CS209

Computer system design and application

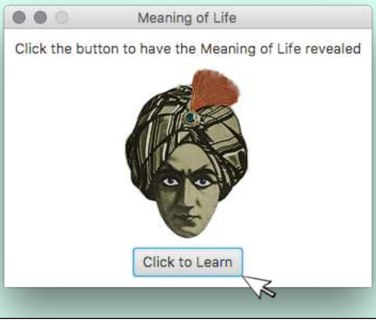
Stéphane Faroult
faroult@sustc.edu.cn

Zhao Yao	zhaoy6@sustc.edu.cn
Liu Zijian	liuzijian47@163.com
Li Guansong	intofor@163.com

MidTerm Exam : Nov 16th

Review Session Nov 8th

For once, the code that was shown during the lecture will not be given here – you'll rewrite it yourself. Just a short reminder of important points.



The Scene class has a `getStylesheets()` method to retrieve a list of stylesheets, to which you may add a new one.

Cascading Style Sheet

You can load in javafx a CSS file, which is a technique borrowed from the web. If the file is missing, a simple warning will appear on the console. "Cascade" means waterfall in French and means that you can have several style sheets taken into account one after each other and overriding parts of the previous one (yours overrides parts of the default one)

```
.root {
      -fx-font-size: 28pt;
}
```

Entries look like this. The -fx- prefix is specific to javafx.

You may use reflection for finding the location (unless it's in a package)

name of my class

"Reflection"

Problem: location?

```
private String directory = Soothsayerfx.class
                           .getProtectionDomain()
                           .getCodeSource()
                           .getLocation()
                           .toString();
scene.getStylesheets().add(directory + "soothsayer.css");
```

No explicit test

No explicit loop

Just events

What is important in a graphical application is that you just declare everything, and there is no procedural logic (if ... and loops) outside event handlers.

Coding forms isn't exactly thrilling.

You can create your widgets by hand, instantiating widgets objects one by one.

You can also use tools to create an XML (called FXML here) file describing widgets and containers. It will be loaded by JavaFx and graphical objects will be created from this static description.

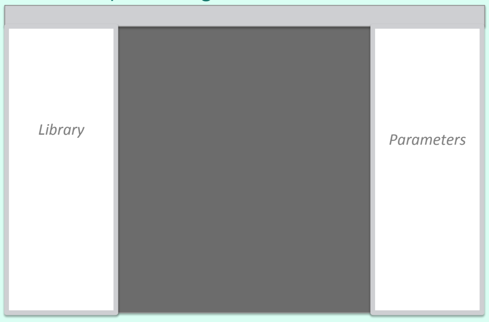
Interface Design Tools

Scene Builder

There are several such tools available, here is one in particular.

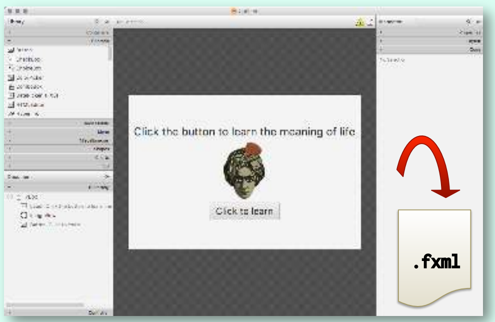
Interface Design Tools

With Scene Builder, you drag widgets from the Library into the middle part and set parameters (spacing, centering, handler to call ...) on the right.



Interface Design Tools

When your interface is designed, you save it to a .fxml file.



Interface Design Tools

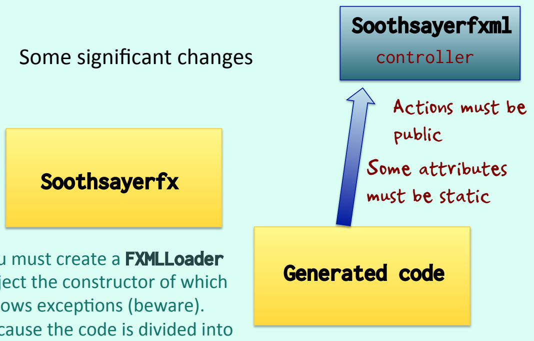
It looks like this (it's text)

```
<?xml version="1.0" encoding="UTF-8"?>

<?import javafx.geometry.Insets?>
<?import javafx.scene.control.Button?>
<?import javafx.scene.control.Label?>
<?import javafx.scene.image.Image?>
<?import javafx.scene.image.ImageView?>
<?import javafx.scene.layout.VBox?>
<?import javafx.scene.text.Font?>


<VBox alignment="CENTER" maxHeight="-Infinity" maxWidth="-
Infinity" minHeight="-Infinity" minWidth="-Infinity"
prefHeight="400.0" prefWidth="600.0" spacing="8.0"
xmlns="http://javafx.com/javafx/8.0.111" xmlns:fx="http://
javafx.com/fxml/1" fx:controller="Soothsayerfxml">
```

