COMP3015
Data Communication and Networking
GUI with AWT and SWING

AWT and SWING

Java provides two packets – *java.awt* and *javax.swing* for creating graphical user interfaces. SWING is built on top of AWT and it provides more components than AWT. In this lab, we mainly use SWING in our examples.

Class javax.swing.JFrame

To create a window, we create a new class that inherits *javax.swing.JFrame*. Consider the following code:

```
import java.awt.Dimension;
import javax.swing.JFrame;

public class MyWindow extends JFrame {

   public MyWindow() {
      this.setTitle("It is my first Java GUI app");
      this.setSize(new Dimension(320, 240));
      this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      this.setVisible(true);
   }

   public static void main(String[] args) {
      new MyWindow();
   }
}
```

In the code above, we create a window using the class **MyWindow**. It has a title and its size is 640 x 480 pixels. A new thread will be created for handling the events of the window. Therefore, the window still shows even if the main thread is completed.

By default, the window will just hide if the user presses its **CLOSE** button. If you want it to be the main window of your application, you need to set the default close operation to **EXIT_ON_CLOSE**. So, your application will be terminated too if the user pressed the CLOSE button of the window.

```
this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

Layout Managers

In Java, the sizes and positions of the components are managed by the layout managers. Although each component can provide size and position information, the layout manager has the final discretion to set the size and position for its child components.

Class java.awt.FlowLayout

It lists the components from left to right, top to bottom.

```
public MyWindow() {
   this.setTitle("It is my first Java GUI app");
   this.setSize(new Dimension(320, 240));
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   Container container = this.getContentPane();

   container.setLayout(new FlowLayout());

   for(int i=0; i<8; i++) {
      JButton b = new JButton("Button " + i);
      container.add(b);
   }

   this.setVisible(true);
}
```

With a flow layout manager, the components are shown with their own size setting. You can specify the size of a component using its **setPreferredSize()** method.

```
b.setPreferredSize(new Dimension(100, 20)); // width: 100, height: 20
```

Class java.awt.BorderLayout

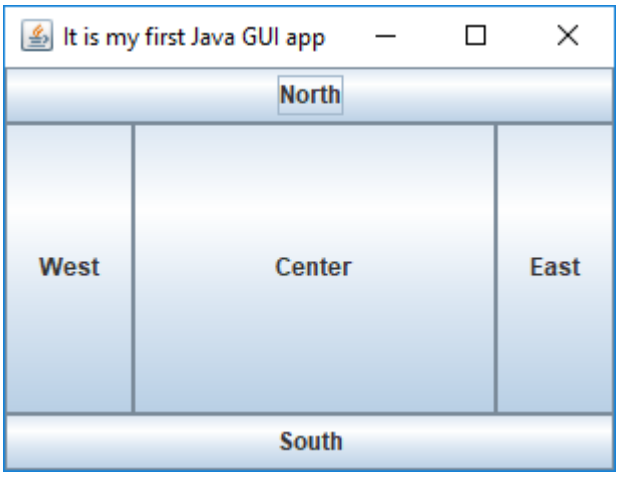
It puts the components in FIVE main regions – **NORTH**, **EAST**, **SOUTH**, **WEST**, and **CENTER**. In the following example, the buttons show in different regions:

```
public MyWindow() {
   this.setTitle("It is my first Java GUI app");
   this.setSize(new Dimension(320, 240));
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   Container container = this.getContentPane();

   JButton btn1 = new JButton("North");
   JButton btn2 = new JButton("East");
   JButton btn3 = new JButton("South");
   JButton btn4 = new JButton("West");
   JButton btn5 = new JButton("Center");

   container.setLayout(new BorderLayout());

   container.add(btn1, BorderLayout.NORTH);
   container.add(btn2, BorderLayout.EAST);
   container.add(btn3, BorderLayout.SOUTH);
   container.add(btn4, BorderLayout.WEST);
   container.add(btn5, BorderLayout.CENTER);

   this.setVisible(true);
}
```

When we add a component to the container with a border layout manager, we need to specify which region will be used for displaying the component.

