

Objects in Programming

Lecture 5

Advanced User Interfaces

Extra material in Canvas



Overview

1. Layout Management
2. Choices
3. Menus
4. Using Time Events for Animations
5. Mouse Events
6. JavaFX

Learning goals

- Be able to implement GUIs making use of a *layout manager*
- Understand and be able to utilize commonly use *graphical components*
- Identify differences between *frames* and *panels*
- Understand and be able to utilize *menus*
- Understand and be able to utilize *time and mouse events*

LAYOUT MANAGEMENT

JFrame and JPanel

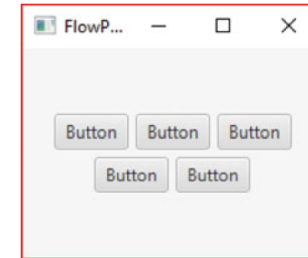
- JFrame: framed window
 - Generally used as a window for hosting stand-alone applications, like an alert window or notification window
 - Contains the title bar
- JPanel: it works as a container to host components
 - Can be considered as general container, which is used in case of complex or bigger functions which require grouping of different components together

Layout Management

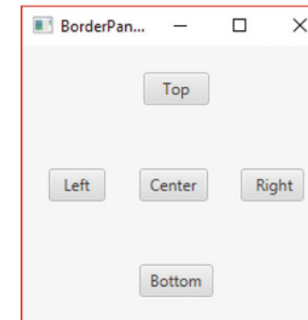
- In Java, you build up user interfaces by adding components into containers such as panels
- Each container has its own layout manager, which determines how components are laid out
- Three useful layout managers are the **flow layout**, **border layout**, and **grid layout**



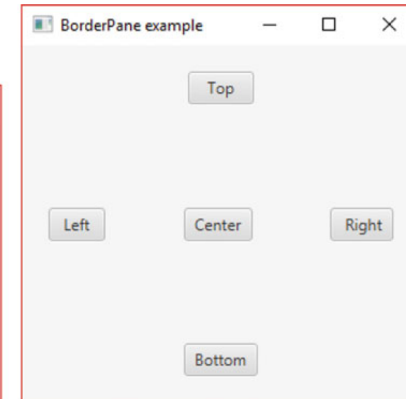
Before resizing



After resizing



Before resizing



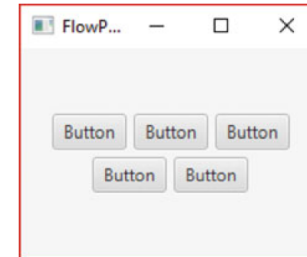
After resizing

Flow Layout

- By default, a JPanel uses a flow layout
- A flow layout simply arranges its components from left to right and starts a new row when there is no room in the current row



Before resizing



After resizing

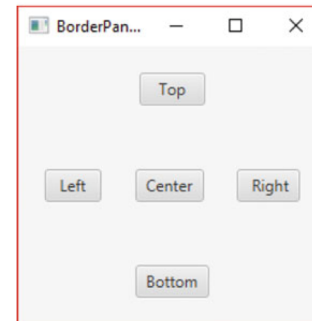
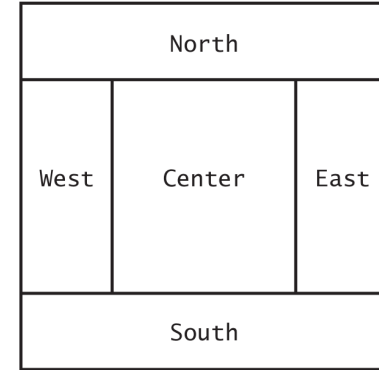
Border Layout

- Another commonly used layout manager is the border layout
- The border layout groups components into 5 areas (CENTER, NORTH, SOUTH, WEST, EAST)
- It is the default layout manager for a frame, but you can also use it in a panel:

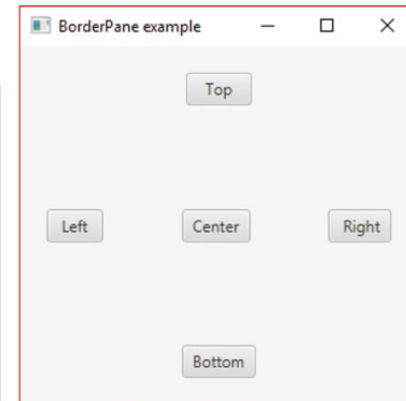
```
panel.setLayout(new BorderLayout());
```

- Now the panel is controlled by a border layout, not the flow layout. When adding a component, you specify the position like this:

```
panel.add(component, BorderLayout.NORTH);
```



Before resizing



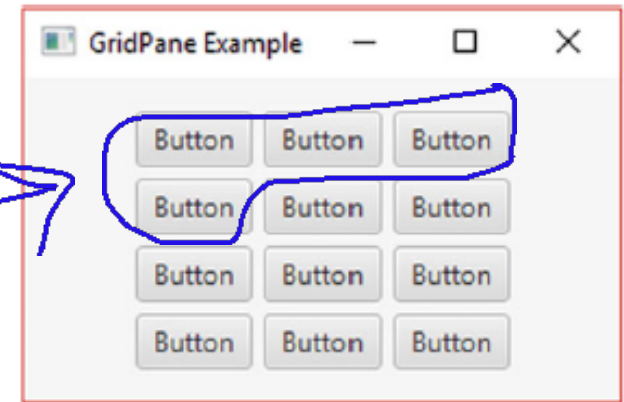
After resizing



Grid Layout

- The grid layout arranges components in a grid with a fixed number of rows and columns
- All components are resized so that they all have the same width and height
- Like the border layout, it expands each component to fill the entire allotted area
- If not desirable, you need to place each component inside a panel
- To create a grid layout, you supply the number of rows and columns in the constructor, then add the components row by row, left to right

```
JPanel buttonPanel = new JPanel();  
buttonPanel.setLayout(new GridLayout(4, 3));  
buttonPanel.add(button1);  
buttonPanel.add(button2);  
buttonPanel.add(button3);  
buttonPanel.add(button4);  
...
```



CHOICES

Choices

- How to present a finite set of choices to the user?
- Which Swing component you use if the choices are mutually exclusive or not?
- What about the amount of space you have for displaying the choices?





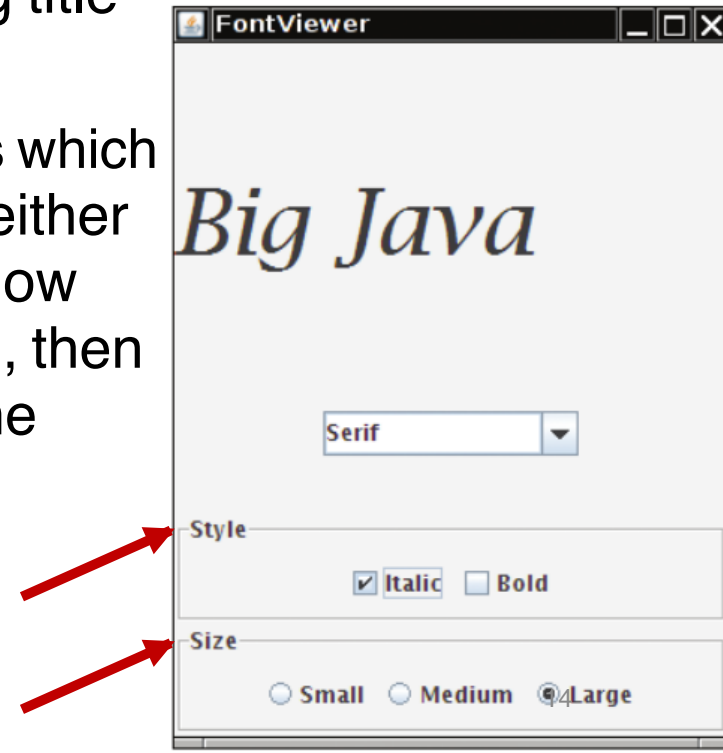
Radio Buttons

- If the choices are mutually exclusive, use a set of radio buttons
- Only one button can be selected at a time



