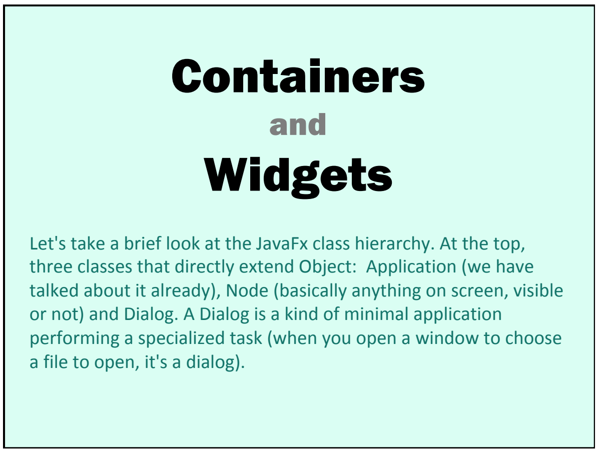
CS209

Computer system design and application

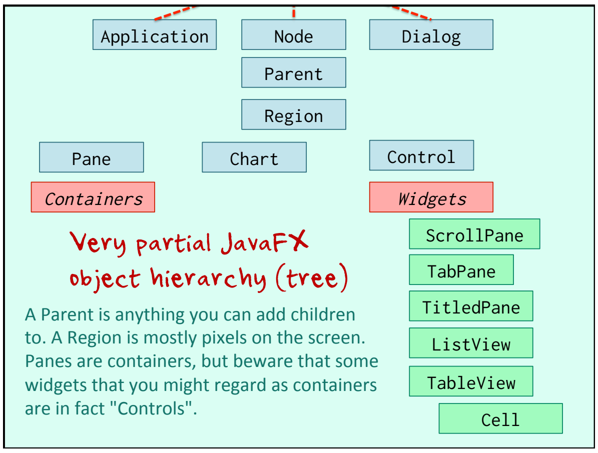
Stéphane Faroult
faroult@sustc.edu.cn

Zhao Yao zhaoy6@sustc.edu.cn



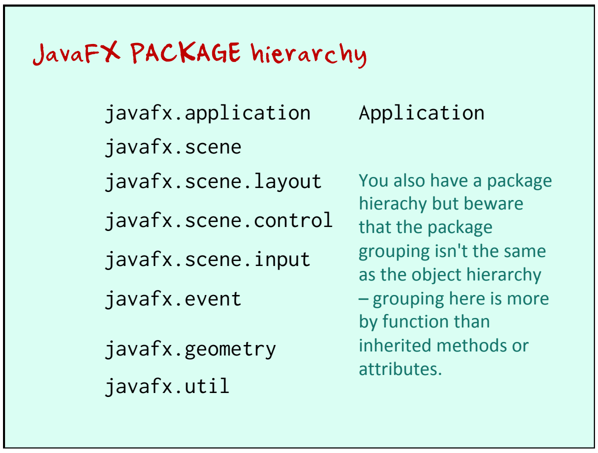
Containers and Widgets

Let's take a brief look at the JavaFx class hierarchy. At the top, three classes that directly extend Object: Application (we have talked about it already), Node (basically anything on screen, visible or not) and Dialog. A Dialog is a kind of minimal application performing a specialized task (when you open a window to choose a file to open, it's a dialog).



Very partial JavaFX object hierarchy (tree)

A Parent is anything you can add children to. A Region is mostly pixels on the screen. Panes are containers, but beware that some widgets that you might regard as containers are in fact "Controls".

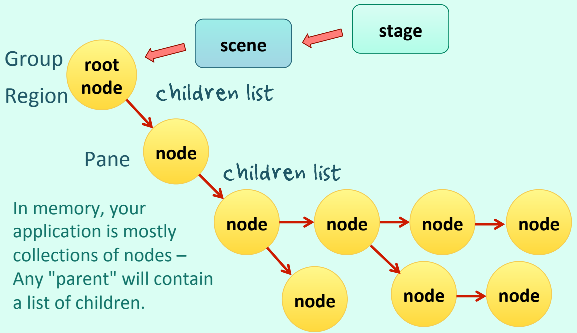


JavaFX PACKAGE hierarchy

```
javafx.application      Application
javafx.scene
javafx.scene.layout
javafx.scene.control
javafx.scene.input
javafx.event
javafx.geometry
javafx.util
```

You also have a package hierachy but beware that the package grouping isn't the same as the object hierarchy – grouping here is more by function than inherited methods or attributes.

In practice
based on a tree of NODES



In memory, your application is mostly collections of nodes – Any "parent" will contain a list of children.

Panes

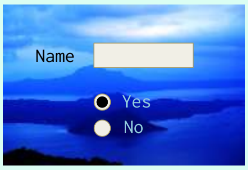
Pane BorderPane + boxes

GridPane

Special case

StackPane

Most often your main Window will be one of those. The StackPane allows to have elements on top of each other, which is mostly interesting for background images.


Name
○ Yes
○ No

Panes

More sophisticated types of panes may be added afterwards

AnchorPane

It's not uncommon to add more advanced Panes to a basic one.

ScrollPane

SplitPane

TabPane

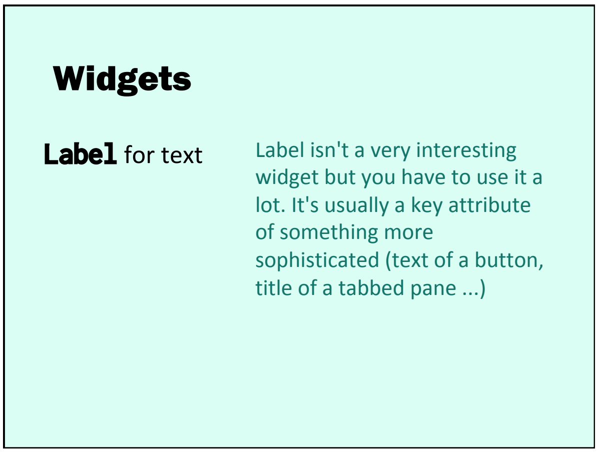
Controls

TitledPane → Accordion

In practice

```
public static void start(Stage stage) {
     stage.setTitle("Window Title");
     Group root = new Group();
     Scene scene = new Scene(root);
     BorderPane pane = new BorderPane();
     root.getChildren().add(pane);

     // Add containers and widgets to pane


     stage.setScene(scene);
     stage.show();
}
```

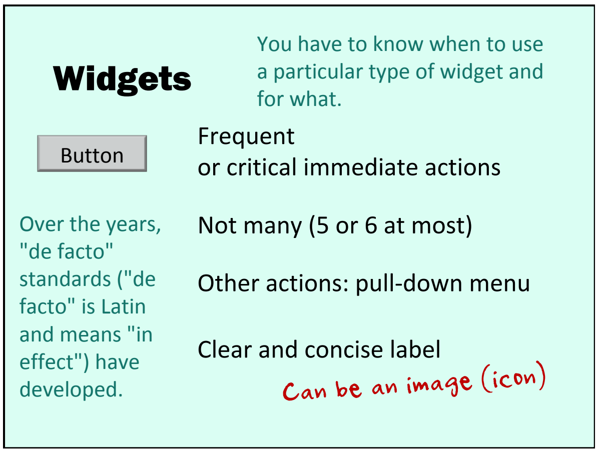
This can be seen as a basic start() method for a javafx program.



Widgets

Label for text

Label isn't a very interesting widget but you have to use it a lot. It's usually a key attribute of something more sophisticated (text of a button, title of a tabbed pane ...)



Widgets

You have to know when to use a particular type of widget and for what.

