جامعة جدة
University of Jeddah

College of Computer Science & Engineering

Department of Computer Science and Artificial Intelligence

CCCS214: Object-Oriented Programming II

Lab 4: A First Look at GUI Applications

OOP2 – Fall 2022

Lab 4: A First Look at GUI Applications

Lab Objectives

- To become familiar with common user-interface components, such as text components, radio buttons and check boxes.
- To understand the use of layout managers to arrange user-interface components in a container.
- To build programs that handle events from user-interface components.

Layout Managers

A layout manager is an object that governs the positions and sizes of components in a container. The layout manager automatically repositions and, in some cases, resizes the components when the container is resized.

To deal with layouts your code should have the following import statement: import java.awt.*;

FlowLayout

Arranges components in rows; this is the default layout manager for JPanel objects. It is the simplest layout manager. The components are arranged in the container from left to right in the order in which they were added. When one row is filled, a new row is started. You can specify the way the components are aligned by using one of three constants: FlowLayout. RIGHT, FlowLayout. CENTER, or FlowLayout.LEFT. You can also specify the gap between components in pixels.

Border Layout

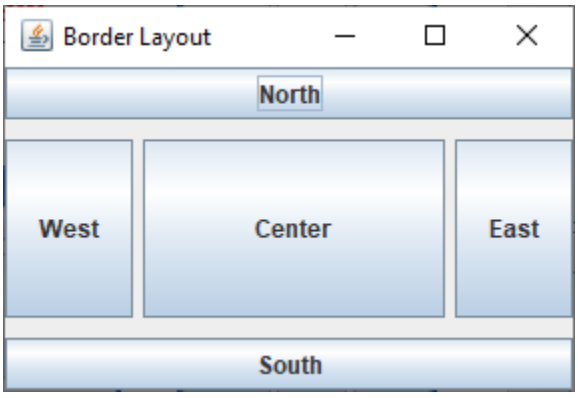
Arranges components in five regions: north, south, east, west, and center; this is the default layout manager for a JFrame object's content pane. The BorderLayout manager divides a container into five areas: East, South, West, North, and Center. Components are added to a BorderLayout by using add(Component, index), where index is a constant BorderLayout.EAST, BorderLayout.SOUTH, BorderLayout.WEST, BorderLayout.NORTH, or BorderLayout.CENTER.

Grid Layout

Arranges components in a grid with rows and columns (a matrix). The components are placed in the grid from left to right, starting with the first row, then the second, and so on, in the order in which they are added. A container that is managed by a GridLayout object is divided into equally sized cells.

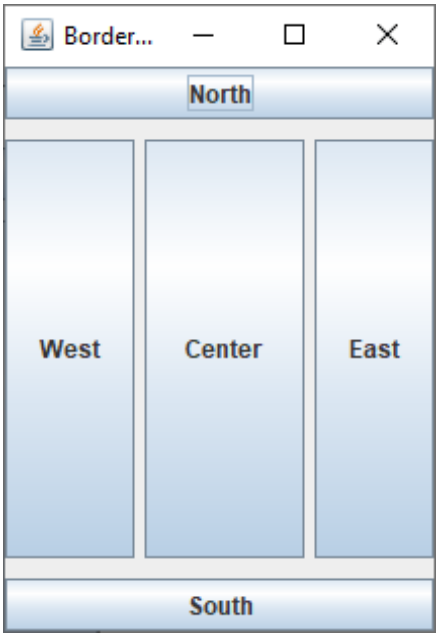
Border window example

Run the java class file **(Borderwindow.java)** with the instructor and review the code of the given example, and you will set this output:



One component (button) is added to each region.

If the user manually resizes the window:

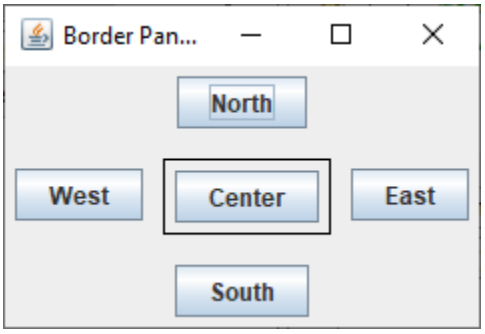


Resizing the frame causes the components to resize and maintain their same regions when one element is added to a region.

Nesting Components in a Layout

You might think that the BorderLayout manager is limiting because it allows only one component per region, and the components that are placed in its regions are automatically resized to fill up any extra space. These limitations are easy to overcome, however, by adding components to panels and then nesting the panels inside the regions.

