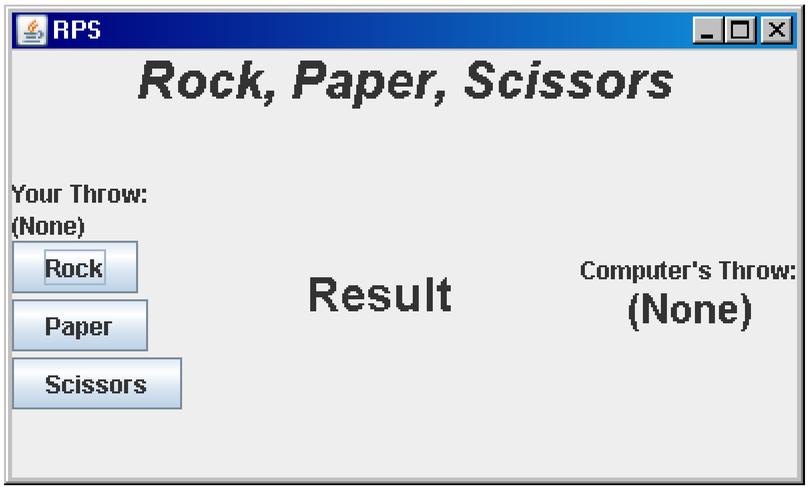
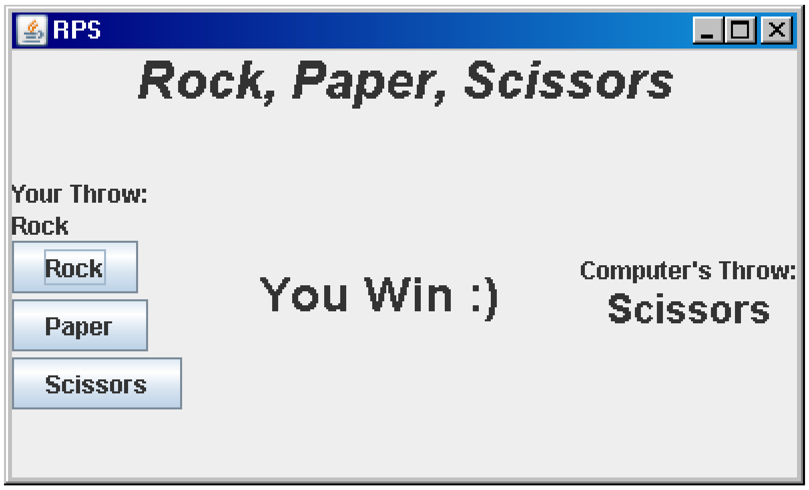
Assignment 6: Class Design, Comparable Interface, Layout Managers

Due Monday October 12, 2009 11:55PM

For this assignment, you'll be making a program that simulates the popular game Rock Paper Scissors. For your reference: http://en.wikipedia.org/wiki/Rock_paper_scissors. The program will allow the player to choose a throw, and then the computer will randomly choose its throw, and the results are displayed to the player. There are three possible throws: rock, paper, and scissors. Rock beats scissors, scissors beats paper, and paper beats rock. If both players throw the same, then it is a tie. **Please be sure to read all of the instructions before beginning.**



The player chose rock by clicking on the Rock button, the computer randomly chose scissors, and thus the player wins:



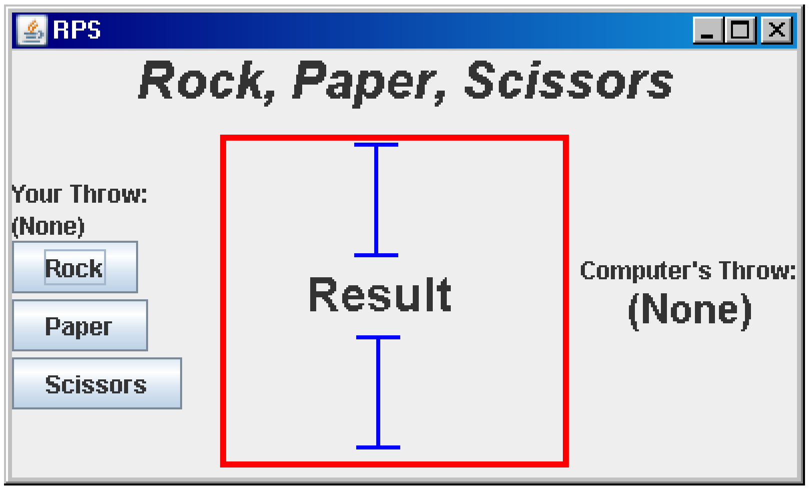
6.1: RPSThrow Class

This class will represent a player’s throw for a round of the game: rock, paper or scissors. Specs:

1. The class should implement the Comparable interface (see step 6).
2. It should have an `enum` for the 3 possible throws.
3. Private instance data to store the throw it represents (using the `enum` from step 2).
4. A static Random variable for the class to generate random numbers (see steps 8 and 10).
5. A constructor that takes in and stores the throw it represents (using the `enum` from step 2).
6. A `compareTo` method that will return (as an `int`) if this throw is “greater than” (ie, beats), is “equal to” (ie, ties), or is “less than” (ie, loses to) another throw. Rock beats scissors, scissors beats paper, and paper beats rock. Hint: from the java API: “`compareTo` returns a negative integer, zero, or a positive integer as this object is less than, equal to, or greater than the specified object.”
7. A `toString` method that will return the string representation of this throw, ie, “Rock”, “Paper”, or “Scissors”. Hint: use the features of `enums`.
8. A static method called `getRandomThrow` that will return an `RPSThrow` that is randomly chosen to be rock, paper, or scissors (this will be for the computer player).
9. In your javadocing for `getRandomThrow`, comment on why the `getRandomThrow` method can and should be static. Be sure to include in your explanation the differences between a static method and an instance method.
10. Javadoc the Random variable, and comment on why this variable can and should be static in its javadocing. Be sure to include in your explanation the differences between a static variable, an instance variable, and a local variable.

6.2: RPSResultPanel Class

The `RPSResultPanel` will be the middle panel in the window, as boxed in red:

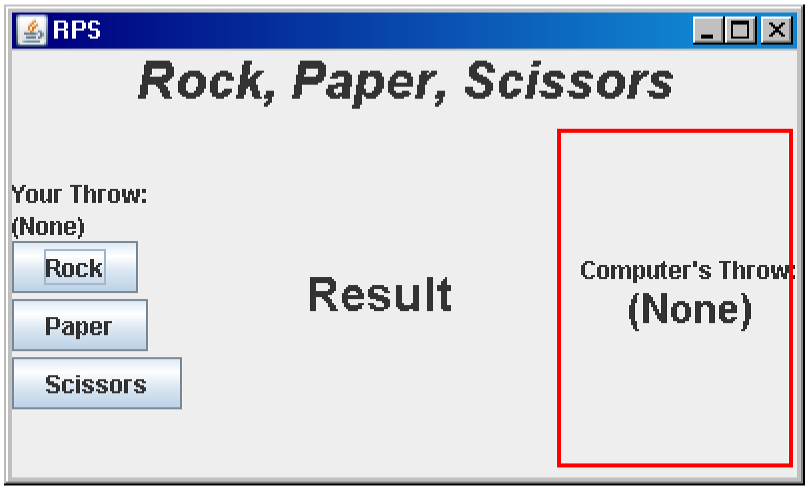


Specs:

1. The class should extend `JPanel`.
2. The label should be stored in a private instance variable.
3. To center the label that displays the result:
   a. Use a BoxLayout that lays components out on the Y axis (or the page axis)
   b. Set the label's X alignment (`setAlignmentX`) to be on the center (`Component.CENTER_ALIGNMENT`) to set its horizontal position.
   c. Add vertical glue above and below the label to force the label to be centered vertically (represented by the blue lines in the above picture). See the Box class: http://java.sun.com/javase/6/docs/api/javax/swing/Box.html
4. A method called `setResult` that takes in a String to set the label's text.
5. In your javadocing for `setResult`, explain how providing this method, rather than providing a getter for the label or leaving the variable public, respects abstraction and encapsulation.