```
   container.setLayout(new BorderLayout());
   container.add(btn1, BorderLayout.NORTH);
```

Note that the grid layout manager overrides the size setting of all managed components. Therefore, the statement of the **setPreferredSize()** method will be ignored.



Class java.awt.GridLayer

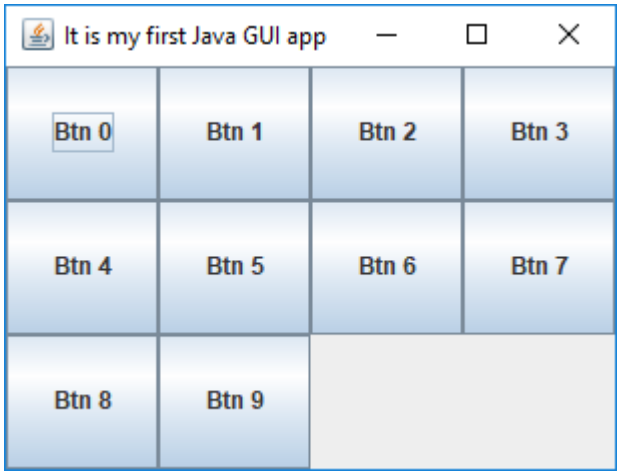
It lists the components in a grid.

```
public MyWindow() {
   this.setTitle("It is my first Java GUI app");
   this.setSize(new Dimension(320, 240));
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   Container container = this.getContentPane();

   container.setLayout(new GridLayout(0,3));

   for(int i=0; i<10; i++) {
      JButton b = new JButton("Btn " + i);
      container.add(b);
   }

   this.setVisible(true);
}
```

We can specify the number of rows or columns by providing values to the constructor of **GridLayout**.

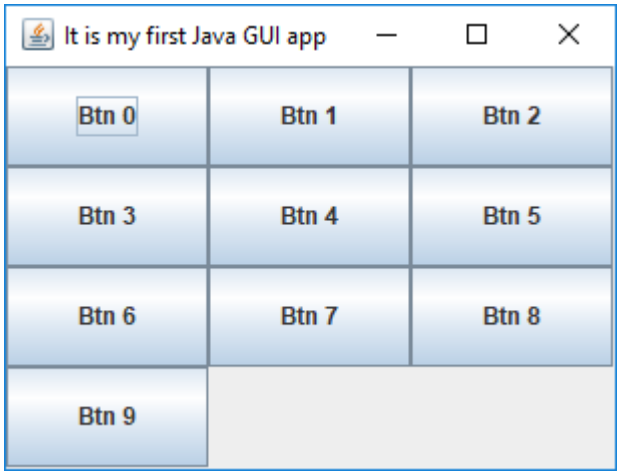
Fix the number of rows to 3:

```
new GridLayout(3,0)
```



Fix the number of columns to 3:

```
new GridLayout(0,3)
```



Note that the grid layout manager overrides the size setting of all managed components. Therefore, the statement of the **setPreferredSize()** method will be ignored.

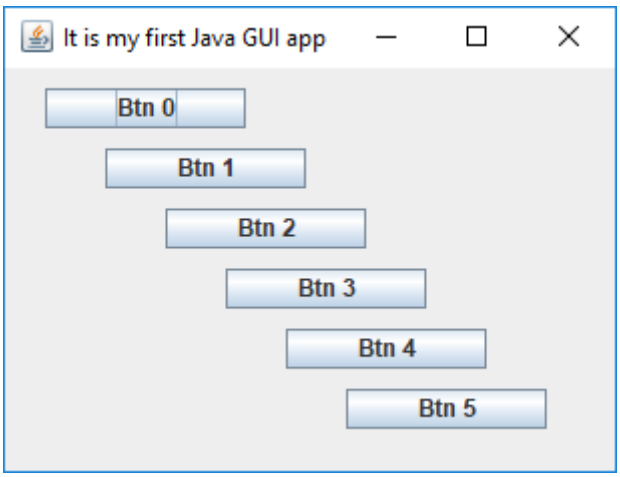
Absolute Position

You may set the layout manager of a container to null. It indicates that you want to put the components to the absolute positions. Therefore, you need to specify the position and size of each component manually.

