CS 213 SOFTWARE METHODOLOGY

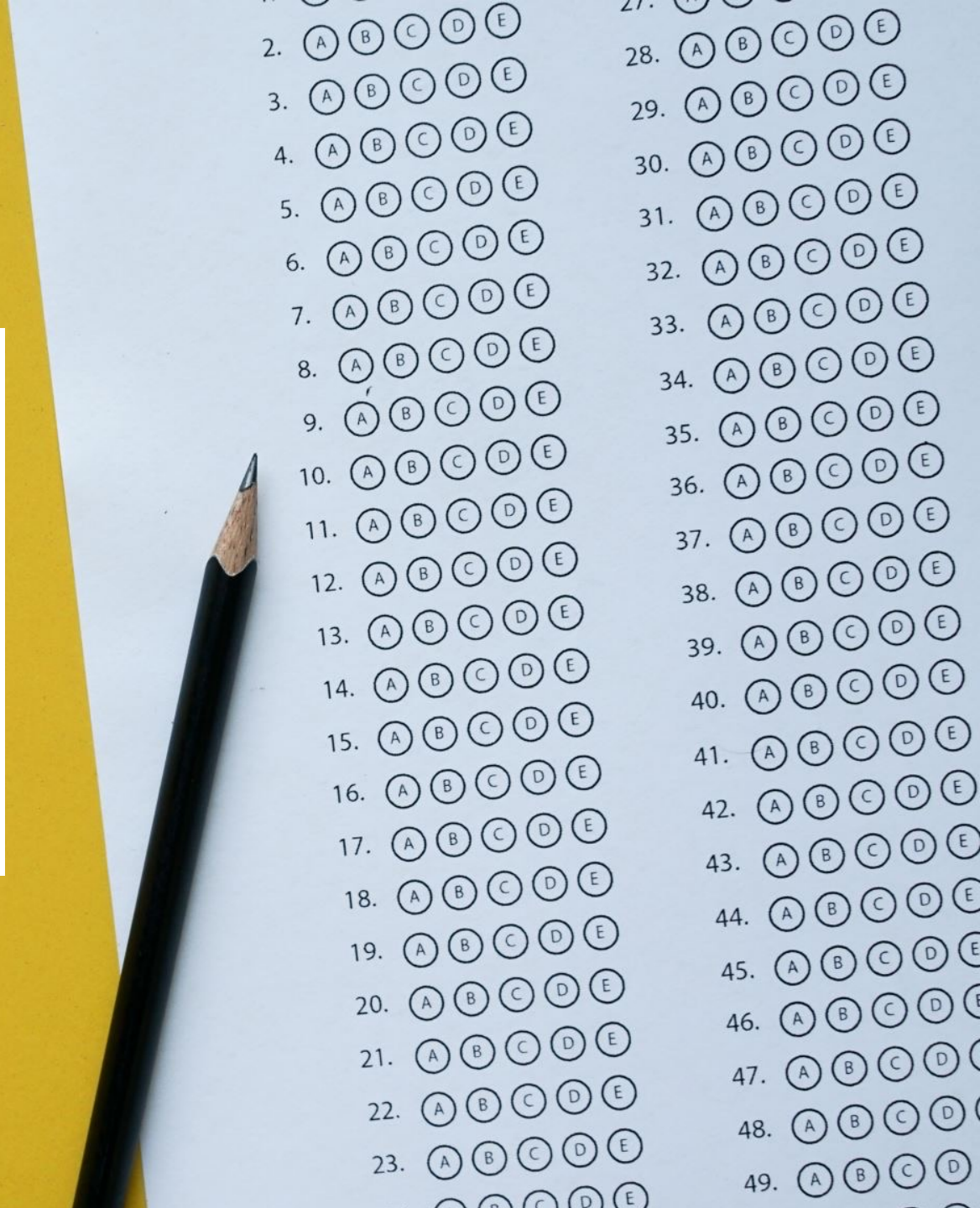
Lily Chang

CS Department @ Rutgers New Brunswick

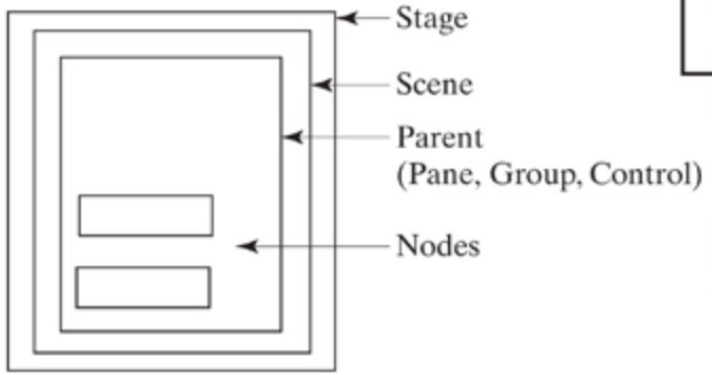
FALL 2023

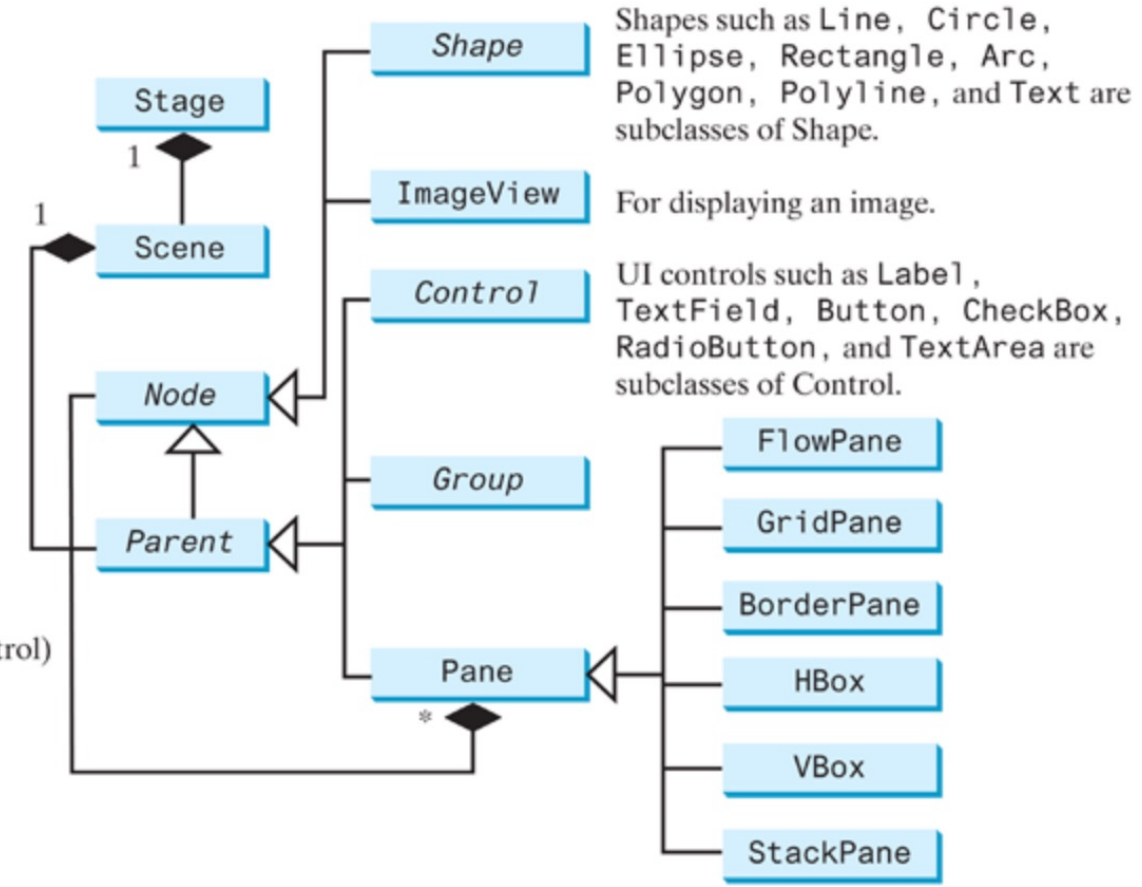
More on JavaFX

Lecture Note #15

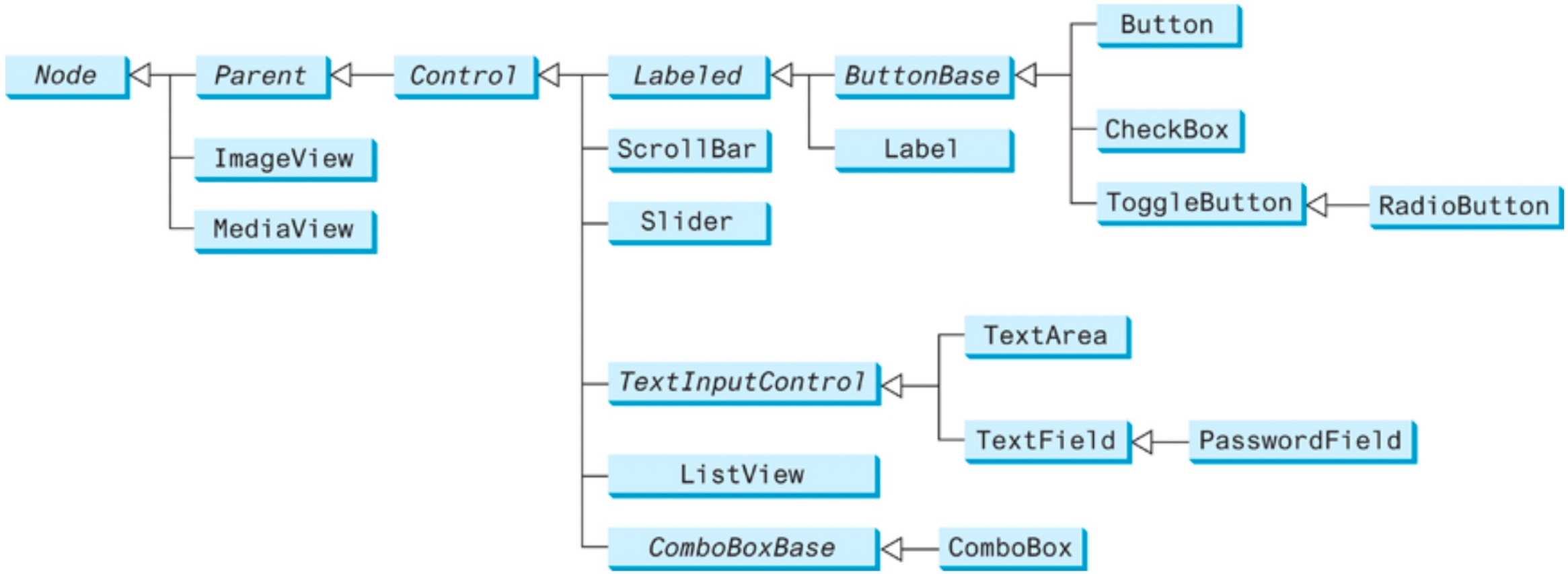


JAVAFX GUI COMPONENTS REVISITED





JAVAFX UI CONTROLS REVISITED



JavaFX Scene Graph

- The JavaFX scene graph is a hierarchical tree of nodes that represents all the visual elements of the application's user interface
- A single element in a scene graph is called a node. Each node has an ID, style class, and bounding volume; except for the root node of a scene graph, each node in a scene graph has a single parent and zero or more children.
- It can also have the following:
  - Effects, such as blurs and shadows
  - Opacity
  - Transforms
  - Event handlers (such as mouse, key and input method)
  - An application-specific state

The Node class

Each item in the scene graph is called a **Node**, which is the base class for scene graph nodes.

Branch nodes are of type Parent, whose concrete subclasses are Group, Region, and Control, or subclasses thereof.

Leaf nodes are classes such as Rectangle, Text, ImageView, MediaView, or other such leaf classes which cannot have children.

Only a single node within each scene graph tree will have no parent, which is referred to as the "root" node.

A node may occur at most once anywhere in the scene graph.

The scene graph must not have cycles.

If a program adds a child node to a Parent (including Group, Region, etc.) and that node is already a child of a different Parent or the root of a Scene, the node is automatically (and silently) removed from its former parent.