Button

Frequent
or critical immediate actions

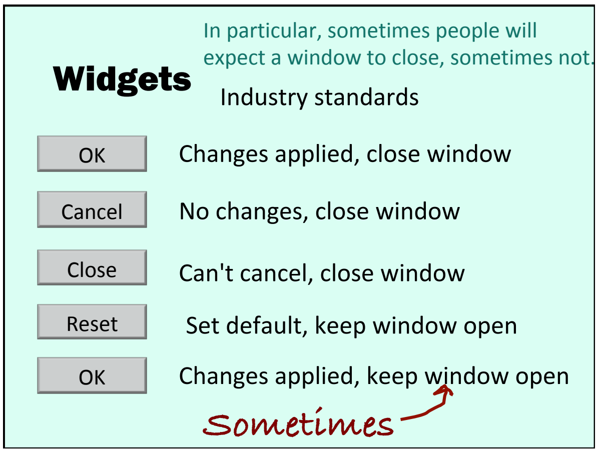
Over the years, "de facto" standards ("de facto" is Latin and means "in effect") have developed.

Not many (5 or 6 at most)

Other actions: pull-down menu

Clear and concise label

Can be an image (icon)



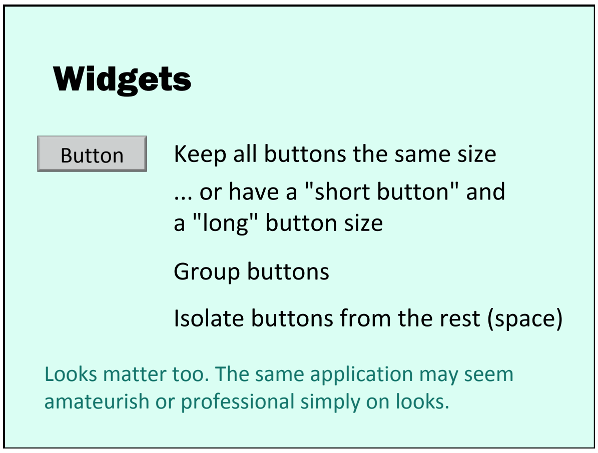
Widgets

In particular, sometimes people will expect a window to close, sometimes not.

Industry standards

OK	Changes applied, close window
Cancel	No changes, close window
Close	Can't cancel, close window
Reset	Set default, keep window open
OK	Changes applied, keep window open

Sometimes



Widgets

Button

Keep all buttons the same size

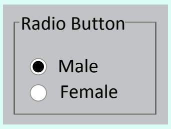
... or have a "short button" and a "long" button size

Group buttons

Isolate buttons from the rest (space)

Looks matter too. The same application may seem amateurish or professional simply on looks.

Widgets



One of several exclusive choices

Usually in a group

In a quiz a radio button will tell you that only one answer is correct ...

ToggleGroup object

`javafx.scene.control.ToggleGroup`

Set of on-off switches in which only one can be on.

ToggleGroup object

`javafx.scene.control.ToggleGroup`

Set of on-off switches in which only one can be on.

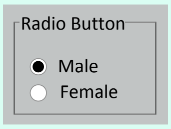
ToggleGroup object

`javafx.scene.control.ToggleGroup`

Set of on-off switches in which only one can be on.

```
ToggleGroup radioGroup = new ToggleGroup();
radioButton1.setToggleGroup(radioGroup);
radioButton2.setToggleGroup(radioGroup);
radioButton3.setToggleGroup(radioGroup);
```

Widgets



Radio Button

● Male

○ Female

One of several exclusive choices

Usually in a group

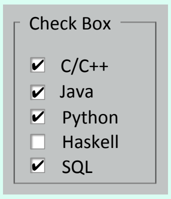
Use vertically

Six options or less

More than six options: ListBox

Avoid Yes/No or On/Off

Widgets



Check Box

☑ C/C++

☑ Java

☑ Python

□ Haskell

☑ SQL

More than several options allowed

Toggling (Yes/No, On/Off)

Use vertically

Ten options or less

Button for "select all"

Alternative: multiple-select ListBox

Widgets Data Entry

Your email

One line: TextField

No echo: PasswordField

Prompt text

Several lines: TextArea

Widgets Special-Purpose Widgets

ColorPicker

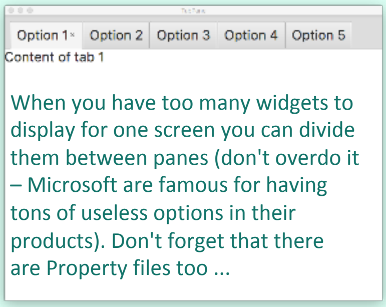
ColorPickers are mostly used for customizing settings (or for drawing applications)

DatePicker

DatePickers are common in business applications. They solve the "which date format should we use" problem.

We'll see other widgets as the need arises ...

Too many widgets



Content of tab 1

TabPane

When you have too many widgets to display for one screen you can divide them between panes (don't overdo it – Microsoft are famous for having tons of useless options in their products). Don't forget that there are Property files too ...

Too many widgets

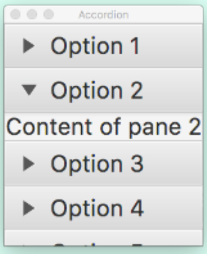
Container of Tabs

TabPane

```java
// Create a TabPane
TabPane pane = new TabPane();
pane.setPrefWidth(800);
pane.setPrefHeight(600);
root.getChildren().add(pane);
Tab tab;
// Create five tabs
for (int i = 1; i <= 5; i++) {
   tab = new Tab();
   tab.setText("Option " + i);
   tab.setContent(new Label("Content of tab " + i));
   pane.getTabs().add(tab);
}
```

Node = any container or widget

Too many widgets



Content of pane 2

Accordion and TitledPanes

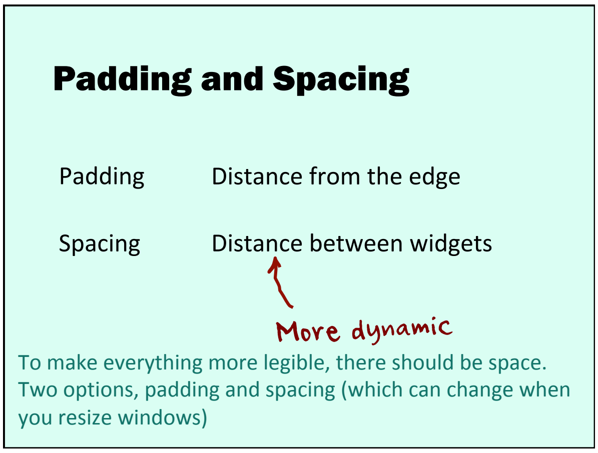
Titled panes are added to an Accordion. Scene Builder uses this.

