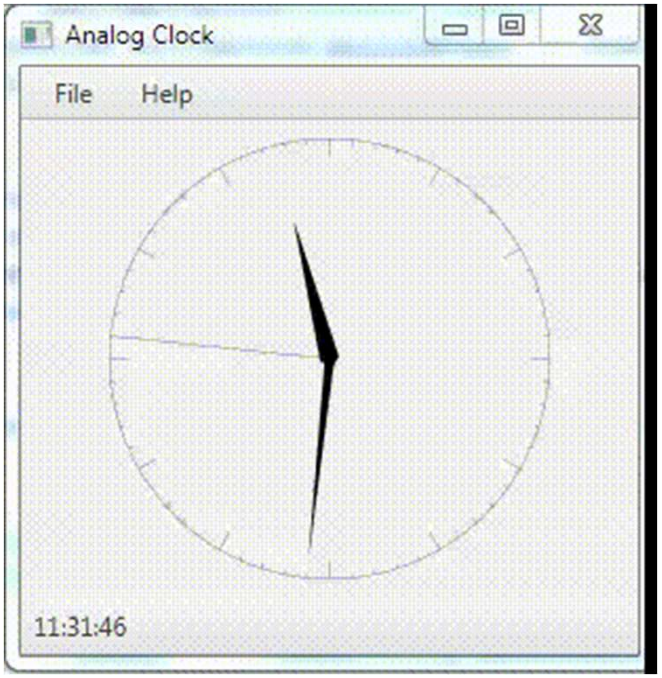
CSCD 370 - GUI Programming with JavaFX

Module 9: Properties, Binding, Media, Printing

Creating Your Own Properties



- The Clock in this example is a custom widget based on Region
  - similar to two of your lab assignments
- How can we make the status bar track the clock?
  - we could give the Clock widget a reference to the status bar, but that would be un-OOP-ish
  - we could give the Clock widget an (optional) callback method (via an interface). That would be OK, but would be a bit old-school-ish, requiring the Clock to check whether the callback has been provided, and then call it.
  - the new-school approach: give the Clock widget some Properties and use a Property change listener (or binding)

Creating Your Own Properties

```
public class Clock extends Region {
   private final Canvas mCanvas = new Canvas();

   // hrs, min, sec are Properties to illustrate binding
   private final IntegerProperty seconds
                 = new SimpleIntegerProperty();
   public IntegerProperty secondsProperty() {return seconds;}
   private final IntegerProperty minutes
                 = new SimpleIntegerProperty();
   public IntegerProperty minutesProperty() {return minutes;}
   private final IntegerProperty hours
                 = new SimpleIntegerProperty();
   public IntegerProperty hoursProperty() {return hours;}
   ...
   private void draw() {
      GregorianCalendar cal = new GregorianCalendar() ;
      hours.set(cal.get(Calendar.HOUR_OF_DAY));
      minutes.set(cal.get(Calendar.MINUTE));
      seconds.set(cal.get(Calendar.SECOND)) ;
      ...
   }
   ...
}
```

The Binding

- **The textProperty of a Label is of type StringProperty**
  - **which implements the interface called Property**
  - **here I *bind* it to a String assembled from the 3 Clock Properties (which implement the Observable interface, and may therefore be used in bindings)**
  - **one of the many useful methods available to numeric properties is asString()**
  - **toString() would NOT work here (try it)**

```
...
// Add a status bar
mStatus = new Label() ;
mStatus.textProperty().bind(
        clock.hoursProperty().asString("%02d")
        .concat(":").concat(clock.minutesProperty().asString("%02d"))
        .concat(":").concat(clock.secondsProperty().asString("%02d"))
        );
ToolBar toolBar = new ToolBar(mStatus) ;
root.setBottom(toolBar) ;
...
```

Things to be Aware of

- This is an example of *High Level Binding*
  - there are two ways to do it
- The so-called *Fluent API*
  - provides many chainable methods (on Observables) that assist in binding computations, such as: asString(), negate(), add(), lessthan(), or(), divide(), etc:

```
prop1.bind(prop2.multiply(prop2).multiply(Math.PI));
```

  - one way to discover what is available is to use the IDE's method prompter
- The *Bindings Class*
  - in which you explicitly use static methods of the Bindings class:

```
prop3.bind(Bindings.multiply(prop1, prop2));
```

