Announcements

- PS 10 (last one of the semester) is ready on the web
- Try to get some help from me and tutors
- Reading assignment for this slide set: Chapter 15

Event-Driven Programming and Animations

Motivations

- Suppose you wish to write a GUI program that lets the user enter a loan amount, annual interest rate, and number of years and click the Calculate button to obtain the monthly payment and total payment.
- You have to use event-driven programming to write the code to respond to the button-clicking event.
- Procedurally written programs execute programs in a linear manner that is insufficient to handle user interaction of window-based programs

Event handling

- Consider a simple graphical program containing two buttons
- We wish to write code that will be executed when the buttons are processed
- The button is an event-source object where the action originates
- You need to create an object capable of handling the action event on a button. This object is called an event handler
Event handling (cont.)

- Event-handling code can be implemented in several ways that we will explore.
- One way is to encapsulate an event-handling method in a new class that implements the `EventHandler<ActionEvent>` interface.
- To handle button presses we need to implement the method `handle(ActionEvent e)`.
- Then we construct an object of our new class and associate it with the button calling the `setOnAction` method of the button object.

Taste of event-driven programming

The example displays a button in the frame.
A message is displayed on the console when a button is clicked.

- See `HandleEvent.java`

Event classes

JavaFX event classes are in the JavaFX.event package.

<table>
<thead>
<tr>
<th>User Action</th>
<th>Source Object</th>
<th>Event Type Fixed</th>
<th>Event Enrichment Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click a button</td>
<td>Button</td>
<td>ActionEvent</td>
<td><code>setOnAction(EventHandler&lt;ActionEvent&gt;)</code></td>
</tr>
<tr>
<td>Press Enter in a text field</td>
<td>TextField</td>
<td>ActionEvent</td>
<td><code>setOnAction(EventHandler&lt;ActionEvent&gt;)</code></td>
</tr>
<tr>
<td>Check or uncheck</td>
<td>Radio button</td>
<td>ActionEvent</td>
<td><code>setOnAction(EventHandler&lt;ActionEvent&gt;)</code></td>
</tr>
<tr>
<td>Check or uncheck</td>
<td>Checkbox</td>
<td>ActionEvent</td>
<td><code>setOnAction(EventHandler&lt;ActionEvent&gt;)</code></td>
</tr>
<tr>
<td>Select a new item</td>
<td>ComboBox</td>
<td>ActionEvent</td>
<td><code>setOnAction(EventHandler&lt;ActionEvent&gt;)</code></td>
</tr>
<tr>
<td>Mouse pressed</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td><code>setOnMousePressed(EventHandler&lt;MouseEvent&gt;)</code></td>
</tr>
<tr>
<td>Mouse released</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td><code>setOnMouseReleased(EventHandler&lt;MouseEvent&gt;)</code></td>
</tr>
<tr>
<td>Mouse clicked</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td><code>setOnMouseClicked(EventHandler&lt;MouseEvent&gt;)</code></td>
</tr>
<tr>
<td>Mouse entered</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td><code>setOnMouseEntered(EventHandler&lt;MouseEvent&gt;)</code></td>
</tr>
<tr>
<td>Mouse exited</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td><code>setOnMouseExited(EventHandler&lt;MouseEvent&gt;)</code></td>
</tr>
<tr>
<td>Mouse moved</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td><code>setOnMouseMoved(EventHandler&lt;MouseEvent&gt;)</code></td>
</tr>
<tr>
<td>Mouse dragged</td>
<td>Node, Scene</td>
<td>MouseEvent</td>
<td><code>setOnMouseDragged(EventHandler&lt;MouseEvent&gt;)</code></td>
</tr>
<tr>
<td>Key pressed</td>
<td>Node, Scene</td>
<td>KeyEvent</td>
<td><code>setOnKeyPressed(EventHandler&lt;KeyEvent&gt;)</code></td>
</tr>
<tr>
<td>Key released</td>
<td>Node, Scene</td>
<td>KeyEvent</td>
<td><code>setOnKeyReleased(EventHandler&lt;KeyEvent&gt;)</code></td>
</tr>
<tr>
<td>Key typed</td>
<td>Node, Scene</td>
<td>KeyEvent</td>
<td><code>setOnKeyDown(EventHandler&lt;KeyEvent&gt;)</code></td>
</tr>
</tbody>
</table>

Common event types
Registering event handlers

- Java uses a delegation-based model for event handling: a source object fires an event, and an object interested in the event handles it, called an event handler or event listener.
- For an object to be a handler for an event on a source object, two things are needed:
  - The handler object must be an instance of the corresponding event-handler interface to ensure that the handler has the correct method for processing the event.
  - The handler object must be registered by the source object that might fire the event.

