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

Events and Change Listeners

Reacts to event occurring elsewhere

"Observable Value" changes

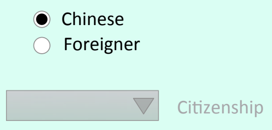
Widely implemented interface

Application:

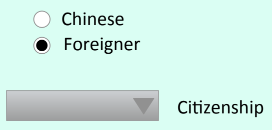
- Selections (radio button groups, lists ...)
- Progress bar

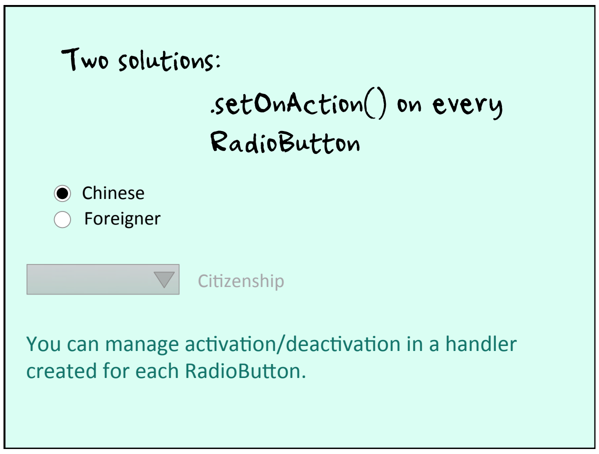
We have talked last time about events, "Change Listeners" are less focused on one particular widget.

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



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).





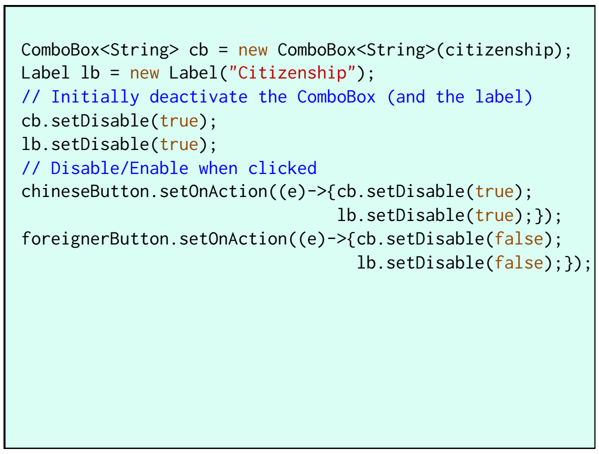
Two solutions:

.setOnAction() on every RadioButton

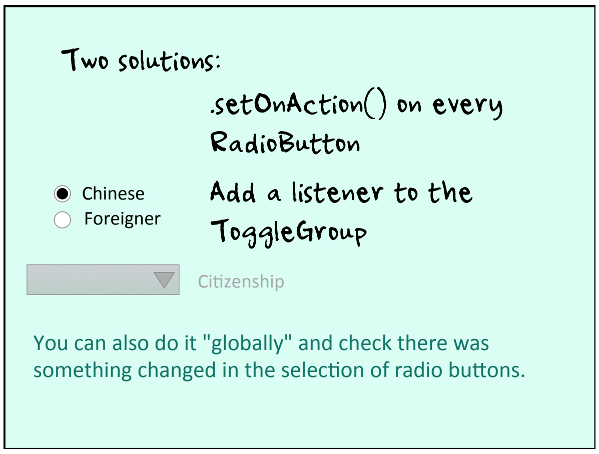
● Chinese
○ Foreigner

Citizenship

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:

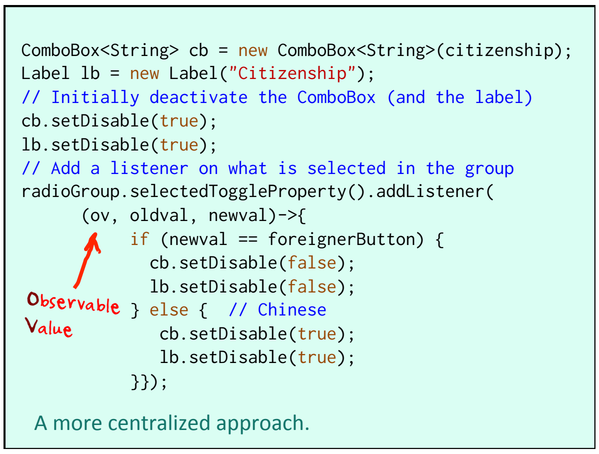
.setOnAction() on every RadioButton

Add a listener to the ToggleGroup

● Chinese
○ Foreigner

Citizenship

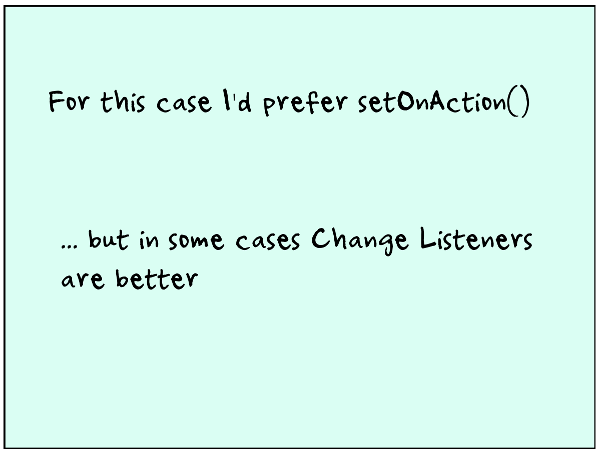
You can also do it "globally" and check there was something changed in the selection of radio buttons.



```
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);
           }});
```

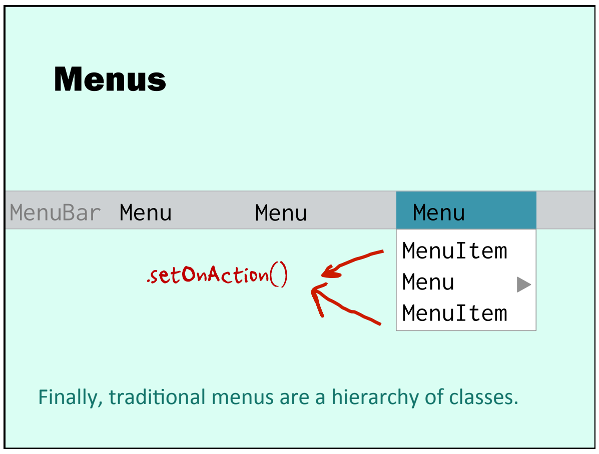
Observable Value

A more centralized approach.