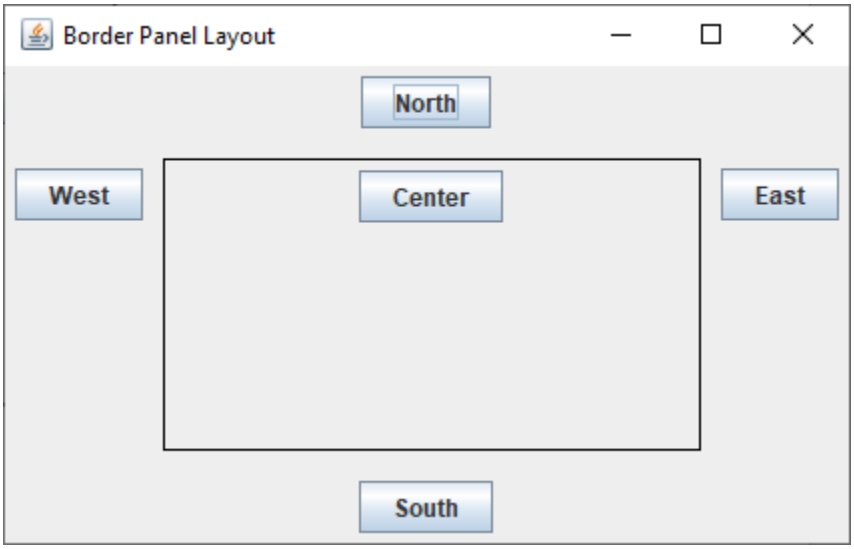
RUN the given (BorderPanelWindow.java) example and review the code with the instructor.



One panel is added to each region and one button is added to each panel (Nesting Components).

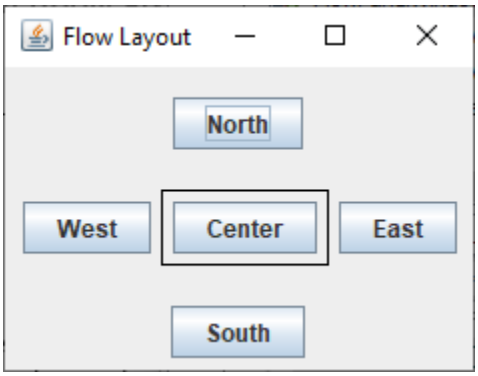
Note that:

- The window uses the BorderLayout with panels (which uses FlowLayout).
- A line border is set to the panel in the center to show you how to add borders and to define the panel area in the frame.
- Resizing the window manually by the user will change the buttons arrangement in the window like:



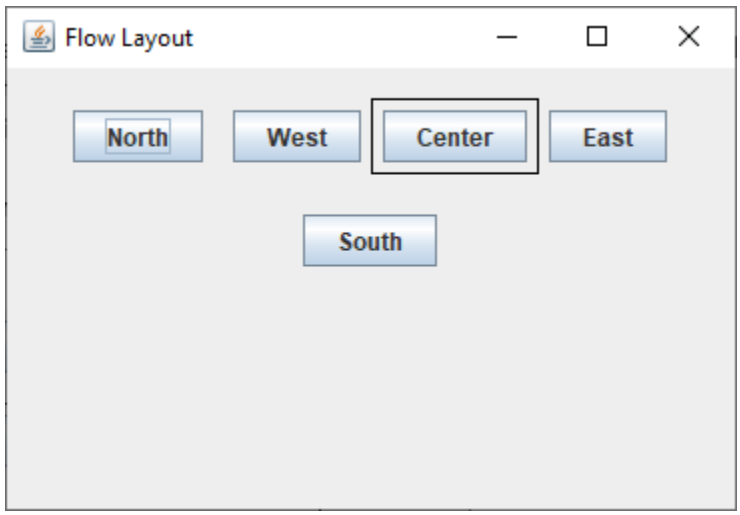
Resizing the frame causes the panels to resize and maintain their same regions when the panels are added to the regions.

