ICT104

Program Design and Development

Lecture 8 – A First Look at GUI applications (continued)

Adopted from: Gaddis & Gaddis (2019) Starting Out with Java: From Control Structures through Objects, 7th Edition.

Focus for this week

A First Look at GUI applications (continued)

- Layout Managers
- Radio Buttons and Check Boxes
- Borders
- Focus on Problem Solving: Extending Classes from JPanel

Activity 1:
Revision Exercise

List any three concepts which you can remember from your previous week class

Layout Managers

- An important part of designing a GUI application is determining the layout of the components
- The term ***layout*** refers to the positioning and sizing of components
- In Java, you do not normally specify the exact location of a component within a window
- A ***layout manager*** is an object that:
  - controls the positions and sizes of components, and
  - makes adjustments when necessary

Layout Managers

- The layout manager object and the container work together
- Java provides several layout managers:
  - **`FlowLayout`** - Arranges components in rows. This is the default for panels
  - **`BorderLayout`** - Arranges components in five regions:
    - **North**, **South**, **East**, **West**, and **Center**
    - This is the default layout manager for a **`JFrame`** object's content pane
  - **`GridLayout`** - Arranges components in a grid with rows and columns

Layout Managers

- The **`Container`** class is one of the base classes that many components are derived from
- Any component that is derived from the **`Container`** class can have a layout manager added to it
- You add a layout manager to a container by calling the **`setLayout`** method.

```
JPanel panel = new JPanel();
panel.setLayout(new BorderLayout());
```

- In a `JFrame` constructor you might use: **`setLayout(new FlowLayout());`**

`FlowLayout` Manager

- `FlowLayout` is the default layout manager for `JPanel` objects

- Components appear horizontally, from left to right, in the order that they were added. When there is no more room in a row, the next components “flow” to the next row

- See example: **FlowWindow.java**

`FlowLayout` Manager

- The `FlowLayout` manager allows you to align components:
  - in the center of each row
  - along the left or right edges of each row
- An overloaded constructor allows you to pass:
  - `FlowLayout.CENTER,`
  - `FlowLayout.LEFT, or`
  - `FlowLayout.RIGHT`
- Example:

```
setLayout(new FlowLayout(FlowLayout.LEFT));
```

`FlowLayout` Manager

- `FlowLayout` inserts a gap of five pixels between components, horizontally and vertically.
- An overloaded `FlowLayout` constructor allows these to be adjusted.
- The constructor has the following format:

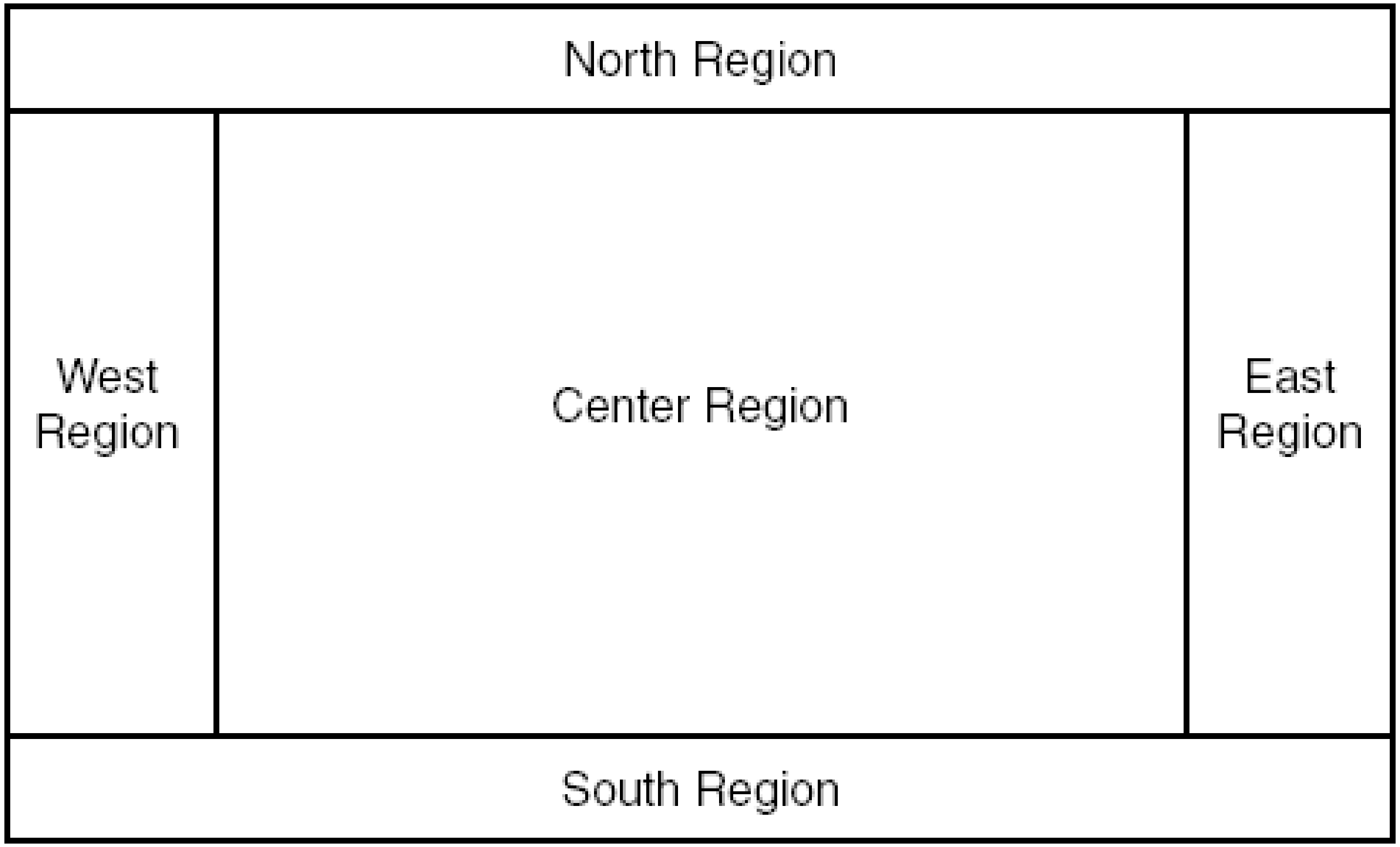
```
FlowLayout(int alignment,
           int horizontalGap,
           int verticalGap)
```

- Example:

```
setLayout(new FlowLayout(FlowLayout.LEFT,
                         10, 7));
```

`BorderLayout` Manager

*`BorderLayout` manages **five** regions where components can be placed*



`BorderLayout` Manager

- A component placed into a container that is managed by a `BorderLayout` must be placed into one of five regions:
  - `BorderLayout.NORTH`
  - `BorderLayout.SOUTH`
  - `BorderLayout.EAST`
  - `BorderLayout.WEST`
  - `BorderLayout.CENTER`

- `See example:` **`BorderWindow.java`**

`BorderLayout` Manager

- Each region can hold only one component at a time
- When a component is added to a region, it is stretched so it fills up the entire region
- `BorderLayout` is the default manager for `JFrame` objects

```
add(button, BorderLayout.NORTH);
```

- If you do not pass a second argument to the add method, the component will be added to the **center** region

`BorderLayout` Manager

- Normally the size of a button is just large enough to accommodate the text that it displays
- The buttons displayed in `BorderLayout` region will not retain their normal size
- The components are stretched to fill all the space in their regions

`BorderLayout` Manager

- If the user resizes the window, the sizes of the components will be changed as well
- `BorderLayout` manager resizes components:
  - placed in the north or south regions may be resized **horizontally** so it fills up the entire region
  - placed in the east or west regions may be resized **vertically** so it fills up the entire region
  - A component that is placed in the center region may be resized both **horizontally** and **vertically,** so it fills up the entire region

`BorderLayout` Manager

- By default, there is no gap between the regions
- An overloaded `BorderLayout` constructor allows horizontal and vertical gaps to be specified (in pixels)
- The constructor has the following format:

```
BorderLayout(int horizontalGap, int
             verticalGap)
```

- Example:

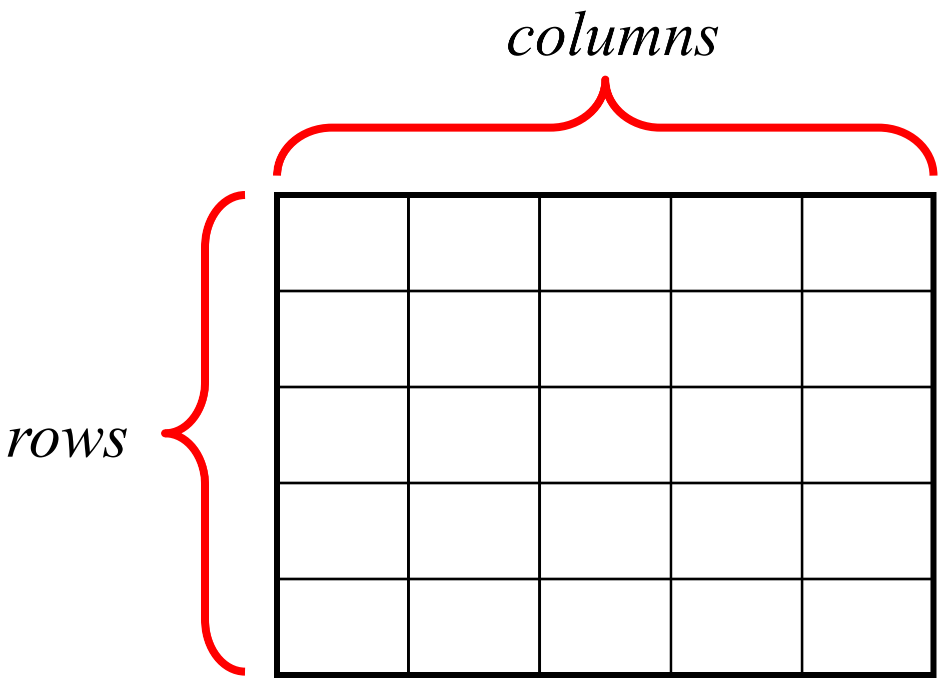
```
setLayout(new BorderLayout(5,10));
```

Nesting Components in a Layout

- Adding components to panels and then nesting the panels inside the regions can overcome the single component limitation of layout regions
- By adding buttons to a `JPanel` and then adding the `JPanel` object to a region, sophisticated layouts can be achieved
- See example: **BorderPanelWindow.java**

`GridLayout` Manager

`GridLayout` creates a grid with rows and columns, much like a spreadsheet. A container that is managed by a `GridLayout` object is divided into equally sized cells



`GridLayout` Manager

- `GridLayout` manager follows some simple rules:
  - Each cell can hold only one component
  - All of the cells are the size of the largest component placed within the layout
  - A component that is placed in a cell is automatically resized to fill up any extra space
- You pass the number of rows and columns as arguments to the `GridLayout` constructor

`GridLayout` Manager

- The general format of the constructor:

  `GridLayout(int` *`rows`*`, int` *`columns`*`)`

- Example:

```
setLayout(new GridLayout(2, 3));
```

- A zero (0) can be passed for one of the arguments but not both.
  - passing 0 for both arguments will cause an `IllegalArgumentException` to be thrown