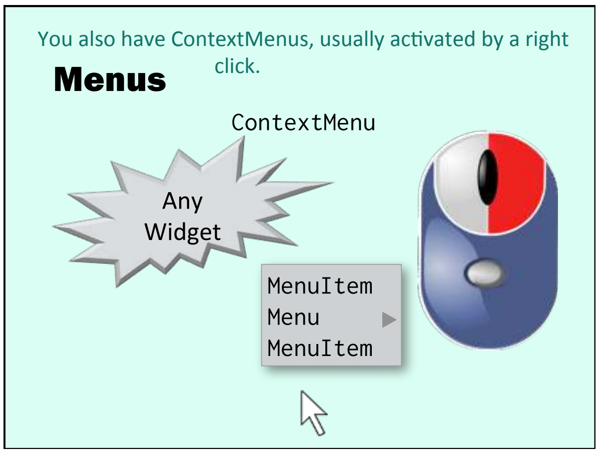
For this case I'd prefer setOnAction()

… but in some cases Change Listeners are better



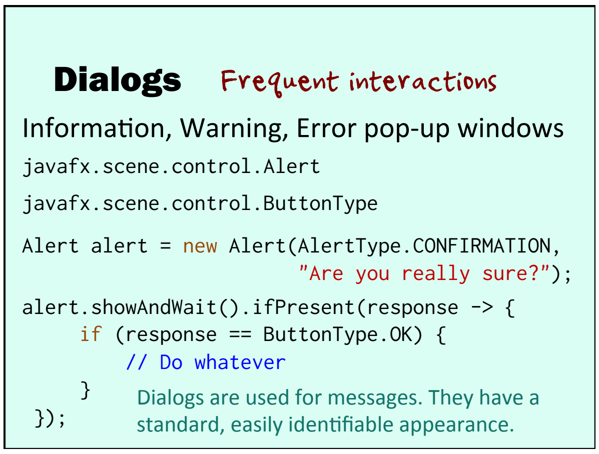
Menus

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).

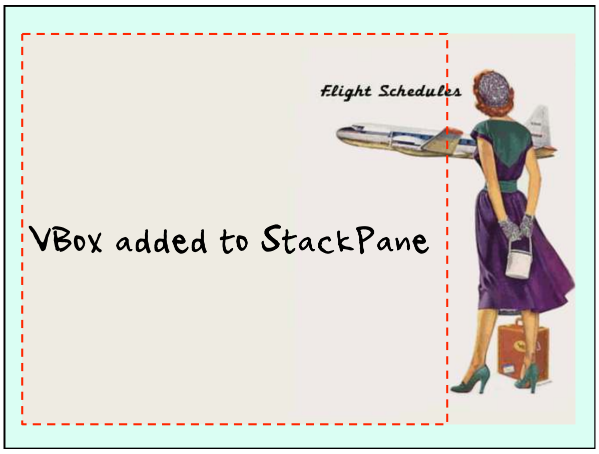
Many other features
but it's a start ...

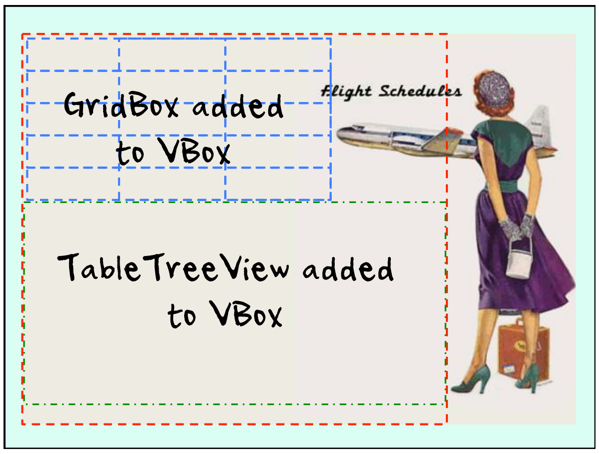
A case study

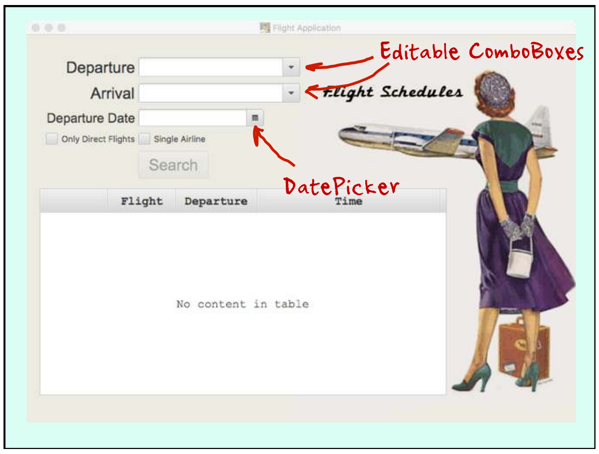
To give you a feeling of what you can do, this is a demo application I have written that searches flights in a database file with around 75,000 flights between around 100 of the busiest airports in the world.

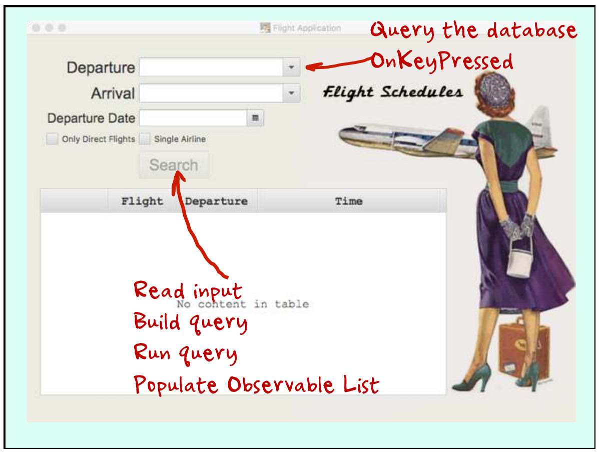
Not resizable

ImageView in a StackPane


Flight Schedules
VBox added to StackPane

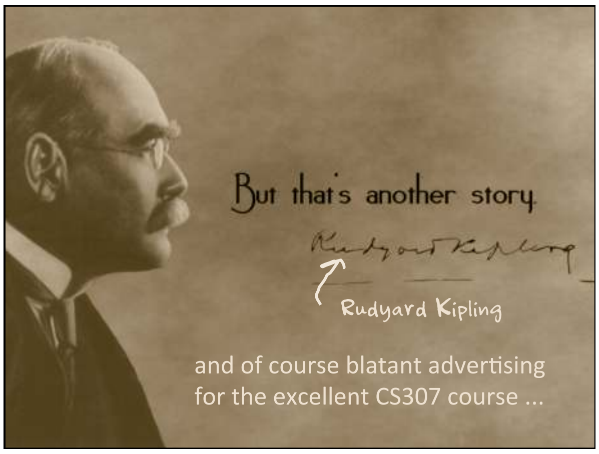

Flight Schedules
GridBox added to VBox
TableTreeView added to VBox


Flight Application
Departure
Arrival
Departure Date
Only Direct Flights
Single Airline
Search
Flight
Departure
Time
No content in table
Flight Schedules
Editable ComboBoxes
DatePicker


Flight Application
Departure
Arrival
Departure Date
Only Direct Flights
Single Airline
Search
Flight
Departure
Time
No content in table
Flight Schedules
Query the database
OnKeyPressed
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.



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

Widgets associated with data

Lists and Combo Boxes

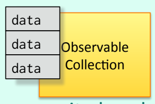
Table Views and Tree Views

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

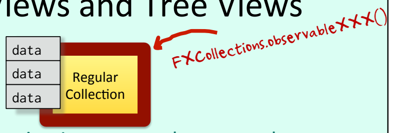


In all cases, we have seen it already, what is on the screen is backed by an "Observable Collection", which is nothing more than a Collection that supports change listeners, which allows refreshing the screen when data changes in the collection.

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.

