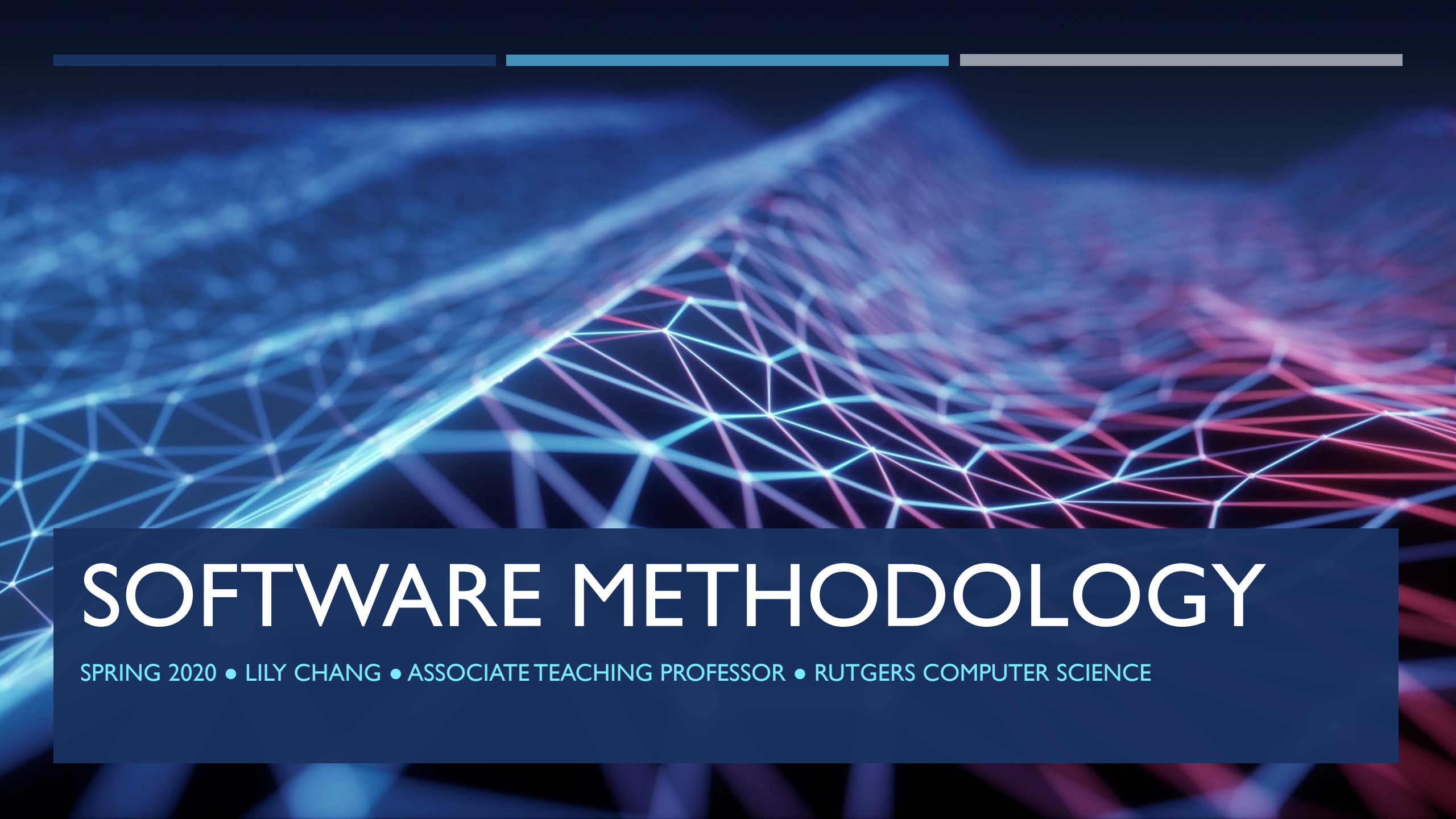


SOFTWARE METHODOLOGY

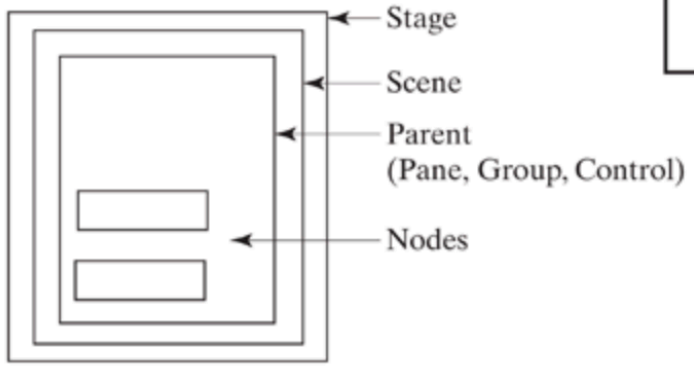
SPRING 2020 • LILY CHANG • ASSOCIATE TEACHING PROFESSOR • RUTGERS COMPUTER SCIENCE

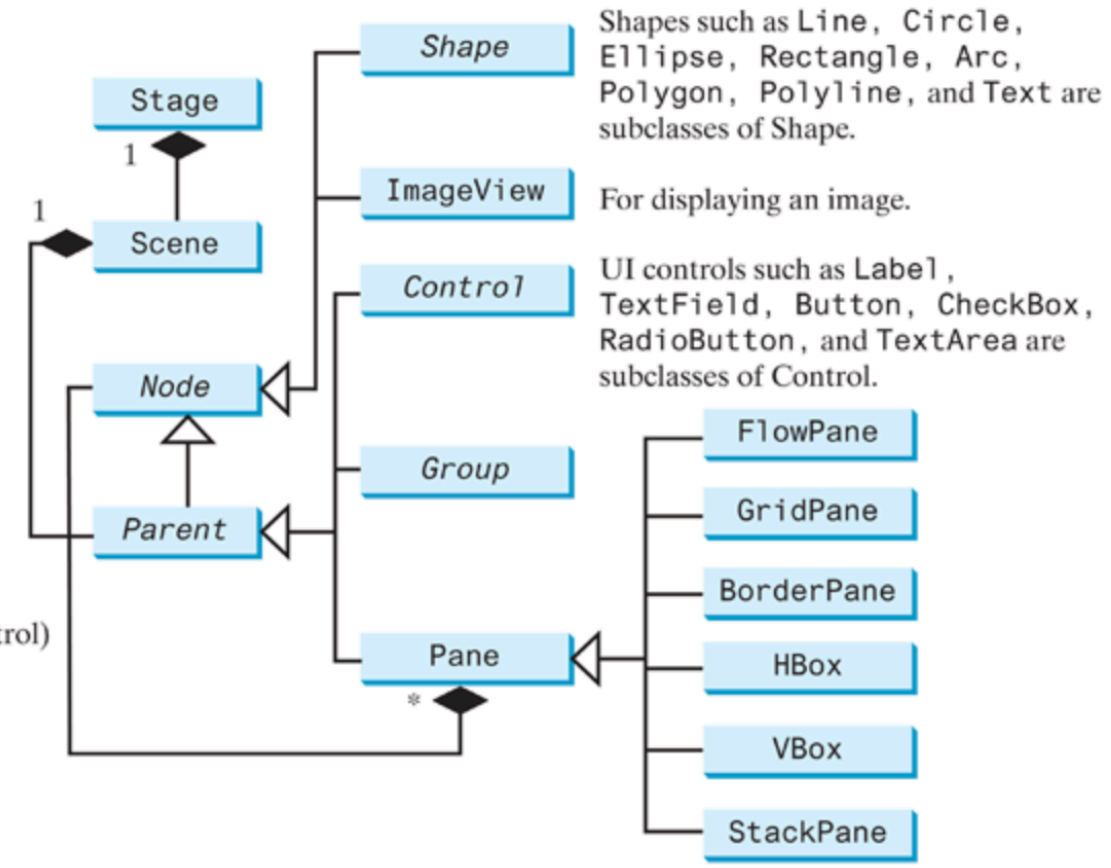
MORE ON JAVAFX CLASSES AND GUI COMPONENTS