Presenting button groups

- **javax.swing.border.TitledBorder**: A class which implements an arbitrary border with the addition of a String title in a specified position and justification
- **javax.swing.border.EtchedBorder**: A class which implements a simple etched border which can either be etched-in or etched-out. If no highlight/shadow colors are initialized when the border is created, then these colors will be dynamically derived from the background color of the component argument passed into the `paintBorder()` method.





Check Boxes

- A check box is a UI component with 2 states: **checked** and **unchecked**
- They are not exclusive
- Check boxes are square and have a check mark when selected
- Because check box settings do not exclude each other, you do not place a set of check boxes inside a button group
- You can use ***isSelected*** method to find out whether a check box is currently checked or not

```
JCheckBox italicBox = new JCheckBox("Italic");  
JCheckBox boldBox = new JCheckBox("Bold");
```



Combo Boxes

- If you have a lot of choices and little space you can use a combo box
- A combination of a list and a text field
- If the combo box is editable you can also type in your own selection
- To make a combo box editable, call the ***setEditable*** method

```
JComboBox combo = new JComboBox();  
combo.addItem("Serif");  
combo.addItem("SansSerif");  
...  
String select = (String) combo.getSelectedItem();
```

Exercise to practise: replace the Radio buttons with a combo box



Homework

- Implement the GUI including:
 - Label
 - Combo box
 - Check boxes
 - Radio buttons
 - **Events to modify the label accordingly**



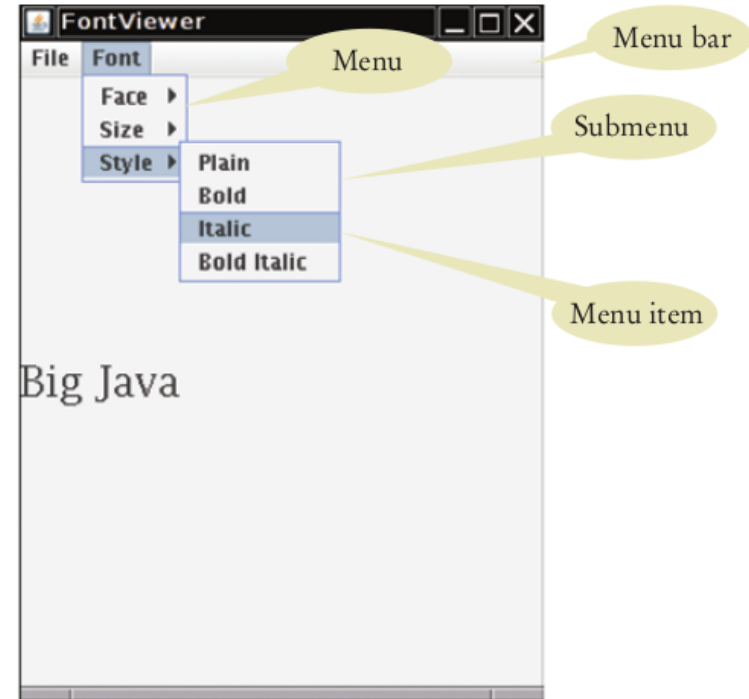
MENUS

Menus

- Anyone who has ever used GUI is familiar with pull-down menus
- At the top of the frame is a **menu bar** that contains the top-level menus
- Each menu is a collection of **menu items** and **submenus**



Menu's example



Menus

- When the user selects a menu item, the menu item sends an action event. Therefore, you must add a listener to each menu item

```
ActionListener listener = new ExitItemListener();  
exitItem.addActionListener(listener);
```

