6.3 – Layout Managers

- Every container is managed by a layout manager object that determines how the components in the container are arranged visually.
  - Every container has a default layout manager, but we can replace it if desired.
- The layout manager is consulted when needed, such as when the container is resized or when a component is added.
- A layout manager determines the size and position of each component.
- Every layout manager has its own rules and properties governing the layout of the components it contains.
- For some layout managers, the order in which you add the components affects their positioning.

6.3 – Predefined Layout Managers

<table>
<thead>
<tr>
<th>Layout Manager</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Border Layout</td>
<td>Organizes components into five areas (North, South, East, West, and Center)</td>
</tr>
<tr>
<td>Box Layout</td>
<td>Organizes components into a single row or column</td>
</tr>
<tr>
<td>Tabbed Layout</td>
<td>Organizes components into one area such that only one area is visible at a time</td>
</tr>
<tr>
<td>Flow Layout</td>
<td>Organizes components from left to right, starting new rows as necessary</td>
</tr>
<tr>
<td>Grid Layout</td>
<td>Organizes components into a grid of rows and columns</td>
</tr>
</tbody>
</table>

6.3 – Flow Layout

- The easiest layout managers to use.
- The JPanel class uses flow layout by default.
- Puts as many components as possible on a row, at their preferred size.
- When a component can not fit on a row, it is put on the next row.

```java
TabbedPane pane = new JTabbedPane();
pane.addTab("Intro", new IntroPanel());
pane.addTab("Flow", new FlowPanel());
pane.addTab("Border", new BorderPanel());
pane.addTab("Grid", new GridPanel());
pane.addTab("Box", new BoxPanel());
frame.getContentPane().add(pane);
```
6.3 – Border Layout

- A border layout has five areas to which components can be added: North, South, East, West, and Center.
- The four outer areas are as large as needed in order to accommodate the component they contain.
- If no components are added to a region, the region takes up no room in the overall layout.
- The Center area expands to fill any available space.

Border Panel

```java
public class BorderPanel extends JPanel {
    public BorderPanel() {
        setLayout(new BorderLayout());
        setBackground(Color.green);
        JButton b1 = new JButton("BUTTON 1");
        ... // code for the other 4 buttons here
        add(b1, BorderLayout.CENTER);
        add(b2, BorderLayout.NORTH);
        add(b3, BorderLayout.SOUTH);
        add(b4, BorderLayout.EAST);
        add(b5, BorderLayout.WEST);
    }
}
```

6.3 – Grid Layout

- A grid layout presents a container’s components in a rectangular grid of rows and columns.
- One component in each cell, all cells are the same size.
- The number of rows and columns in a grid layout is established by using parameters to the constructor when the layout manager is created.
- As components are added to the grid layout, they fill the grid from left to right, top to bottom.
- No way to explicitly assign a component to a particular location other than the order in which they are added to the container.

Grid Panel

```java
public class GridPanel extends JPanel {
    public GridPanel() {
        setLayout(new GridLayout(2,3));
        setBackground(Color.green);
        JButton b1 = new JButton("BUTTON 1");
        ... // code for the other 4 buttons here
        add(b1);
        add(b2);
        add(b3);
        add(b4);
        add(b5);
    }
```
6.3 – Box Layout

- A box layout organizes components either vertically or horizontally, in one row or one column
- It is easy to use and when combined with other layout managers, can produce complex GUI designs
- Components are organized in the order in which they are added to the container
- There are no gaps between the components in a box layout

Box Panel

```java
public class BoxPanel extends JPanel {
    public BoxPanel() {
        setLayout(new BoxLayout(this, BoxLayout.Y_AXIS));
        setBackground(Color.green);
        JButton b1 = new JButton("BUTTON 1");
        ... // code for the other 4 buttons here
        add(b1);
        add(Box.createRigidArea(new Dimension(0, 10)));
        add(b2);
        add(Box.createVerticalGlue());
        add(b3);
        add(b4);
        add(Box.createRigidArea(new Dimension(0, 20)));
        add(b5); }
```

6.3 – Containment Hierarchies

- The way components are grouped into containers, and the way those containers are nested within each other, establishes the containment hierarchy for a GUI
- For any Java GUI program, there is generally one primary (top-level) container, such as a frame or applet
- The top-level container often contains one or more containers, such as panels
- These panels may contain other panels to organize the other components as desired

6.7 – GUI Design

- Keep in mind our goal is to solve a problem
- Fundamental ideas of good GUI design include
  - knowing the user
  - preventing user errors
  - optimizing user abilities
  - being consistent
- We should design interfaces so that the user can make as few mistakes as possible