Homework 3

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

- JButton
- JTextField
- JComboBox

According to Oracle, the listed components generate the following events. Jbuttons generate ActionEvents, ChangeEvents, and ItemEvents (Oracle, 2019b). JtextFields generate ActionEvents, ChangeEvents, DocumentEvents, and UndoableEditEvents, with the last two being indirectly generated via the JTextField's getDocument() method. JcomboBoxes generate ActionEvents and ItemEvents.

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?

According to Oracle, JTable directly implements TableModelListener, Scrollable, TableColumnModelListener, ListSelectionListener, CellEditorListener, Accessible, and RowSorterListener (Oracle, 2020a). The following are the methods required by these interfaces.

- TableModelListener
  - public void tableChanged(TableModelEvent e)
- Scrollable
  - public Dimension getPreferredScrollableViewportSize()
  - public int getScrollableUnitIncrement(
    Rectangle visibleRect, int orientation, int direction)
  - public int getScrollableBlockIncrement(
    Rectangle visibleRect, int orientation, int direction)
  - public boolean getScrollableTracksViewportWidth()
  - public boolean getScrollableTracksViewportHeight()
- TableColumnModelListener
  - public void columnAdded(TableColumnModelEvent e)
  - public void columnRemoved(TableColumnModelEvent e)
  - public void columnMoved(TableColumnModelEvent e)
  - public void columnMarginChanged(TableColumnModelEvent e)
  - public void columnSelectionChanged(TableColumnModelEvent e)
- ListSelectionListener
  - public void valueChanged(ListSelectionEvent e)
- CellEditorListener
  - public void editingStopped(ChangeEvent e)
  - public void editingCanceled(ChangeEvent e)
- Accessible
  - public AccessibleContext GetAccessibleContext()
- RowSorterListener
  - public void sorterChanged(RowSorterEvent e)

3. (10 pts) Address how the differences among these various layout managers, focusing on their behavior as their container is resized:

a. FlowLayout
b. GridLayout
c. BorderLayout
d. GridBagLayout
e. Absolute Positioning (explain the basic steps required for this manager as well)

In my answer, the default configurations for each layout manager is assumed.

a. FlowLayout places components aligned to the top and centered horizontally in the container. If the container is to narrow, it will start a new row. While resizing, FlowLayout will attempt to put as many components in one row as possible (Oracle, 2019a).

b. GridLayout arranges components in the specified number of rows and columns, starting at the top-left going left to right. Components in a GridLayout all have the same dimensions: all compenents have the exact same height and width. While resizing, the components retain this equivalency of dimensions. For example, when shrinking horizontally, all components widths change the same amount (Oracle, 2019a).

c. BorderLayout separates the area of the container into five sections: PAGE_START (top), PAGE_END (bottom), LINE_START (left), LINE_END (right), and CENTER. All areas use only as much space as needed for their components, with the CENTER getting as much of the remaining space as possible. When enlarging, if the outer sections need more space, they will take it, but no more than needed, and the CENTER will get most of the space. When shrinking, LINE_START and LINE_END will cover up the CENTER as horizontal space runs out, and PAGE_START and PAGE_END will cover up everything else as vertical space runs out (Oracle, 2019a).

d. GridBagLayout, like GridLayout, arranges components in a grid. The components in this layout manager can use up different numbers of rows and or columns, of which they may or may not use up the whole space. In GridBagLayout, horizontal and vertical weights, from 0.0 to 1.0, can be applied to the separate components. This dictates how resizing affects the components: larger numbers equals more space. In the event of conflict or equal number, the right column and bottom row will tend to get the space (Oracle, 2019a).

e. In Absolute Positioning, there technically is no layout manager. As such, the resizing of a container has no effect on the size or position of the components. The easiest way to set up absolute positioning is to create an Insets object and assigning it the Insets object of the container. For example, Insets insets = panel.getInsets(). These insets would set the outside boundaries from which components can be placed. Components position is set using the setBounds(int x, int y, int width, int height) method of the component. Using the insets to set a component at 5 from the left and top with a width of 20 and height of 10, one would use, for example, button.setBounds(5 + insets.left, 5 + insets.top, 20, 10). Components can be set based off of insets.right and insets.bottom, but these should be in the form of subtraction, like insets.right - 5.

