Class 18 Notes

At this point you should be able to:

- Set up a frame by extending JFrame
- Add JButtons and JLabels to a JFrame
    - Instantiate a JButton/JLabel with a String
    - Instantiate a JBbutton/JLabel with a picture
- Understand the
    - BorderLayout Manager (default for JFrame)
    - FlowLayout Manager
    - GridLayout Manager
    - No layout manager (set setLayout(null)

Panels

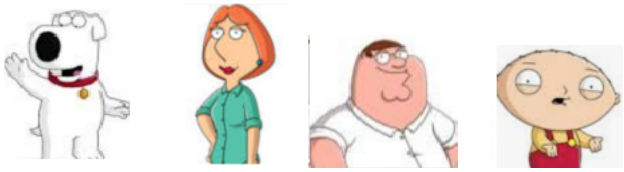
Most Swing applications do not place components directly in a frame. Instead, components are grouped together and placed in ***panels***. Then the panels are placed in the frame.

A *panel* is an invisible container used for arranging and organizing components.

Think of it like this:

You have a bunch of family photos and you decide to make a collage. So you put them in a single picture frame then hang the frame – a single unit-- on the wall.

For example, here are a few **individual** pictures of members of the Griffin family:



We can place them in a single picture frame and hang the frame on the wall.

A panel is like the picture frame. The wall is like the JFrame. We arrange components (such as JButtons and JLabels) in a panel then place the panel- in a JFrame. We can place several panels in a JFrame with various components (check boxes, text fields etc)

Some basic facts about a panel, in Java, a **JPanel**

- A JPanel can have a layout manager.
- The **default layout manager of a JPanel is FlowLayout** but can be changed
- Components are placed in panels and the panels are subsequently added to a frame.
- For example, one panel may hold a group of buttons and another, a group of checkboxes or even a mix.

So the basic process is

1. Place components into a JPanel (Place the photos in a picture frame)
2. Place the JPanel in the JFrame (Hang the picture frame on the wall)

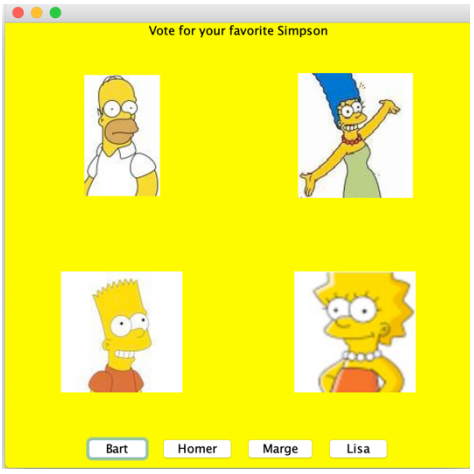
Swing's JPanel class extends JComponent.
Two constructors of JPanel are:

- JPanel()
  instantiates a JPanel object with **FlowLayout as the default layout manager.**
  Panel myPanel = newJPanel(); // has FlowLayout

- JPanel (LayoutManager layoutManager)
  instantiates a JPanel object with the specified layout manager.
  JPanel myPanel = newJPanel(new GridLayout(3,5));

- You can also use the setLayout(...) method with a JPanel.
  JPanel p = new JPanel();
  p.setLayout(new GridLayout(3,7));

Example: Create this frame

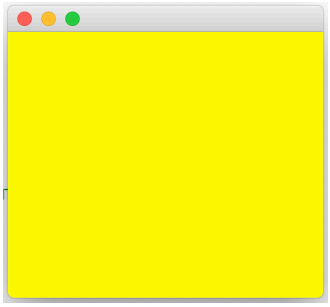


Notice:

- In the NORTH region is a centered **label** with the string " Vote for your favorite Simpson"
- In the CENTER is a JPanel with GridLayout (2,2) that has four picture JLabels
- In the SOUTH is a JPanel with four JButtons

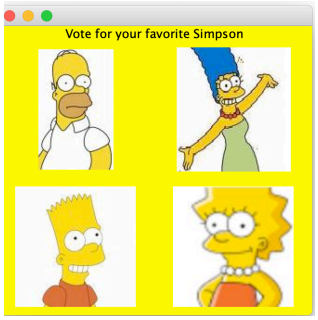
Before looking at the code lets look at the steps:

First, we extend JFrame, setting the size , position and possibly the background color:

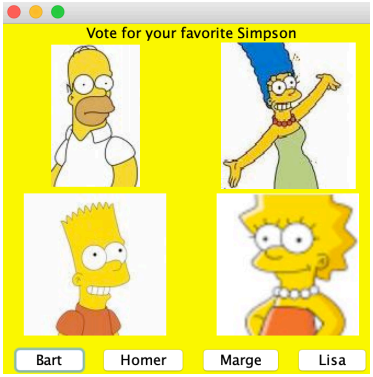


Next we create a JLabel with the string "Vote for your favorite Simpson" and place it directly on the JFrame in the NORTH. See below

Next create a JPanel with GridLayout(2,2). Make four JLabels with the images "homer.jpg" etc. and place each one in the panel. Then add the panel to the JFrame.