Definition of a TableView

```
TableView<RowType> tableView = new TableView<RowType>();
tableView.setItems(FXCollections
                    .observableArrayList(rowTypeArrayList));
```

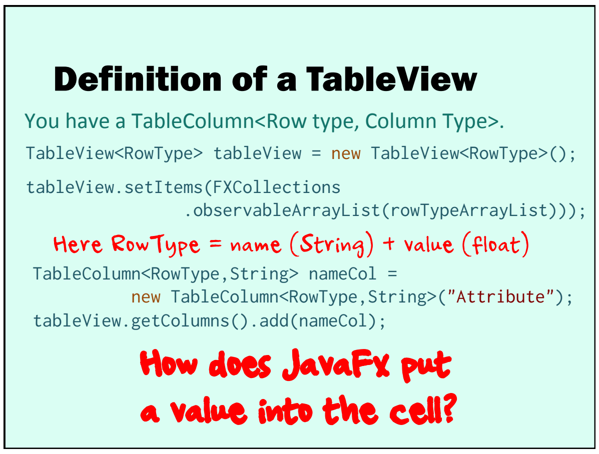
Must define columns

What do we show?

How do we show it?

You apply the same setItems() method as with ListViews but this time it's not enough.

You may not want to show every attribute in the object. You may want to show them in a particular order, or displayed in a particular way. You must define columns.



Definition of a TableView

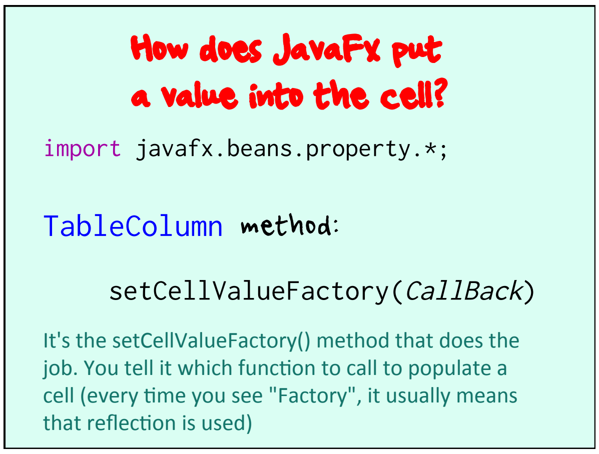
You have a TableColumn<Row type, Column Type>.

```
TableView<RowType> tableView = new TableView<RowType>();
tableView.setItems(FXCollections
                 .observableArrayList(rowTypeArrayList)));
```

Here RowType = name (String) + value (float)

```
TableColumn<RowType,String> nameCol =
           new TableColumn<RowType,String>("Attribute");
tableView.getColumns().add(nameCol);
```

How does JavaFx put a value into the cell?



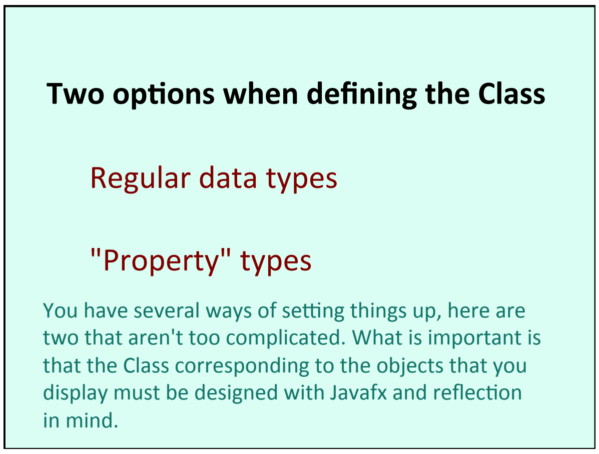
How does JavaFx put a value into the cell?

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

`TableColumn` method:

```
setCellValueFactory(CallBack)
```

It's the setCellValueFactory() method that does the job. You tell it which function to call to populate a cell (every time you see "Factory", it usually means that reflection is used)

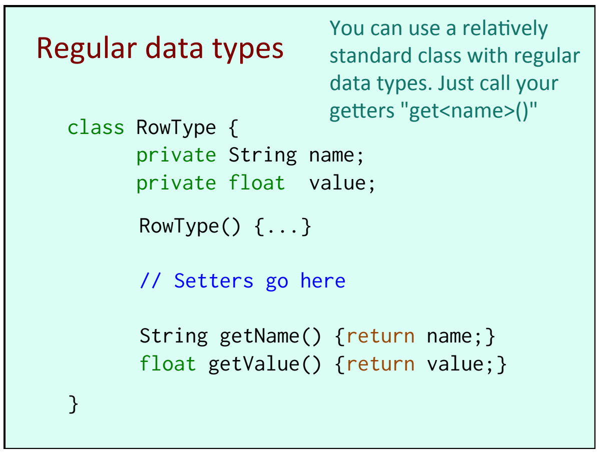


Two options when defining the Class

Regular data types

"Property" types

You have several ways of setting things up, here are two that aren't too complicated. What is important is that the Class corresponding to the objects that you display must be designed with Javafx and reflection in mind.



Regular data types

You can use a relatively standard class with regular data types. Just call your getters "get<name>()"

```
class RowType {
       private String name;
       private float  value;

       RowType() {...}

       // Setters go here

       String getName() {return name;}
       float getValue() {return value;}
}
```

Because the Factory method takes a function that returns a special type, you must wrap what the getter returns.

```
TableColumn<RowType,String> nameCol =
          new TableColumn<RowType,String>("Attribute");
tableView.getColumns().add(nameCol);
nameCol.setCellValueFactory((cd)->
          new ReadOnlyStringWrapper(cd.getValue()
                                      .getName()));

TableColumn<RowType,Number> valCol =
          new TableColumn<RowType,Number>("Value");
tableView.getColumns().add(valCol);
valCol.setCellValueFactory((cd)->
          new ReadOnlyFloatWrapper(cd.getValue()
                                     .getValue()));
```

getValue() returns the object of the current row.

Beware that the wrappers for numbers all return a Number type that is the parent of Integer, Float, Double.

Mine!

"Property" types

The other option is to use wrapper property types that must be instantiated in setters.

```
  class RowType {
        private SimpleStringProperty name;
        private SimpleFloatProperty  value;

        RowType() {...}

        // Setters go here

        SimpleStringProperty nameProperty(){
               return name;}

        SimpleFloatProperty valueProperty() {
               return value;}
  }
```

new SimpleXXXProperty()

Note method names.

```
TableColumn<RowType,String> nameCol =
          new TableColumn<RowType,String>("Attribute");
tableView.getColumns().add(nameCol);
valCol.setCellValueFactory(new
         PropertyValueFactory<RowType,String>("name"));

TableColumn<RowType,Float> valCol =
          new TableColumn<RowType,Float>("Value");
tableView.getColumns().add(valCol);
valCol.setCellValueFactory(new
         PropertyValueFactory<RowType,Float>("value"));
```

Then you can use "PropertyValueFactory()" that returns a suitable callback. No problem with "Number".

The easy solution ...

Only String types

After all it's just for displaying ...

Unless you want the table to be editable (as a spreadsheet), a convenient solution is to return everything as a String (at least in the <attr>Property() method). In particular, one often wants numerical values to be right-aligned. A TableView doesn't do it, but you can format a number to be right-aligned in a String.

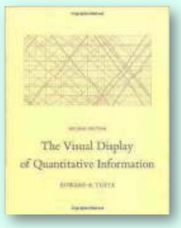
Price	123.50

Graphics in Java

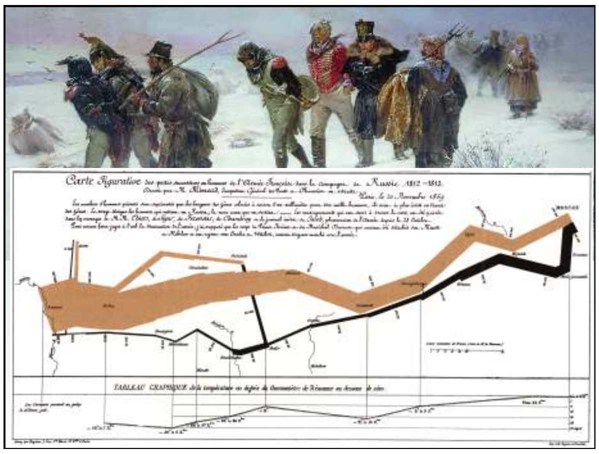
Let's now switch to what most monitoring tools use intensely: graphics. We'll only talk of relatively classic graphics that are often used in business applications.