An accordion is this musical instrument (also known as "the poor man's piano")

Too many widgets

Accordion and TitledPanes

```java
// Create an Accordion
Accordion accordion = new Accordion();
root.getChildren().add(accordion);
TitledPane pane;
// Create five titled panes
for (int i = 1; i <= 5; i++) {
  pane = new TitledPane();
  pane.setText("Option " + i);
  pane.setContent(new Label("Content of pane " + i));
  accordion.getPanes().add(pane);
}
```

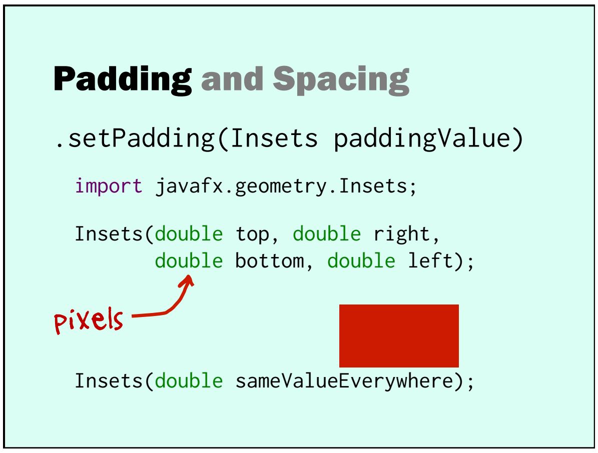


Padding and Spacing

Padding	Distance from the edge
Spacing	Distance between widgets

More dynamic

To make everything more legible, there should be space. Two options, padding and spacing (which can change when you resize windows)

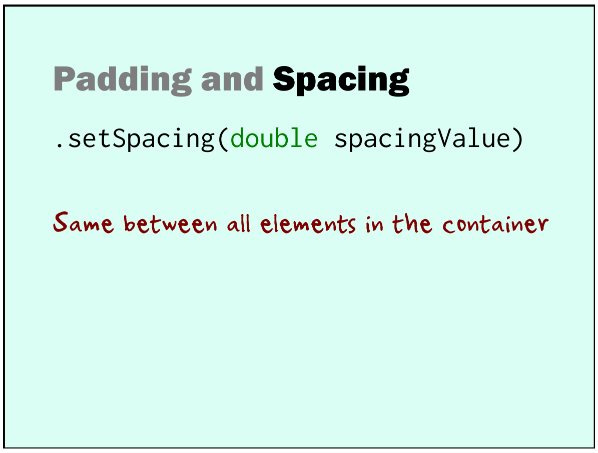


Padding and Spacing

```
.setPadding(Insets paddingValue)

  import javafx.geometry.Insets;

  Insets(double top, double right,
         double bottom, double left);
```

pixels

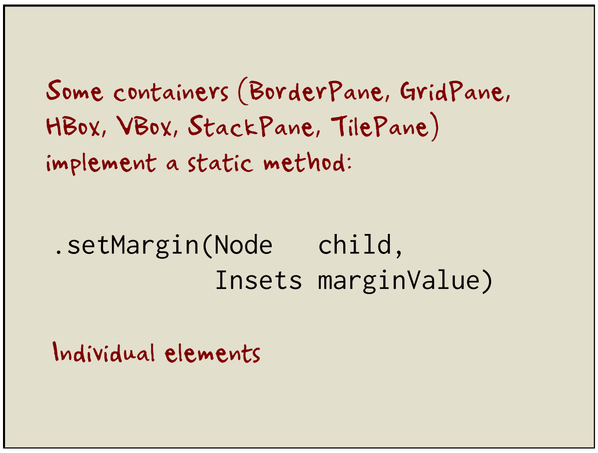
```
  Insets(double sameValueEverywhere);
```



Padding and Spacing

```
.setSpacing(double spacingValue)
```

Same between all elements in the container



Some containers (BorderPane, GridPane, HBox, VBox, StackPane, TilePane) implement a static method:

```
.setMargin(Node   child,
           Insets marginValue)
```

Individual elements

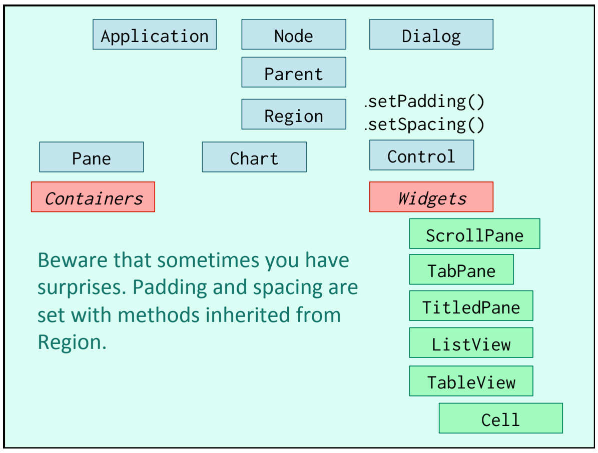
Padding and Spacing

```
.setSpacing(double spacingValue)
```

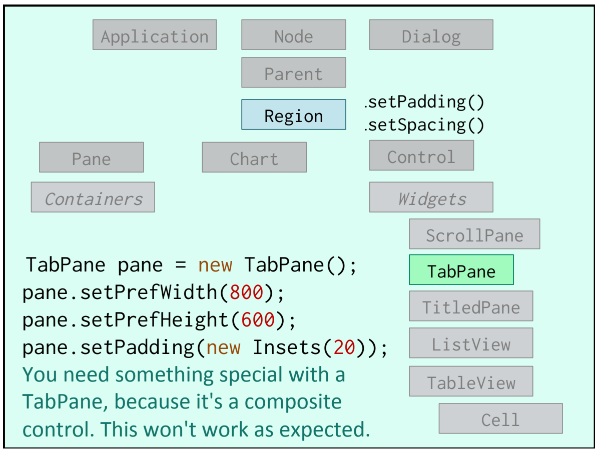
Same between all elements in the container

Used to compute initial size

When the window is first displayed, it may have a size you set, or the size may be computed. Of course, a lot of things will change in spacing if you broaden the window for instance.

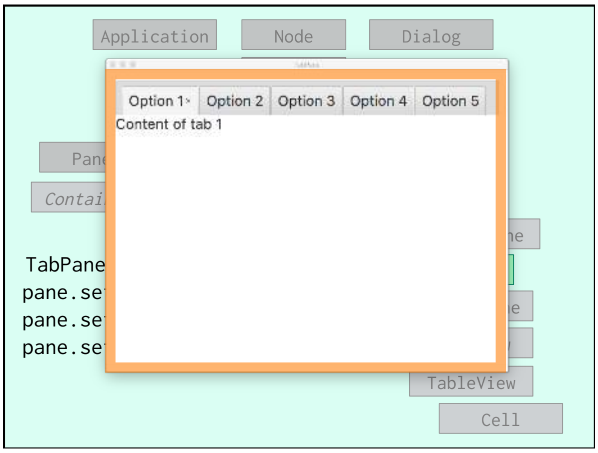


Beware that sometimes you have surprises. Padding and spacing are set with methods inherited from Region.



```
TabPane pane = new TabPane();
pane.setPrefWidth(800);
pane.setPrefHeight(600);
pane.setPadding(new Insets(20));
```

You need something special with a TabPane, because it's a composite control. This won't work as expected.



Content of tab 1

Solution:

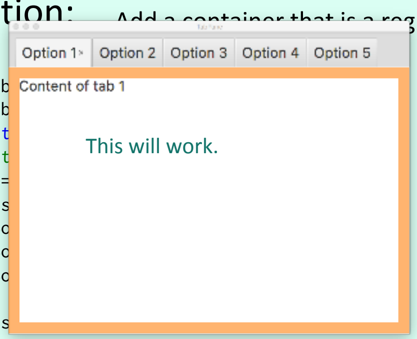
Add a container (region) to the tab, eg VBox

```
Tab  tab;
VBox tabBox;
// Create five tabs
for (int i = 1; i <= 5; i++) {
   tab = new Tab();
   tab.setText("Option " + i);
   tabBox = new VBox();
   tabBox.setPadding(new Insets(20));
   tabBox.getChildren()
         .add(new Label("Content of tab " + i));
   tab.setContent(tabBox);
   pane.getTabs().add(tab);
}
```

Solution:

Add a container that is a region

```
Tab  tab
VBox tab
// Creat
for (int
   tab =
   tab.s
   tabBo
   tabBo
   tabBo

   tab.s
   pane.getTabs().add(tab);
}
```



Content of tab 1

This will work.

