COE 312 – Data Structures

Welcome to Exam II
Wednesday January 08, 2014

Instructor: Dr. Wissam F. Fawaz

Name: _____

Student ID: _____

Instructions:

1. This exam is **Closed Book**. Please do not forget to write your name and ID on the first page.
2. You have exactly **90 minutes** to complete the **6** required problems.
3. Read each problem carefully. If something appears ambiguous, please write your assumptions.
4. Points allocated to each problem are shown in square brackets.
5. Put your answers in the space provided only. No other spaces will be graded or even looked at.

Good Luck!!

Problem 1: Multiple Choice Questions (**20 minutes)** [16 points]

1) Which of the following event descriptions best describes the *mouse entered* event?
   a. The mouse button is pressed down
   b. The mouse button is pressed down and released without moving the mouse in between
   c. **The mouse pointer is moved onto a component**
   d. None of the above
2) Which of the following best describes a *timer*?
   a. It starts when a GUI component is first created, and ends when component is destroyed
   b. **It generates action events at regular intervals**
   c. It determines the amount of time it takes to execute a method
   d. All of the above
3) Which of the following is a proper recursive definition of x raised to the y^{th} power?
   a. $x^y = x^{y-1} * x$ for all y
   b. $x^y = x^{y-1} * x$ for $y > 1$; $x^y = x$ for $y = 1$
   c. $x^y = x^{y-2} * x^2$ for $y > 2$; $x^y = x$ for $y = 1$ and $x^y = 1$ for $y = 0$
   d. **None of the above**
4) Which of the following statements makes the text of `JTextField` uneditable?
   a. **`textField.setEditable(false);`**
   b. `textField.setUneditable(true);`
   c. All of the above
   d. None of the above
5) Which method determines if a `JCheckBox` is selected?
   a. `getSelected`
   b. `selected`
   c. `isChosen`
   d. **None of the above**
6) The logical relationship between radio buttons is maintained by objects of which of the following classes?
   a. `MutualExclusionGroup`
   b. `RadioButtonGroup`
   c. `Group`
   d. **None of the above**
7) Which of the following statements about anonymous inner classes is **false**?
   a. They are declared without a name
   b. They typically appear inside a method declaration
   c. They can access their top-level class's methods
   d. **None of the above**
8) If the characters 'D', 'B', 'C', 'A' are placed in a queue (in that order), and then removed one at a time from the queue, in what order will they be removed?
   a. `ACBD`
   b. **`DBCA`**
   c. `DCAB`
   d. None of the above
9) Consider the following method declaration

```
void exam(int i) {
     if(i>1) {
          exam(i/2);
          exam(i/2);}
     System.out.print("*");}
```

How many asterisks are printed by the method call `exam(5)`?

a. `3`
b. `4`
c. **`7 (Correct answer)`**
d. None of the above

10) Which of the following `SwingWorker` methods executes a long computation in a worker thread?
    a. `execute`
    b. `done`
    c. **`doInBackground`**
    d. none of the above
11) Which of the following is used to perform all the tasks that manipulate GUI components, because they are not thread safe
    a. Event-handling thread
    b. **Event-dispatch thread**
    c. GUI-handling thread
    d. Component-dispatch thread
12) The number of calls to recursively calculate the Fibonacci value of 7 is
    a. 7
    b. 13
    c. **41**
    d. 29
13) The first argument to the method `showMessageDialog` of `JOptionPane` specifies
    a. The title
    b. The message
    c. The icon
    d. **The parent window**
14) Which of the following statements about a `JLabel` is **false**?
    a. Applications can change a label's content after creating it
    b. A `JLabel` can display an image
    c. **A `JLabel` can display text of any length**
    d. None of the above
15) A `JEditorPane` can be used to render
    a. Multi-line plain text
    b. HTML-formatted data
    c. An image
    d. **All of the above**
16) Which `JFileChooser` method returns the file the user selected?
    a. `getFile`
    b. `getOpenDialog`
    c. `showOpenDialog`
    d. **None of the above**

Problem 2: True or false (**10 minutes)** [14 points]

1. Consider the case where a method called `f1` calls a different method called `f2`, which then calls the original method `f1`. In this case, `f1` is a recursive method.