Excellent book

Flickr:
Leonard Ling



If you have the opportunity to find this book in a library, take a look at it. The title isn't glamorous but the book is remarkable. What Tufte cites as one of the best graphics ever created is on the next slides and displays the 1812 disastrous French Russian campaign (numbers have been debated, but it's not the point). On a 2D surface you have a map, the size of the army, time, temperature (when retreating) and it remains remarkably legible. It wasn't done by a program ...


Carte Figurative des pertes successives en hommes de l'Armée Française dans la campagne de Russie 1812-1813.
TABLEAU GRAPHIQUE de la température en degrés du thermomètre de Réaumur au dessous de zéro.

Charts

Much used in business applications...

```
import javafx.scene.chart.*;

class PieChart
```

Pie charts are hated by Tufte.

```
class XYChart
```

X can be a real X or the name of a category.

```
AreaChart
BarChart
BubbleChart
LineChart
ScatterChart
StackedAreaChart
StackedBarChart
```

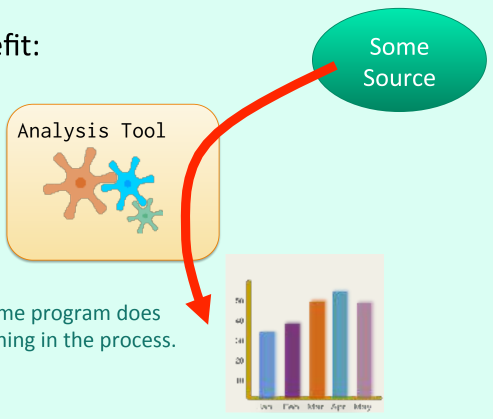
Benefit:

Very often people collect data with a program, generate a .csv or .txt file, load the data in Excel and create charts in Excel.

Writing a tool that collects source data and generates graphics for reports without having to use Excel

This is pretty time-consuming (even if you have macros, because you have to open files and so forth). Charts are therefore useful not only in an interactive application, but also to help generate reports.

Benefit:



The same program does everything in the process.

A Bar Chart example

Charts are widgets backed by data (they can also be updated dynamically), which means as usual Observable Lists.

Backed by an ObservableList

```
ObservableList<XYChart.Data<XType,YType>>
```

```
#EVENT TIME_WAITED PCT_WAITS WAIT_CLASS
"db file sequential read",2172313,87.370,"User I/O"
"db file scattered read",130611,5.250,"User I/O"
"log file switch (checkpoint incomplete)",84041,3.380,"Configuration"
"enq: TX - row lock contention",34906,1.400,"Application"
"log file switch completion",19113,0.770,"Configuration"
"direct path write temp",11049,0.440,"User I/O"
"log file sync",10550,0.420,"Commit"
"read by other session",8530,0.340,"User I/O"
"control file sequential read",3638,0.150,"System I/O"
"db file parallel read",2477,0.100,"User I/O"
"direct path read temp",2332,0.090,"User I/O"
"SQL*Net more data to client",2234,0.090,"Network"
"SQL*Net message to client",1453,0.060,"Network"
"buffer busy waits",566,0.020,"Concurrency"
"Streams AQ: qmn coordinator waiting for slave to start",490,0.020,"Other"
"SQL*Net more data from client",437,0.020,"Network"
"control file heartbeat",392,0.020,"Other"
"direct path read",240,0.010,"User I/O"
"latch free",147,0.010,"Other"
"enq: CF - contention",127,0.010,"Other"
```

My data comes from this file. What I want to chart is highlighted.

```
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.stage.Stage;
import javafx.collections.*;

import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;

import java.nio.file.Paths;
import java.nio.file.Files;
import java.io.BufferedReader;
import java.io.IOException;
```

This is what I need for the chart

This is what I need for reading the file

```
import java.nio.file.Paths;
import java.nio.file.Files;
import java.io.BufferedReader;
import java.io.IOException;

public class BarChartExample extends Application {
    private final String dataFile = BarChartExample.class
                                   .getClassLoader()
                                   .getResource("data.txt")
                                   .toString().replace("file:",
                                                       "");

    private static ObservableList<XYChart.Data<String,Number>>
          data = FXCollections.observableArrayList();
```

I prepare the name of my file and an ObservableList where I'm going to sore what I read from the file.

```
    private static ObservableList<XYChart.Data<String,Number>>
          data = FXCollections.observableArrayList();

    static void loadData(String file) {
        // Here it's loaded from a file, it could
        // as well be queried from a database (this type
        // of data is obtained by querying system tables).
        // We are only interested by the first and third
        // fields from each line (event name and percentage)
        try (BufferedReader reader
              = Files.newBufferedReader(Paths.get(file))) {
          String   line = null;
```

Going to split each line on commas.

```
          String[] fields;
          while ((line = reader.readLine()) != null) {
            if (!line.startsWith("#")) { // not a comment
              fields = line.split(",");
              data.add(new
XYChart.Data<String,Number>(fields[0].replace("\"",
                                              ""),
                                   new Float(fields[2])));
            }
          }
        } catch (IOException x) {
          System.err.format("IOException: %s%n", x);
        }
    }
```

Adding the first field (X) and 3rd field (Y) of each row to the collection. Note that each object in the collection is XYChart.Data<X type, Y type>

```
    @Override
    public void start(Stage stage) {
         stage.setTitle("Technical Bar Chart");
         final CategoryAxis xAxis = new CategoryAxis();
         final NumberAxis yAxis = new NumberAxis();
         final BarChart<String,Number> bc =
                   new BarChart<String,Number>(xAxis,yAxis);
         bc.setTitle("Database Waits");
         xAxis.setLabel("Event");
         yAxis.setLabel("Percentage of Waits");
         bc.setLegendVisible(false);
```

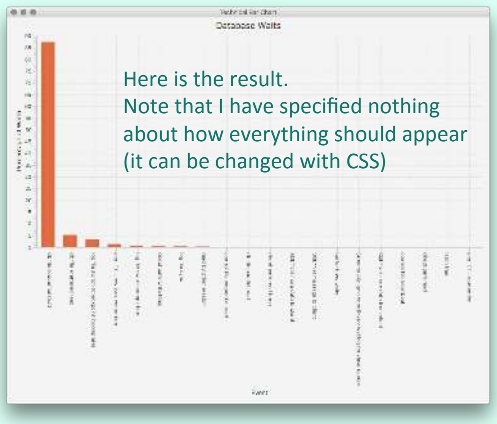
Because I only have ONE series of data

You can have several values (series) for Y associated with every X, it's not my case.

```
          loadData(dataFile);
          XYChart.Series<String,Number> series =
                    new XYChart.Series<String,Number>();
          series.setData(data);

          Scene scene  = new Scene(bc,1000,800);
          bc.getData().add(series);
          stage.setScene(scene);
          stage.show();
        }

        public static void main(String[] args) {
          launch(args);
        }
}
```

Used as root node

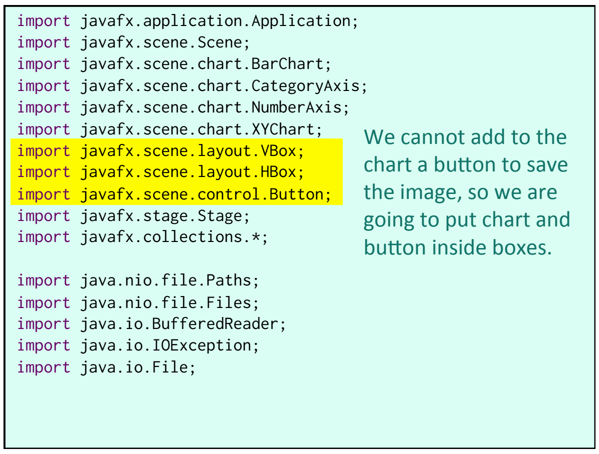
There is a setData() wich reminds of setItems(). Note that here the BarChart is used as root node. It works because a Chart is a child of Region, like a Pane.