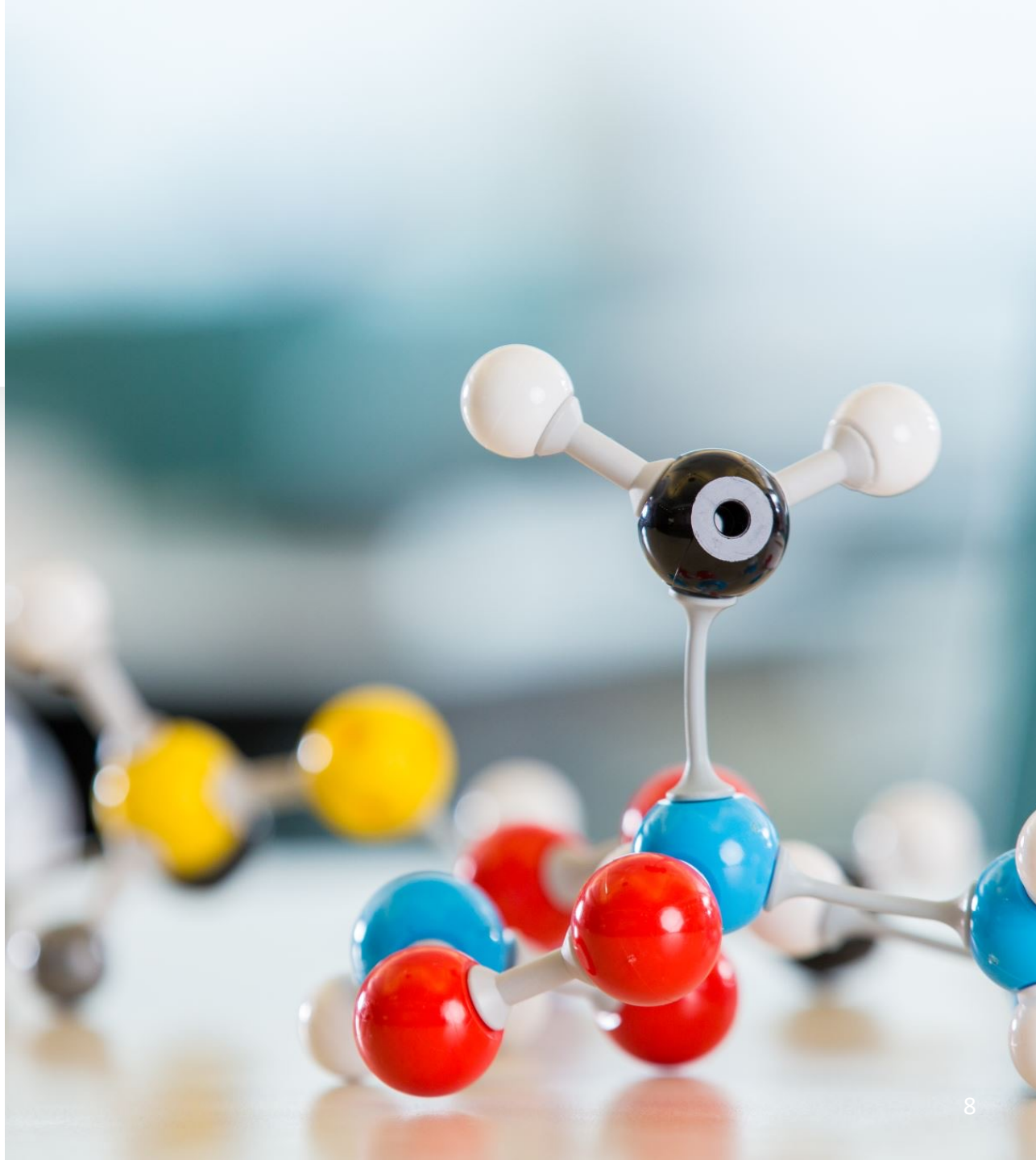
It is possible to rearrange the structure of the scene graph, for example, to move a subtree from one location in the scene graph to another.

The Node class

- The Node class defines a traditional computer graphics "local" coordinate system in which the x axis increases to the right and the y axis increases downwards.
- Any Node can have transformations applied to it. These include translation, rotation, scaling, or shearing.
  - A **translation** transformation is one which shifts the origin of the node's coordinate space along either the x or y axis; for example, applying a Translate with a shift of 10 along the x axis (0, 0) → (10, 0)
  - A **rotation** transformation is one which rotates the coordinate space of the node about **a specified "pivot" point**, causing the node to appear rotated; for example, apply a Rotate with a 90 degree rotation (**angle**=90) and a pivot at the origin (pivotX=0, pivotY=0).
  - A **scaling** transformation causes a node to either appear larger or smaller depending on the **scaling factor**; for example, apply a Scale with scale factors (x=2.0, y=2.0) and a pivot at the origin (pivotX=0, pivotY=0), the node will grow double in size
  - A **shearing** transformation, sometimes called **a skew**, effectively rotates one axis so that the x and y axes are no longer perpendicular.