JAVAFX CLASSES RELEVANT TO PROGRAM 4

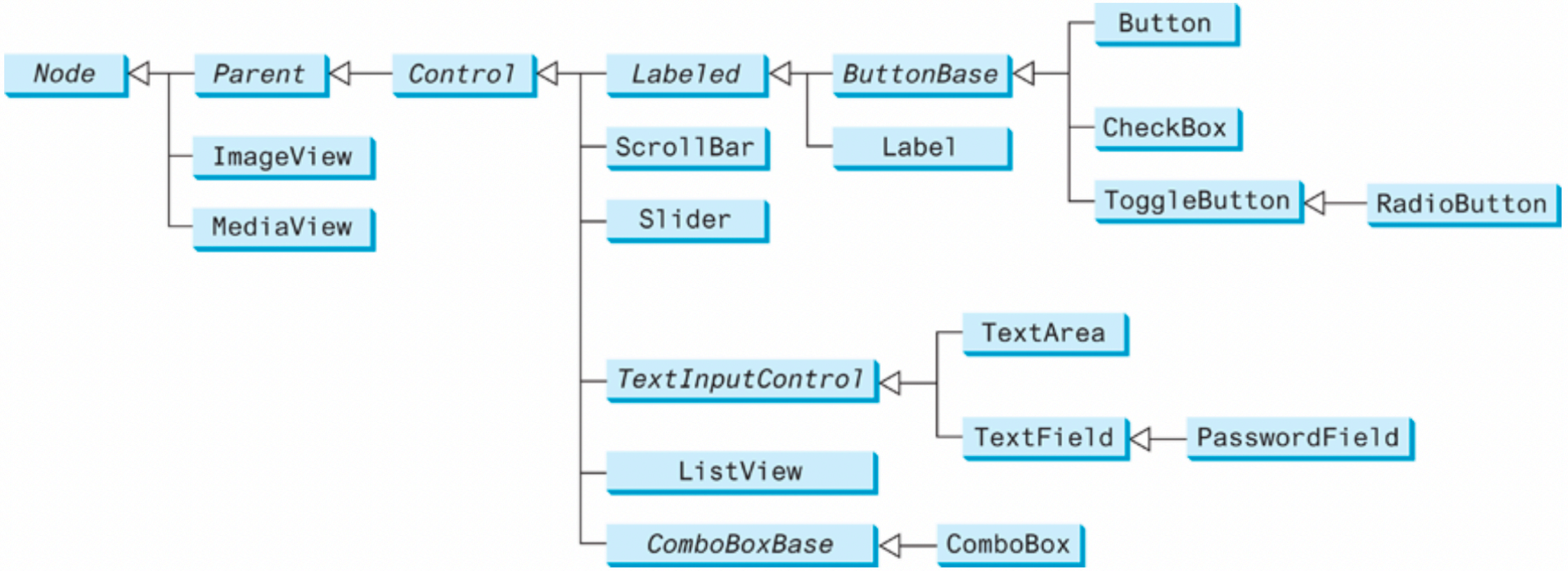
- Class Image
- Class ImageView
- Enum SelectionMode
- Class ComboBox<T>
- Class ListView<T>
- Interface ObservableList<E>
- Interface Initializable

JAVAFX GUI COMPONENTS REVISITED





JAVAFX UI CONTROLS REVISITED



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 or not 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,

```
Image image1 = new Image("/flower.png", true);  //file in classpath
Image image2 = new Image("file:flower.png”);    //file in project folder
Image image3 =
    new Image(file:///Users/MyName/Documents/eclipse/Program4/pizza.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/13/javafx.graphics/javafx/scene/image/ImageView.html

CLASS IMAGEVIEW

- Example,