Here is the result.
Note that I have specified nothing about how everything should appear (it can be changed with CSS)

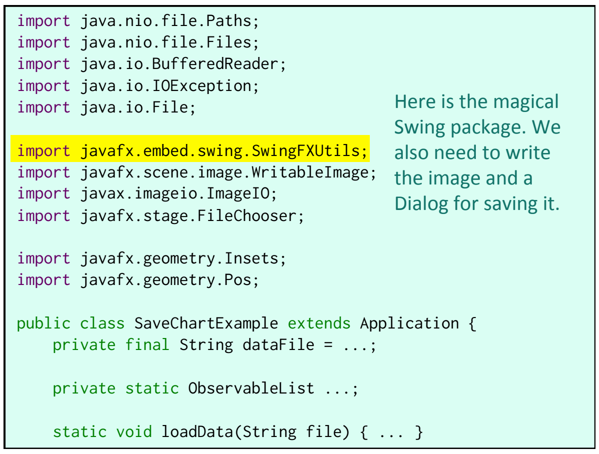
Generating Image Files

Having an image on screen is nice, but if you want to include it into a report taking a screenshot isn't the most convenient. You can save a chart to a file, using a component that actually comes from Swing ...



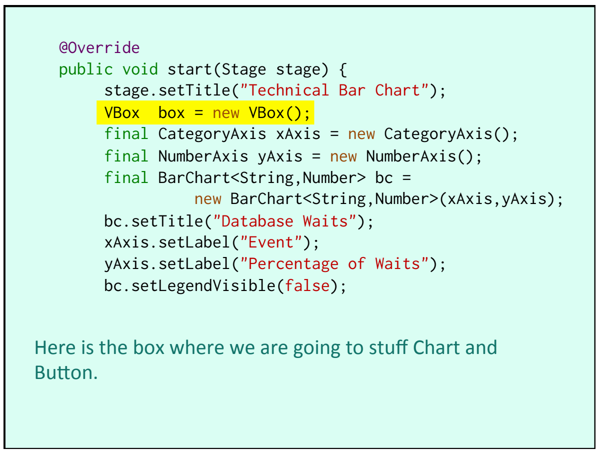
```java
import javafx.application.Application;
import javafx.scene.Scene;
import javafx.scene.chart.BarChart;
import javafx.scene.chart.CategoryAxis;
import javafx.scene.chart.NumberAxis;
import javafx.scene.chart.XYChart;
import javafx.scene.layout.VBox;
import javafx.scene.layout.HBox;
import javafx.scene.control.Button;
import javafx.stage.Stage;
import javafx.collections.*;

import java.nio.file.Paths;
import java.nio.file.Files;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.File;
```

We cannot add to the chart a button to save the image, so we are going to put chart and button inside boxes.



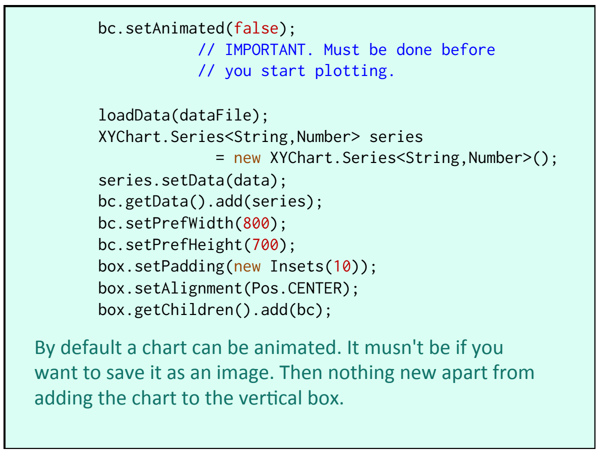
```java
import java.nio.file.Paths;
import java.nio.file.Files;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.File;

import javafx.embed.swing.SwingFXUtils;
import javafx.scene.image.WritableImage;
import javax.imageio.ImageIO;
import javafx.stage.FileChooser;

import javafx.geometry.Insets;
import javafx.geometry.Pos;

public class SaveChartExample extends Application {
    private final String dataFile = ...;

    private static ObservableList ...;

    static void loadData(String file) { ... }
```

Here is the magical Swing package. We also need to write the image and a Dialog for saving it.



```java
    @Override
    public void start(Stage stage) {
         stage.setTitle("Technical Bar Chart");
         VBox  box = new VBox();
         final CategoryAxis xAxis = new CategoryAxis();
         final NumberAxis yAxis = new NumberAxis();
         final BarChart<String,Number> bc =
                   new BarChart<String,Number>(xAxis,yAxis);
         bc.setTitle("Database Waits");
         xAxis.setLabel("Event");
         yAxis.setLabel("Percentage of Waits");
         bc.setLegendVisible(false);
```

Here is the box where we are going to stuff Chart and Button.



```java
          bc.setAnimated(false);
                     // IMPORTANT. Must be done before
                     // you start plotting.

          loadData(dataFile);
          XYChart.Series<String,Number> series
                       = new XYChart.Series<String,Number>();
          series.setData(data);
          bc.getData().add(series);
          bc.setPrefWidth(800);
          bc.setPrefHeight(700);
          box.setPadding(new Insets(10));
          box.setAlignment(Pos.CENTER);
          box.getChildren().add(bc);
```

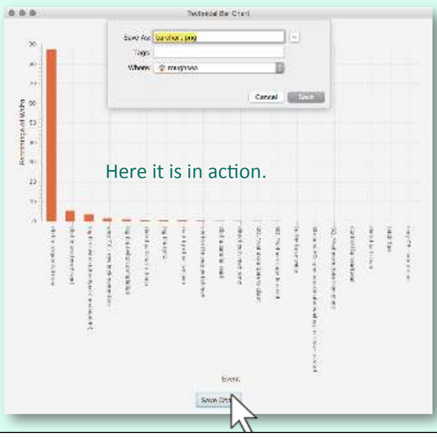
By default a chart can be animated. It musn't be if you want to save it as an image. Then nothing new apart from adding the chart to the vertical box.

```
HBox hbox = new HBox();
hbox.setPadding(new Insets(5));
hbox.setAlignment(Pos.CENTER);
Button saveButton = new Button("Save Chart");
hbox.getChildren().add(saveButton);
box.getChildren().add(hbox);

saveButton.setOnAction((e)->{
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Save Chart");
    fileChooser.setInitialFileName("barchart.png");
    File selectedFile =
               fileChooser.showSaveDialog(stage);
```

Another box (mostly to control padding) to add the button, then the action: when you click on the button, the dialog opens with a default filename.

```
saveButton.setOnAction((e)->{
    FileChooser fileChooser = new FileChooser();
    fileChooser.setTitle("Save Chart");
    fileChooser.setInitialFileName("barchart.png");
    File selectedFile =
               fileChooser.showSaveDialog(stage);
    if (selectedFile != null) {
      try {
        WritableImage snap = bc.snapshot(null, null);
        ImageIO.write(SwingFXUtils.fromFXImage(snap,
                      null),
                      "png", selectedFile);
      } catch (IOException exc) {
        System.err.println(exc.getMessage());
      }
    }
  });
```

If you didn't click "Cancel" in the dialog (which would return null) you can take a snapshot in the program and save it.
... Skipping the end of the program ...