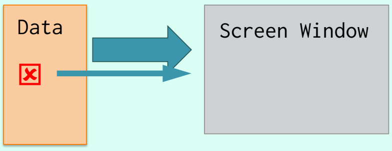
Widgets associated with data

Special collections

Widgets designed for displaying data are backed by special collections (which are regular collections wrapped into something special)

WHY?

Widgets associated with data



"Observable"

WHY?

The reason is that the interface must be able to "monitor" data and get an event when it changes.

Widgets associated with data

class FXCollections

One static method per collection in java.util.Collections

ArrayList

Wrapper

```
FXCollections.observableArrayList(arr)
```

Returns an ObservableList

Widgets associated with data

Lists and Combo Boxes

Table Views and Tree Views

It mostly concerns these widgets.

Widgets associated with data

Lists and Combo Boxes

```
javafx.scene.control.ListView<T>
```

ListView Object

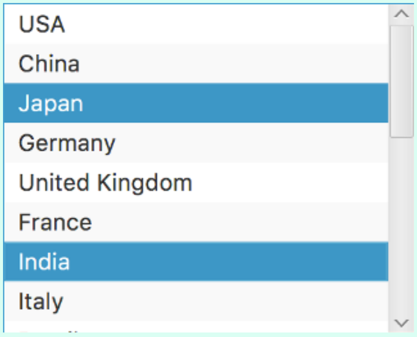
Scrollable
Can be editable

Associated with an ObservableList<T>

Explicit list of
items or Collection

```
ObservableList<T> choices =
                FXCollections.observableArrayList( );

ListView<T> list = new ListView<T>(choices);
```

Depending on options, a selection can be single or multiple.



```
list.getSelectionModel()
     .setSelectionMode(SelectionMode.MULTIPLE);
                           SelectionMode.SINGLE
```

Retrieving what was selected

```
ObservableList<T> selected = list.getSelectionModel()
                                          .getSelectedItems();
ListIterator<T> iter = selected.listIterator();
while (iter.hasNext()) {
    ...
}
```

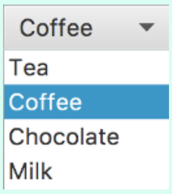
Usual collection handling to retrieve (in the handler possibly activated by a click on a button) what was selected.

ComboBox Object

ListView + TextField (editable) or Label

Single choice

This one is a variant allowing to insert new values in addition to those already known. You could overwrite "Coffee" with "Latte".



I'd use this to ask you about your high-school. However it can lead to multiple slightly different entries for the same thing.

ComboBox Object

ListView + TextField (editable) or Label

Single choice

Created like a ListView

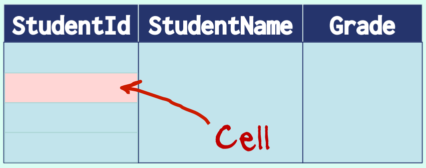
```
T selected = combobox.getValue();
```

Nothing special about using it inside the program.

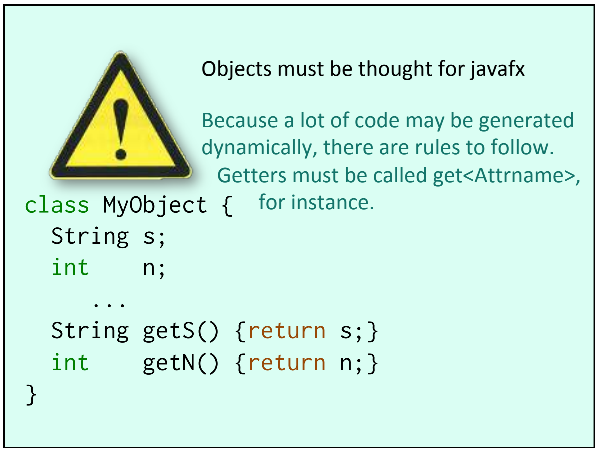
Widgets associated with data

Table Views and Tree Views

Data usually retrieved from a database



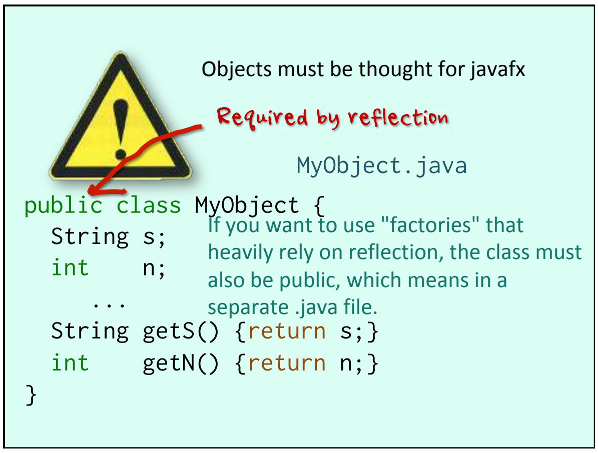
StudentId	StudentName	Grade



Objects must be thought for javafx

Because a lot of code may be generated dynamically, there are rules to follow. Getters must be called get<Attrname>, for instance.

```
class MyObject {
  String s;
  int    n;
     ...
  String getS() {return s;}
  int    getN() {return n;}
}
```



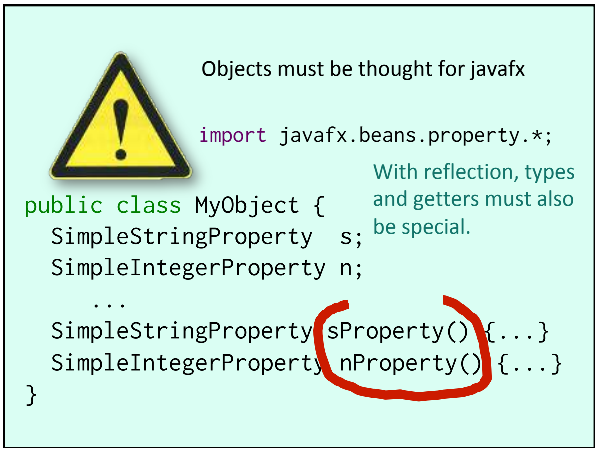
Objects must be thought for javafx

Required by reflection

MyObject.java

```
public class MyObject {
  String s;
  int    n;
     ...
  String getS() {return s;}
  int    getN() {return n;}
}
```

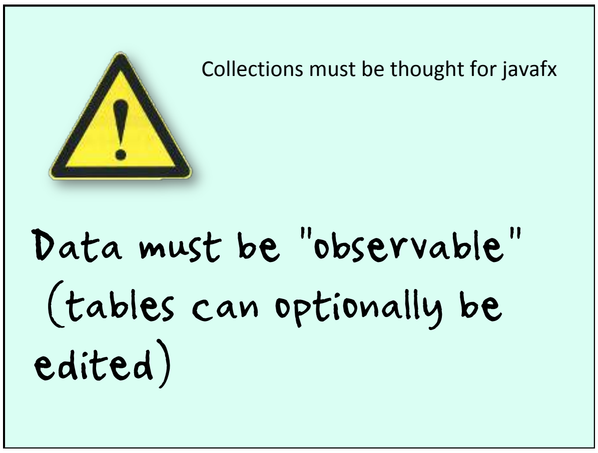
If you want to use "factories" that heavily rely on reflection, the class must also be public, which means in a separate .java file.