```
Image image = new Image("flower.png");
ImageView iv1 = new ImageView();
iv1.setImage(image); //display the image as is on the ImageView
```

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

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

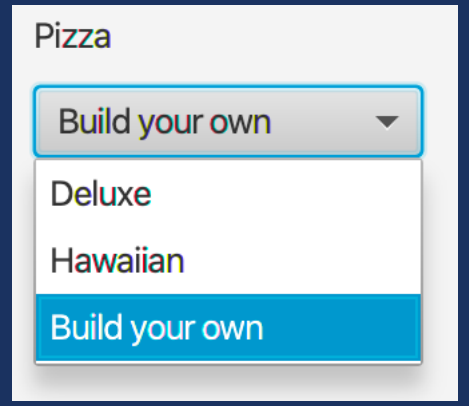
Enum Constant	Description
`MULTIPLE`	Allows for one or more contiguous range of indices to be selected at a time.
`SINGLE`	Allows for only one item to be selected at a time.

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 popup 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 popup 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.

CLASS COMBOBOX<T>

- Example

```
ObservableList<String> items =
        FXCollections.observableArrayList(“Red”, ”Green”, “Blue”);
ComboBox<String> comboBox = new ComboBox<>();
comboBox.setItems(items);
```

- Another example

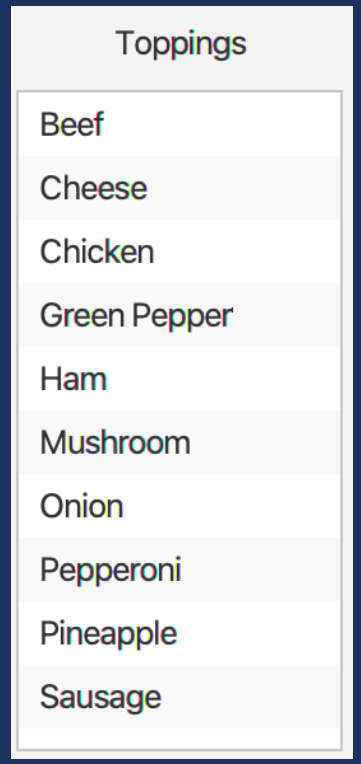
```
ComboBox<Color> cmb = new ComboBox<>();
cmb.getItems().addAll(
      Color.RED,
      Color.GREEN,
      Color.BLUE);
```

CLASS COMBOBOX<T>

- Other useful methods

```
String selected = combobox.getSelectionModel().getSelectedItem();
combobox.setValue(”rectangle"); //set the default selected item
```

CLASS LISTVIEW<T>



Module javafx.controls

Package javafx.scene.control

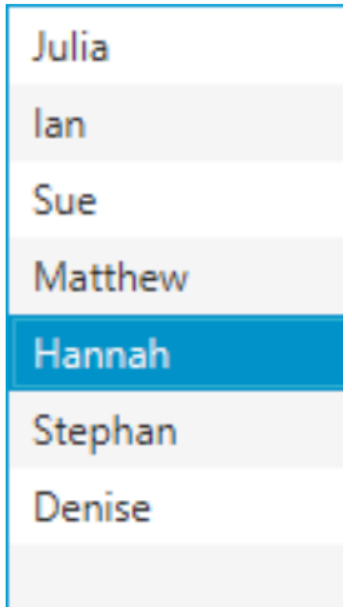
Class ListView<T>

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.
- Populating a ListView
  - 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);
```



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>

- Othr useful methods of class ListView<T>

```
ObservableList<String> myItems =
                        listview.getSelectionModel().getSelectedItems();
listview.getItems().remove(item); //remove item from the listview
listview.getItems().add(item);    //add item to the listview
```

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 ObservableLis**t 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