Here it is in action.

2D Graphics

Charts are 2D graphics (you also have 3D charts but if you ever read Tufte you'll never want to use them), but in charts you haven't full freedom to draw whatever you want on the screen. If you want to draw you should you a Canvas object. "Canvas" was the name of the cloth used in the old days for making ship sails. Put on a wooden frame, this is what western artists started to use around the 17th century for painting, hence the name in graphical interfaces.

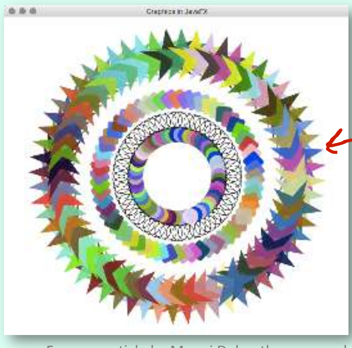
Canvas object

Lines

Geometrical shapes

On a canvas, you mostly draw lines and shapes.

Examples you usually find



Check figure 15.32 in the book (close)

Whenever you look for a Canvas example, you find a program generating something like this.

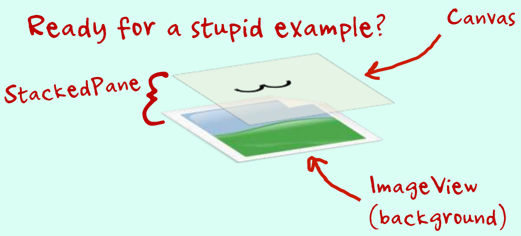
From an article by Manoj Debnath on www.developer.com

I have decided to have a stupid example of my own. Background image, and Canvas on which I draw two half circles close to each other (part-circles shapes are called "arc", which is French for "bow"

)

arc (de triomphe), originally to honor victorious armies in Rome, idea copied in Paris

Ready for a stupid example?



```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.scene.*;
import javafx.scene.layout.*;
import javafx.scene.paint.*;
import javafx.scene.canvas.*;
import javafx.scene.shape.*;
import javafx.stage.Stage;
import javafx.stage.Screen;
import javafx.scene.image.*;
import java.net.URL;

public class StupidCanvasExample extends Application {

    public static void main(String[] args) {
        launch(args);
    }
```

A couple of new packages to import.

```
public void start(Stage stage) {
    double width;
    double height;
    double x;
    double y;

    stage.setTitle("StupidCanvasExample");
    stage.setResizable(false);
    Group root = new Group();
    Scene scene = new Scene(root);
    StackPane pane = new StackPane();
    URL url = this.getClass()
                  .getClassLoader()
                  .getResource("background.jpeg");
```

StackPane to put image and Canvas (transparent) on top of it.

```
    if (url != null) {
      Image image = new Image(url.toString());
      width = image.getWidth();
      height = image.getHeight();
      ImageView iv = new ImageView(image);
      pane.getChildren().add(iv);

      final Canvas canvas = new Canvas(width, height);
      GraphicsContext gc = canvas.getGraphicsContext2D();
```

To draw on a Canvas, you need the associated "GraphicsContext". This is where you define, among other things, line thickness and colours.

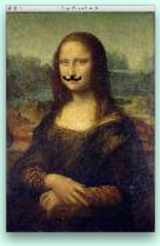
```
      gc.setStroke(Color.BLACK);
      gc.setLineWidth(height * 0.01);
      x = 0.42 * width - width / 36.0;
      y = 0.285 * height;
      gc.strokeArc(x, y,
                   width / 18.0, height / 40.0,
                   180, 180, ArcType.OPEN);
      x += width / 18.0;
      gc.strokeArc(x, y,
                   width / 18.0, height / 40.0,
                   180, 180, ArcType.OPEN);
      pane.getChildren().add(canvas);
      // Make canvas disappear when clicked
      canvas.setOnMouseClicked((e)->{
         canvas.setVisible(false);
      });
```

There are multiple ways to define colours. For basic colors you can use an enum.

"Stroke" refers to lines. When you draw, you give the position of the top left corner, plus parameters that depend on the shape drawn.

```
    }
    root.getChildren().add(pane);
    stage.setScene(scene);
    stage.show();
  }
}
```

And there you go. All the art, of course, is in the choice of the suitable background image.

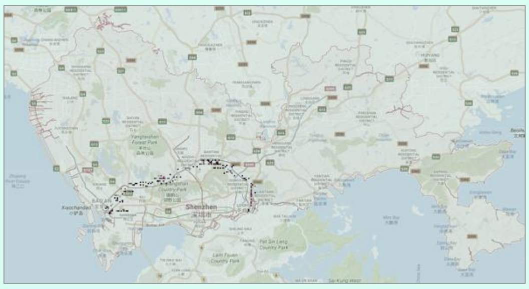


From stupid to usable in real life

Use a map as background

Draw routes

If instead of using Mona Lisa you use a map, you can create some really interesting applications with canvases (other than a drawing tool).



You could have for instance one Canvas per Metro line, stack all of them, and use buttons to make a line appear or disappear. That said, working with maps is not very easy because what you want to plot are usually places for which you know latitude and longitude.

Problems with maps

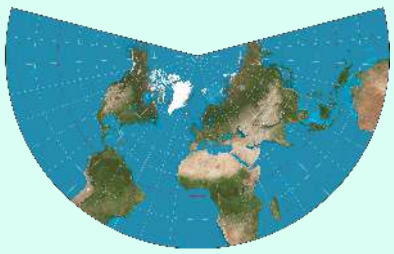
1. Projection

There are multiple ways to project a latitude and longitude on a flat screen, from the relatively simple cylindrical projection.

Problems with maps

1. Projection

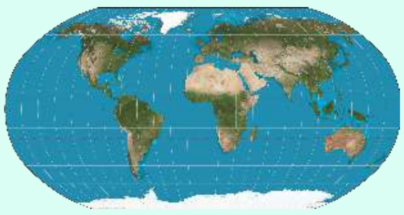
... or conical projection (that makes here southen regions look far bigger than they are)



Problems with maps

1. Projection

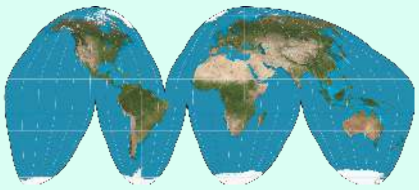
To projections that attempt to keep the surfaces right (but not the angles)



Problems with maps

1. Projection

To projections that try to achieve the "let's take an orange peel and lay it flat on the table" effect.

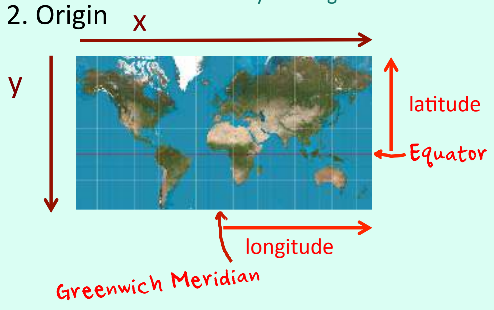


As you can guess, finding the (x,y) matching a given longitude and latitude can be mathematically challenging.