Objects must be thought for javafx

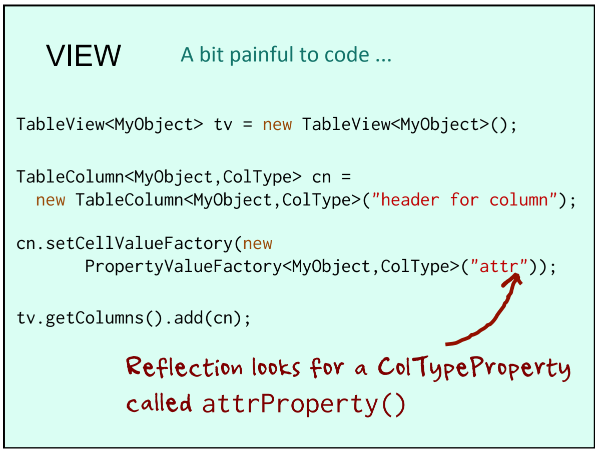
```
import javafx.beans.property.*;

public class MyObject {
  SimpleStringProperty  s;
  SimpleIntegerProperty n;
     ...
  SimpleStringProperty sProperty() {...}
  SimpleIntegerProperty nProperty() {...}
}
```

With reflection, types and getters must also be special.



Collections must be thought for javafx

Data must be "observable" (tables can optionally be edited)

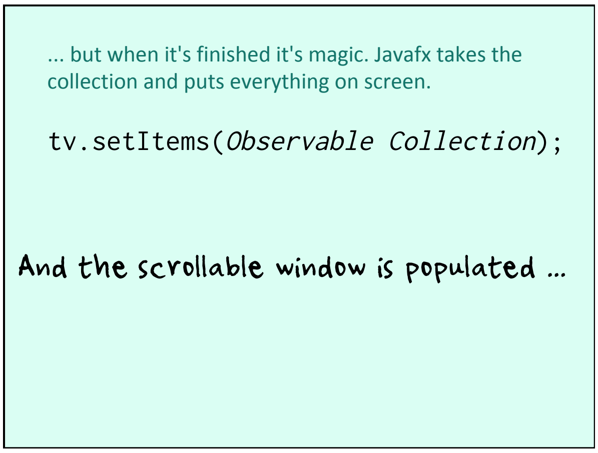


VIEW

A bit painful to code ...

```
TableView<MyObject> tv = new TableView<MyObject>();

TableColumn<MyObject,ColType> cn =
  new TableColumn<MyObject,ColType>("header for column");

cn.setCellValueFactory(new
       PropertyValueFactory<MyObject,ColType>("attr"));

tv.getColumns().add(cn);
```

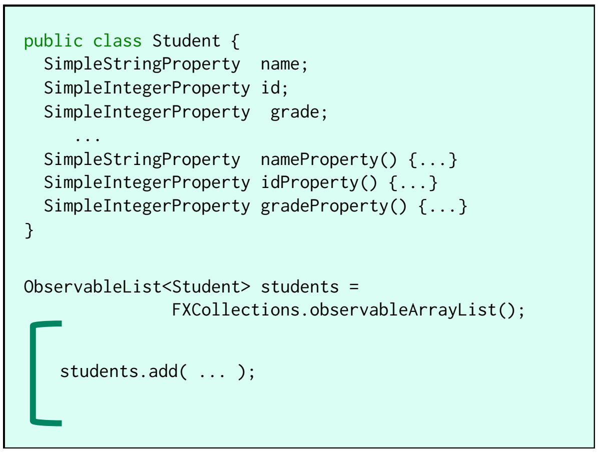
Reflection looks for a ColTypeProperty called `attrProperty()`



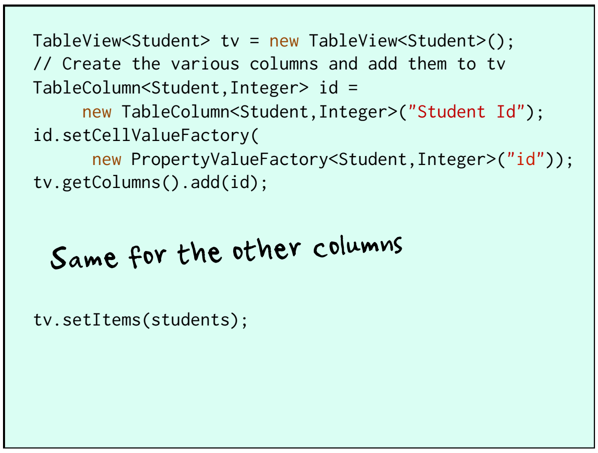
... but when it's finished it's magic. Javafx takes the collection and puts everything on screen.

```
tv.setItems(Observable Collection);
```

And the scrollable window is populated ...



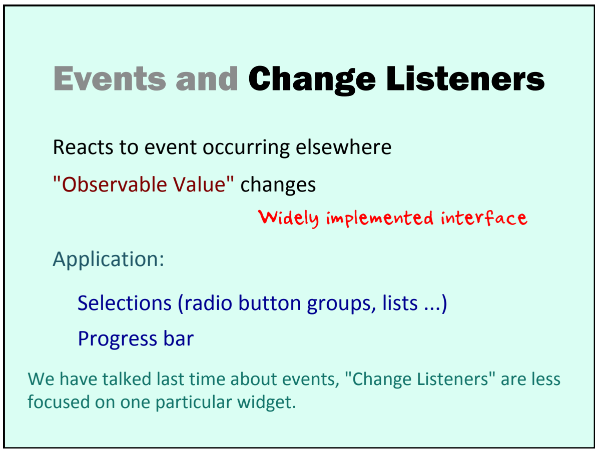
```
public class Student {
  SimpleStringProperty  name;
  SimpleIntegerProperty id;
  SimpleIntegerProperty  grade;
     ...
  SimpleStringProperty  nameProperty() {...}
  SimpleIntegerProperty idProperty() {...}
  SimpleIntegerProperty gradeProperty() {...}
}

ObservableList<Student> students =
               FXCollections.observableArrayList();

    students.add( ... );
```



```
TableView<Student> tv = new TableView<Student>();
// Create the various columns and add them to tv
TableColumn<Student,Integer> id =
     new TableColumn<Student,Integer>("Student Id");
id.setCellValueFactory(
      new PropertyValueFactory<Student,Integer>("id"));
tv.getColumns().add(id);
```

Same for the other columns

```
tv.setItems(students);
```



Events and **Change Listeners**

Reacts to event occurring elsewhere

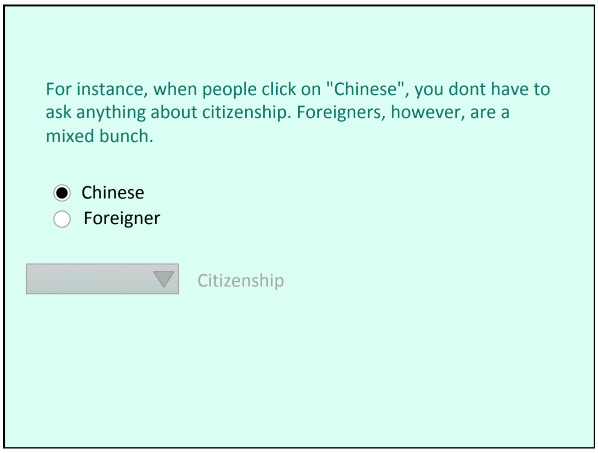
"Observable Value" changes

Widely implemented interface

Application:

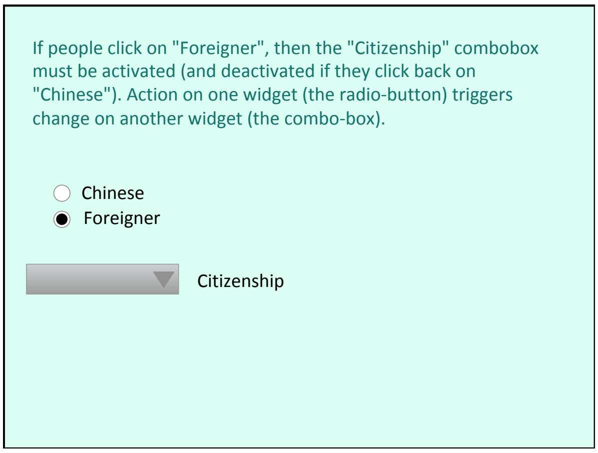
- Selections (radio button groups, lists ...)
- Progress bar

We have talked last time about events, "Change Listeners" are less focused on one particular widget.



