**Homework 3**

**Name:** Sairam Soundararajan

1. (10 pts) What events do the following components generate:

* JButton – The JButton will generate an ActionEvent event type in which the event is delivered to any listener that has been registered with the button as an ActionListener.
* JTextField – An ActionEvent is generated. To respond to such an event, one can register an ActionListener with a text field by using the text field’s addActionListener() method.
* JComboBox – Like the past two components, JComboBox generates an ActionEvent and the user can respond using an ActionListener with the JComboBox.

2. (10 pts) What methods does JTable implement which are required by the interfaces implemented by the JTable class beyond those interfaces implemented by the various parent classes of JTable?

imageUpdate – required by ImageObserver

getFont, postEvent, remove – required by MenuContainer

getAccessibleContext() – required by Accessible

editingCanceled – required by CellEditorListener

valueChanged – required by ListSelectionListener

sorterChanged – required by RowSorterListener

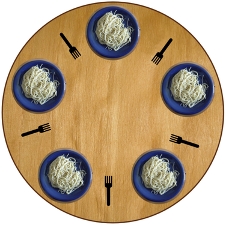
columnAdded, columnRemoved, columnMarginChanged, columnMoved, columnSelectionChanged – required by TableColumnModelListener

tableChanged – required by TableModelListener

getPreferredScrollableViewportSize, getScrollableBlockIncrement, getScrollableTracksViewportHeight, getScrollableTracksViewportWidth, getScrollableUnitIncrement – required by Scrollable  
  
3. (10 pts) Address how the differences among these various layout managers, focusing on their behavior as their container is resized:

1. FlowLayout – Used by JPanel; lays out components in a single row, beginning a new row if its container is not adequately wide enough.
2. GridLayout – creates several components of equal size and displays them in the desired number of rows and columns.
3. BorderLayout – Every content pane is initialized to use a BorderLayout, which places components in up to five areas: top, bottom, left, right, and center. Extra space is placed in the center area.
4. GridBagLayout – sophisticated, yet flexible layout manager; aligns components by placing them within a grid of cells, allowing components to traverse more than a single cell. Rows can have different heights, while grid columns can have different widths.
5. Absolute Positioning (explain the basic steps required for this manager as well)
   1. Set the container’s layout manager to null by calling setLayout(null)
   2. Call the Component class’s setbounds method for each of the container’s children
   3. Call the Component class’s repaint method

4. (20 pts) (Ex 1.8.2) The dining philosophers problem was invented by E. W. Dijkstra, a concurrency pioneer, to clarify the notions of deadlock and starvation freedom. Imagine five philosophers who spend their lives just thinking and feasting. They sit around a circular table with five chairs. The table has a big plate of rice. However, there are only five chopsticks (in the original formulation forks) available, as shown in Fig. 1.5. Each philosopher thinks. When he gets hungry, he sits down and picks up the two chopsticks that are closest to him. If a philosopher can pick up both chopsticks, he can eat for a while. After a philosopher finishes eating, he puts down the chopsticks and again starts to think.



a. What is wrong with everybody doing the following - other than that the philosophers never get up from the table?

1. think for a while
2. get left chopstick
3. get right chopstick
4. eat for a while
5. return left chopstick
6. return right chopstick
7. return to 1

The problem with everyone doing the following is that they can reach a deadlock. For example, if at the start, every philosopher stops thinking at the same time and picks up their left chopstick, then all 5 philosophers will have a left chopstick but there will be no right chopstick. Thus, they will be infinitely stuck in the 2nd state.

b. How can the above be fixed to avoid deadlocks?

One solution to prevent deadlocks is the following:

1. Think for a while
2. Pick up left and right chopstick. If none are available, return to 1
3. Eat for a while
4. Return left and right chopstick
5. Return 1

c. Is your solution starvation free? Literally!

The solution is not starvation free. Timing is a possible issue. It is possible that each time a philosopher is ready to eat, one of the adjacent philosophers has already begun eating.

5. (20 pts) What methods must a class implementing the java.util.concurrent.locks.Lock interface implement? Describe some of the expected characteristics of each of the methods of this interface?

@Override  
**public void** lock() {  
   
}

* Simply acquires the lock. If the lock is unavailable, the current thread will be disabled for thread scheduling purposed and remains inactive until the lock has been acquired  
    
  @Override  
  **public void** lockInterruptibly() **throws** InterruptedException {  
    
  }
* Acquires the lock unless the current thread is interfered.   
    
  @Override  
  **public boolean** tryLock() {  
   **return false**;  
  }
* Acquires the lock on the condition that it is free at time of invocation. Also acquires the lock if it is available and returns immediately with the value true. The method will promptly return a false value if the lock is unavailable.  
    
  @Override  
  **public boolean** tryLock(**long** time, TimeUnit unit) **throws** InterruptedException {  
   **return false**;  
  }
* Acquires the lock if it is free within the allocated wait time and the current hasn’t been interrupted.

@Override  
**public void** unlock() {  
  
}

* Simply releases the lock

@Override  
**public** Condition newCondition() {  
 **return null**;  
}

* Returns a new condition instance that is attached to this Lock instance.

6. (10 pts) Explain what the JVM does when it encounters a synchronized directive. Hint: consider carefully what is synchronized.

When the JVM resolved a symbolic reference to a method, it determines whether or not the method is synchronized. If so, the JVM will them gain a lock before invoking the method.

7. (10 pts) What happens when the JVM encounters a wait () call?

The thread will subsequentially wait until it can regain ownership of the monitor and proceed with execution

8. (10 pts) Describe the environment in which a wait () call is legal?

A wait() call must always be placed in a synchronized block. It must also accompany a notify() in order for the intercommunication between threads to work adequately.

**Grading Rubric:**

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Meets** | **Does not meet** |
| Problem 1 | **10 points** Lists the events associated with each provided component. | **0 points** Does not list the events associated with each provided component. |
| Problem 2 | **10 points** Lists the methods JTable implements.  Lists the methods which are required by the interfaces implemented by the JTable class beyond those interfaces implemented by the various parent classes of JTable. | **0 points** Does not list the methods JTable implements.  Does not list the methods which are required by the interfaces implemented by the JTable class beyond those interfaces implemented by the various parent classes of JTable. |
| Problem 3 | **10 points** Addresses the differences among the various layout managers.  Focuses on their behavior as their container is resized. | **0 points** Does not address the differences among the various layout managers.  Does not focus on their behavior as their container is resized. |
| Problem 4 | **20 points** Explains what is wrong with everybody doing the actions provided.  Explains how the actions be fixed to avoid deadlocks.  Explains if the solution provided is starvation free. | **0 points** Does not explain what is wrong with everybody doing the actions provided.  Does not explain how the actions be fixed to avoid deadlocks.  Does not explain if the solution provided is starvation free. |
| Problem 5 | **20 points** Explains what methods a class implementing the java.util.concurrent.locks.Lock interface must implement.  Describes some of the expected characteristics of each of the methods of this interface. | **0 points** Does not explain what methods a class implementing the java.util.concurrent.locks.Lock interface must implement.  Does not describe some of the expected characteristics of each of the methods of this interface. |
| Problem 6 | **10 points** Explains what the JVM does when it encounters a synchronized directive. | **0 points** Does not explain what the JVM does when it encounters a synchronized directive. |
| Problem 7 | **10 points** Explains what happens when the JVM encounters a wait () call. | **0 points** Does not explain what happens when the JVM encounters a wait () call. |
| Problem 8 | **10 points** Describes the environment in which a wait () call is legal. | **0 points** Does not describe the environment in which a wait () call is legal. |