these "imports" will no longer appear in the .java file

Interface Design Tools

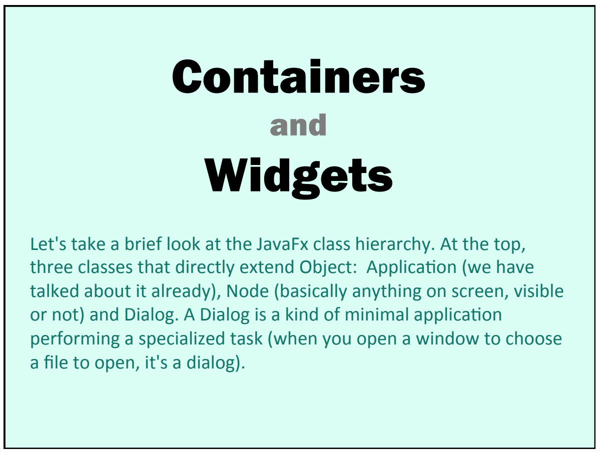
Some significant changes



Actions must be public

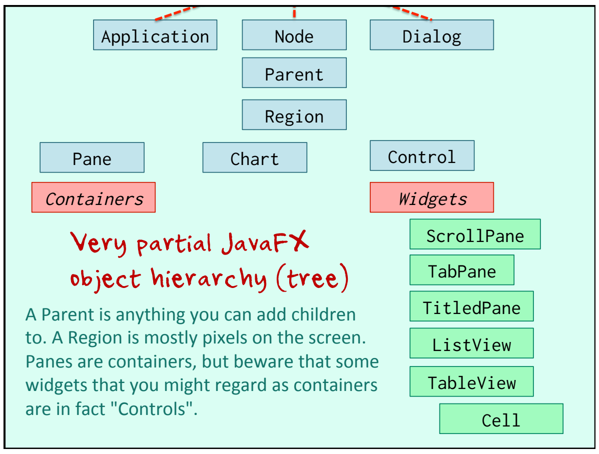
Some attributes must be static

You must create a **FXMLLoader** object the constructor of which throws exceptions (beware). Because the code is divided into new modules, actions must be **public**.



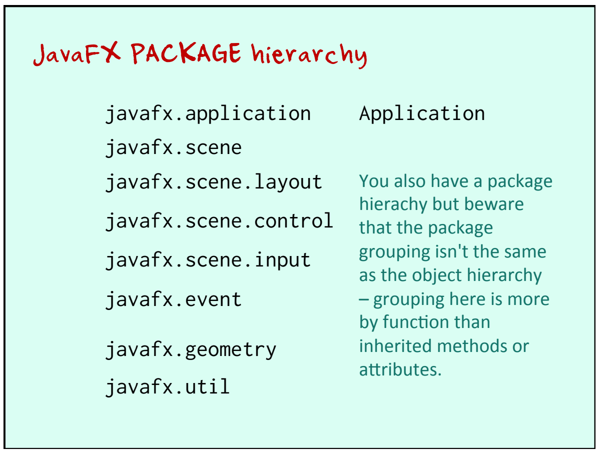
Containers and Widgets

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)

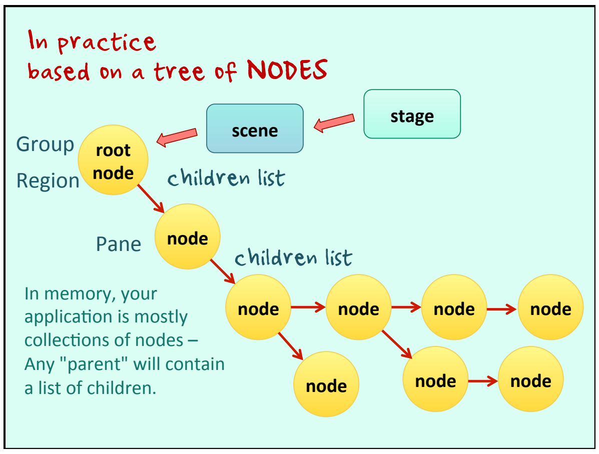
A Parent is anything you can add children to. A Region is mostly pixels on the screen. Panes are containers, but beware that some widgets that you might regard as containers are in fact "Controls".