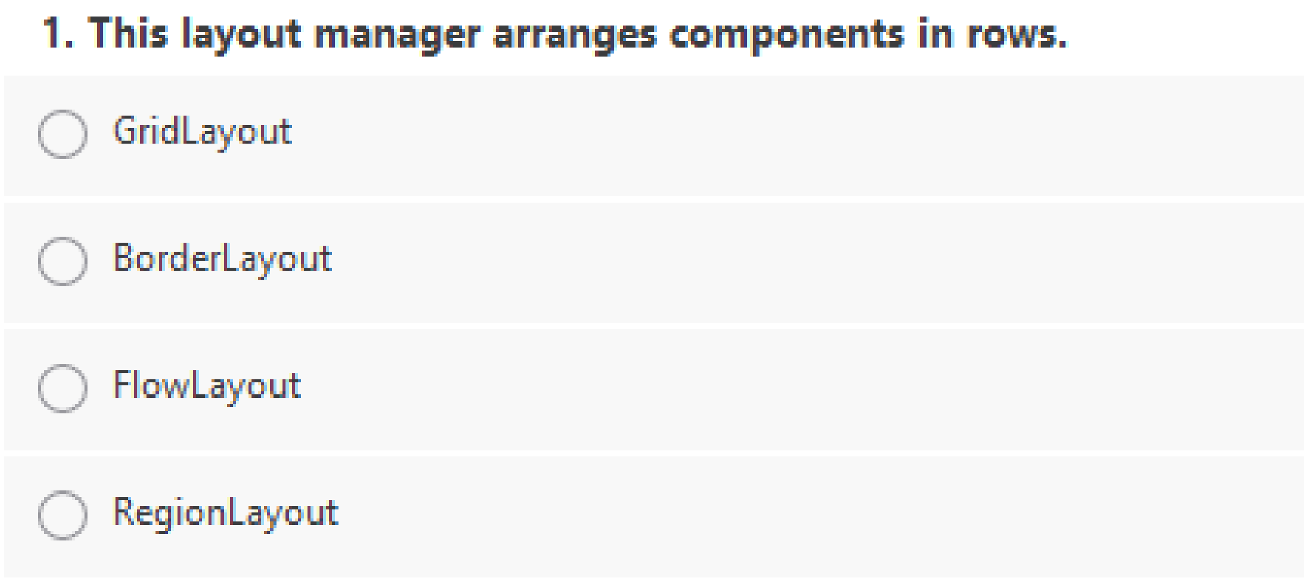
`GridLayout` Manager

- Components are added to a `GridLayout` in the following order (for a 5×5 grid):

1	*2*	*3*	*4*	*5*
6	*7*	*8*	*9*	*10*
11	*12*	*13*	*14*	*15*
16	*17*	*18*	*19*	*20*
21	*22*	*23*	*24*	*25*

- Example: **GridWindow.java**
- `GridLayout` also accepts nested components
- Example: **GridPanelWindow.java**

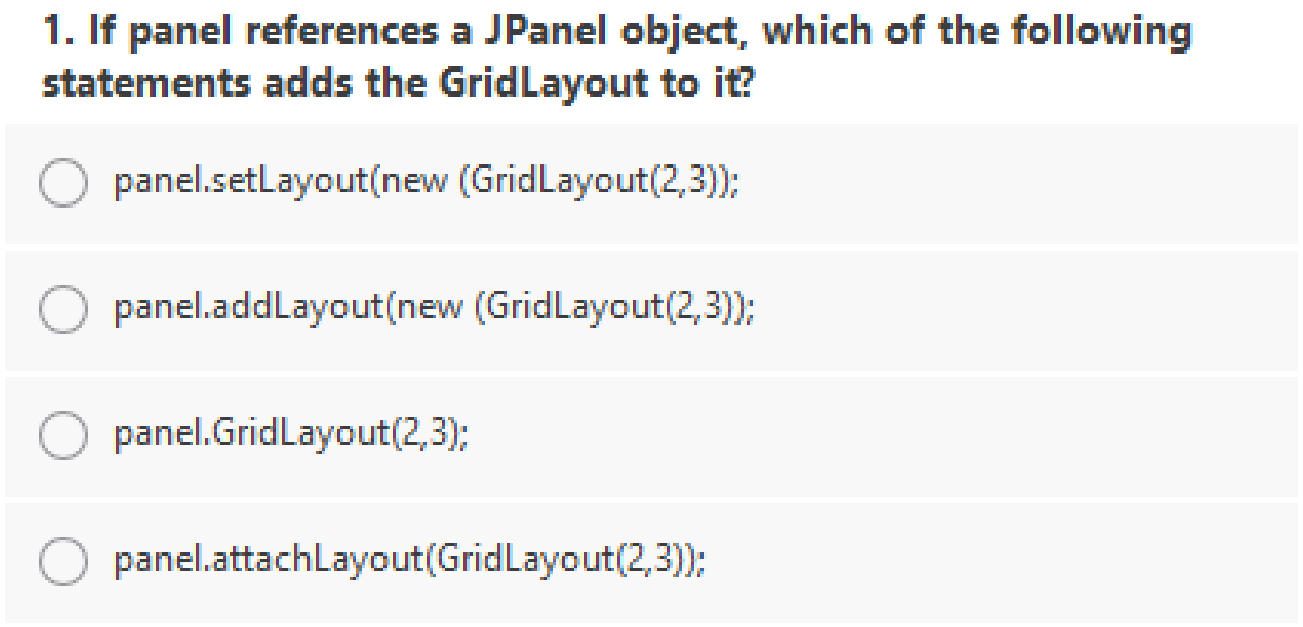
Activity 2: Poll



1. This layout manager arranges components in rows.

- ○ GridLayout
- ○ BorderLayout
- ○ FlowLayout
- ○ RegionLayout

Activity 3: Poll



1. If panel references a JPanel object, which of the following statements adds the GridLayout to it?

- ○ panel.setLayout(new (GridLayout(2,3));
- ○ panel.addLayout(new (GridLayout(2,3));
- ○ panel.GridLayout(2,3);
- ○ panel.attachLayout(GridLayout(2,3));