If we try to design the previous example using the FlowLayout only, see the code in the java class (FlowPanelWindow) run it and we will get this output:

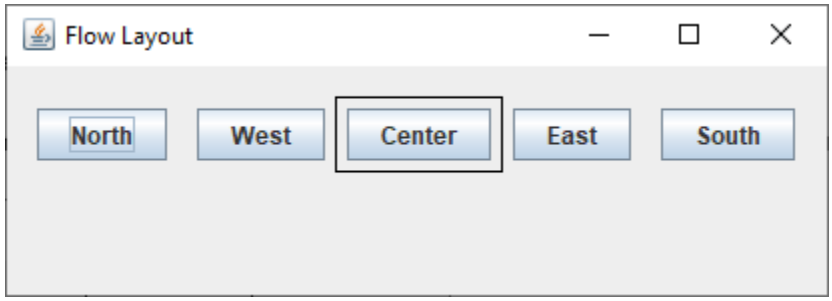


Note that:

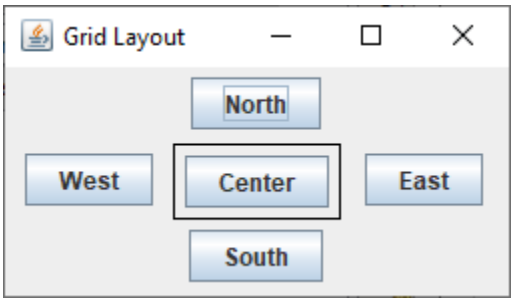
- The window uses the BorderFlowLayout with panels, Four panels are added (Three for each raw and one to hold the center button and add border around it).
- We used fixed size to ensure that the buttons are displayed the same way it is in the previous example.
- Resizing the window manually by the user will change the buttons arrangement in the window like:



or



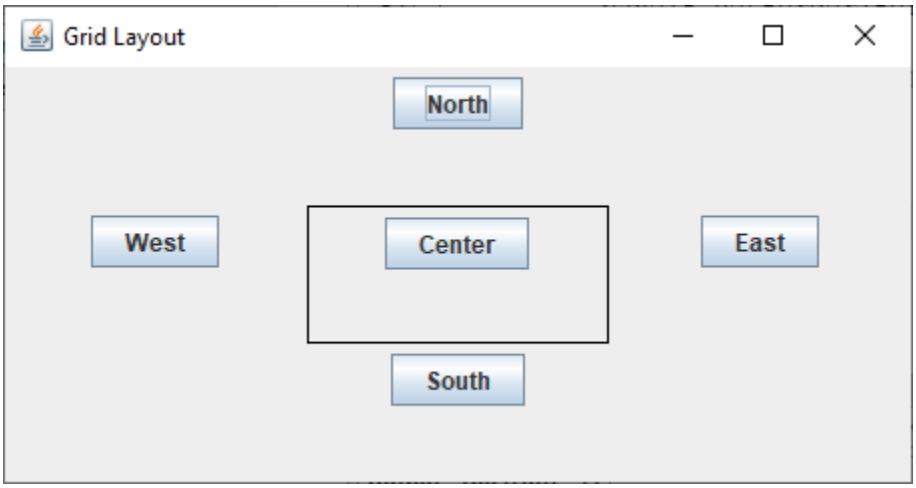
And if we try to design it using the GridLayout, see the code in the java class (GridPanelWindow)



Note that:

- Nine panels are added (five to contain the buttons in each desired cell and four empty panels to guarantee the correct buttons distribution on the grid).
- We can use the pack() method since the grid and the panels will hold the places of the buttons.

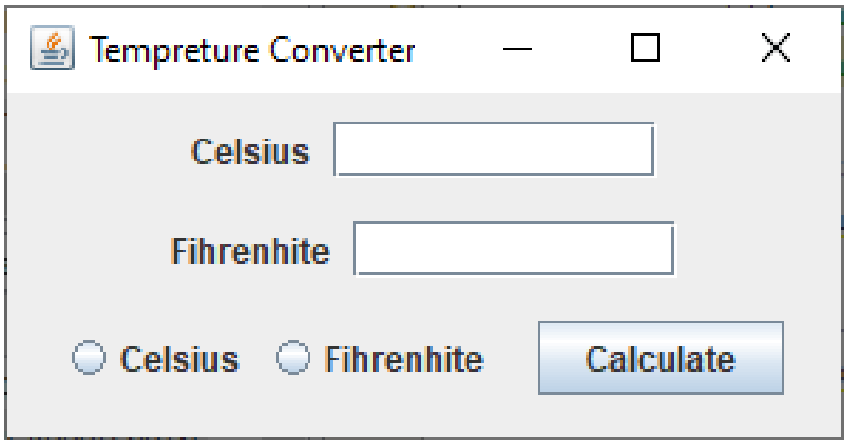
If the user manually resizes the window:



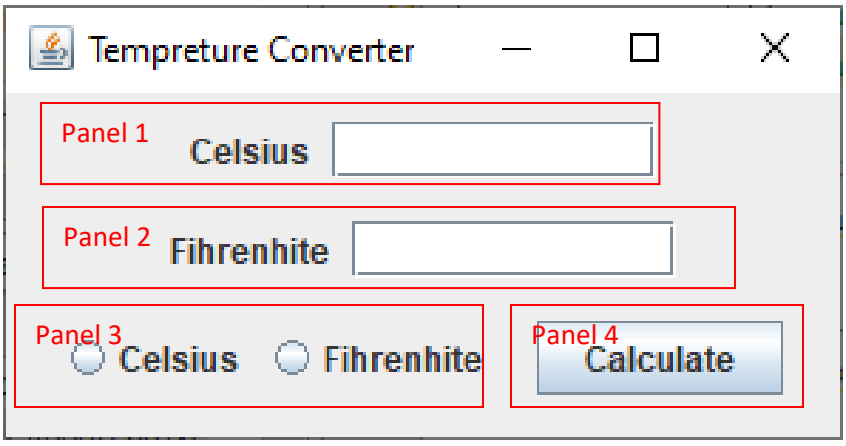
Resizing the frame causes the panels to resize and fill their cells when the panels are added to the grid.

Task # 1: (Temperature Converter- FlowLayout)

Create the following GUI by filling the missing code in the (TempConvertorT1.java) file which uses the FlowLayout. It is a GUI application that converts Celsius temperatures to Fahrenheit and vice versa. You do not have to provide any functionality.

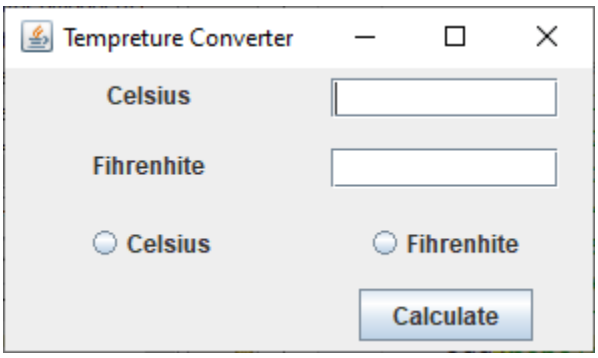


Design the panels and components as follows:

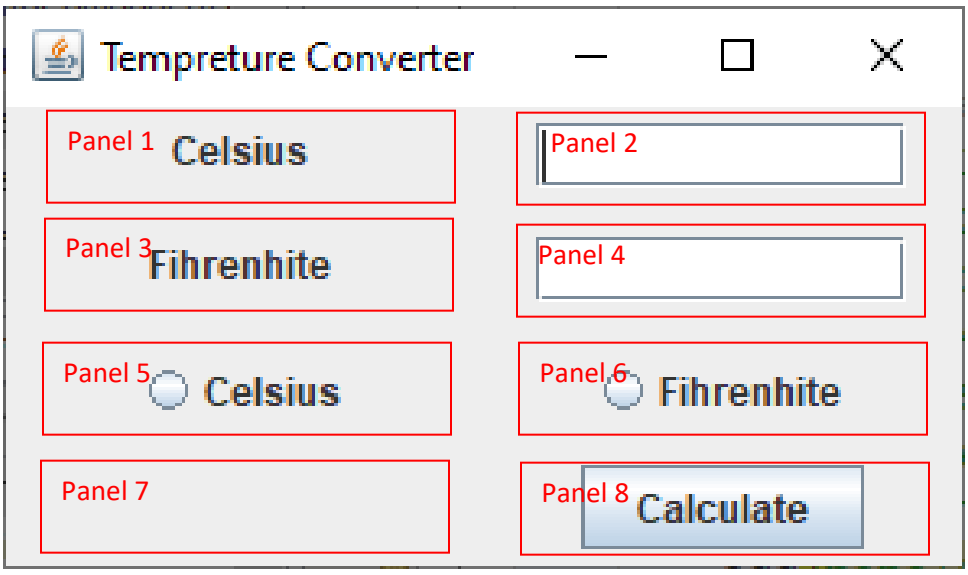


Task # 2: (Temperature converter- GridLaout)

Make a new version of Task1 file and do the necessary code changes to make the frame use a GridLayout so you can get this output:



Design the panels and components as follows:



You do not have to provide any functionality.

Task # 3: (Temperature converter- FlowLayout- Functioning)

Add functionality to the Calculate button in Task # 1 GUI, and make the program calculate the Fahrenheit temperature from the entered Celsius temperature if the user choice was Fahrenheit. And make the program calculate the Celsius temperature from the entered Fahrenheit temperature if the user choice was Celsius.

Use the formulas:

$$C = \frac{5}{9} * (F - 32)$$

$$F = \frac{9}{5} C + 32$$

Task # 4: Create the following GUI. You do not have to provide any functionality.