Finally, create a JPanel to hold the buttons. The JPanel has FlowLayout by default. Create four buttons adding each to the panel. Then add the panel in the SOUTH section of the JFrame



Here is the code—color coded with the above instructions.

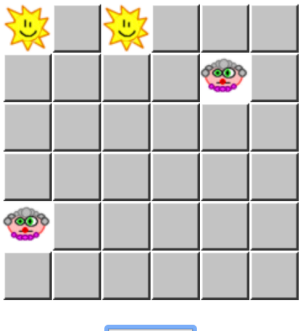
```
1.  import javax.swing.*;
2.  import java.awt.*;
3.  public class Simpsons extends JFrame
4.  {
5.    public Simpsons()
6.    {
7.      super();  // calls default of JFrame
8.      setBounds(0,0, 500,500);
9.      setBackground(Color.YELLOW);  // may not work like this on all systems

10.     // Make the label  and place it the NORTH section
11.     JLabel top = new JLabel("Vote for your favorite Simpson", SwingConstants.CENTER);
12.     add( top, BorderLayout.NORTH); // add label  (top) to the frame


13.     //  Make a JPanel to hold the four pictures, it is named picturePanel
14.     JPanel picturePanel = new JPanel();  // has FlowLayout by default

15.     // change  the layout manager of the panel to GridLayout
16.     picturePanel.setLayout(new GridLayout(2,2));
17.     // could also say
18.     // GridLayout grid = new GridLayout(2,2);
19.    // picturePanel.setLayout(grid);

20.      // Make a JLabel with the image homer.jpg and add it to picturePanel
21.      JLabel homer = new JLabel(new ImageIcon("homer.jpg"));
22.      picturesPanel.add(homer);   // notice it is picturePanel.add(..)

23.     // Make a JLabel with  the image marge.jpg and add it to picturePanel
24.     JLabel marge = new JLabel(new ImageIcon("marge.jpg"));
25.     picturePanel.add(marge);

26.     // Make a JLabel with the image bart.jpg and add it to picturePanel
27.     JLabel bart = new JLabel(new ImageIcon("bart.jpg"));
28.     picturePanel.add(bart);

29.     // Make a JLabel with the imagelisa.jpg and add it to picturePanel
30.     JLabel lisa= new JLabel(new ImageIcon("lisa.jpg"));
31.     picturePanel.add(lisa);

32.     //Now add picturePanel panel to the frame (by default at BorderLayout.CENTER):
33.     add(picturePanel);  // or this.add(picturePanel, BorderLAyout.CENTER)
```

```
34.     // Now make another JPanel for the four buttons, call it buttonPanel
35.     JPanel buttonPanel = new JPanel();  // default is FlowLayout

36.     // make the four buttons and add them to the buttonPanel
37.     JButton bart = new JButton("Bart");
38.     buttonPanel.add(bart);

39.     JButton homer = new JButton("Homer");
40.     buttonPanel.add(homer);

41.     JButton marge = new JButton("Marge");
42.     buttonPanel.add(marge);

43.     JButton lisa = new JButton("Lisa");
44.     buttonPanel.add(lisa);

45.      // add the buttonPanel to the SOUTH
46.     add(buttons, BorderLayout.SOUTH);
47.    setVisible (true)
48.  }

49.   public static void main(String[] args)
50.   {
51.     JFrame frame = new Simpsons();
52.   }
53.  }
```

Here is another example which is slightly more complex. Make sure you can follow it.

Example

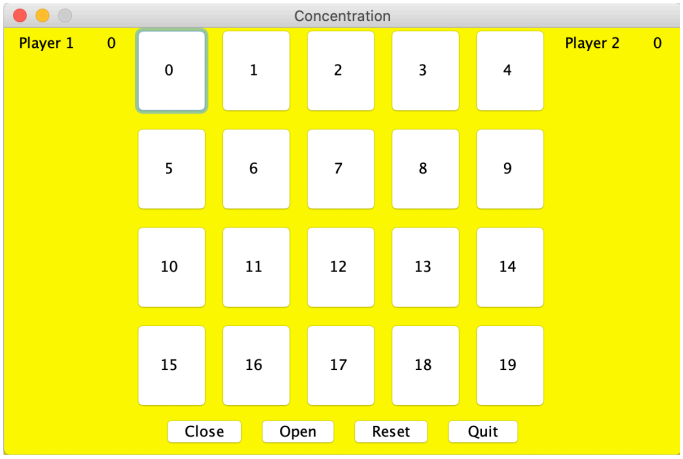
The game *Concentration* utilizes a frame with a number "doors "(doors can be buttons). Each door/button hides a picture. There are different pairs of identical pictures. For example, there may be a sun hidden by the first and third and a face hidden by two other buttons. (See below) The idea is to click two buttons and if the pictures match you get a point. If when a player clicks two buttons, the pictures do not match the pictures are once again hidden. Two players compete for the most points.