Radio Buttons

- *Radio buttons* allow the user to select one choice from several possible options
- The `JRadioButton` class is used to create radio buttons
- `JRadioButton` constructors:
  - `JRadioButton(String` *`text)`*
  - `JRadioButton(String` *`text, boolean selected)`*

Button appears already selected when true

- Example:

```
JRadioButton radio1 = new
  JRadioButton("Choice 1");
```

or

```
JRadioButton radio1 = new JRadioButton(
    "Choice 1", true);
```

Button Groups

- Radio buttons normally are grouped together
- In a radio button group only one of the radio buttons in the group may be selected at any time
- Clicking on a radio button selects it and automatically deselects any other radio button in the same group
- An instance of the `ButtonGroup` class is a used to group radio buttons

Button Groups

- The `ButtonGroup` object creates the ***mutually exclusive*** relationship between the radio buttons that it contains

```
JRadioButton radio1 = new
  JRadioButton("Choice 1",true);
JRadioButton radio2 = new
  JRadioButton("Choice 2");
JRadioButton radio3 = new
  JRadioButton("Choice 3");
ButtonGroup group = new ButtonGroup();
group.add(radio1);
group.add(radio2);
group.add(radio3);
```

Button Groups

- `ButtonGroup` objects are not containers like `JPanel` objects, or content frames
- If you wish to add the radio buttons to a panel or a content frame, you must add them individually:

```
panel.add(radio1);
panel.add(radio2);
panel.add(radio3);
```

Radio Button Events

- `JRadioButton` objects generate an action event when they are clicked
- To respond to an action event, you must write an action listener class, just like a `JButton` event handler
- See example: **MetricConverter.java**

Determining Selected Radio Buttons

- The `JRadioButton` class’s `isSelected` method returns a `boolean` value indicating if the radio button is selected

```
if (radio.isSelected())
{
   // Code here executes if the
   // radio button is selected
}
```

Selecting a Radio Button in Code

- It is also possible to select a radio button in code with the `JRadioButton` class’s `doClick` method
- When the method is called, the radio button is selected just as if the user had clicked on it
- As a result, an action event is generated

```
radio.doClick();
```

Activity 4: Discussion question

How many radio buttons can be selected at the same time as the result of the following code?

```
hours = new JRadioButton("Hours");
minutes = new JRadioButton("Minutes");
seconds = new JRadioButton("Seconds");
days = new JRadioButton("Days");
months = new JRadioButton("Months");
years = new JRadioButton("Years");
timeOfDayButtonGroup = new ButtonGroup();
dateButtonGroup = new ButtonGroup();
timeOfDayButtonGroup.add(hours);
timeOfDayButtonGroup.add(minutes);
timeOfDayButtonGroup.add(seconds);
dateButtonGroup.add(days);
dateButtonGroup.add(months);
dateButtonGroup.add(years);
```

Check Boxes

- A *check box* appears as a small box with a label appearing next to it
- Like radio buttons, check boxes may be selected or deselected at run time
- When a check box is selected, a small check mark appears inside the box
- Check boxes are often displayed in groups but they are not usually grouped in a `ButtonGroup`

Check Boxes

- The user is allowed to select any or all of the check boxes that are displayed in a group
- The `JCheckBox` class is used to create check boxes
- Two `JCheckBox` constructors:

```
JCheckBox(String text)
JCheckBox(String text, boolean
   selected)
```

Check appears in box if true

- Example:

```
JCheckBox check1 = new
   JCheckBox("Macaroni");
```

or

```
JCheckBox check1 = new
   JCheckBox("Macaroni",
   true);
```

Check Box Events

- When a `JCheckBox` object is selected or deselected, it generates an *item event*
- Handling item events is similar to handling action events
- Write an *item listener* class, which must meet the following requirements:
  - It must implement the `ItemListener` interface
  - It must have a method named `itemStateChanged`
    - This method must take an argument of the `ItemEvent` type

Check Box Events

- Create an object of the class
- Register the item listener object with the `JCheckBox` component
- On an event, the `itemStateChanged` method of the item listener object is automatically run
  - The event object is passed in as an argument