```
public MyWindow() {
   this.setTitle("It is my first Java GUI app");
   this.setSize(new Dimension(320, 240));
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   Container container = this.getContentPane();

   container.setLayout(null);

   for(int i=0; i<6; i++) {
      JButton b = new JButton("Btn " + i);
      b.setBounds(20 + 30 * i, 10 + 30 * i, 100, 20);
      container.add(b);
   }

   this.setVisible(true);
}
```

The **setBounds()** method is used to set the position and size for a component.

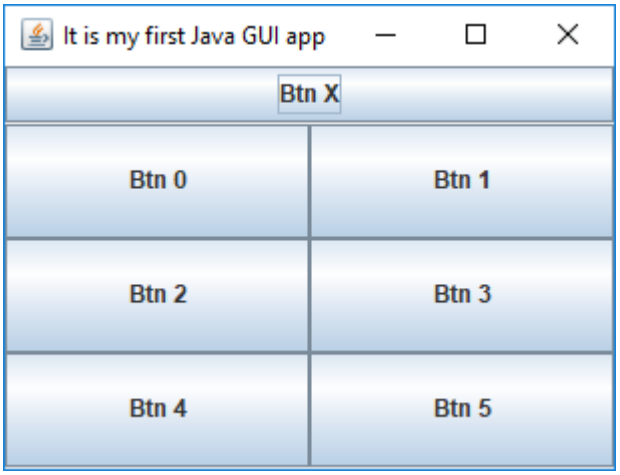
```
component.setBounds(x, y, width, height);
```



Class javax.swing.JPanel

JPanel is a container that is used for storing other components. We usually put multiple containers with different layout managers together (some containers are nested in others) to create a complex graphical user interface. If you want to have a new container, you may use **JPanel**.

```
public MyWindow() {
   this.setTitle("It is my first Java GUI app");
   this.setSize(new Dimension(320, 240));
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   Container container = this.getContentPane();

   container.setLayout(new BorderLayout());

   JButton btn = new JButton("Btn X");
   container.add(btn, BorderLayout.NORTH);

   JPanel panel = new JPanel();
   panel.setLayout(new GridLayout(0,2));
   container.add(panel, BorderLayout.CENTER);

   for(int i=0; i<6; i++) {
      JButton b = new JButton("Btn " + i);
      b.setBounds(20 + 30 * i, 10 + 30 * i, 100, 20);
      panel.add(b);
   }

   this.setVisible(true);
}
```



The button labeled “Btn X” is added to the content pane of the window directly.

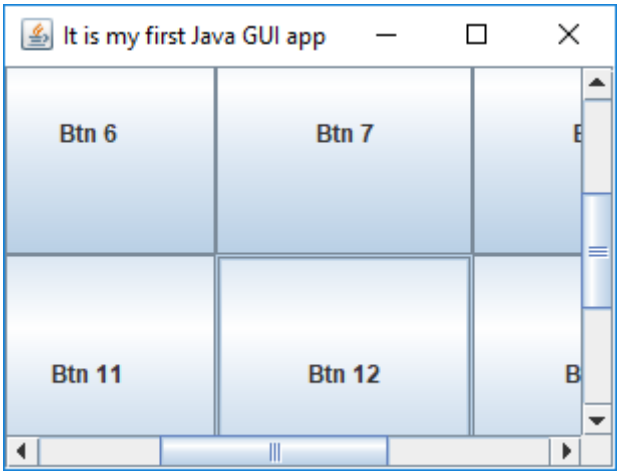
The buttons labeled “Btn 0” to “Btn 5” are added to a **JPanel**.

Class javax.swing.JScrollPane

JScrollPane provides scrollbars and manages the view area. Users can use the scrollbars to change the view area of its child panel.

