C212 Lab 8

Last Updated 10/20/2021

Objectives

- Mouse input
- Keyboard input
- Interfaces, Event Handling, Buttons
- Drawing

Lab Instructions

Please complete the following exercises. For each, review the requirements given below and include all stated specifications. Please compress your project, which contains all of your files, into a .zip and submit it through Canvas. The grading scheme will also be provided in Canvas, which you may review as you work.

Required Files to Submit (by task)

- **Mouse Listener**
  - MouseListenerDemo.java
  - MouseListenerImplementation.java
  - MouseListenerAnswers.txt
  - Any additional files you write to complete this task
- **Key Listener**
  - KeyListenerDemo.java
  - KeyListenerImplementation.java
  - KeyListenerAnswers.txt
  - Any additional files you write to complete this task
- **Interfaces, Buttons, Event Handlers**
  - BlackjackPlayer.java
  - Any additional files you write to complete this task
- **Drawing**
  - FlagDisplay.java
  - Flag.java
  - Any additional files you write to complete this task

Suggestions

- Comment the start of each class with what its purpose is.
- Comment each function with the appropriate JavaDoc style of comments. Look at the Lab 03 solutions for examples.

- Separate each of the 4 tasks into their own package to help keep track of your progress across each.

1. Mouse Listener

Task

- Create a class called ***MouseListenerDemo*** that will hold your *main* method. This method will be used to run your program. The main method should do the following:
    - Create a *JFrame*.
    - Create a *JPanel* and add the panel to the *JFrame*.
    - Register your *MouseListenerImplementation* to the *JPanel*.
    - Display the frame to the user. Make sure the details provided below actually occur when the user interacts with the *JPanel*.
    - End program when user closes the *JFrame*.
- Create a class called ***MouseListenerImplementation*** that implements the *MouseListener* interface.
    - Everytime the user *presses* the mouse, it should increment a counter by 1 and print that to the console.
    - Everytime the user *releases* the mouse, it should print the (x, y) coordinates to the console in the form: "(*x*, *y*) - Mouse was released".
    - Everytime the user *clicks* the mouse, print out a message of your choice. One example could be "I love java! I know MouseListeners and Interfaces!"
- Create a text file called **MouseListenerAnswers** that answers the following questions:
    - What is the difference between *pressing* and *clicking* a mouse in Java?
    - How does the use of interfaces help with interchangeability of classes? For example, what if we wanted to have a different implementation of *MouseListener* in the future?

Helpful Sources/Hints

- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/java/awt/event/MouseEvent.html
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/java/awt/event/MouseListener.html
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/javax/swing/JFrame.html
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/javax/swing/JPanel.html
- https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html

2. Key Listener

Tasks

- Create a class called ***KeyListenerDemo*** that will hold your *main* method. This method will be used to run your program. The main method should do the following:
    - Create a *JFrame*.
    - Create a *JPanel* and add the panel to the *JFrame*.
    - Register your *KeyListenerImplementation* to the *JPanel*.
    - Display the frame to the user. Make sure the details provided below actually occur when the user interacts with the *JPanel*.
    - End program when user closes the *JFrame*.
- Create a class called ***KeyListenerImplementation*** that implements the *KeyListener* interface.
    - Everytime the user *presses* a key, it should increment a counter by 1 and print that to the console in the form: “You have pressed the keyboard *n* times”.
    - Everytime the user *releases* a key, it should print the key pressed and ASCII integer value to the console in the form: “letter, ascii integer value”.
    - Everytime the user *clicks* a key, do the following depending on key:
        - If the user clicks ‘e’, exit the application.
        - If the user clicks ‘h’, print “Hello World” to the console.
        - All other input should be echoed back to the console.
- Create a text file called **KeyListenerAnswers** that answers the following questions:
    - What is the difference between *pressing* and *clicking* a key in Java?
    - How could an implementation of the *KeyListener* interface be used to handle multiple characters entered in a row rather than just one character at a time?

Helpful Sources/Hints

- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/java/awt/event/KeyListener.html
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/java/awt/event/KeyEvent.html
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/javax/swing/JFrame.html
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/javax/swing/JPanel.html
- https://www.oracle.com/technical-resources/articles/java/javadoc-tool.html
- MouseListenerDemo and KeyListenerDemo classes are mostly the same, just with their respective listener’s registered.