Problems with maps

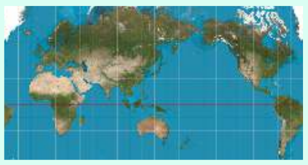
2. Origin

Additionally the origins are different ...

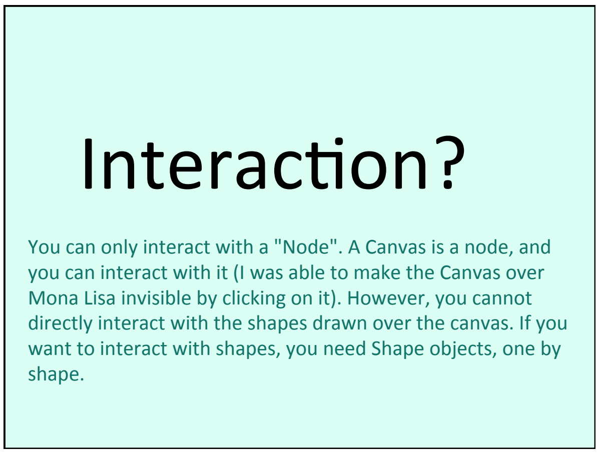


Problems with maps

2. Origin

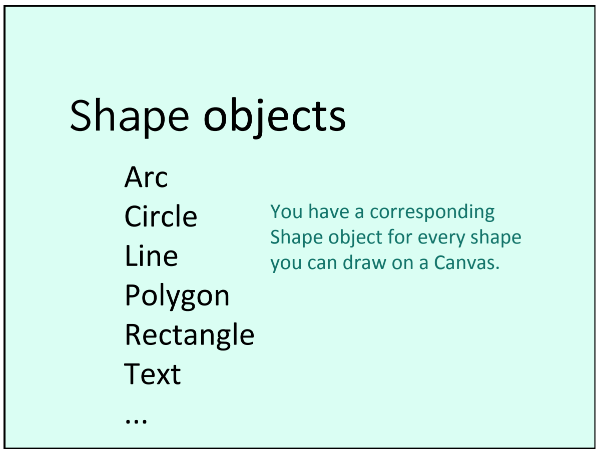


... and it may be further moved if you don't want to see the Greenwich Meridian (or the Equator) right in the middle. You need to write methods that know how to translate latitude and longitude depending on several parameters.



Interaction?

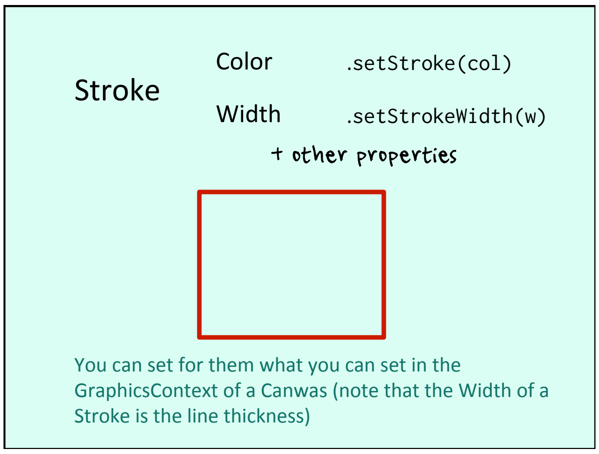
You can only interact with a "Node". A Canvas is a node, and you can interact with it (I was able to make the Canvas over Mona Lisa invisible by clicking on it). However, you cannot directly interact with the shapes drawn over the canvas. If you want to interact with shapes, you need Shape objects, one by shape.



Shape objects

- Arc
- Circle
- Line
- Polygon
- Rectangle
- Text
- ...

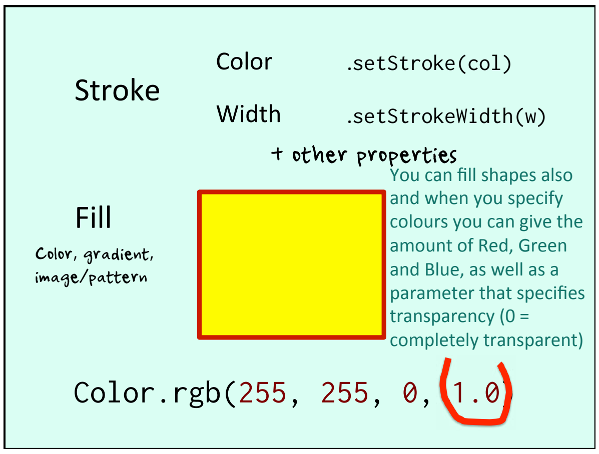
You have a corresponding Shape object for every shape you can draw on a Canvas.



Stroke	Color	`.setStroke(col)`
	Width	`.setStrokeWidth(w)`

+ other properties

You can set for them what you can set in the GraphicsContext of a Canwas (note that the Width of a Stroke is the line thickness)



Stroke	Color	`.setStroke(col)`
	Width	`.setStrokeWidth(w)`

+ other properties

Fill

Color, gradient, image/pattern

You can fill shapes also and when you specify colours you can give the amount of Red, Green and Blue, as well as a parameter that specifies transparency (0 = completely transparent)

```
Color.rgb(255, 255, 0, 1.0)
```

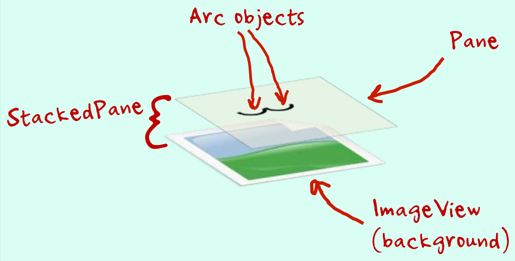
Shapes are nodes ...

CLICKABLE!

With Shapes you can click on every individual shape.

Let's redo the Mona Lisa example with shapes instead of a Canvas.

Stupid example, revisited



```
Arc arc = new Arc();
arc.setCenterX(0.42 * width);
arc.setCenterY(0.288 * height);
arc.setRadiusX(width / 36.0);
arc.setRadiusY(width / 36.0);
arc.setStartAngle(180.0);
arc.setLength(180.0);
arc.setType(ArcType.OPEN);
arc.setStroke(Color.BLACK);
arc.setStrokeWidth(height * 0.01);
arc.setFill(Color.rgb(255, 255, 255, 0.0));
shapePane.getChildren().add(arc);
```

More code than with a simple drawing (... but we could create a method for that)

Same with arc2

```
arc.setOnMouseClicked((e)->{
  Random rand = new Random();
  int r = rand.nextInt(256);
  int g = rand.nextInt(256);
  int b = rand.nextInt(256);
  Color col = Color.rgb(r, g, b);
  arc.setStroke(col);
  arc2.setStroke(col);
});
```

I associate the same action to a click on each arc, that changes the color for both arcs.

Canvas or individual shapes?

Depends on how many elements

Possible to check where a Canvas was clicked

Shapes are good if you have few of them. Otherwise everything can become slow (and it may become difficult to make sure you clicked at the right place).

3D Graphics

I won't talk about 3D graphics because it's beginning to become very specific to advanced applications. Let's just say that you have packages in JavaFX for 3D graphics as well.