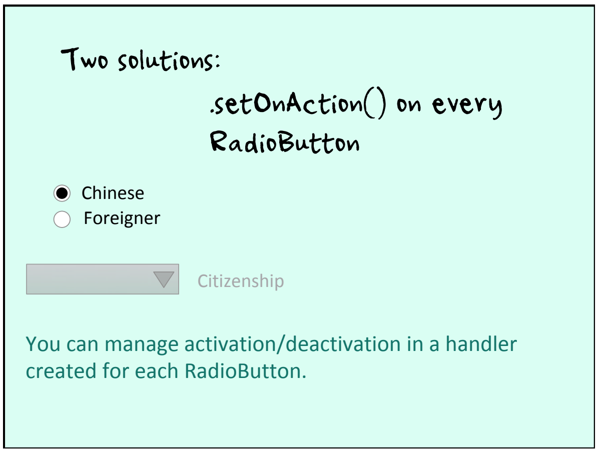
For instance, when people click on "Chinese", you dont have to ask anything about citizenship. Foreigners, however, are a mixed bunch.

● Chinese
○ Foreigner



If people click on "Foreigner", then the "Citizenship" combobox must be activated (and deactivated if they click back on "Chinese"). Action on one widget (the radio-button) triggers change on another widget (the combo-box).

○ Chinese
● Foreigner

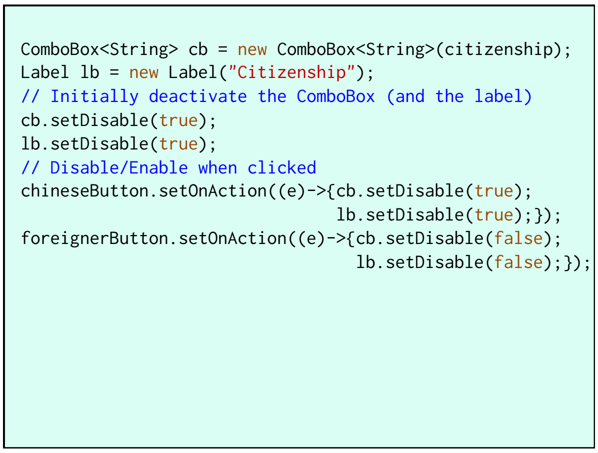


Two solutions:

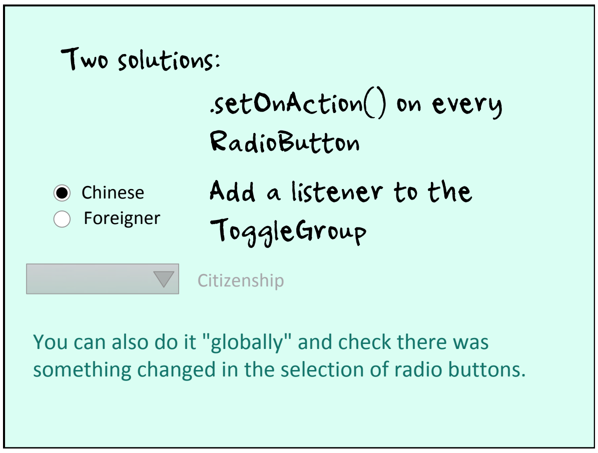
.setOnAction() on every RadioButton

● Chinese
○ Foreigner

You can manage activation/deactivation in a handler created for each RadioButton.



```
ComboBox<String> cb = new ComboBox<String>(citizenship);
Label lb = new Label("Citizenship");
// Initially deactivate the ComboBox (and the label)
cb.setDisable(true);
lb.setDisable(true);
// Disable/Enable when clicked
chineseButton.setOnAction((e)->{cb.setDisable(true);
                                lb.setDisable(true);});
foreignerButton.setOnAction((e)->{cb.setDisable(false);
                                  lb.setDisable(false);});
```



Two solutions:

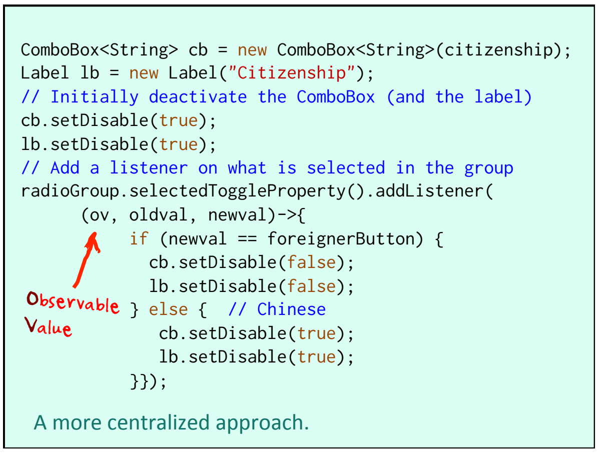
● Chinese
○ Foreigner

.setOnAction() on every RadioButton

Add a listener to the ToggleGroup

Citizenship

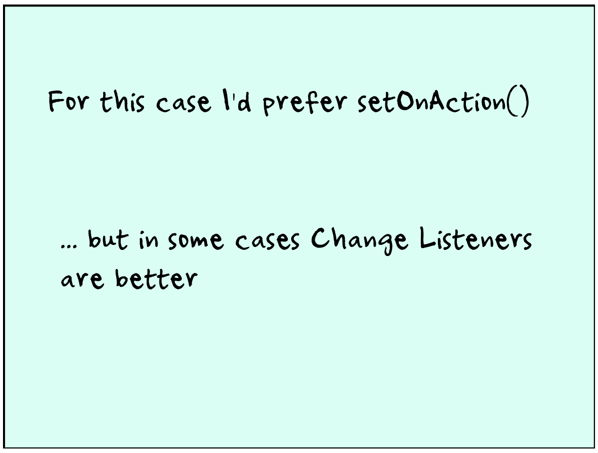
You can also do it "globally" and check there was something changed in the selection of radio buttons.



```
ComboBox<String> cb = new ComboBox<String>(citizenship);
Label lb = new Label("Citizenship");
// Initially deactivate the ComboBox (and the label)
cb.setDisable(true);
lb.setDisable(true);
// Add a listener on what is selected in the group
radioGroup.selectedToggleProperty().addListener(
      (ov, oldval, newval)->{
           if (newval == foreignerButton) {
             cb.setDisable(false);
             lb.setDisable(false);
           } else {  // Chinese
              cb.setDisable(true);
              lb.setDisable(true);
           }});
```

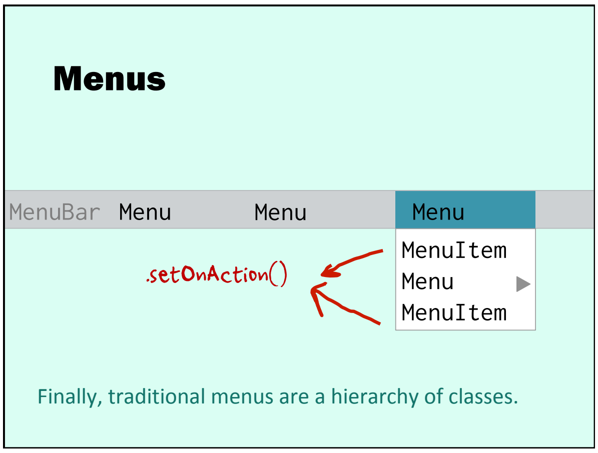
Observable Value

A more centralized approach.