- And of course you can mix them:

```
prop1.bind(Bindings.multiply(prop2,
                             prop2.multiply(Math.PI)));
```

Things to be Aware of

- **All Bindings in JavaFX support *lazy computation***
  - **if a property that is part of a dependency changes, the dependent result is not immediately calculated**
  - **it is marked as invalid, and recalculated only when the value is needed**
  - **obviously, the dependent value will be recalculated if you call getValue() on it, but ...**
  - **registering a ChangeListener on the dependent value it will force *eager computation***
  - **in this clock example the Label's textProperty is computed eagerly because it is used in an active Scene graph**
  - **you can listen for invalidation events as well as change events (an invalidation event does not force computation)**
- **Bidirectional Bindings are possible**
  - **but probably to be avoided**

Sound – 4 Types

- Much simplified since Swing (see example)
  - and much easier than in Android
- Simple Beep

```
Toolkit.getDefaultToolkit().beep();
```

- Platform Sounds
  - Toolkit can also be used to retrieve Desktop settings, including various desktop sounds
- AudioClip (aac, mp3, pcm, wav)
  - can be played with minimal latency, usable immediately
  - fire and forget, and can be played again prior to finishing
  - appropriate only for short clips as the uncompressed audio is held in memory
- Media and MediaPlayer
  - suitable for long audio as it decompresses into memory on the fly
  - built-in support for the usual operations (play, pause, stop, etc)
  - supports video (mp4, vp6) as well, in which case pair it with a MediaView

Platform Sounds

- **I've only tested this on Windows**
  - **desktop sound properties are returned as Runnables**

```
Runnable runnable = (Runnable)
   Toolkit.getDefaultToolkit()
          .getDesktopProperty("win.sound.exit");
if (runnable != null)
   runnable.run();
```

(not working in Win 10 on my machine)

win.sound.asterisk
win.sound.close
win.sound.default
win.sound.exclamation
win.sound.exit
win.sound.hand
win.sound.maximize
win.sound.menuCommand
win.sound.menuPopup
win.sound.minimize
win.sound.open
win.sound.question
win.sound.restoreDown
win.sound.restoreUp
win.sound.start

- **To get a listing of all Windows property names:**

```
String propnames[] = (String[])
   Toolkit.getDefaultToolkit().getDesktopProperty("win.propNames");
for (int i=0 ; i<propnames.length ; i++)
   System.out.println(propnames[i]);
```

AudioClip

- **Put your clip in the package directory**

```
// Must be in the package directory, which is inside
// the src directory (and is the name of your package).
// Directly in the src directory is not an option like
// it is for Images.
AudioClip clip = new
AudioClip(getClass().getResource("myhorn.wav").toString());
clip.play();
```

- **See also:**
  - **stop()**
  - **setCycleCount(): the number of times the clip is played on each call to play()**
  - **there are set and get methods to control: (relative) volume, balance, rate (1/8th to 8x normal),**
    pan (similar to balance, but mixes both channels left/right)
    priority (relative to other clips)
  - **there are overloads of play() that take (relative) volume, balance, rate, pan (stereo width), and priority**

Media and MediaPlayer

```
// here we're relative to the project root
String path = "InTheMood.mp3";

// this converts it to an absolute path
Media media = new Media(new File(path).toURI().toString());

// attach it to a media player
mediaPlayer = new MediaPlayer(media);
```

(see AudioEx)

- **MediaPlayer methods of particular interest:**
  - **play(), pause()**
  - **stop(): in which case play() starts over at the beginning**
  - **seek(), setStartTime(), setStopTime()**
  - **setAutoPlay(), setCycleCount(), setRate()**
  - **setVolume(), setMute(), setBalance(),**
  - **setOnEndOfMedia(), setOnRepeat(), etc.**
  - **interestingly, no setPan()**

Video

- **Media and MediaPlayer also handle video:**

```
String path = "MyVideo.mp4";
// convert it to an absolute path
Media media = new Media(new File(path).toURI().toString());

// attach it to a media player
mediaPlayer = new MediaPlayer(media);
```