JavaFX UI Control Classes Relevant to Project 4

- Class Image
- Class ImageView
- Enum SelectionMode
- Class ComboBox<T>
- Class ListView<T>
- Interface ObservableList<E>
- Interface Initializable



Class Image

- The Image class represents graphical images and is used for loading images from a specified URL.
  - file://<host>/<path>, OR
  - http://<host>:<port>/<path>?<searchpart>
  - Supported image formats are BMP, GIF, JPEG and PNG
- Images can be resized as they are loaded (for example to reduce the amount of memory consumed by the image). The application can specify the quality of filtering used when scaling, and whether to preserve the original image's aspect ratio.
- All URLs supported by URL can be passed to the constructor. If the passed string is not a valid URL, but a path instead, the Image is searched on the classpath in that case.
- Use ImageView for displaying images loaded with this class. The same Image instance can be displayed by multiple ImageViews.

Class Image

Import the JavaFX Package – javafx.scene.image.Image;

There are 6 constructors, below is one of them

- **Image(String url)** – constructs an Image with content loaded from the specified url.

For example, put your image files under the resource folder

```
//file in resource folder, background loading is true
Image image1 = new Image(“file:src/main/resources/com/example/project4/flower.png", true);
Image image2 = new Image("file:src/main/resources/com/example/project4/flower.png”);
  //file with a full path; try to avoid an absolute path
  Image image3 = new  Image(file:///Users/yourName/Documents/CS213/Project4/donuts.jpeg);
```

Class ImageView

- The ImageView is a Node used for painting images loaded with Image class.
- This class allows resizing the displayed image (with or without preserving the original aspect ratio) and specifying a viewport into the source image for restricting the pixels displayed by this ImageView.
- Reference the Javadoc for the properties of this class by following the link below
  - https://openjfx.io/javadoc/17/javafx.graphics/javafx/scene/image/ImageView.html

Class ImageView

- For example,

```
Image image =
    new Image("file:src/main/resources/com/example/project4/flower.png");
        ImageView imageview = new ImageView();
        imageview.setImage(image);  //display the image in the ImageView
```

- You can call the setter methods to set the properties of an ImageView, or set the properties in the Scene Builder

Image Button

- In SceneBuilder
  - Add a Button to the Scene Graph
  - Add an ImageView as the child node of the Button
  - set the Button property “content display”
- Without SceneBuilder

```
Button imageButton = new Button(“Button”);
ImageView imview = new ImageView("file:donut.bmp");
imageButton.setContentDisplay(ContentDisplay.TOP);
imageButton.setGraphic(imview);
```

Enum SelectionMode

- An enumeration used to specify how many items may be selected in a MultipleSelectionModel (a class in JavaFX)
- For example, specifying single or multiple selection mode in a ComboBox or ListView

Enum Constant Summary

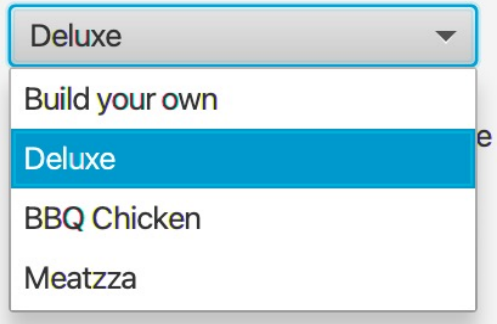
Enum Constants

Enum Constant	Description
`MULTIPLE`	Allows for one or more contiguous range of indices to be selected at a time.
`SINGLE`	Allows for only one item to be selected at a time.

Package javafx.scene.control

Class ComboBox<T>

java.lang.Object
- javafx.scene.Node
  - javafx.scene.Parent
    - javafx.scene.layout.Region
      - javafx.scene.control.Control
        - javafx.scene.control.ComboBoxBase<T>
          - javafx.scene.control.ComboBox<T>



CLASS COMBOBOX<T>

Class ComboBox<T>

- An implementation of the ComboBoxBase abstract class for the most common form of ComboBox, where a dropdown list is shown to users providing them with a choice that they may select from.
- On top of ComboBoxBase, the ComboBox class introduces additional API. Most importantly, it adds an items property that works in much the same way as the ListView items property. In other words, it is the content of the items list that is displayed to users when they click on the ComboBox button.

Class ComboBox<T>

The value property is not constrained to items contained within the items list - it can be anything as long as it is a valid value of type T.

By default, when the popup list is showing, the maximum number of rows visible is 10, but this can be changed by modifying the **visibleRowCount** property.