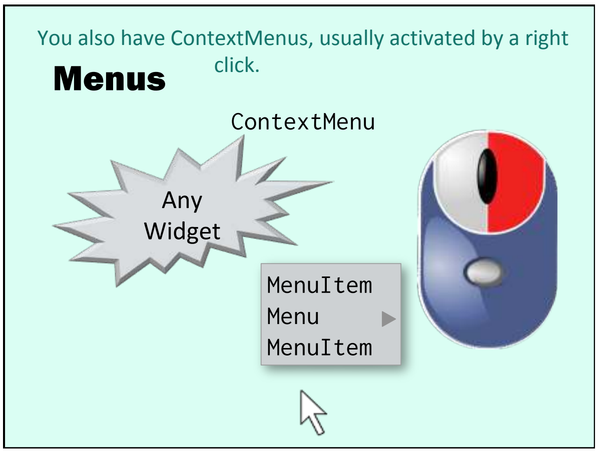
For this case I'd prefer setOnAction()

... but in some cases Change Listeners are better



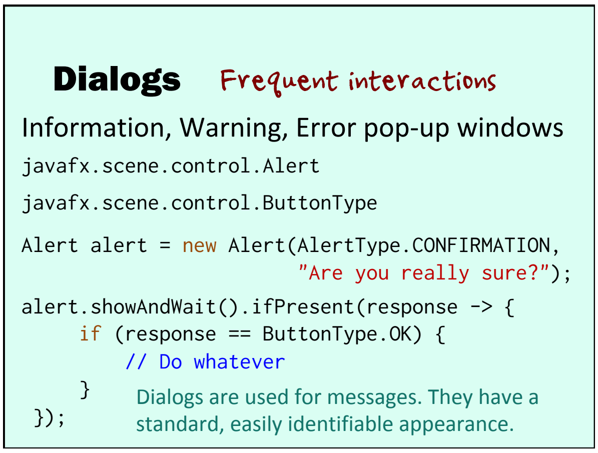
Menus

Finally, traditional menus are a hierarchy of classes.



You also have ContextMenus, usually activated by a right click.

Menus

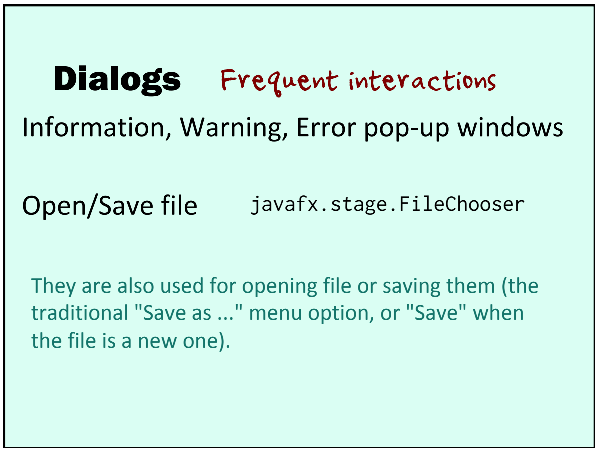


Dialogs *Frequent interactions*

Information, Warning, Error pop-up windows

```
javafx.scene.control.Alert
javafx.scene.control.ButtonType
Alert alert = new Alert(AlertType.CONFIRMATION,
                        "Are you really sure?");
alert.showAndWait().ifPresent(response -> {
     if (response == ButtonType.OK) {
         // Do whatever
     }
 });
```

Dialogs are used for messages. They have a standard, easily identifiable appearance.



Dialogs *Frequent interactions*

Information, Warning, Error pop-up windows

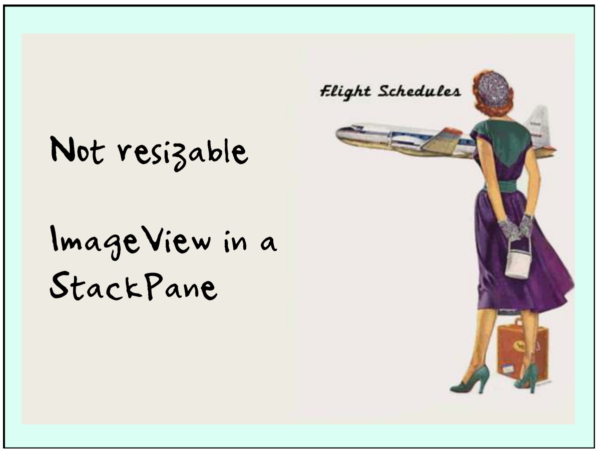
Open/Save file `javafx.stage.FileChooser`

They are also used for opening file or saving them (the traditional "Save as ..." menu option, or "Save" when the file is a new one).

Many other features
but it's a start ...

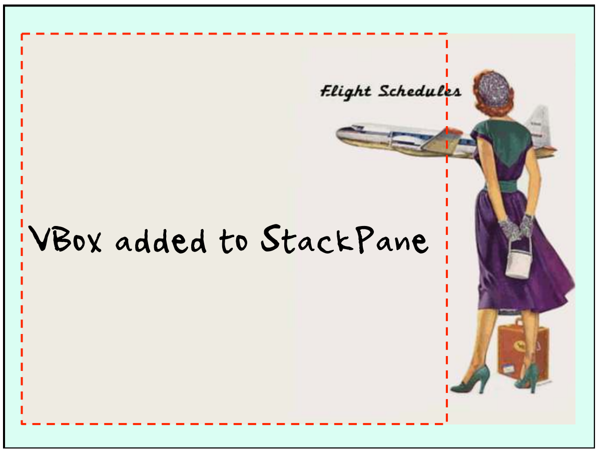
A case study

To give you a feeling of what you can do, this is a demo application I have written that searches flights in a database file with around 75,000 flights between around 100 of the busiest airports in the world.