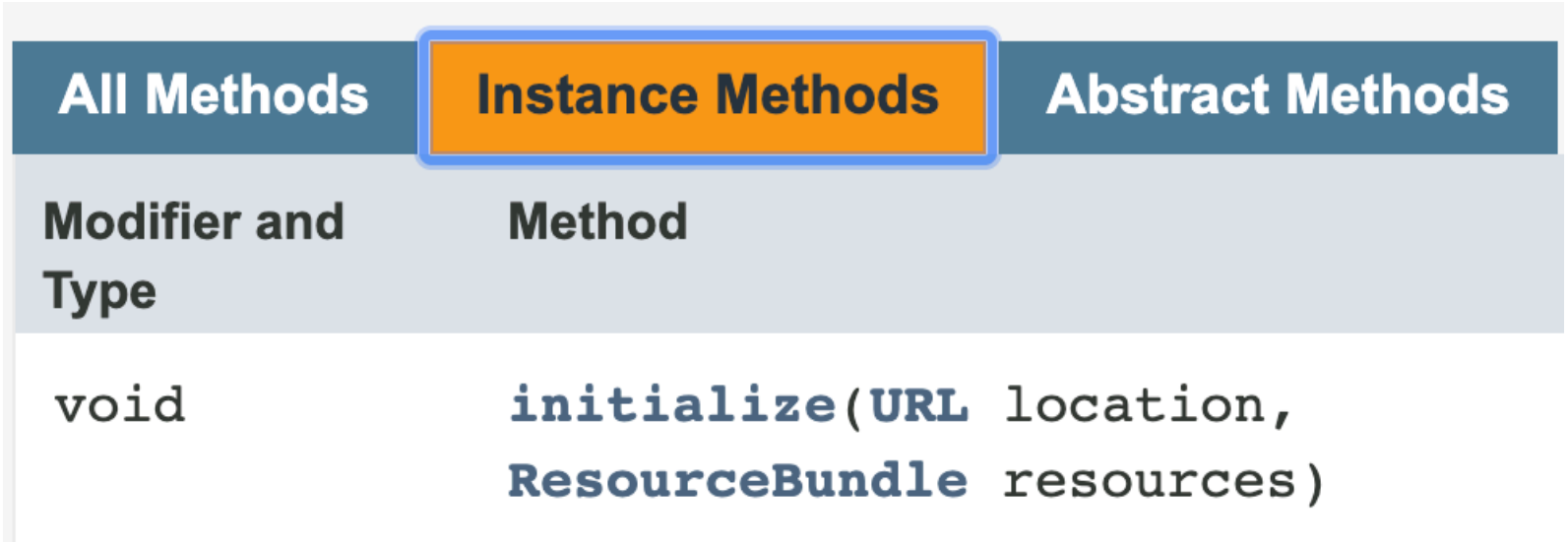
Modifier and Type	Method
boolean	`addAll(E... elements)`
void	`addListener(ListChangeListener<? super E> listener)`
void	`remove(int from, int to)`
boolean	`removeAll(E... elements)`
void	`removeListener(ListChangeListener<? super E> listener)`
boolean	`retainAll(E... elements)`
boolean	`setAll(E...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

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.
- It is recommended that the injection approach be used whenever possible.

INTERFACE INITIALIZABLE

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



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
    }
    ...
}
```

PASSING DATA BETWEEN CONTROLLERS

- In general, we would like to have one controller for each fxml file (GUI)
- Often times we need to share information among controllers in a software system
  - Call the methods in another controller or share the data between controllers
  - For example, login View and pass the username and password to the next view
- Each controller is a Java class (encapsulation), thus you need to
  - Create an instance of the controller class if you want to communicate with (reference) it
  - Get the reference of the controller you will be communicating (referencing)
  - Once you get the reference of the controller, you can call the methods in the controller

PASSING DATA BETWEEN CONTROLLERS

- For example,

```
//In View1Controller, which create a new stage and loads View2
FXMLLoader loader =
        new FXMLLoader(getClass().getResource("View2.fxml"));
View2Controller controller2 = loader.getController();
controller2.method1();
//could be a method passing "this" to a method in controller2
Controller2.method2();
...
```

```
//In View2Controller
...
View1Controller controller1 = refereceOfController1;
...
controller1.method1();
Controller1.method2();
...
```

THANK YOU