If the number of items in the ComboBox is less than the value of visibleRowCount, then the items size will be used instead so that the dropdown list is not exceedingly long.

It is possible to modify the selection model that is used, although this is likely to be rarely changed, because the ComboBox enforces the need for a SingleSelectionModel instance, and it is not likely that there is much need for alternate implementations.

- Example

```
ObservableList<String> items =
    FXCollections.observableArrayList("Red", "Green", "Blue");
ComboBox<String> comboBox = new ComboBox<>();
comboBox.setItems(items); //synchronize the list in comboBox and
items
```

- Another example

```
ComboBox<Color> cmb = new ComboBox<>()
cmb.getItems().addAll(
        Color.RED,
        Color.GREEN,
        Color.BLUE);
```

Class ComboBox<T>

Class ComboBox<T>

Other useful methods

```
//get the item selected by the user

String selected =

        combobox.getSelectionModel().getSelectedItem();

//set the default selected item

combobox.setValue(”rectangle");

//set the default selected item to be the first item of the list

combobox.getSelectionModel().select(0);
```

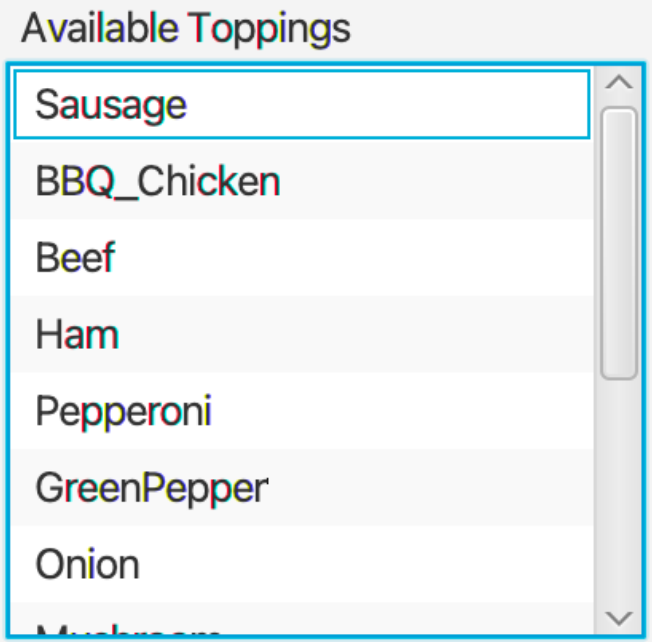
Class ListView<T>

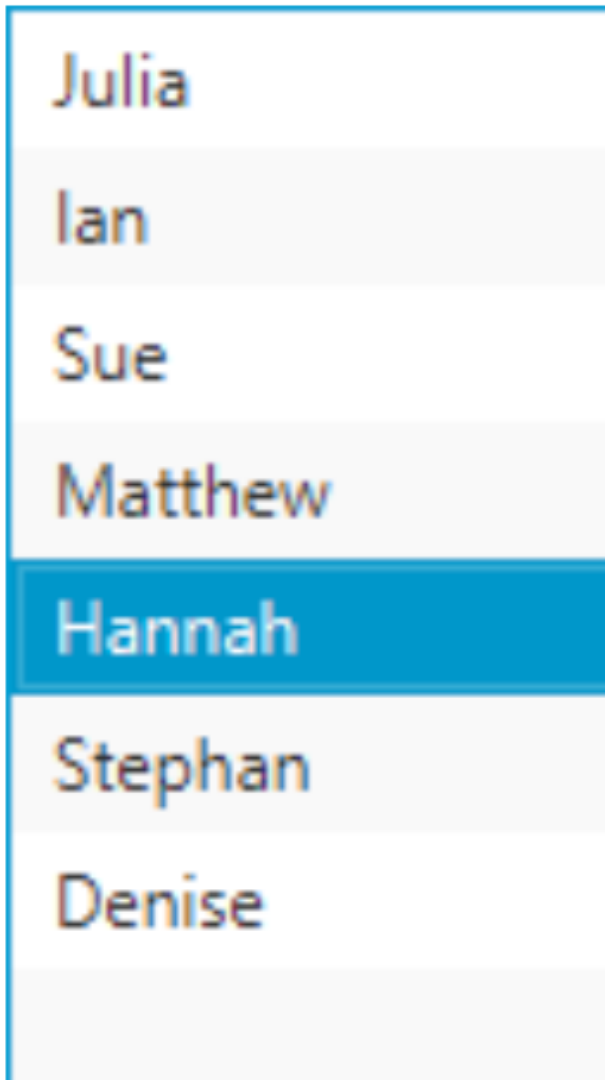
Module javafx.controls

Package javafx.scene.control

Class ListView<T>

java.lang.Object
- javafx.scene.Node
  - javafx.scene.Parent
    - javafx.scene.layout.Region
      - javafx.scene.control.Control
        - javafx.scene.control.ListView<T>





