dispatch Thread (EDT) is waiting for a GUI event, and both workers are waiting.
b. The Event Dispatch Thread (EDT) is waiting for a GUI event, one worker is running, and the other worker is waiting.
c. The Event Dispatch Thread (EDT) is waiting for a GUI event, and both workers are running.
d. Any of the above.

(1.5 marks)

21. If multiple variables are updated in the body of a `synchronized` method, these changes are

a. Not necessarily atomic, but always visible to all other threads
b. Always atomic, but not necessarily visible to all other threads
c. Always atomic, and always visible to all other threads
d. Not necessarily atomic, and not necessarily visible to all other threads

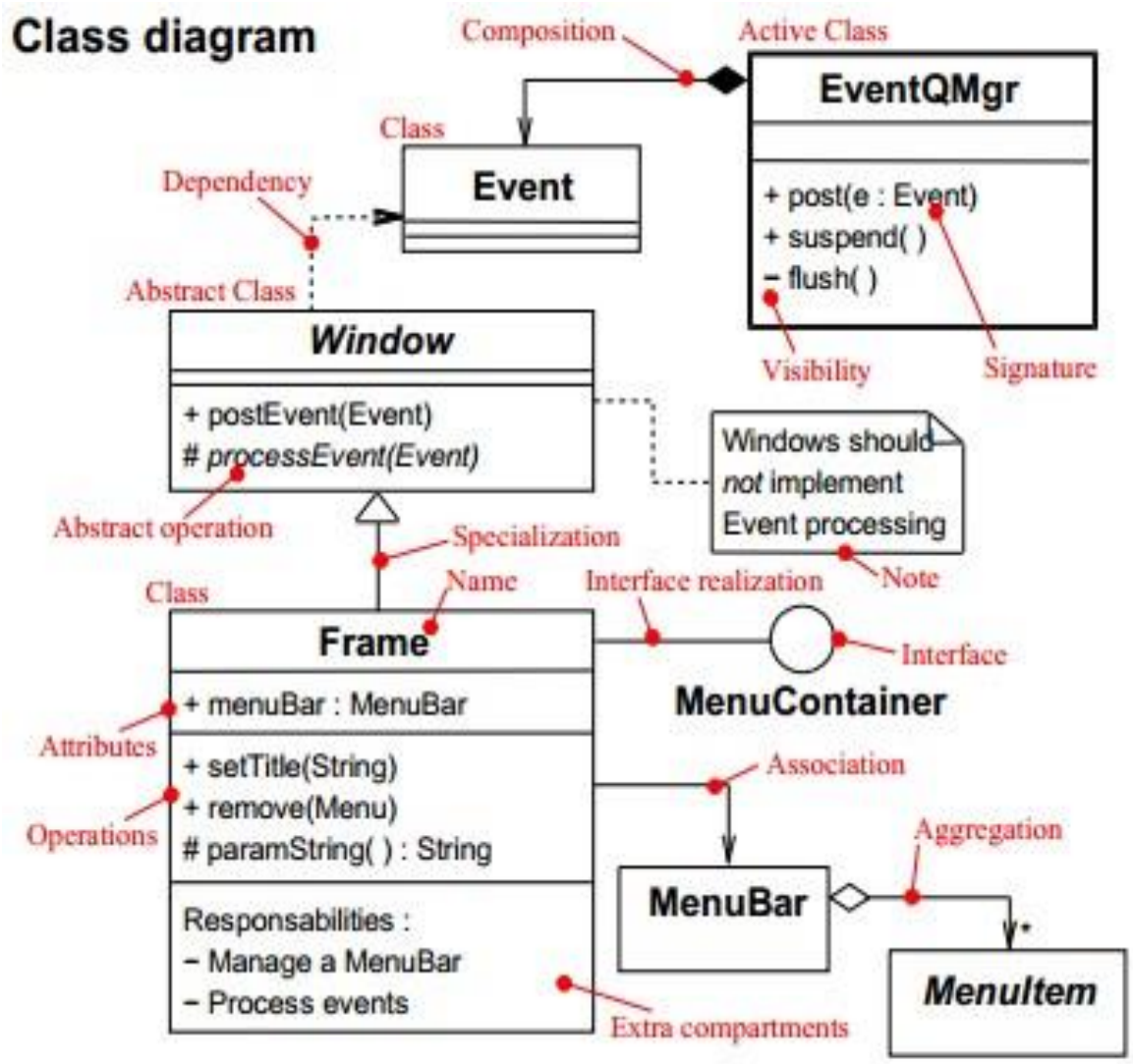
(1.5 marks)