3. Interfaces, Event Handling, Buttons

Task

- Create a class ***BlackjackPlayer***...
    - that has static fields:
        - int[] handTotals
        - boolean bust
        - JLabel totalsLabel
        - JButton hit
        - JButton stay
    - that will hold your *main* method. This method will be used to run your program. The main method should do the following:
        - Randomly generate a starting hand for the player (see below)
        - Create a *JFrame*
        - Create an overarching container *JPanel* mainPanel that you eventually add to the *JFrame*
        - Create two *JPanels*, statusPanel and buttonsPanel, which you add to mainPanel
        - Create a *JLabel* dealerLabel and instantiate it with some value (see below) and add it to statusPanel
        - Instantiate totalsLabel with the correct value(s) (see below) and add it to statusPanel
        - Instantiate the two *JButtons*, hit and stay, and add them to buttonsPanel
        - Add the unique *ActionListeners* to the buttons, HitButtonListener and StayButtonListener, described later
        - Display the frame to the user. Make sure the details provided actually occur when the user interacts with the *JPanel*.
        - End program when user closes the *JFrame*.
    - that has an inner static class HitButtonListener, which implements the *ActionListener* interface.
        - When the action is performed, deal the player a new card, checking if the player has “busted” -- gone over 21.
        - If the player busts, disable both buttons and, instead of displaying the player’s hand value(s), update the text of totalsLabel to “Bust!” Otherwise, simply update the text of totalsLabel with the new value(s).
    - that has an inner static class StayButtonListener, which implements the *ActionListener* interface
        - When the action is performed, disable both buttons.
- Program simple rules for dealing cards to the user in the style of the game Blackjack, also known as 21.
    - Your Blackjack functionality must encompass the following:
        - “Hit” means the user wants to be dealt another card. “Stay” means the user chooses not to receive any more cards.

        - In Blackjack, the user is dealt two cards to begin with. Suit is irrelevant, we care only about the cards' values. Therefore, you only need to track their hand value(s) as integers. You can randomly deal the user cards however you'd like, but do NOT simply use a uniform random range of [1, 10], since 10, J, Q, and K are distinct cards, thus the odds of receiving a 10 are not the same as receiving any other number.
        - When the user hits, if both of their totals exceed 21, they "bust."
        - The user can continue to hit so long as they have not busted. They can choose to stay at any time so long as they have not busted.
        - Face cards (jack, queen, king) are worth 10. Numeric cards are worth their listed value.
        - Aces can be worth 1 or 11 -- whichever is more beneficial to the player at any given moment. Thus, you must keep track of TWO hand totals in handTotals[] simultaneously. The first value in the array treats aces as 1, while the second treats them as 11.
        - When displaying the user's hand value(s), only list multiple values if the two values are different (i.e. they have an ace), and the second value is not greater than 21. Otherwise, only display the first value. You can separate them with some character and put them in the same String.
        - The dealer starts with two cards, one face up and one face down. For simplicity, only generate one card for the dealer, which you're allowed to do simply by generating a random integer in the range [2, 10] for this particular case. Then, you can have dealerLabel display a string like "Dealer has X + ???", where X is the randomly generated number.
    - Your Blackjack functionality need not encompass the following from the actual game of Blackjack:
        - There is no action on the part of the dealer -- the program does nothing more once the user has busted or chosen to stay. In this program, simply deal cards to the user according to their choice. The program does NOT play out the rest of the game.
            - Typically, the goal of the game is to get as high of a hand value without exceeding 21 as possible, in order to beat the dealer.
        - If the player is dealt a hand value of 21 in the actual game, they have no need to choose to stay or hit, since they've already achieved their best hand. In the program, you can allow the user to still interact with the buttons when one (or both) of their totals are 21.
        - You need not worry about the formatting of the different panels and elements so long as all elements are clearly visible in the starting frame dimensions and not obstructing each other.

Helpful Sources/Hints

- https://bicyclecards.com/how-to-play/blackjack/
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/java/awt/event/ActionListener.html

- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/javax/swing/JPanel.html
- https://docs.oracle.com/en/java/javase/16/docs/api/java.desktop/javax/swing/JButton.html

4. Drawing

Task

- Create a class ***FlagDisplay*** that will hold your *main* method. This method will be used to run your program. The main method should do the following:
    - Create a *JFrame*.
    - Give the frame appropriate dimensions for a flag.
    - Give the frame a title, corresponding to the name of the flag's country (or a made-up country name for a made-up flag).
    - Create a Flag *JComponent* and add it to the frame.
    - Display the frame to the user.
    - End program when user closes the *JFrame*.
- Create a class ***Flag*** that extends *JComponent*. Within the class, do the following:
    - Provide a void method paintComponent(Graphics g). This method uses the Graphics object provided as its argument to draw different shapes in the *JComponent*. In particular, choose an existing tricolor vertical flag, such as the flag of Belgium, and draw it. You may also choose to create your own flag, using whatever colors you want so long as two adjacent vertical bars do not have the same color. If you'd like, you may also add more elements to your flag, such as the crest in the middle of the Mexican flag, though this is not required.
    - You should draw your flag using a combination of Graphics methods, including but not necessarily limited to (see the links for more):
        - g.fillRect(int x, int y, int width, int height)
        - g.setColor(Color c)
    - Your flag should be drawn in such a way that it fills the frame and maintains a coherent ratio of bar thickness when the frame is first drawn. You don't need to worry about resizing the flag if the user resizes the window once the flag is drawn.

Helpful Sources/Hints

- https://docs.oracle.com/javase/10/docs/api/java/awt/Graphics.html
- https://en.wikipedia.org/wiki/List_of_flags_by_design#Tricolour_and_other_tribands