Determining Selected Check Boxes

- The `isSelected` method will determine whether a `JCheckBox` component is selected
- The method returns a `boolean` value

```
if (checkBox.isSelected())
{
   // Code here executes if the check
   // box is selected.
}
```

- See example: **ColorCheckBoxWindow.java**

Selecting Check Boxes in Code

- It is possible to select check boxes in code with the `JCheckBox` class's `doClick` method.
- When the method is called, the check box is selected just as if the user had clicked on it.
- As a result, an item event is generated.

```
checkBox.doClick();
```

Activity 5: Discussion question

Assume that the variable `checkBox` *references a* `JCheckBox` *object. To determine whether the check box has been selected, use the following code:*

A) `if (isSelected(checkBox)) {/*code to execute, if selected*/}`

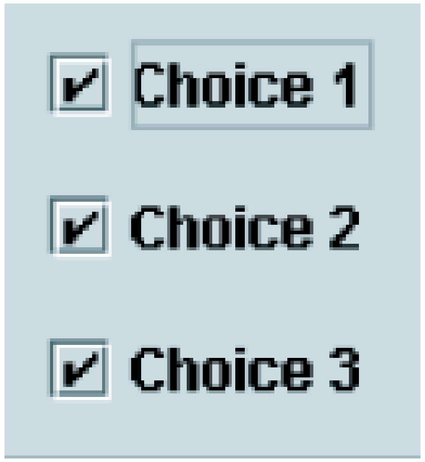
B) `if (checkBox.isSelected()) {/*code to execute, if selected*/}`

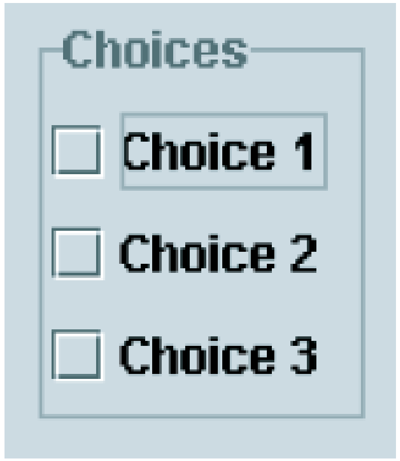
C) `if (checkBox) {/*code to execute, if selected*/}`

D) `if (checkBox.doClick()) {/*code to execute, if selected*/}`

Borders

- Windows have a more organized look if related components are grouped inside borders





- You can add a border to any component that is derived from the `JComponent` class
  - Any component derived from `JComponent` inherits a method named `setBorder`

Borders

- The `setBorder` method is used to add a border to the component
- The `setBorder` method accepts a `Border` object as its argument
- A `Border` object contains detailed information describing the appearance of a border
- The `BorderFactory` class, which is part of the `javax.swing` package, has static methods that return various types of borders

Border	`BorderFactory` Method	Description
Compound border	`createCompoundBorder`	A border that has two parts: an inside edge and an outside edge. The inside and outside edges can be any of the other borders.
Empty border	`createEmptyBorder`	A border that contains only empty space.
Etched border	`createEtchedBorder`	A border with a 3D appearance that looks "etched" into the background.
Line border	`createLineBorder`	A border that appears as a line.
Lowered bevel border	`createLoweredBevelBorder`	A border that looks like beveled edges. It has a 3D appearance that gives the illusion of being sunken into the surrounding background.
Matte border	`createMatteBorder`	A line border that can have edges of different thicknesses.
Raised bevel border	`createRaisedBevelBorder`	A border that looks like beveled edges. It has a 3D appearance that gives the illusion of being raised above the surrounding background.
Titled border	`createTitledBorder`	An etched border with a title.

The Brandi’s Bagel House Application

- A complex application that uses numerous components can be constructed from several specialized panel components, each containing other components and related code such as event listeners

- See **Brandi's Bagel House** example

Summary of today's lesson

A First Look at GUI applications (continued)

- Layout Managers
- Radio Buttons and Check Boxes
- Borders
- Focus on Problem Solving: Extending Classes from JPanel

Activity 6: Reflection Exercise

List any four concepts you have learnt in today’s lesson

Activity 7: Homework Exercise

Write a program using Eclipse or NetBeans to implement any one concept you have learnt in today’s lesson