22. In a Swing application, performance problems are likely to arise if

a. The Event Dispatch Thread (EDT) calls `InvokeLater()`
b. The Event Dispatch Thread (EDT) calls `InvokeAndWait()`
c. An initial thread calls `InvokeLater()`
d. An initial thread calls `InvokeAndWait()`

(1.5 marks)

Appendix 1



Appendix 2: Button Demo, from the Java Tutorials

```
package components;

import javax.swing.AbstractButton;
import javax.swing.JButton;
import javax.swing.JPanel;
import javax.swing.JFrame;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyEvent;

/*
 * ButtonDemo.java requires the following files:
 *   images/right.gif
 *   images/middle.gif
 *   images/left.gif
 */
@SuppressWarnings("serial")
public class ButtonDemo extends JPanel
                        implements ActionListener {
```

```
    protected JButton b1, b2, b3;

    public ButtonDemo() {

        b1 = new JButton("Disable middle button", null);
        b1.setVerticalTextPosition(AbstractButton.CENTER);
        b1.setHorizontalTextPosition(AbstractButton.LEADING); //aka LEFT,
for left-to-right locales
        b1.setMnemonic(KeyEvent.VK_D);
        b1.setActionCommand("disable");

        b2 = new JButton("Middle button", null);
        b2.setVerticalTextPosition(AbstractButton.BOTTOM);
        b2.setHorizontalTextPosition(AbstractButton.CENTER);
        b2.setMnemonic(KeyEvent.VK_M);

        b3 = new JButton("Enable middle button", null);
        //Use the default text position of CENTER, TRAILING (RIGHT).
        b3.setMnemonic(KeyEvent.VK_E);
        b3.setActionCommand("enable");
        b3.setEnabled(false);

        //Listen for actions on buttons 1 and 3.
        b1.addActionListener(this);
        b3.addActionListener(this);

        b1.setToolTipText("Click this button to disable the middle
button.");
        b2.setToolTipText("This middle button does nothing when you click
it.");
        b3.setToolTipText("Click this button to enable the middle
button.");

        //Add Components to this container, using the default FlowLayout.
        add(b1);
        add(b2);
        add(b3);
    }

    public void actionPerformed(ActionEvent e) {
        if ("disable".equals(e.getActionCommand())) {
            b2.setEnabled(false);
            b1.setEnabled(false);
            b3.setEnabled(true);
        } else {
            b2.setEnabled(true);
            b1.setEnabled(true);
            b3.setEnabled(false);
        }
    }


    /**
     * Create the GUI and show it.  For thread safety,
     * this method should be invoked from the
     * event-dispatching thread.
     */
    private static void createAndShowGUI() {

        //Create and set up the window.
```

```
        JFrame frame = new JFrame("ButtonDemo");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        //Create and set up the content pane.
        ButtonDemo newContentPane = new ButtonDemo();
        newContentPane.setOpaque(true); //content panes must be opaque
        frame.setContentPane(newContentPane);

        //Display the window.
        frame.pack();
        frame.setVisible(true);
    }

    public static void main(String[] args) {
        //Schedule a job for the event-dispatching thread:
        //creating and showing this application's GUI.
        javax.swing.SwingUtilities.invokeLater(new Runnable() {
            public void run() {
                createAndShowGUI();
            }
        });
    }
}
```