```
public MyWindow() {
   this.setTitle("It is my first Java GUI app");
   this.setSize(new Dimension(320, 240));
   this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

   Container container = this.getContentPane();
   container.setLayout(new BorderLayout());

   JPanel panel = new JPanel();
   panel.setLayout(new GridLayout(0,5));
   panel.setPreferredSize(new Dimension(640, 480));

   JScrollPane sp = new JScrollPane(panel,
         JScrollPane.VERTICAL_SCROLLBAR_AS_NEEDED,
         JScrollPane.HORIZONTAL_SCROLLBAR_AS_NEEDED);

   container.add(sp, BorderLayout.CENTER);

   for(int i=0; i<20; i++) {
      JButton b = new JButton("Btn " + i);
      panel.add(b);
   }

   this.setVisible(true);
}
```

In the example code above, an inner panel is used for containing buttons and its size is 640 x 480. The window's size is 320 x 480. We know that the inner panel is larger than the window. If we do not use a scroll pane, the area exceeded cannot be viewed. The scrollbars of the scroll pane allow users to change the view area.



Commonly Used UI Components

Class javax.swing.JLabel

JLabel is used for showing a text label.

```
JLabel label = new JLabel("I am a JLabel");
```

JLabel supports HTML syntax:

```
JLabel label = new JLabel("<html><div style='text-align: center;'>I am<br/>a
JLabel</div></html>");
```

Class javax.swing.JButton and class java.awt.event.ActionListener

JButton is used for creating a button. With an action listener, the button can interact with users.

```
JButton btn = new JButton("Go...");
container.add(btn, BorderLayout.CENTER);

btn.addActionListener(new ActionListener(){

   @Override
   public void actionPerformed(ActionEvent e) {
      System.out.println("Oh! your pressed me!");
   }

});
```

Classs javax.swing.JToggleButton

JToggleButton is like **JButton** but it has an additional attribute to store its current stage – on or off.

```
JToggleButton btn = new JToggleButton("Go...");

container.add(btn, BorderLayout.SOUTH);

btn.addActionListener(new ActionListener(){

   @Override
   public void actionPerformed(ActionEvent e) {
      JToggleButton btn = (JToggleButton) e.getSource();
      if (btn.isSelected())
         System.out.println("I am on now!");
      else
         System.out.println("I am off now!");
   }

});
```

Class javax.swing.JTextField and class java.awt.event.KeyListener

JTextBox is used for creating a text input box. With a key listener, the key event related to the text field will be captured.

```
JTextField txt = new JTextField();
txt.setPreferredSize(new Dimension(200, 30));
txt.addKeyListener(new KeyListener() {

    @Override
    public void keyTyped(KeyEvent e) {
        System.out.println("You pressed and then released key #" + e.getKeyChar());
    }

    @Override
    public void keyPressed(KeyEvent e) {
        System.out.println("You pressed key #" + e.getKeyCode());
    }

    @Override
    public void keyReleased(KeyEvent e) {
        System.out.println("You released key #" + e.getKeyCode());
    }

});
```

Note: the keyTyped() method of the key listener cannot capture key codes.

You may use its **getText()** method and **setText()** method to get or set its value.

```
JButton btn = new JButton("Print");
btn.addActionListener(new ActionListener() {

    @Override
    public void actionPerformed(ActionEvent e) {
        System.out.println(txt.getText());
        txt.setText("");
    }

});
```

Class javax.swing.JTextArea

JTextArea is like **JTextField** but it allows multiple lines display.

Class javax.swing.JList

JList is used for creating an item selection list.