6.3: RPSComputerPanel Class

The `RPSComputerPanel` will be the panel that displays what the computer chose:

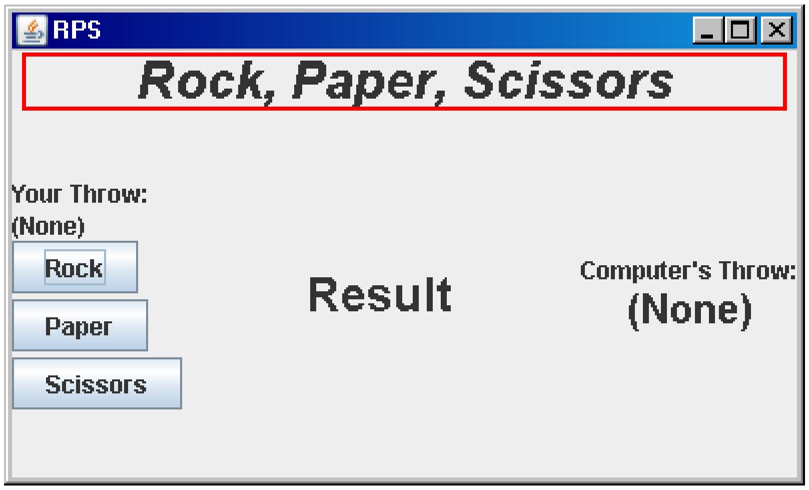


Specs:

1. The class should extend `JPanel`.
2. The labels should be stored in private instance variables.
3. Center the labels as you did for `RPSResultPanel`.
4. Provide a method called `setComputerThrow` that takes an `RPSThrow` parameter, that will update the label to display what the computer chose.

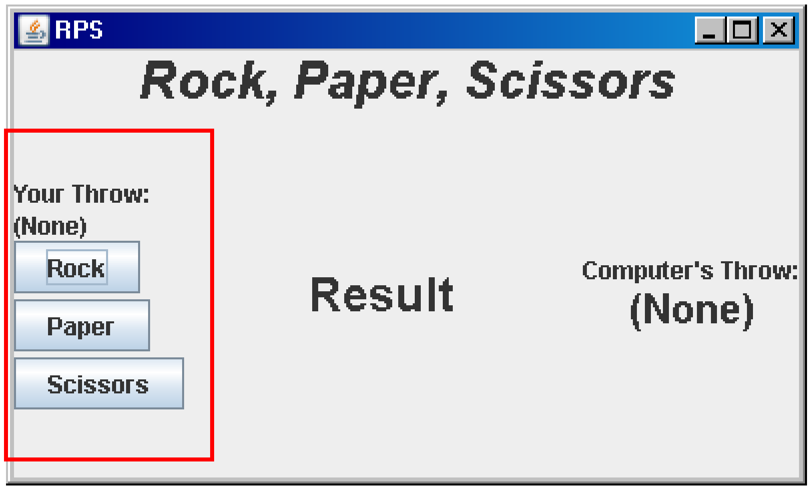
6.4: RPSTitlePanel Class

This simple subclass of `JPanel` will display the title of the game. Use a Box Layout that lays components out on the X axis (or the line axis). Add horizontal glue before and after the label to center it horizontally:



6.5: RPSPlayerPanel Class

This will be the most complicated panel, because it will have buttons to allow the player to choose his or her throw:



1. Use a Box Layout using the Y (or page) axis to lay the components out.
2. It should have a constructor that takes in an `RPSComputerPanel` and an `RPSResultPanel`. These will be needed for step 3.
3. When the player clicks one of the buttons, the program should:
    a. Create an `RPSThrow` object, depending on which button was pressed.
    b. Get a random throw for the computer.
    c. Determine who won the round, using the `RPSThrow` object's `compareTo` method.
    d. Update the player throw label, result label, and computer throw label appropriately.

6.6: RPSMain

This class will have the main method that brings all the panels together:

1. Create a `JFrame` and set its default close operation to exit on close.
2. Create instances of `RPSResultPanel`, `RPSComputerPanel`, and `RPSPlayerPanel`.
3. Recall that the default layout manager for `JFrames` is Border Layout:
    a. Add the result panel to the `CENTER` area
    b. Add the player panel to the `LINE_START` area.
    c. Add the computer panel to the `LINE_END` area.
    d. Add the title panel to the `PAGE_START` area.
4. Set the preferred size of the frame to something reasonable (the examples above are 400x240)
5. Pack and make the frame visible.

Turn-in Procedure

Turn in the following files on T-Square. Double-check that you have *submitted* and not just saved as draft (if the submission is successful, you will receive an e-mail from T-Square within a few minutes). Also, be sure all of your files compile and run.

- RPSMain.java
- RPSThrow.java
- RPSPlayerPanel.java
- RPSComputerPanel.java
- RPSResultPanel.java
- RPSTitlePanel.java
- Any other files needed to run your submission

All .java files should have a brief descriptive javadoc comment.

Don't forget your collaboration statement. You should include a statement with every homework you submit, even if you worked alone.