- You add action listeners only to menu items, not to menus or the menu bar
- When the user clicks on a menu name and a submenu opens, no action event is sent



Using variables from an inner class method

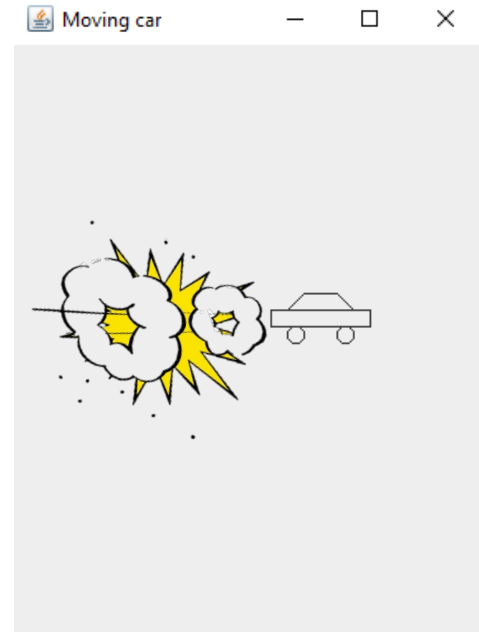
USING TIMER EVENTS FOR ANIMATIONS

Using Timer Events for Animations

- The **Timer** class in the **javax.swing** package generates a sequence of action events, spaced at even time intervals
 - This is useful whenever you want to send continuous updates to a component
 - E.g., in an animation you may want to update a scene 10 times per second and redisplay the image to give the illusion of movement
1. When you use a timer you specify the frequency of the events and an object of a class that implements the **ActionListener** interface
 2. Place whatever action you want to occur inside the **actionPerformed** method
 3. Start the timer



Let's start our car!





Example – refreshing components

Homework



MOUSE EVENTS

Mouse Events

- If you write programs that show drawings, and you want the users to manipulate the drawings with a mouse, then you need to listen to mouse events
- Mouse listeners are more complex than action listeners



Mouse Listener

- A mouse listener must implement the `MouseListener` interface, which contains the following 5 methods

```
public interface MouseListener {  
    void mousePressed(MouseEvent event);  
    void mouseReleased(MouseEvent event);  
    void mouseClicked(MouseEvent event);  
    void mouseEntered(MouseEvent event);  
    void mouseExited(MouseEvent event);  
}
```

- It often happens that a particular listener specifies actions only for one or two of the listener methods. Nevertheless, all 5 methods of the interface must be implemented
 - You can also use adapters (see Lecture 4)

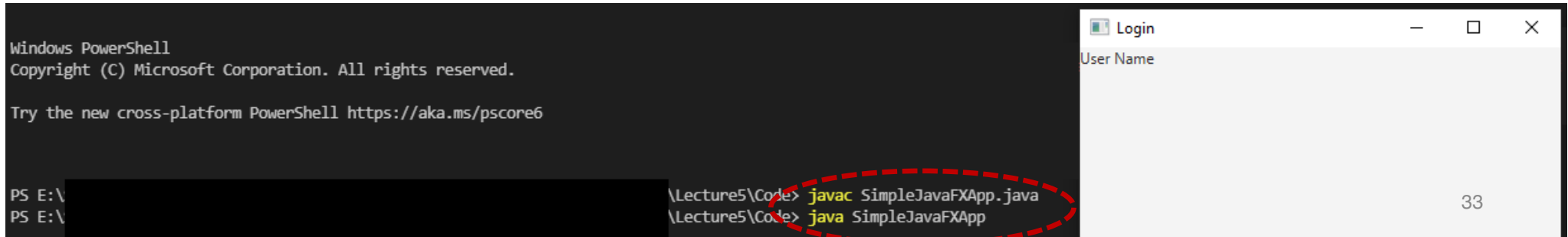
JAVAFX

JavaFX

- JavaFX is a library that enables you to create and deploy a rich client application
- The main purpose of this library is to enable a developer to create a consistent look and feel across a variety of platforms such as cell phone, browser, car dashboard, and so forth