The delegation model: example

```java
Button btOK = new Button("OK");
OKHandlerClass handler = new OKHandlerClass();
btOK.setOnAction(handler);
```

The delegation model

Event objects

- Contains information about the event, such as:
  - (x, y) location of the mouse when a button was clicked
  - event source that was interacted with (EventObject.getSource())
  - what key on the keyboard was pressed among others
- Listeners use them to properly respond to the event
- Examples on the next slides
First version for ControlCircle (no listeners)

Now let us consider writing a program that uses two buttons to control the size of a circle.

- See ControlCircleWithoutEventHandling.java

Second version for ControlCircle (with listener for Enlarge)

Now let us consider writing a program that uses two buttons to control the size of a circle.

- See ControlCircle.java

Loan calculator

- See LoanCalculator.java
- See Loan.java

MouseEvent

- See MouseEventDemo.java
The KeyEvent Class

```java
KeyEvent
+getCharacter(): String
+getCode(): KeyCode
+getText(): String
+isAltDown(): boolean
+isControlDown(): boolean
+isMetaDown(): boolean
+isShiftDown(): boolean
```

- `getCharacter()` returns the character associated with the key in this event.
- `getCode()` returns the key code associated with the key in this event.
- `getText()` returns a string describing the key code.
- `isAltDown()` returns true if the Alt key is pressed on this event.
- `isControlDown()` returns true if the Control key is pressed on this event.
- `isMetaDown()` returns true if the mouse Meta button is pressed on this event.
- `isShiftDown()` returns true if the Shift key is pressed on this event.

Example: Control circle with mouse and key

- See ControlCircleWithMouseAndKey.java

The KeyCode constants

<table>
<thead>
<tr>
<th>Constant</th>
<th>Description</th>
<th>Constant</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>HOME</td>
<td>The Home key</td>
<td>CONTROL</td>
<td>The Control key</td>
</tr>
<tr>
<td>END</td>
<td>The End key</td>
<td>SHIFT</td>
<td>The Shift key</td>
</tr>
<tr>
<td>PAGE_UP</td>
<td>The Page Up key</td>
<td>BACK_SPACE</td>
<td>The Backspace key</td>
</tr>
<tr>
<td>PAGE_DOWN</td>
<td>The Page Down key</td>
<td>CAPS</td>
<td>The Caps Lock key</td>
</tr>
<tr>
<td>UP</td>
<td>The up-arrow key</td>
<td>NUM_LOCK</td>
<td>The Num Lock key</td>
</tr>
<tr>
<td>DOWN</td>
<td>The down-arrow key</td>
<td>ENTER</td>
<td>The Enter key</td>
</tr>
<tr>
<td>LEFT</td>
<td>The left-arrow key</td>
<td>UNDEFINED</td>
<td>The keyCode unknown</td>
</tr>
<tr>
<td>RIGHT</td>
<td>The right-arrow key</td>
<td>F1 to F12</td>
<td>The function keys from F1 to F12</td>
</tr>
<tr>
<td>ESCAPE</td>
<td>The Esc key</td>
<td>0 to 9</td>
<td>The number keys from 0 to 9</td>
</tr>
<tr>
<td>TAB</td>
<td>The Tab key</td>
<td>A to Z</td>
<td>The letter keys from A to Z</td>
</tr>
</tbody>
</table>

Animation

JavaFX provides the Animation class with the core functionality for all animations.

```java
Animation
@Autowired
private Animation animation;
```

- `autoReverse: BooleanProperty` defines whether the animation reverses direction on alternating cycles.
- `cycleCount: IntegerProperty` defines the number of cycles in this animation.
- `rate: DoubleProperty` defines the speed and direction for this animation.
- `status: ReadOnlyObjectProperty` denotes the status of the animation.

Functions:
- `pause(): void` pauses the animation.
- `play(): void` plays the animation from the current position.
- `stop(): void` stops the animation and resets the animation.
PathTransition

The **PathTransition** class animates the moves of a node along a path from one end to the other over a given time.

```
PathTransition
```

```
 PathTransition(Duration duration, Node node, OrientationType orientation, PathTransition.OrientationType path, Object type)
 PathTransition()
```

```
 fadeTransition(Duration duration, Node node, DoubleProperty fromValue, DoubleProperty byValue)
```

Timeline

**PathTransition** and **FadeTransition** define specialized animations.

The **Timeline** class can be used to program any animation using one or more **KeyFrames**.

Each **KeyFrame** is executed sequentially at a specified time interval.

**Timeline** inherits from **Animation**.

FadeTransition

The **FadeTransition** class animates the change of the opacity in a node over a given time.

```
FadeTransition(Duration duration, Node node, DoubleProperty fromValue, DoubleProperty byValue)
```

Clock Animation

- See **ClockAnimation.java**
- See **ClockPane.java**
Bouncing ball

- See `BounceBallControl.java`
- See `BallPane.java`