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

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

c. Is your solution starvation free? Literally!

a. Assuming the philosopher’s take the same amount of time to complete step 1, all five will pick up the left chopstick successfully. They will then attempt to pick up the right chopstick, but it no longer exists. As such, they must wait for the philosopher on the right to put down the left chopstick. The philosopher on the right is busy waiting for their right chopstick, so there is a deadlock.

b. This can easily be fixed by having even philosophers switch step 2 and step 3 and pick up the right chopstick first, as well as having philosophers only continue to pick up the second chopstick if and only if successful in picking up the first. In other words, all philosophers should wait until able to successfully complete a step before continuing to the next.

c. This is starvation free, because even in the case that all philosophers attempt to eat at the same time, not all philosophers will be successful in picking up the first chopstick. Since an even philosopher will attempt to pick up the same chopstick as the philosopher to the right, only one can be successful. Assuming the odd philosopher got the chopstick, that philosopher’s right chopstick would still be available. Once again, this chopstick would go to the philosopher that gets it first. Whether it’s the even or odd philosopher, is unimportant, because at least one philosopher is eating. It also doesn’t matter if a philosopher retains hold on its first chopstick, because it will gain access to the second as soon the the neighboring philosopher finishes eating. This may not be the most efficient, but it does prevent deadlock.

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?

A class implementing the java.util.concurrent.locks.Lock interface must implement the following (Oracle, 2020b).

- public void lock()
  - If the lock is not available, the thread should wait until it is available.
- public void lockInterruptibly()
  - If the lock is not available, the thread should wait until it is available or the thread is interrupted and interruption of lock acquisition is supported.
- public Condition newCondition()
  - "Returns a new Condition instance that is bound to this Lock instance."
- public boolean tryLock()
  - If the lock is available, it returns true. Otherwise, it returns false.
- public boolean tryLock(long time, TimeUnit unit)
  - If the lock is available, it returns true. Otherwise, it waits the specified amount of time and returns false if the time expires. This should also return false if the thread was interrupted during the time and true if the lock is acquired during the time.
- public void unlock()
  - This should release the lock for other threads.

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

According to Bill Venners (1997), when the JVM encounters a synchronized block, it will use two opcodes, monitorenter and monitorexit, to acquire and release the lock with the object reference specified for that block. If, instead, the JVM encounters a synchronized method, it will not use the opcodes. In this case, the JVM will either acquire the lock for the target of the method – the instance – or the class depending on if it is an instance or static method, respectively. Regardless, the JVM will acquire the lock before invoking the block or method, and release the lock after.

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

When the JVM encounters a wait() call, the thread with a lock on the objectref will release its lock on that objectref until another thread invokes the notify() method on that objectref.

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

A wait() call is only legal within a synchronized directive and can only be invoked on that directive's objectref, with the directive being either a block or method.

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	**0 points** Does not explain 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.	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.

References

Oracle. (2019a). *A Visual Guide to Layout Managers*. The Java™ Tutorials. https://docs.oracle.com/javase/tutorial/uiswing/layout/visual.html.

Oracle. (2019b). *Listeners Supported by Swing Components*. The Java™ Tutorials. https://docs.oracle.com/javase/tutorial/uiswing/events/eventsandcomponents.html.

Oracle. (2020a, July 09). *JTable (Java Platform SE 8)*. Java™ Platform, Standard Edition 8 API Specification. https://docs.oracle.com/javase/8/docs/api/java/awt/event/MouseAdapter.html

Oracle. (2020b, July 09). *Lock (Java Platform SE 8)*. Java™ Platform, Standard Edition 8 API Specification. https://docs.oracle.com/javase/8/docs/api/java/util/concurrent/locks/Lock.html

Venners, B. (1997, July 1). How the Java virtual machine performs thread synchronization. *Under the Hood, JavaWorld*. https://www.infoworld.com/article/2076971/how-the-java-virtual-machine-performs-thread-synchronization.html