- **The only extra step is to tie the MediaPlayer to a MediaView**
  - **and maybe go thru some gyrations to control scaling**

```
mediaView = new MediaView(mediaPlayer);

// use available width, set height to HD aspect ratio
mediaView.setFitWidth(sceneWidth);
mediaView.setFitHeight(sceneWidth/16.0*9.0);
```

- **see example**

Printing

- **Printing a Node on a single page is easy**
  - pagination is application-dependent and can get considerably more tricky
  - Java provides no support for Print Preview, and not a lot of help out there, so I'll show you my simple way of doing that
- **Printer class**
  - static methods getAllPrinters() and getDefaultPrinter()
  - non-static method to createPageLayout()
- **PrinterJob class**
  - static method to createPrinterJob(Printer p)
  - job.getJobSettings().setPageLayout(PageLayout pl)
  - job.showPrintDialog() to give the user a chance to select a printer or cancel the job
  - job.printPage(Node n) returns a success code
  - on success, call job.endJob() to actually spool the job
  - you'll likely want to scale your root Node to the page size

Print Example (Canvas)

```
// need a PageLayout in order to scale our stuff to the page
Printer printer = Printer.getDefaultPrinter();
PageLayout pageLayout = printer.createPageLayout(Paper.NA_LETTER,
          PageOrientation.PORTRAIT, Printer.MarginType.DEFAULT);

// scale our drawing so that it fills the page width or height
double destW = pageLayout.getPrintableWidth() ;
double destH = pageLayout.getPrintableHeight() ;
double srcW = mPermCanvas.getWidth();
double srcH = mPermCanvas.getHeight();
double scaleX =  destW / srcW ;
double scaleY =  destH / srcH ;
double scale = (scaleX>scaleY) ? scaleY : scaleX ;

// need a Canvas to draw the scaled stuff on
Canvas temp = new Canvas(destW, destH) ;
WritableImage image = mPermCanvas.snapshot(null, null);
GraphicsContext gc = temp.getGraphicsContext2D() ;
gc.drawImage(image, 0, 0, scale*srcW, scale*srcH) ;

// now create a printer job and pop up the print dialog
PrinterJob job = PrinterJob.createPrinterJob(printer);
if (job != null && job.showPrintDialog(mPermCanvas.getScene().getWindow())) {
   job.getJobSettings().setPageLayout(pageLayout);  // another get vs. set
   boolean success = job.printPage(temp);
   if (success)
      job.endJob();  // this is what actually spools the job
}
```

Print Preview Example

```
(...) // default printer, pageLayout, widths, heights, and scales as previous
double leftM = pageLayout.getLeftMargin();
double topM = pageLayout.getTopMargin() ;
double rightM = pageLayout.getRightMargin() ;
double bottomM = pageLayout.getBottomMargin() ;
double totalW = destW + leftM + rightM ;
double totalH = destH + topM + bottomM ;

if (preview) {
   temp = new Canvas(totalW, totalH) ;  // show the whole page for preview
   WritableImage image = mPermCanvas.snapshot(null, null);
   GraphicsContext gc = temp.getGraphicsContext2D() ;
   gc.setFill(Color.WHITE);
   gc.fillRect(0, 0, totalW, totalH);
   gc.setStroke(Color.LIGHTGRAY);
   gc.strokeRect(leftM, topM, destW, destH);  // outline the paper boundaries
   gc.drawImage(image, leftM, topM, scale*srcW, scale*srcH) ;  // include margins
   Alert alert = new Alert(Alert.AlertType.INFORMATION) ;
   alert.setTitle("Print Preview");
   alert.setHeaderText("Print Preview (with margins)");
   alert.getDialogPane().setExpandableContent(new ScrollPane(temp));
   alert.getDialogPane().setExpanded(true);
   ButtonType butPrint = new ButtonType("Print") ;
   ButtonType butCancel = new ButtonType("Cancel", ButtonBar.ButtonData.CANCEL_CLOSE);
   alert.getButtonTypes().setAll(butPrint, butCancel) ;
   Optional<ButtonType> result = alert.showAndWait() ;
   if (result.get()==butCancel) return ;
}
(...) // continue to print as usual
```

Print Preview Example

- this example gives a Preview for the default Printer
- but the user can still change printers after pressing Print, which shows the print dialog