By: Erfan Y, Peter L, Filip M

### SpringLayout Erfan:

The SpringLayout is a layout manager that controls the location of components relative to a container’s edge or edges of another component. SpringLayouts are good for precise placement of components relative to others. It does this by allowing the user to set constraints for the component which represents a Spring object (controlling the distance, vertically and horizontally, between different components. By default, SpringLayout creates constraints that make their associated component have the minimum, preferred, and maximum and actual value. These default constraints cause a mess (overlapping) so it would be recommended to apply constraints individually to each component. Directional constraints to define the way that the component appears using SpringLayout.EAST,SOUTH,WEST,NORTH.

SpringLayouts could be useful in scenarios where components should be relative to others such as creating a Frame that changes sizes (width and length) while still having the components be relative to each other. An example of this could be dynamic text that would go off of the screen if the length was too long, yet if it was constraint using SpringLayout it would be able to show the text, based on the length, relative to the container.

Some cautions when using SpringLayout, since it is a “low level” GUI manager, it requires a lot of hard coding to get it set up. Meaning that just adding components isn't going to be effective, each component would have to have its respective constraints (location, relevance, and springs that define the components appearance (SOUTH,WEST, etc..), padding, etc..) or else the output would be overlapping and overall messy. This is because the layout does not actively make space for new components making it hard to program a lot of components. Another thing to watch out for is relating components using constraints to components that are null, as this would produce a NullPointerException.

### FlowLayout Erfan:

FlowLayout is a very simple Layout manager unlike SpringLayout, it's the default layout of JPanel objects and does not require a lot of hard coding to set up. This Layout manager places components in a row sized using preferred size. If an overflow occurs, then multiple rows are constructed for the components to wrap around. By default Flow Layouts are centered horizontally, yet the alignment can be changed (Alignment: L to R, R to L, leading, trailing, and center). The spacing of the components can also be manipulated using Hgap and Vgap mutators. Flow Layouts are used often, mostly just when there are a view components and the program requires simplicity, an example would be a pop up dialog box that displays information and an exit button. So mainly simple menus and panels would use FlowLayout as a means of not making the code to messy or complicated. Because of this, FlowLayouts cannot really define specific placements for components, and it also most often tends to make components smaller than the preferred size causing some components to disappear sometimes. Mainly, simplicity is its power and weakness.

### GroupLayout Peter:

* Description of GroupLayout: GroupLayout lays out its components in a simple manner. For the x- and y-axes, it lays the components out sequentially, or parallel to each other, as decided by the programmer.
* Because GroupLayout deals with the x- and y-axes separately, it is easier than setting the layout using 2 dimensions at the same time.
* GroupLayout is a powerful tool because it can not only lay out components with a sequential or parallel group, it can also lay out other groups in this manner.
* In addition, GroupLayout supports the addition of automatic computer-generated gaps, or manual gaps. This can suit different application’s needs.
* GroupLayout supports dynamically swapping and changing the visibility of components using the replace() and setHonorsVisibility() method.
* As discussed above, GroupLayout is good for aligning items in rows or columns with its use of sequential and parallel groups. It also can make switching different components easier, which is useful if a component needs to be swapped or hidden often.
* Cautions:
  + GroupLayout cannot be used on more than one container at the same time, which can be a common mistake when it is used on a JFrame. This is because the setLayout method forwards the GroupLayout to the ContentPane, resulting it being used on a JFrame and ContentPane at the same time.
  + Because GroupLayout handles the x- and y-dimension separately, each component needs to be added to the horizontal and vertical group twice. Failure to do this will result in an error.

### GridBagLayout Peter:

* Description of GridBagLayout: GridBagLayout is a very powerful and flexible layout manager. It places components in rows and columns, with their own specific width and height. Because of the layout’s flexibility, not all rows have the same height and not all columns have the same width.
* GridBagConstraints allows you to:
  + set the internal padding for components
  + set the external padding for components
  + determine how to fill a cell with a component in the event it’s smaller than the cell
  + determine where in the cell the component will be placed
  + set the number of rows tall and columns wide the component will be
  + set the location of the component
* GridBagLayout can be used instead of GroupLayout because it is just as powerful and flexible as GroupLayout.
* Some cautions with GridBagLayout:
  + Reusing the GridBagConstraints may cause bugs due to some fields being neglected and not reset. It is important to know which fields in the GridBagConstraints are default and which ones aren’t.
  + Using GridBagConstraints.RELATIVE and GridBagConstraints.REMAINDER can lead to unexpected behaviour.