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.



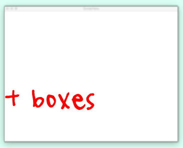
In practice based on a tree of NODES

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

Panes

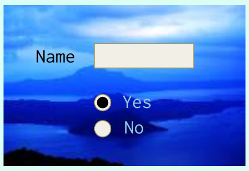
Pane BorderPane + boxes

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.

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

Not many (5 or 6 at most)

Other actions: pull-down menu

Clear and concise label
Can be an image (icon)

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

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

Widgets

Button

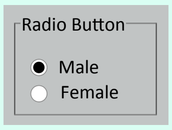
Keep all buttons the same size
... or have a "short button" and a "long" button size

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.

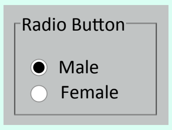
ToggleGroup object

`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



One of several exclusive choices

Usually in a group

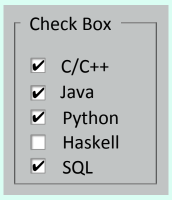
Use vertically

Six options or less

More than six options: ListBox

Avoid Yes/No or On/Off

Widgets



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

One line: TextField

No echo: PasswordField

Prompt text

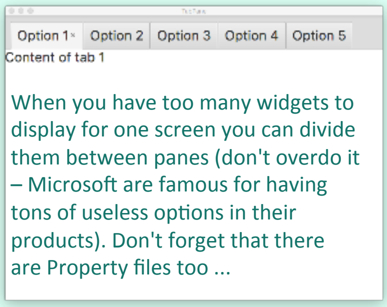
Several lines: TextArea

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



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 ...

TabPane

Too many widgets

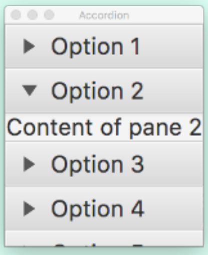
Container of Tabs

TabPane

```
// 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



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

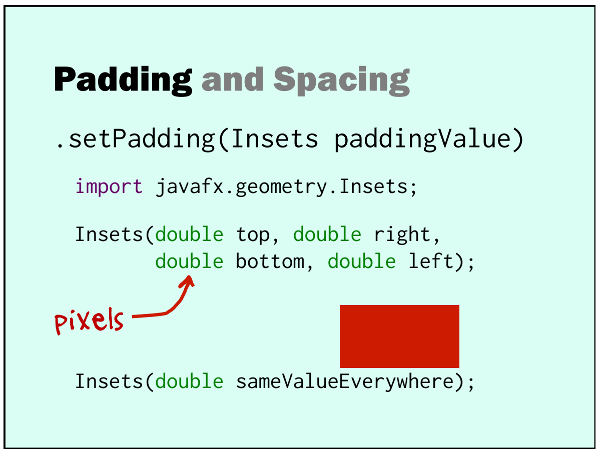
```
// Create an Accordion
Accordion accordion = new Accordion();
root.getChildren().add(accordion);
TitledPane pane;
// Create five titled panes
for (int i = 1; i <= 5; i++) {
  pane = new TitledPane();
  pane.setText("Option " + i);
  pane.setContent(new Label("Content of pane " + i));
  accordion.getPanes().add(pane);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

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)