```
String[] data = new String[10];
for (int i = 0; i < data.length; i++)
   data[i] = "Item " + i;

JList<String> listView = new JList<String>(data);

JScrollPane sp = new JScrollPane(listView);
container.add(sp, BorderLayout.CENTER);

JButton btn = new JButton("Go...");
container.add(btn, BorderLayout.SOUTH);

btn.addActionListener(new ActionListener() {

   @Override
   public void actionPerformed(ActionEvent e) {
      System.out.println(listView.getSelectedValue());
   }

});
```

JList uses the data stored in the String array to construct a list. In the example code above, we put a list in a scroll pane. So, users can use the vertical scrollbar to scroll the list. We can check which item is selected by using **JList's getSelectedValue()** method.

Class Graphics

JPanel can be used for containing other components. It can also be used for displaying images. To do so, we need to override its **paint()** method with the **Graphics.drawImage()** method.

Here is the procedure:

1. Read the image data from the image file. Assumed that the file image.jpg is stored in the working directory.

   ```
   BufferedImage colorImage = ImageIO.read(new File("image.jpg"));
   ```

2. Create a panel with a customized **paint()** method. Use **Graphics.drawImage()** method to render the image.

   ```
   JPanel panel = new JPanel() {
      @Override
      public void paint(Graphics g) {
         super.paint(g);
         g.drawImage(colorImage, 0, 0, this);
      }
   };
   ```

3. Resize the panel to fit the image.

```
panel.setPreferredSize(new Dimension(colorImage.getWidth(),
      colorImage.getHeight()));
```

4. Call **panel.repaint()** to re-render the image. Note that you should not call the **paint()** method directly.

Methods of Graphics/Graphics2D

JPanel can be used to draw others using different methods provided by the **Graphics**/**Graphics2D** class:

Function	Example
Set color	`g.setColor(Color.black);`
	Set the color to black. All following draw or fill statements will be affected.
Set stroke	`g.setStroke(new BasicStroke(3));`
	Set the line width to 3 pixels. All following draw or fill statements will be affected.
Draw rectangle	`g.drawRect(0, 0, 20, 40);`
	Draw a 20 x 40 rectangle at position (0, 0). The origin of the rectangle is its top left corner.
Fill rectangle	`g.fillRect(0, 0, 20, 40);`
	Fill a 20 x 40 rectangle at position (0, 0).
Draw arc	`g.drawArc(0, 0, 20, 40, 0, 360);`
	Draw a 20 x 40 ellipse at position (0, 0). The starting angle is zero degree. The ending angle is 360 degree.
Fill arc	`g.fillArc(0, 0, 20, 20, 0, 180);`
	Fill a half circle at position (0, 0). The starting angle is zero degree and the ending angle is 180 degree.

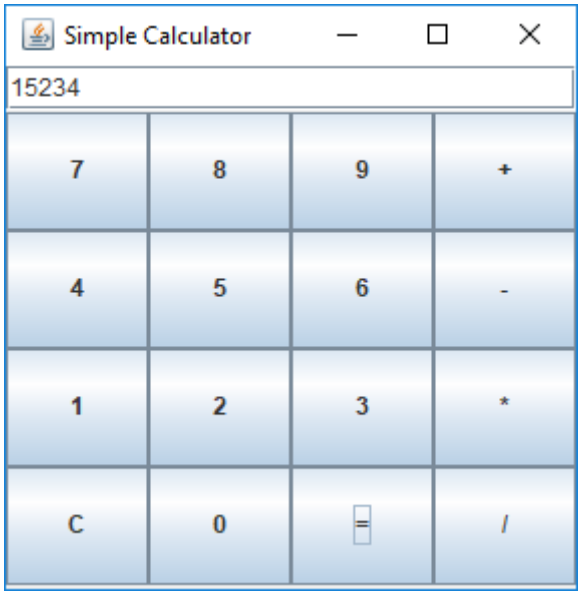
To enable anti-aliasing, we use:

```
RenderingHints rh = new RenderingHints(RenderingHints.KEY_ANTIALIASING,
      RenderingHints.VALUE_ANTIALIAS_ON);
g2.setRenderingHints(rh);
```

****Note: You can find the example code in the Project template.**

Exercise 1

Create a simple calculator as follows.



Exercise 2 (Optional)

Add action listeners to the buttons so that the calculator can be used for calculation.