Answer: **True** False

2. Every line in a `catch` block is guaranteed to be executed when an exception is thrown.

Answer: True **False**

3. If a class implements an interface, then it cannot extend another class.

Answer: True **False**

4. The `peek` operation of the `java.util.Stack` class returns a reference to the top element.

Answer: **True** False

5. By default, `JRadioButton`s operate as a group, providing a set of mutually exclusive options.

Answer: True **False**

6. Consider following method.

```
public void foo(int x) {
     if(x>0) foo(x-2);
     System.out.println(x + " ");}
```

The method call `foo(4)` prints: `2 4`

Answer: True **False**

7. `JOptionPane`'s `showMessageDialog` method creates a modal input dialog box.

Answer: True **False**

8. A Polygon object in Java can be defined by arrays of x and y coordinates.

Answer: **True** False

9. Similarly to all Java programs, swing Java applications must have a `main` method.

Answer: **True** False

10. The following method correctly multiplies two integer values so long as both are non-negative:

```
public int multiply(int a, int b) {
      return (b > 0) ? a + multiply(a, b-1) : 0;
}
```

Answer: **True** False

11. Some problems such as the Towers of Hanoi puzzle are easier to solve iteratively than recursively.

Answer: True **False**

12. It is possible for a recursive method having a base case to yield infinite recursion.

Answer: **True** False

13. The pop and front methods can be used to remove elements from stacks and queues respectively.

Answer: True **False**

14. An `ArrayList` can be used to implement a stack data structure.

Answer: **True** False

Problem 3: Recursion (**10 minutes)** [10 points]

1) A binomial coefficient $\binom{n}{k}$ is the coefficient of the x^k term in the polynomial expansion of the binomial power $(1+x)^n$. The binomial coefficient $\binom{n}{k}$ can be defined recursively as follows:

$$\binom{n}{k} = \begin{cases} \binom{n-1}{k-1} + \binom{n-1}{k}, if\ n,k > 0 \\ 1, \quad if\ n > 0\ and\ k = 0 \\ 0, \quad if\ n = 0\ and\ k \geq 0 \end{cases}$$

Write a method `computeBinomialCoeff(int n, int k)` that computes $\binom{n}{k}$ with a recursive approach.

```
public int computeBinomialCoeff(int n, int k) {

     if(n == 0)
          return 0;
     if(k == 0)
          return 1;
     return
computeBinomialCoeff(n-1,k-1)+computeBinomialCoeff(n-1, k);
     }
```

2) What does the following method do?

```
public int mysteryMethod(int a, int b) {
     if(b == 1)
          return a;
     else
          return a * mysteryMethod(a, b-1);
}
```

Answer: ***It returns a^b***

Problem 4: Stacks and queues (**10 minutes)** [10 points]

1) Write a method to move all the nodes from a stack `s` to a queue `q`. You can use any of the methods defined in the Stack and Queue ADT.

```
public void moveStackToQueue(Stack s, Queue q) {
     while(!q.isEmpty())
          s.push(q.dequeue());

}
```

2) Write a method that returns the number of elements stored in a stack. In this question, we assume that there is no `size` function defined in the Stack ADT. In other words, you are allowed to use all the methods of the Stack ADT but the `size` method in your solution.

```
public int stackSize(Stack s) {
     int size = 0 ;
     while( !q.isEmpty()) {
          q.dequeue() ;
          size++ ;
     }
     return size ;









}
```

Problem 5: Timer class (**20 minutes)** [20 points]