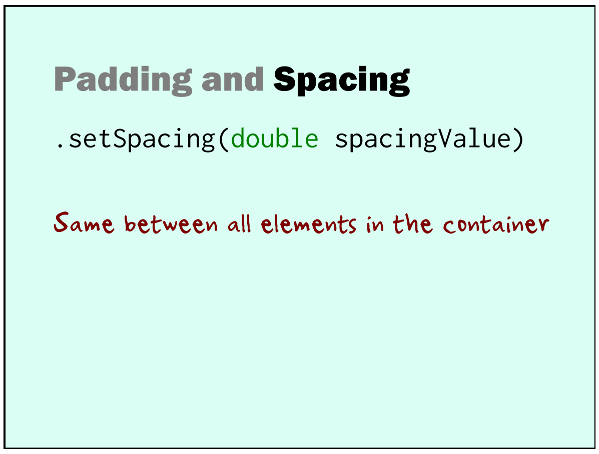
Padding and Spacing

```
.setPadding(Insets paddingValue)

  import javafx.geometry.Insets;

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

pixels

[redacted]

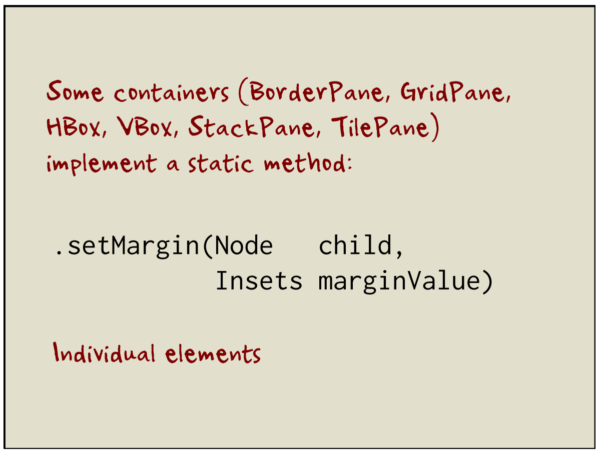
```
  Insets(double sameValueEverywhere);
```



Padding and Spacing

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

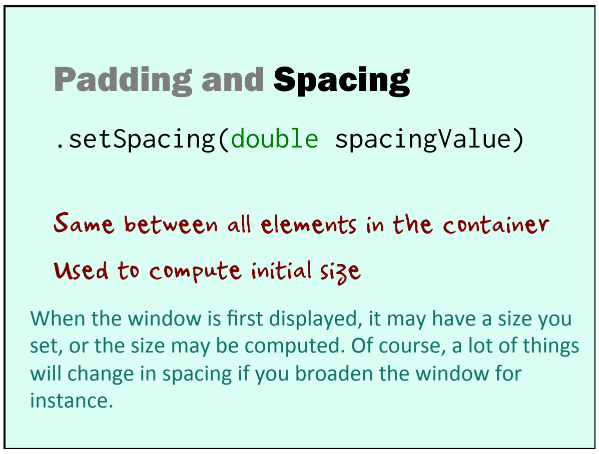
Same between all elements in the container



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

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

Individual elements



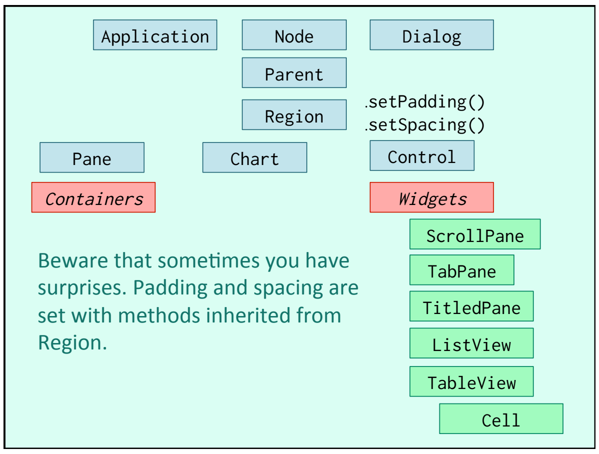
Padding and Spacing

```
.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.



Application Node Dialog

Parent

Region .setPadding() .setSpacing()

Pane Chart Control

Containers *Widgets*

ScrollPane

TabPane

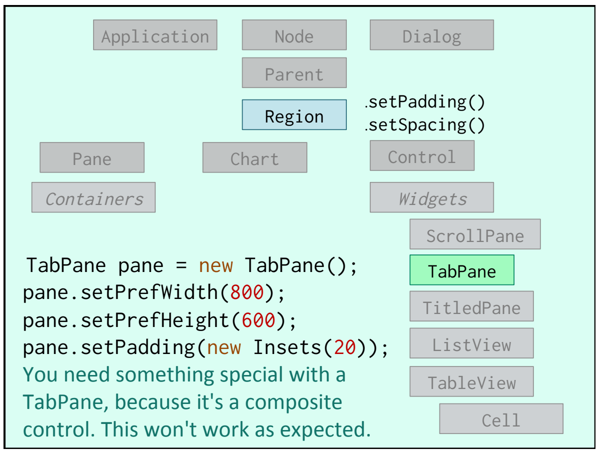
TitledPane

ListView

TableView

Cell

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



Application Node Dialog

Parent

Region .setPadding() .setSpacing()