Class ListView<T>

- A ListView displays a horizontal or vertical list of items from which the user may select, or with which the user may interact.
- The ObservableList is automatically observed by the ListView, such that any changes that occur inside the ObservableList will be automatically shown in the ListView itself. For example

```
ObservableList<String> names =
    FXCollections.observableArrayList(
            "Julia", "Ian", "Sue", "Matthew", "Hannah",
            "Stephan", "Denise");

ListView<String> listView = new ListView<String>(names);

OR,

listView.setItems(names); //synchronize the listView and
names
```

Class ListView<T>

- ListView Selection
  - To track selection and focus, it is necessary to become familiar with the **SelectionModel** classes.
  - A ListView has at most one instance of this class
  - The default SelectionModel used when instantiating a ListView is an implementation of the MultipleSelectionModel abstract class; however, the selectionMode property, the **default value is SelectionMode.SINGLE.**
  - To enable multiple selection in a default ListView instance, you can do the following:
- listView.getSelectionModel().setSelectionMode(SelectionMode.MULTIPLE);

Class ListView<T>

- Other useful methods of class ListView<T>

```
//get the selected items if the item type is String
String item =
     listview.getSelectionModel().getSelectedItem();
//remove item from the listview object
listview.getItems().remove(item);
//add item to the listview object
listview.getItems().add(item);
```

- If the data source of the ListView object is an observable list, the selected item will also be removed from the observable list; that is, the ListView object and the ObserbableList object are synchronized

PACKAGE JAVAFX.COLLECTIONS

Hierarchy For Package javafx.collections

Package Hierarchies:
All Packages

Class Hierarchy

- java.lang.**Object**
  - java.util.**AbstractCollection**<E> (implements java.util.Collection<E>)
    - java.util.**AbstractList**<E> (implements java.util.List<E>)
      - javafx.collections.**ObservableListBase**<E> (implements javafx.collections.ObservableList<E>)
        - javafx.collections.**ModifiableObservableListBase**<E>
  - javafx.collections.**FXCollections**
  - javafx.collections.**ListChangeListener.Change**<E>
  - javafx.collections.**MapChangeListener.Change**<K,V>
  - javafx.collections.**ObservableArrayBase**<T> (implements javafx.collections.ObservableArray<T>)
  - javafx.collections.**SetChangeListener.Change**<E>
  - javafx.collections.**WeakListChangeListener**<E> (implements javafx.collections.ListChangeListener<E>, javafx.beans.WeakListener)
  - javafx.collections.**WeakMapChangeListener**<K,V> (implements javafx.collections.MapChangeListener<K,V>, javafx.beans.WeakListener)
  - javafx.collections.**WeakSetChangeListener**<E> (implements javafx.collections.SetChangeListener<E>, javafx.beans.WeakListener)

Class FXCollections

Utility class that consists of static methods that are 1:1 copies of java.util.Collections methods.

The wrapper methods (like synchronizedObservableList or emptyObservableList) has exactly the same functionality as the methods in Collections, **with exception that they return ObservableList** and are therefore suitable for methods that require ObservableList on input.

The utility methods are here mainly for performance reasons. All methods are optimized in a way that they yield only limited number of notifications.

PACKAGE JAVAFX.COLLECTIONS

Interface Hierarchy

- javafx.collections.**ArrayChangeListener**<T>
- java.lang.**Iterable**<T>
  - java.util.**Collection**<E>
    - java.util.**List**<E>
      - javafx.collections.**ObservableList**<E> (also extends javafx.beans.Observable)
    - java.util.**Set**<E>
      - javafx.collections.**ObservableSet**<E> (also extends javafx.beans.Observable)
- javafx.collections.**ListChangeListener**<E>
- java.util.**Map**<K,V>
  - javafx.collections.**ObservableMap**<K,V> (also extends javafx.beans.Observable)
- javafx.collections.**MapChangeListener**<K,V>
- javafx.beans.**Observable**
  - javafx.collections.**ObservableArray**<T>
    - javafx.collections.**ObservableFloatArray**
    - javafx.collections.**ObservableIntegerArray**
  - javafx.collections.**ObservableList**<E> (also extends java.util.List<E>)
  - javafx.collections.**ObservableMap**<K,V> (also extends java.util.Map<K,V>)
  - javafx.collections.**ObservableSet**<E> (also extends java.util.Set<E>)
- javafx.collections.**SetChangeListener**<E>

INTERFACE OBSERVABLELIST<E>

A list that allows listeners to track changes when they occur.