JavaFX

- The library is shipped along with the Java SDK by default (no separate installation required) in versions before Java 11
 - For newer versions (**JavaFX JDK and SDK!!!**):
<https://docs.oracle.com/javafx/2/installation/jfxpub-installation.htm>
 - For its access from Visual Studio Code:
<https://marketplace.visualstudio.com/items?itemName=shrey150.javafx-support>
 - Setting up VS Code:
<https://www.youtube.com/watch?v=H67COH9F718>
<https://linuxtut.com/en/66fbd4a6b3db8e7c2851/>
- If you face issue to set up VS Code, just compile and run the files directly from the command line:



The screenshot shows a Windows PowerShell terminal window on the left and a JavaFX Login window on the right. The PowerShell window displays the standard copyright notice and a prompt to try the new cross-platform PowerShell. Below this, two command-line prompts are shown: 'PS E:\>' followed by a redacted path, and another 'PS E:\>' followed by the path '\Lecture5\Code>'. The commands 'javac SimpleJavaFXApp.java' and 'java SimpleJavaFXApp' are entered on the same line, with the entire command line circled in red. The JavaFX Login window on the right has a title bar with a minus, maximize, and close button. It contains a 'User Name' label and a large text input field. The number '33' is visible in the bottom right corner of the JavaFX window.

```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Try the new cross-platform PowerShell https://aka.ms/pscore6

PS E:\>
PS E:\> \Lecture5\Code> javac SimpleJavaFXApp.java
           \Lecture5\Code> java SimpleJavaFXApp
```

JavaFX and Swing

- Swing has been around for quite some time with the same purpose of providing UI needs of Java
- Swing offers excellent flexibility and capability in creating a GUI
- Swing classes are not built to leverage graphical hardware components. This reduces performance and lacks efficiency when dealing with complex graphics
- JavaFX brought a fresh new UI framework as a complete UI library

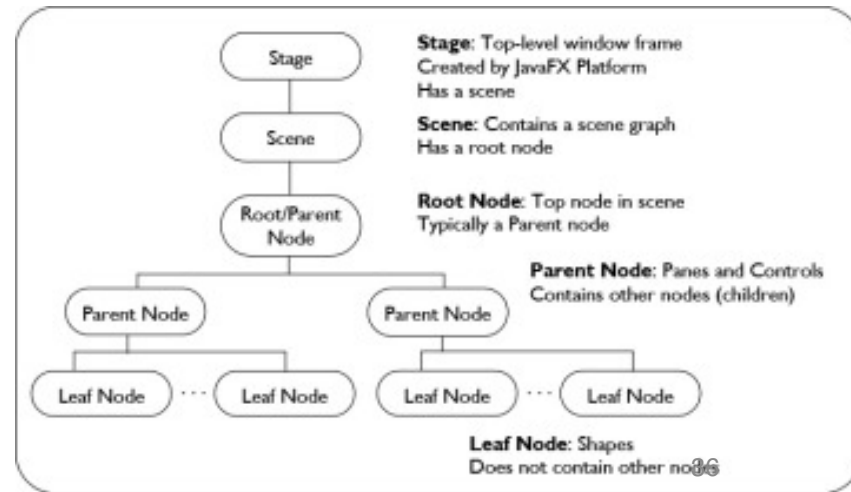
Layouts and UI Controls

- Layouts should have consistent arrangements of a UI control such as Buttons, Texts, Shapes, within the viewable area
- JavaFX layouts are simpler and more intuitive to implement than Swing layouts
- Some common layouts are:
 - `javafx.scene.layout.Hbox`**: Lays out UI controls within a horizontal box
 - `javafx.scene.layout.Vbox`**: Lays out UI controls within a vertical box
 - `javafx.scene.layout.FlowPane`**: UI controls are arranged in a flow that wraps at the flow pane interior
 - `javafx.scene.layout.BorderPane`**: UI controls are laid out in the left, top, right, bottom, and centre position of the scene
 - `javafx.scene.layout.GridPane`**: Lays out UI controls in a tabular fashion, in grids of rows and columns