Pane Chart Control

Containers *Widgets*

ScrollPane

TabPane

TitledPane

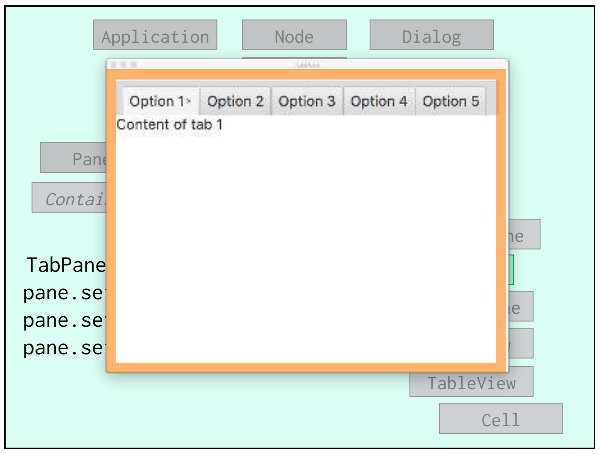
ListView

TableView

Cell

```
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.



Application Node Dialog

Option 1 Option 2 Option 3 Option 4 Option 5

Content of tab 1

Pane

Contai

TabPane

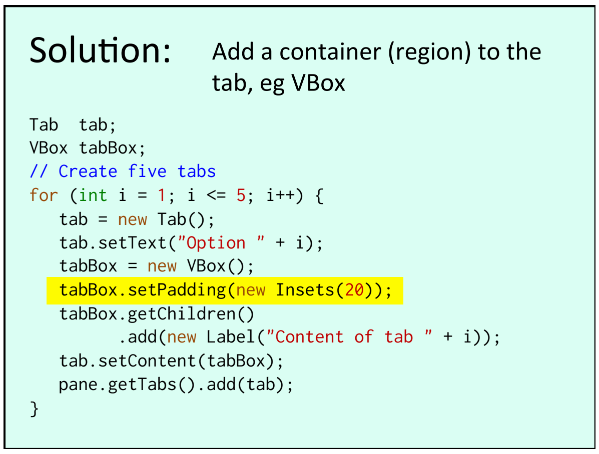
pane.se

pane.se

pane.se

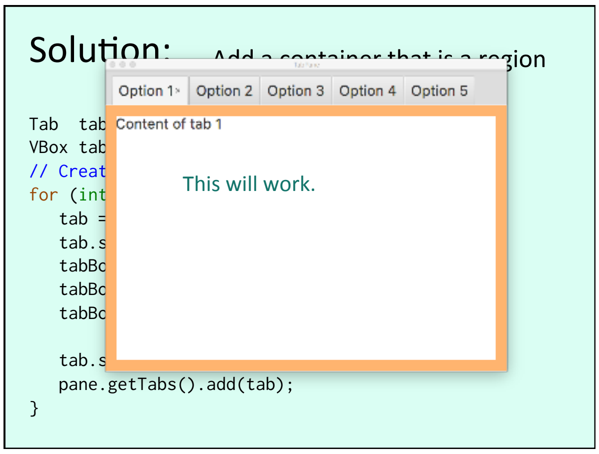
TableView

Cell



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

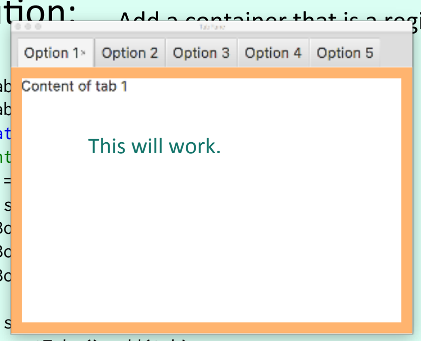
```
Tab  tab;
VBox tabBox;
// Create five tabs
for (int i = 1; i <= 5; i++) {
   tab = new Tab();
   tab.setText("Option " + i);
   tabBox = new VBox();
   tabBox.setPadding(new Insets(20));
   tabBox.getChildren()
         .add(new Label("Content of tab " + i));
   tab.setContent(tabBox);
   pane.getTabs().add(tab);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



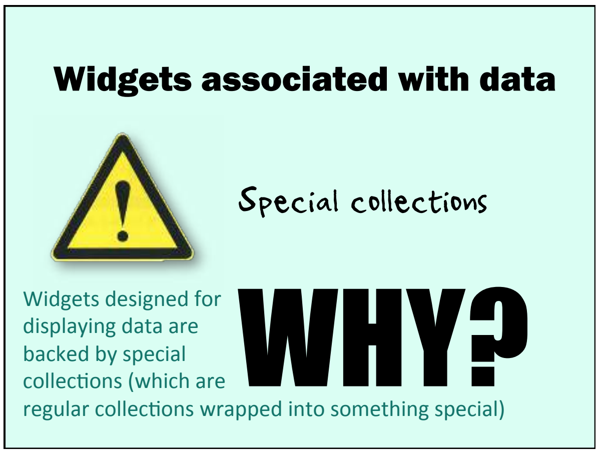
Solution:

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);
}
```



This will work.

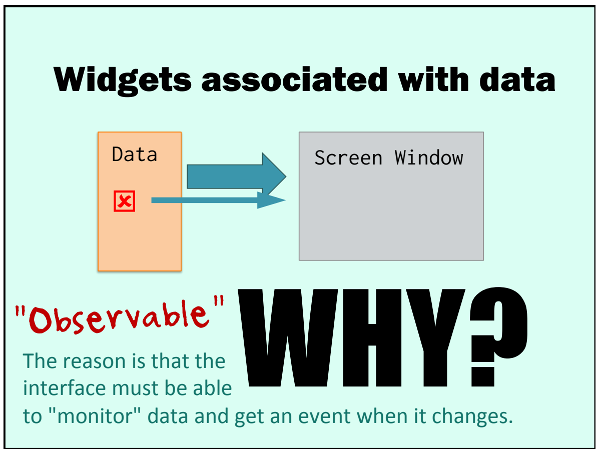


Widgets associated with data

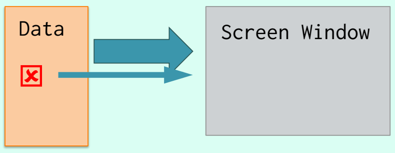
Special collections

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

WHY?



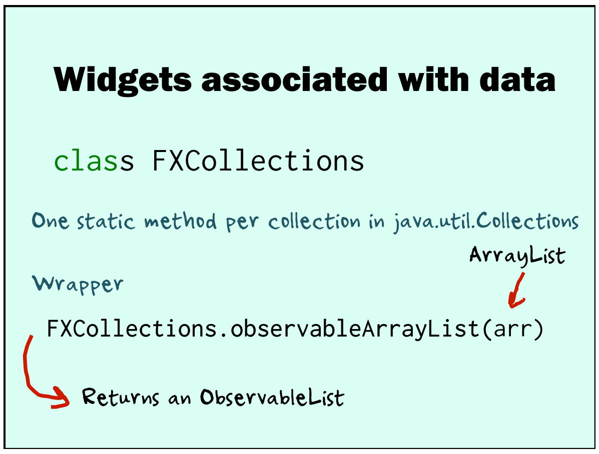
Widgets associated with data



"Observable"

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

WHY?



Widgets associated with data

```
class FXCollections
```

One static method per collection in java.util.Collections

Wrapper

ArrayList

```
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

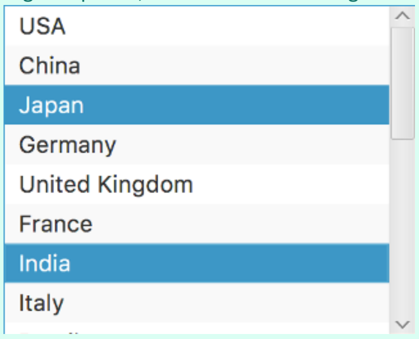
Scrollable
Can be editable

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()) {
    ...
}
```

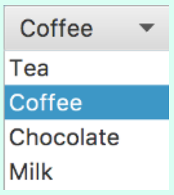
Usual collection handling to retrieve (in the handler possibly activated by a click on a button) what was selected.

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

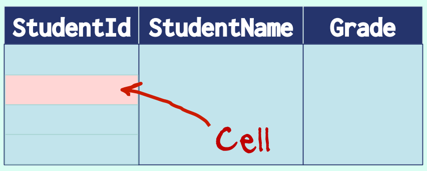
```
T selected = combobox.getValue();
```

Nothing special about using it inside the program.

Widgets associated with data

Table Views and Tree Views

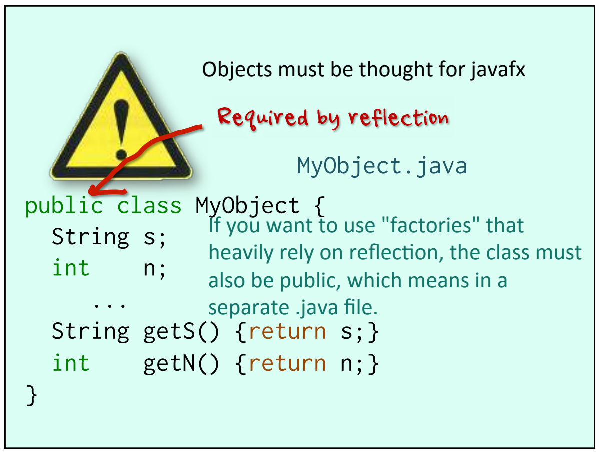
Data usually retrieved from a database



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;}
}
```