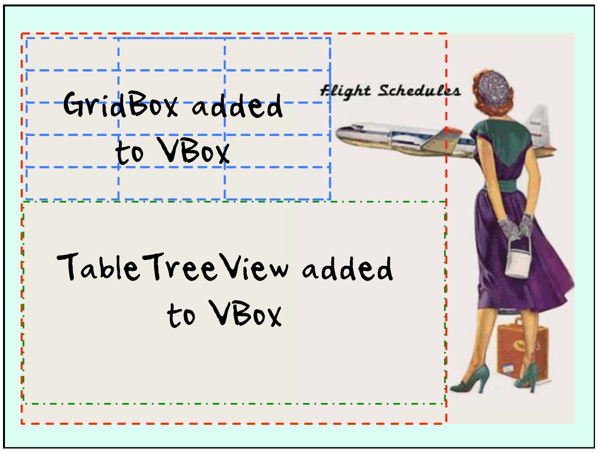


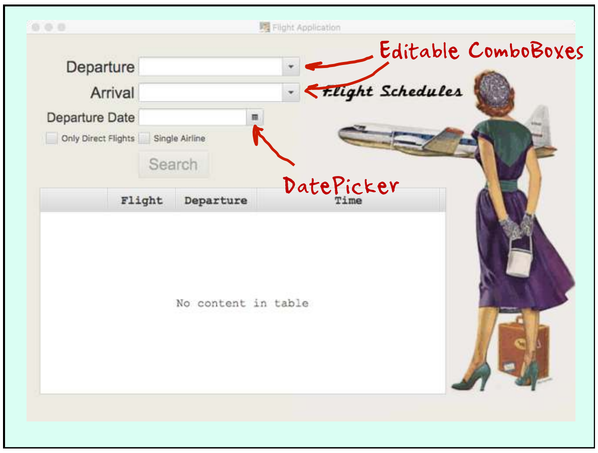
Not resizable

ImageView in a StackPane



VBox added to StackPane

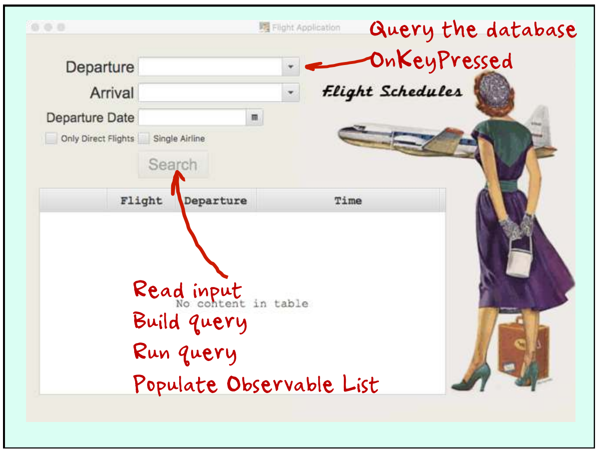




Departure
Arrival
Departure Date
□ Only Direct Flights □ Single Airline

Flight	Departure	Time

No content in table

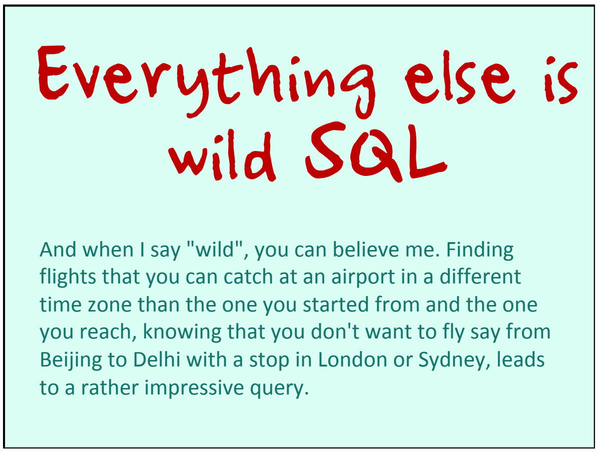


Departure
Arrival
Departure Date
□ Only Direct Flights □ Single Airline

Flight	Departure	Time

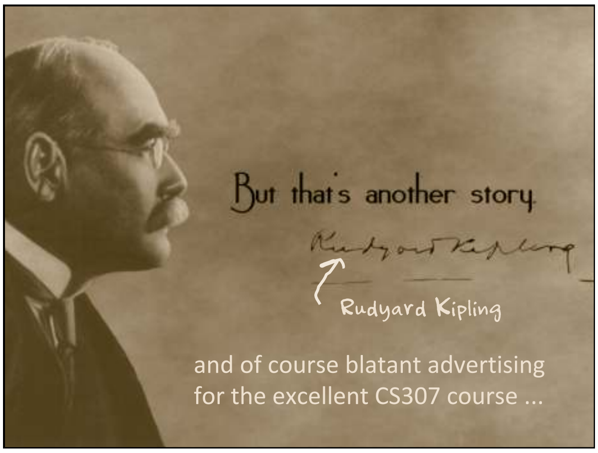
No content in table

Read input
Build query
Run query
Populate Observable List



Everything else is wild SQL

And when I say "wild", you can believe me. Finding flights that you can catch at an airport in a different time zone than the one you started from and the one you reach, knowing that you don't want to fly say from Beijing to Delhi with a stop in London or Sydney, leads to a rather impressive query.



But that's another story

Rudyard Kipling

Rudyard Kipling

and of course blatant advertising for the excellent CS307 course ...

Widgets associated with data

Lists and Combo Boxes

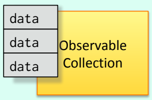
Table Views and Tree Views

You have just seen in the demo application Combo boxes (airport selection) and a TreeView (very like a TableView except that a row can be a child of another row). Now that we have seen them in action, let's come back to how they are coded.

Widgets associated with data

Lists and Combo Boxes

Table Views and Tree Views

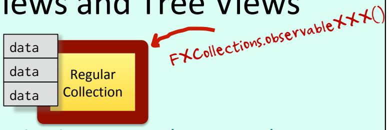


In all cases, we have seen it already, what is on the screen is backed by an "Observable Collection", which is just a Collection that supports change listeners, which allows refreshing the screen when data changes in the collection.

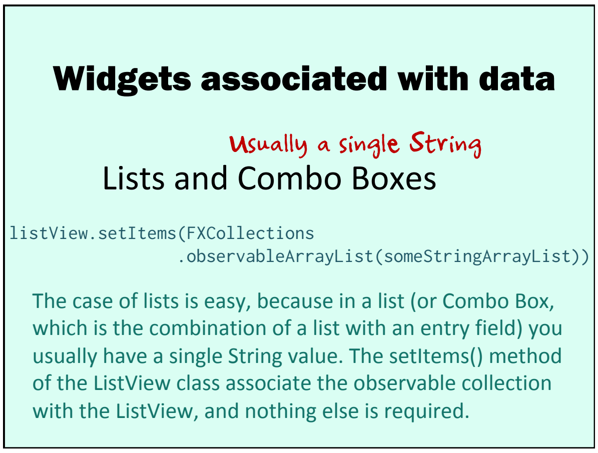
Widgets associated with data

Lists and Combo Boxes

Table Views and Tree Views



An observable collection is no more than a regular collection wrapped into the suitable call to a static FXCollections method.



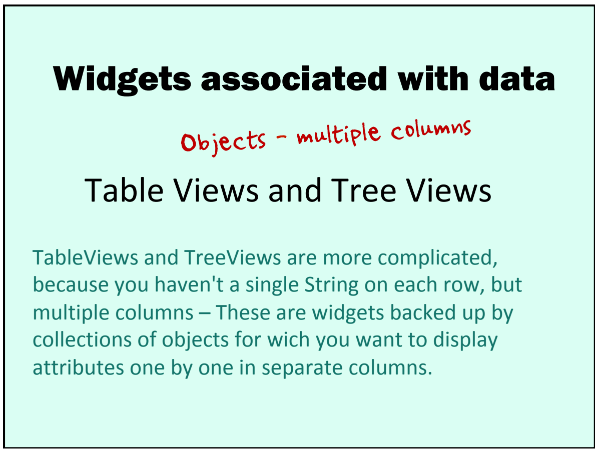
Widgets associated with data

Usually a single String

Lists and Combo Boxes

```
listView.setItems(FXCollections
                  .observableArrayList(someStringArrayList))
```

The case of lists is easy, because in a list (or Combo Box, which is the combination of a list with an entry field) you usually have a single String value. The setItems() method of the ListView class associate the observable collection with the ListView, and nothing else is required.



Widgets associated with data

Objects – multiple columns

Table Views and Tree Views

TableViews and TreeViews are more complicated, because you haven't a single String on each row, but multiple columns – These are widgets backed up by collections of objects for wich you want to display attributes one by one in separate columns.