A Concentration game in progress

You can play the game online here: https://www.mathsisfun.com/games/memory/index.html

In this example, we create a frame that can be used for *Concentration*. Of course at this time the buttons will do nothing. Here is the frame that we will create:



On our version , the buttons are numbered and there are four buttons at the bottom: Close, Open, Reset, Quit. Of course at this point, they do nothing.

There are labels in the EAST and WEST giving the scores: Currently, Player 1 and Player 2 each has score 0.

The program (color coded with the actual code)

- **creates an array of twenty numbered buttons and places each in a panel which is added to CENTER**
- **creates four buttons: Close, Open, Reset, and Quit, places in them in a JPanel. The JPanel that is added to SOUTH**
- **creates two labels, player1 and score1, the first displays the text "Player 1" and the other "0," the initial score for Player 1. The two labels are placed in a panel and then the panel in the WEST**
- **creates two labels, player2 and score2, one displays "Player 2" and the o "0", the initial score for Player 2, and places the labels in a panel and adds the panel to the EAST**

```
import java.awt.*;
import javax.swing.*;

public class Concentration extends JFrame
{
    public Concentration()
    {
        super("Concentration");
        setBounds(0,0,600,400);
        setBackground(Color.YELLOW);  // may not work on all systems


        // Make  a JPanel and add each numbered JButton to the panel
        JPanel numberPanel = new JPanel(new GridLayout(4,5,10,10)); //10 is the space between buttons

         // Create an array of 20 buttons and add each to the panel
        JButton[] button = new JButton[20];
        for( int i = 0; i <20; i++)
        {
            button[i] = new JButton(i+" "); // the label i+" " is a String
            numberPanel.add(button[i]);
```

```
        }
        // add the panel to the frame at center
        add(numberPanel, BorderLayout.CENTER);

         // Create a J Panel for the four buttons at the bottom
         JPanel bottomPanel  = new JPanel(new FlowLayout());

         // Create the four bottom row buttons and add each to the panel;
        JButton buttonClose = new JButton("Close");
        bottomPanel.add(buttonClose);

        JButton buttonOpen = new JButton("Open");
        bottomPanel.add(buttonOpen);

        JButton buttonReset = new JButton("Reset");
        bottomPanel.add(buttonReset);

         JButton buttonQuit = new JButton("Quit");
         bottomPanel.add(buttonQuit);

        // add the panel in the south
         add(bottomPanel, BorderLayout.SOUTH);

         // Make a JPanel for the two labels in the West --player1 and score 1
         JPanel player1Panel = new JPanel(new FlowLayout());

         // Make 2 labels for player1 and score1  adding each to the panel
        JLabel player1 = new JLabel("  Player 1");
        player1Panel.add(player1);
        JLabel score1 =  new JLabel("      0   ");
        player1Panel.add(score1);

         // add player1Panel to WEST
          add(player1Panel, BorderLayout.WEST);

        // Make a JPanel for the two labels in the EAST player2 and score2
         JPanel player2Panel = new JPanel(new FlowLayout());

         // Make labels and add to the panel
        JLabel player2 = new JLabel("  Player 2");
        player2Panel.add(player2);
        JLabel score2 =  new JLabel("      0   ");
        player2Panel.add(score2);

         // add player2Panel to EAST
          add(player2Panel, BorderLayout.EAST);

        setResizable(false);  // cannot resize the game
        setVisible(true);
    }
    public static void main(String[] args)
    {
        JFrame game = new Concentration();
        game.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

Here is another example. This is a program that displays several of the original Star Wars characters. as


```
import javax.swing.*;
import java.awt.*;

public class StarWars extends JFrame
  {
    public StarWars()
    {
      super("Star Wars");
      setBounds(120,120,600,700);
      setBackground(Color.WHITE);

      JLabel logo = new JLabel(new ImageIcon("sw.jpg"));
      add(logo);  // uses BorderLayout.CENTER by default
```

```
    JPanel p1 = new JPanel();  // default layout is flow
    GridLayout grid = new GridLayout(2,1);
    p1 .setLayout(grid);  //changes the layout manager to grid