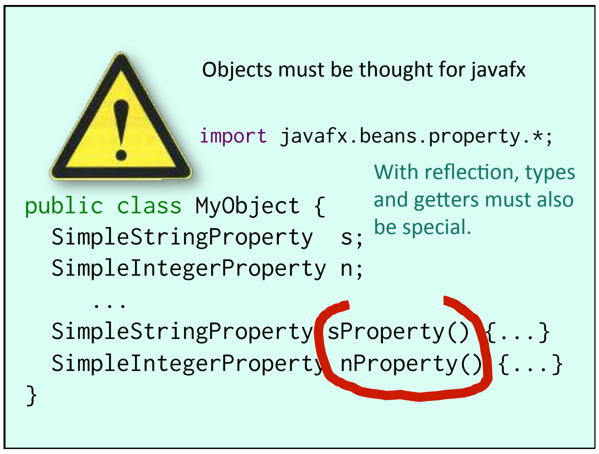
Objects must be thought for javafx

Required by reflection

MyObject.java

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

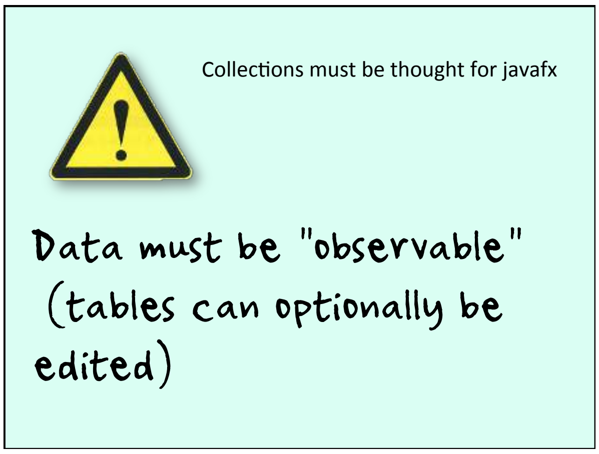
If you want to use "factories" that heavily rely on reflection, the class must also be public, which means in a separate .java file.



Objects must be thought for javafx

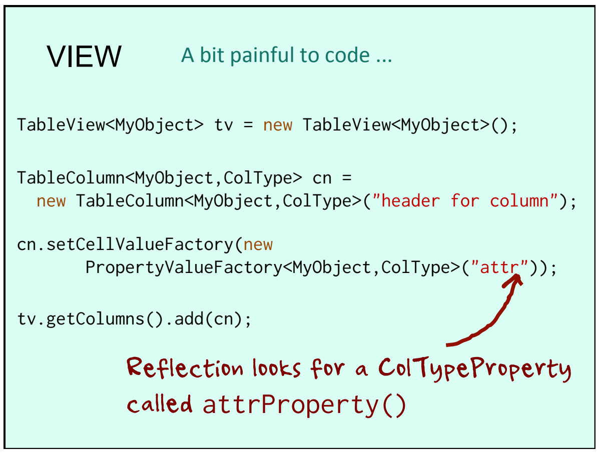
```
import javafx.beans.property.*;

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

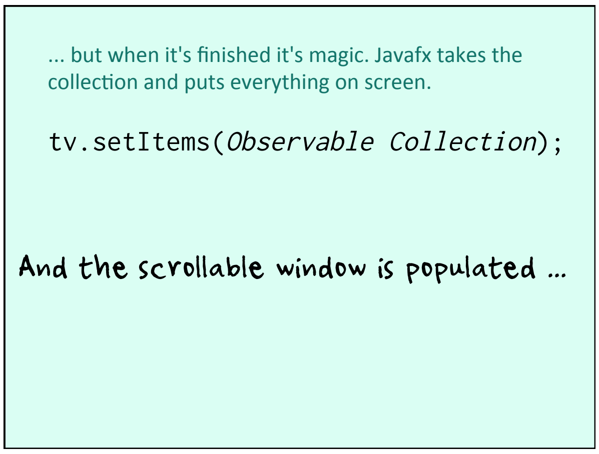
Data must be "observable" (tables can optionally be edited)



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);
```