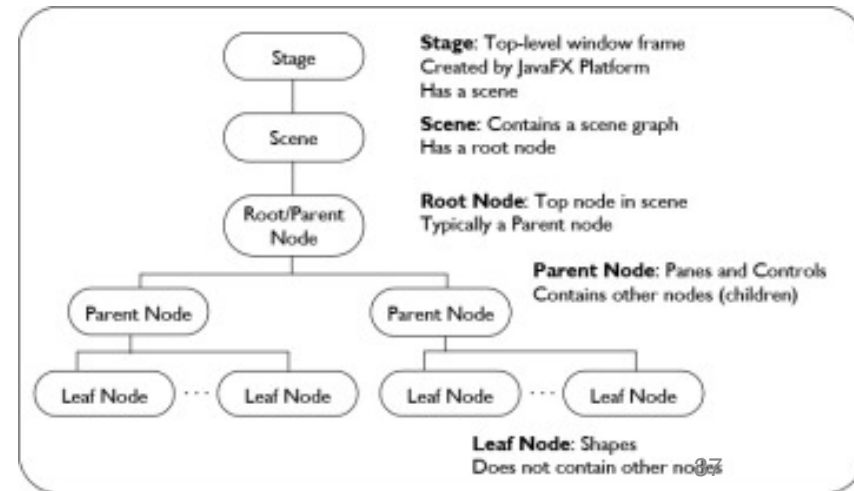
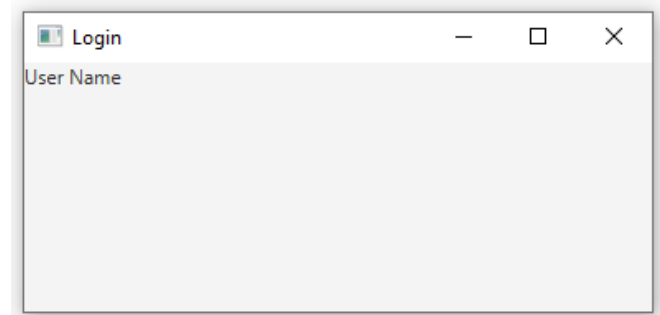
JavaFX example

- JavaFX *Stage* (`javafx.stage.Stage`): represents a window in a JavaFX desktop application
- JavaFX *Scene*: inside a JavaFX Stage you can insert a JavaFX Scene which represents the content displayed inside a window (inside a *Stage*).
- Launching a JavaFX application: it creates a root Stage object which is passed to the `start(Stage primaryStage)` method of the root class of your JavaFX application. This *Stage* object represents the primary window of your JavaFX application. You can create new *Stage* objects (in life time), if your application needs to open more windows.





JavaFX example



JavaFX Media

- The JavaFX media API enables a developer to incorporate audio and video capabilities into a Java application
- The Media API is designed to be cross platform; that means multimedia content can be implemented in an equivalent manner while coding across multiple devices (tablet, media player, TV, etc.)
- The **MediaPlayer** class provides the controls for media playing but does not provide any view
- A view can be realized with the help of **MediaView**



JavaFX Media example

JavaFX Web

- JavaFX provides capabilities to interoperate HTML5 contents with the help of **WebKit** rendering engine
- Similar to MediaView, WebView is used to manage WebEngine and display its content
- JavaFX can harness HTML5's rich Web content to create a Web user interface resembling the native desktop



JavaFX Web example

Learning goals

- Be able to implement GUIs making use of a *layout manager*
- Understand and be able to utilize commonly use *graphical components*
- Identify differences between *frames* and *panels*
- Understand and be able to utilize *menus*
- Understand and be able to utilize *time and mouse events*