All Methods	Instance Methods	Abstract Methods	Default Methods

Modifier and Type	Method
boolean	**addAll**(**E**... elements)
void	**addListener**(**ListChangeListener**<? super **E**> listener)
void	**remove**(int from, int to)
boolean	**removeAll**(**E**... elements)
void	**removeListener**(**ListChangeListener**<? super **E**> listener)
boolean	**retainAll**(**E**... elements)
boolean	**setAll**(**E**... elements)
boolean	**setAll**(**Collection**<? extends **E**> col)

class FXCollections
https://openjfx.io/javadoc/17/javafx.base/javafx/collections/FXCollections.html

static <E> ObservableList<E>	observableArrayList()
static <E> ObservableList<E>	observableArrayList(E... items)
static <E> ObservableList<E>	observableArrayList(Collection<? extends E> col)
static <E> ObservableList<E>	observableArrayList(Callback<E,Observable[]> extractor)
static ObservableFloatArray	observableFloatArray()
static ObservableFloatArray	observableFloatArray(float... values)
static ObservableFloatArray	observableFloatArray(ObservableFloatArray array)
static <K,V> ObservableMap<K,V>	observableHashMap()
static ObservableIntegerArray	observableIntegerArray()
static ObservableIntegerArray	observableIntegerArray(int... values)
static ObservableIntegerArray	observableIntegerArray(ObservableIntegerArray array)
static <E> ObservableList<E>	observableList(List<E> list)
static <E> ObservableList<E>	observableList(List<E> list, Callback<E,Observable[]> extractor)
static <K,V> ObservableMap<K,V>	observableMap(Map<K,V> map)
static <E> ObservableSet<E>	observableSet(E... elements)
static <E> ObservableSet<E>	observableSet(Set<E> set)

Interface Initializable

Controller initialization interface.

NOTE, this interface has been superseded by automatic injection of location and resources properties into the controller.

FXMLLoader will now automatically call any suitably annotated no-arg initialize() method defined by the controller.

Interface Initializable

- Called to initialize a controller after its root element has been completely processed.

All Methods	Instance Methods	Abstract Methods
Modifier and Type	**Method**	
`void`	`initialize(URL location, ResourceBundle resources)`	

Initialize() method

```
public class Controller implements Initializable {
    ...
    /**
      location - used to resolve relative paths for the root object,
      or null if the location is not known.
      resources - used to localize the root object, or null if the root object
      was not localized.
    */
    public void initialize(URL location, ResourceBundle resource) {
        //any statements included here will be executed first when the controller is
       //first invoked
    }
        ...
}
//OR, simply have an initialize() method in the controller;
//this method will be performed when the controller is first invoked
public void initialize() { .... }
```

Sharing Data between Controllers

- In general, we would like to have one controller for each view.fxml file (GUI layout file)
- We need to share data among controllers in a software system
  - Call the methods in another controller or share the object references between controllers
  - For example, user login GUI controller passes the username and password to the other controller for the next step
- When the FXLLoader loaded the .fxml, the associated controller is also instantiated
- To get the references of the objects defined in the controller, you need to get the reference to the controller first
- Once you get the reference to the controller, you can call the public methods provided in that controller

Example – Sharing data between Controllers

```
//In MainController
FXMLLoader loader = new
  FXMLLoader(getClass().getResource("SecondView.fxml"));
      ...

//get the reference of the controller defined in the
//fxml file
SecondViewController secondViewController =
          loader.getController();
      ...
//pass the reference through a setter method in
//SecondViewController
//this is the reference to the MainController
secondViewController.setMainController(this);
```

```
//In SecondViewController
private MainController mainController;

public void setMainController(MainController controller)
{
   //get the reference of the MainController object
   //so you can call the public methods to
   //reference the instance variables
   //declared in MainController
       mainController = controller;
}
```