    // add two labels to the panel
    JLabel label1= new JLabel(new ImageIcon("sw1.jpg"),SwingConstants.CENTER);
    p1.add(label1);
    JLabel label2= new JLabel(new ImageIcon("sw2.jpg"),SwingConstants.CENTER);
    p1.add(label2);

    add(p1, BorderLayout.EAST);  // add the panel to the frame in the EAST

    // This is similar to the  above  code except the panel is in the WEST
    JPanel p2 = new JPanel();
    p2 .setLayout(grid);
    JLabel label3= new JLabel(new ImageIcon("sw3.jpg"),SwingConstants.CENTER);
    p2.add(label3);
    JLabel label4= new JLabel(new ImageIcon("sw4.jpg"),SwingConstants.CENTER);
    p2.add(label4);
    add(p2, BorderLayout.WEST);


    JPanel p3 = new JPanel();  //  uses FlowLayout by default
    // make two labels with pictures
    JLabel label5 = new JLabel(new ImageIcon("sw5.jpg"));
    JLabel label6 = new JLabel(new ImageIcon("sw6.jpg"));

    // add the labels to the panel
    p3.add(label5);
    p3.add(label6);
    // place the panel in the frame in the NORTH
    add(p3, BorderLayout.NORTH);

   // This is similar to the  above  code except the panel is in the SOUTH
    JPanel p4 = new JPanel();
    JLabel label7 = new JLabel(new ImageIcon("sw7.jpg"));
    JLabel label8 = new JLabel(new ImageIcon("sw8.jpg"));
    p4.add(label7);
    p4.add(label8);
    add(p4, BorderLayout.SOUTH);

    setVisible(true);

}

  public static void main(String[] args)
   {
    StarWars s = new StarWars();
   }
}
```

Some Very Basic Graphics

The JFrame class has a method paint(..) that draws a JFrame on the screen.
You do not call paint(), like the garbage collector, it is called by the system.

So when you create a (visible) JFrame such as

- The paint() method paints or "renders" the frame on your screen.
- If you move or resize the frame, the paint(..) method repaints it on the screen.
- If you minimize it and bring it back the paint() method repaints it on the screen.
- **You cannot call paint directly**, the system does that when it is necessary.
- When a frame needs to be rendered on your screen, the Java Virtual Machine takes care of it for you. It is like the garbage collector.

The other swing components, such as JPanel, JButton and JLabel, are rendered/drawn not by paint() but by a method called **paintComponent()**

So
- paint(…) draws JFrames and
- paintComponent(…) draws JPanels, JButtons, JLabels etc

The *paint*() and *paintComponent*() Methods

- The method, paint(...), that draws or *renders* a JFrame on the screen. When a frame is first displayed, the system calls paint(...), and paint(...) does the drawing.

- paintComponent(...),draws other Swing components such as JButtons, JLabels, or JPanels.

- When a user resizes, moves, covers, or uncovers a component, the paint(...) or paintComponent(...) method redraws the component. The method call is automatic.

- Like the garbage collector, paint(...) and paintComponent(...) work behind the scenes. An application does not explicitly invoke paint(...) or paintComponent(...). That's done by the system.

For example, here is a JFrame that we created earlier

Here is a JFrame with five JButtons.

To render it on the screen,

- the system first calls paint(…), which paints a plain, unadorned, boring,
- JFrame and then calls paintComponent() to render each of the five JButtons.
- If you move or resize the frame, the same actions occur.
- So paint(..) draws the basic frame and then paintComponent(..) draws each button.

More formally, these two methods are declared as:

```
void paint(Graphics g);
void paintComponent(Graphics g);
```

So what is the parameter Graphics g ?

Every component that can be drawn on the screen (JButton, JLabel, JFrame, JPanel) has an associated Graphics object that holds information about the component such as color, size, and font. Remember, objects hold data.

When a component is to be drawn the Graphics object for that component is automatically passed to paint(..) or paintCompoent(..). So paint(..) or paintComponent(..) knows how to draw the object-- the size, color, etc.

As I said above, the Graphics parameter, g, supplies paint(...) and paintComponent(...) with information about how to draw a particular component.

For example, certain information about the font, drawing color, and location are encapsulated in g. When your program renders a JFrame on the screen

paint(Graphics g) is called

And g is an object that has information for painting the frame.

When a JButton is rendered

paintComponent(Graphics g)

is called (automatically) and g is an object that has the information about how to draw the JButton.

A component's Graphics object is also called the component's *graphics context*. Without the graphics context g, paint(...) cannot do its job; paint(...) needs information.

So, the paint(...) and paintComponent(...) methods use the information contained in the Graphics object g to render a component. And, when the system calls paint(...) or paintComponent(...), it also sends along the graphics context of the particular component via the parameter g.