Audio and Video

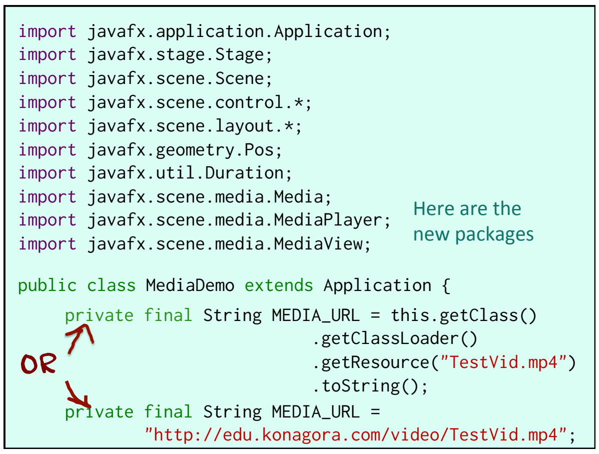
You can also play audio and video in JavaFX. It's not very different from images. With images you have an Image object, and the ImageView that shows it on screen. With audio and video, you have a Media object, you have a MediaView, and between the two you have a MediaPlayer object with controls allowing you to start, pause, stop, rewind and so forth.

Very much like Images

`Media`

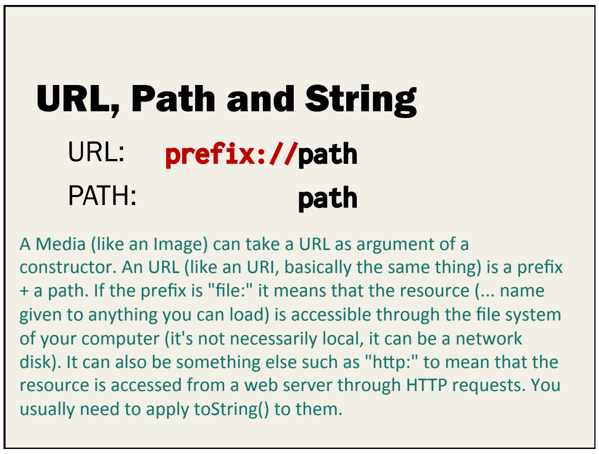
`MediaPlayer` $\leftarrow$ Controls

`MediaView`



```
import javafx.application.Application;
import javafx.stage.Stage;
import javafx.scene.Scene;
import javafx.scene.control.*;
import javafx.scene.layout.*;
import javafx.geometry.Pos;
import javafx.util.Duration;
import javafx.scene.media.Media;
import javafx.scene.media.MediaPlayer;
import javafx.scene.media.MediaView;

public class MediaDemo extends Application {
     private final String MEDIA_URL = this.getClass()
                                 .getClassLoader()
                                 .getResource("TestVid.mp4")
                                 .toString();
OR
     private final String MEDIA_URL =
             "http://edu.konagora.com/video/TestVid.mp4";
```

Here are the new packages

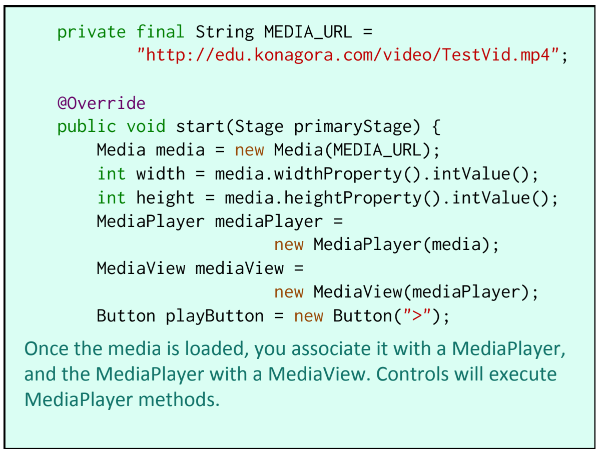


URL, Path and String

URL: **prefix://path**

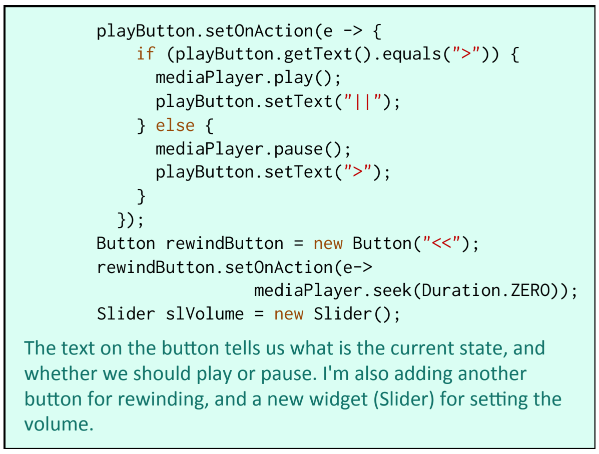
PATH: **path**

A Media (like an Image) can take a URL as argument of a constructor. An URL (like an URI, basically the same thing) is a prefix + a path. If the prefix is "file:" it means that the resource (... name given to anything you can load) is accessible through the file system of your computer (it's not necessarily local, it can be a network disk). It can also be something else such as "http:" to mean that the resource is accessed from a web server through HTTP requests. You usually need to apply toString() to them.



```
    private final String MEDIA_URL =
            "http://edu.konagora.com/video/TestVid.mp4";

    @Override
    public void start(Stage primaryStage) {
        Media media = new Media(MEDIA_URL);
        int width = media.widthProperty().intValue();
        int height = media.heightProperty().intValue();
        MediaPlayer mediaPlayer =
                          new MediaPlayer(media);
        MediaView mediaView =
                          new MediaView(mediaPlayer);
        Button playButton = new Button(">");
```

Once the media is loaded, you associate it with a MediaPlayer, and the MediaPlayer with a MediaView. Controls will execute MediaPlayer methods.



```
        playButton.setOnAction(e -> {
            if (playButton.getText().equals(">")) {
              mediaPlayer.play();
              playButton.setText("||");
            } else {
              mediaPlayer.pause();
              playButton.setText(">");
            }
          });
        Button rewindButton = new Button("<<");
        rewindButton.setOnAction(e->
                        mediaPlayer.seek(Duration.ZERO));
        Slider slVolume = new Slider();
```

The text on the button tells us what is the current state, and whether we should play or pause. I'm also adding another button for rewinding, and a new widget (Slider) for setting the volume.

```
        slVolume.setPrefWidth(150);
        slVolume.setMaxWidth(Region.USE_PREF_SIZE);
        slVolume.setMinWidth(30);
        slVolume.setValue(50);
        mediaPlayer.volumeProperty()
                   .bind(slVolume.valueProperty()
                                 .divide(100));
        HBox hBox = new HBox(10);
        hBox.setAlignment(Pos.CENTER);
        hBox.getChildren().addAll(playButton,
                                  rewindButton,
                                  new Label("Volume"),
                                  slVolume);
        BorderPane pane = new BorderPane();
```

Other than geometry (size) I give the range and initial value for the slider (0 to 100, initially 50) and "bind" it to the MediaPlayer. There is an implicit ChangeListener behind, to change the volume when the slider moves.

```
        BorderPane pane = new BorderPane();
        pane.setCenter(mediaView);
        pane.setBottom(hBox);
        Scene scene = new Scene(pane, 750, 500);
        primaryStage.setTitle("MediaDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
```

Controls are in a box, everything is added to a BorderPane (that controls placement as top/right/bottom/left and center) and we are ready to go.