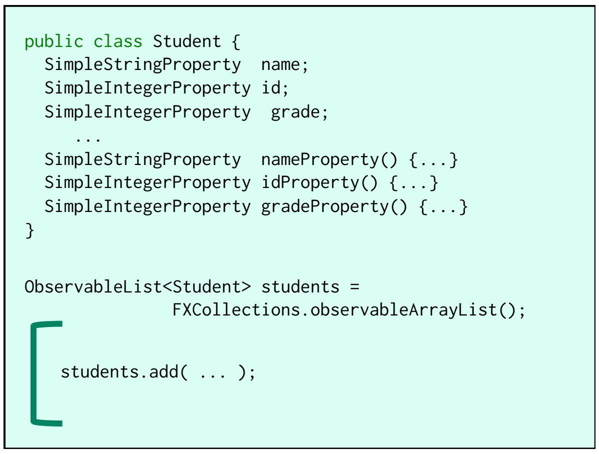
Reflection looks for a ColTypeProperty called attrProperty()



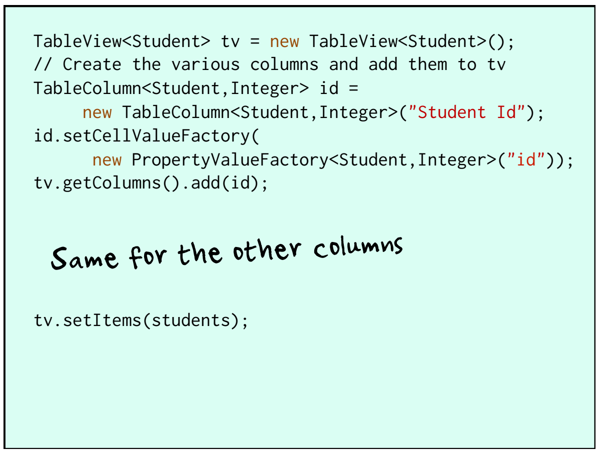
... but when it's finished it's magic. Javafx takes the collection and puts everything on screen.

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

And the scrollable window is populated ...



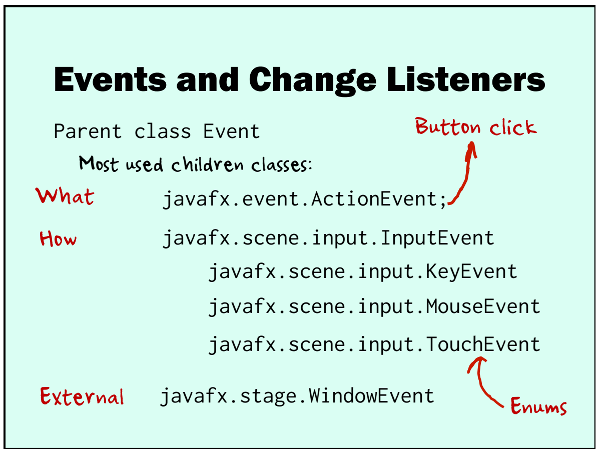
```
public class Student {
  SimpleStringProperty  name;
  SimpleIntegerProperty id;
  SimpleIntegerProperty  grade;
     ...
  SimpleStringProperty  nameProperty() {...}
  SimpleIntegerProperty idProperty() {...}
  SimpleIntegerProperty gradeProperty() {...}
}

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

Parent class Event   Button click

Most used children classes:

What   javafx.event.ActionEvent;

How   javafx.scene.input.InputEvent
    javafx.scene.input.KeyEvent
    javafx.scene.input.MouseEvent
    javafx.scene.input.TouchEvent

External   javafx.stage.WindowEvent   Enums

There are different rules for the system to find the target – widget that has the focus for key events, cursor position for mouse events, etc, and of course if an element is hidden by another it's the one on top that is considered to be the target.

Events and Change Listeners

Determine the Event Target

`EventTarget` interface
- `Window`
- `Scene`
- `Node`

"Bubbles up" like exceptions

Event handler ≅ **catch**

Events and Change Listeners

Tons of

`.setOnSomeAction()` methods for Nodes

```
KeyPressed
KeyReleased
KeyTyped
MouseClicked
MouseExited
...
```

Events and Change Listeners

`.setOnAction()` method for Buttons
- Radio Buttons
- Check Boxes

Events and Change Listeners

Lambda expressions!

```
Button btn = new Button();
btn.setText("Say 'Hi'");
btn.setOnAction((e)->{
            System.out.println("Hi!");
        });
```

You can also define Event Filters to intercept (block) some events or override them

Events and Change Listeners

Instead of `.setOnxxxx()` methods, you can use `addEventHandler()` for unusual actions

```
button.addEventHandler(MouseEvent.MOUSE_ENTERED,
                       (e)->{moveWindow(stage);});
```

Moving the window away when you try to click on the button isn't usual.

"Change Listeners" next time ...