Design and implement an application that works as a stopwatch. Include a label that shows the time (in seconds) as it increments. Include buttons that allow the user to start and stop the time and reset the label to zero. *Hint*: use the `Timer` class to control the timing of the stopwatch.

```
import javax.swing.JFrame;
public class StopWatchFrame {
    public static void main(String[] args){
        JFrame frame = new JFrame ("Stop Watch");
        frame.setDefaultCloseOperation (JFrame.EXIT_ON_CLOSE);
        frame.getContentPane().add (new StopWatchPanel());
        frame.pack();
        frame.setVisible (true);
    }
} // end of StopWatchFrame class

import javax.swing.*;
import java.awt.event.*;
import java.awt.*;
public class StopWatchPanel extends JPanel {
     // Declare instance variables here
     private JLabel timeDisplay;
     private Timer timer;
     private int time;

     // Complete implementation of constructor
      public StopWatchPanel() {
           time = 0;
           timeDisplay = new JLabel("0");
           add(timeDisplay,BorderLayout.NORTH);
           JPanel buttonPanel = new JPanel();

           JButton button;
           ButtonActionListener buttonPressed = new
           ButtonActionListener();
           button = new JButton("Start");
           button.addActionListener(buttonPressed);
           buttonPanel.add(button);
           button = new JButton("Stop");
           button.addActionListener(buttonPressed);
           buttonPanel.add(button);
           button = new JButton("Reset");
           button.addActionListener(buttonPressed);
           buttonPanel.add(button);
           add(buttonPanel, BorderLayout.CENTER);

           timer = new Timer(1000, new TimerActionListener());

     } // end of StopWatchPanel constructor method
```

```
      // Start the stop watch.
      private void startTime() {
            timer.start();
      }
      // Stop the stop watch.
      private void stopTime() {
            timer.stop();
      }
      // Reset the stop watch.
      private void resetTime() {
            timer.stop();
            timeDisplay.setText("0");
            time = 0;
      }
      // Increment the display by 1 second.
      private void incrementDisplay() {
            time++;
            timeDisplay.setText(new Integer(time).toString());
      }
      // Represents the action listener for the timer
      private class TimerActionListener
            implements ActionListener {

            public void actionPerformed(ActionEvent actionEvent){
                  incrementDisplay();
            }
      } // end of private class TimerActionListener
      // Represents the action listener for the buttons
      private class ButtonActionListener
            implements ActionListener {
            // determine which button was pressed and execute
            // appropriate command
            public void actionPerformed(ActionEvent evt) {
                  JButton source = (JButton)evt.getSource();
                  String text = source.getText();
                  if (text.equals("Start"))
                        startTime();
                  else
                    if (text.equals("Stop"))
                        stopTime();
                  else
                      if (text.equals("Reset"))
                         resetTime();




            }
       } // end of private class ButtonActionListener
} // end of class StopWatchPanel
```

Problem 6: Stack (**20 minutes)** [30 points]

Write a class called `ArrayBasedStack` that realizes a stack data structure by means of an array. Equip this class with a constructor, declare the instance variables, and implement all the methods of the Stack ADT. You are required in the process to define the necessary interface and create an exception class representing the errors that might arise when processing the elements of this stack data structure.

```
 public interface Stack {
     public int size();
     public boolean isEmpty();
     public void push(Object elemnt) throws StackException;
     public Object pop() throws StackException;
     public Object top() throws StackException;
}

public class StackException extends Exception {
     public StackException(String msg) {
          super(msg);
     }
}

import java.util.Arrays;
public class ArrayBasedStack implements Stack{
     private Object[] arr;
     private int top;
     public ArrayBasedStack(int capacity) {
          arr = new Object[capacity];
          top = -1;
          Arrays.fill(arr, null);
     }
     public int size() {
          return top+1;
     }
     public boolean isEmpty() {
          return top==-1;
     }
     public void push(Object element) throws
     StackException {
          if(size() == arr.length)
              throw new StackException("Stack is full");
          ++top;
          arr[top] = element;
     }
```

```
    public Object pop() throws StackException {
        if(isEmpty())
        throw new StackException("Stack is Empty");
        Object toReturn = arr[top];
        arr[top] = null;
        top--;
        return toReturn;
    }
    public Object top() throws StackException {
        if(isEmpty())
        throw new StackException("Stack is Empty");
        return arr[top];
    }
}
```

Appendix: Classes and Methods

1. **Methods related to Stack and Queue ADTs:**

Stack
int size() boolean isEmpty() void push(Object) throws StackException Object pop() throws StackException Object top() throws StackException
Queue
int size() boolean isEmpty() void enqueue(Object) throws QueueException Object dequeue() throws QueueException Object front() throws QueueException

2. **Classes related to GUI-based applications:**

JButton	**JLabel**
JButton(String) void setEnabled(boolean) void setMnemonic(char) void addActionListener(ActionListener)	JLabel() JLabel(String) String getText() void setText(String)
Timer	**ActionEvent**
Timer(int delay, ActionListener) void start() void stop() void restart() boolean isRunning()	String getActionCommand() Object getSource() int getModifiers() long getWhen()