1 JavaFX Events and Animation

CST141

2 Event Handling

- GUI components generate *events* when users interact with controls
- Typical events (there are dozens) include:
  - Clicking the mouse
  - Moving the mouse
  - Typing in a text box (TextField)

3 Event Listeners

- To process an event, the programmer must:
  - Define one or more event handler classes/methods
  - Register (declare) an event listener
- When an event occurs, GUI component notifies the listener by *calling* the event's handling method(s)

4 The EventHandler Interface (Page 1)

- EventHandler is a Java API interface used to manage *event listening* and *event handling* for Buttons and other JavaFX GUI components
- Objects instantiated from a class that implements the EventHandler interface "are" event handlers, e.g. "*Is an* EventHandler"
- Imported from javafx.event package:

  import javafx.event.EventHandler;

5 The EventHandler Interface (Page 2)

- Format (*nested* inside the Application class):

  private class *EventHandlerClassName* implements EventHandler<ActionEvent>

  { …

  - implements rather than extends
- Example:

  private class ButtonEventHandler implements EventHandler<ActionEvent>

  { …

6 The ActionEvent Class (Page 1)

- ActionEvent is the generic <*SubType*> for the interface EventHandler and represents action information for a GUI object like a Button
- Imported from javafx.event package:

  import javafx.event.ActionEvent;

7 The ActionEvent Class (Page 2)

- Example:

  private class ButtonEventHandler implements EventHandler<ActionEvent>

  {

```
    @Override
      public void handle(ActionEvent e)
    { …
```

8 □ The handle Method (Page 1)

- The abstract method handle() is a member of the EventHandler interface and *must be defined* in any class that implements it
- If a user clicks a Button and "event listening" is activated for that object, the method handle() automatically is called

9 □ The handle Method (Page 2)

- A parameter variable "e" of type ActionEvent is defined for the method and provides access to ActionEvent methods and properties
- Example:

```
private class ButtonEventHandler implements EventHandler<ActionEvent>
{
    @Override
      public void handle(ActionEvent e)
    { …
```

10 □ Instantiating an EventHandler Object

- To instantiate the object, an EventHandler *class* must have been defined previously
- Format:

  EventHandlerClass eventHandlerObject = new *EventHandlerConstructor*();
- Example:

```
ButtonEventHandler eventHandler = new ButtonEventHandler();
```

11 □ The setOnAction Method (Page 1)

- Method of a Button (and other "action listener" GUI components) that assign an EventHandler object to the component
- The "event handler" object instantiated from the EventHandler is the *argument* to the method
- This method effectively *activates* event listening
- Must be executed for every GUI component that will be an *event listener*

12 □ The setOnAction Method (Page 2)

- Format:

  guiComponentObject.setOnAction(*eventHandlerObject*);
- Example:

```
button.setOnAction(eventHandler);
```

  – The GUI component 'button' is a Button

13 □ Steps to Create Event Handler (*Summary*)

- The event handler method:
  1. Create a "nested" class that implements interface EventHandler (within JavaFX

Application class)

2. Create a method handle() in that class

- Register event listening in the start() method:

3. Instantiate an object from the class that implements the interface EventHandler
4. For each Button call the method setOnAction()

15 The getSource Method

- Method of an ActionEvent object that "points" to *address* of the object *that initiated the event*
- Format:

  actionEventObject.getSource()

  – *actionEventObject* is the parameter variable "e" in method handle()
- Example:

```
public void handle(ActionEvent e)
{
   if (e.getSource() == buttonOK) ...
```

17 The getText Method

- Returns the String property currently stored in a TextField (or another GUI component that has a text property) object
- For a TextField, the text property is the value currently displayed in the text box
- Format:

  textFieldObject.getText()
- Example:

```
String sFirst = firstNumber.getText();
```

18 The setText Method

- Sets the contents of a TextField object (or some other GUI component that has a text property) to a *new value*
- Format:

  textFieldObject.setText(*string*)
- Example:

```
resultField.setText(resultString);
```

19 The setEditable Method

- Sets a boolean property that determines if a TextField object may be edited by a user
- Frequently is set to false if the object will be used exclusively for *output*
- Format:

  textFieldObject.setEditable(true/false)
- Example:

```
resultField.setEditable(false);
```

21 The selectAll Method

- A method of class TextField (inherited from class TextComponent ← TextField) that

selects all the text in the object
- – As if it had been selected with a mouse

- Format:

```
node.selectAll();
```

- Example:

```
inputAge.selectAll();
```

- – In this example variable inputAge is a TextBox

28 Anonymous Inner Classes(Page 1)

- An anonymous inner class is an *event handler* without a name
  - – Located inside the definition of the application window in the start() method
- Defined within the setOnAction() method
- Combines creating the object (Button or other object) with defining of the class

29 Anonymous Inner Classes(Page 2)

- Format:

```
ClassName object = new ConstructorName(…);
object.setOnAction(new EventHandler<ActionEvent>
{
   @Override
   public void handle(ActionEvent event)
   {
      statements
   }
}
);
```

30 Anonymous Inner Classes(Page 3)

- Format:

```
Button buttonOK = new Button("OK");
buttonOK.setOnAction(new EventHandler<ActionEvent>
{
   @Override
   public void handle(ActionEvent event)
   {
      System.out.println("OK clicked");
   }
}
);
```

32 Lambda Expression Event Handling (Page 1)

- Lambda expression event handling is a new feature in Java 8 which *replaces* the anonymous inner class with a *more consise syntax*
- Also defined within setOnAction() method combining creation of object (Button or

other node) with a single *method* that replaces the class

33 □ Lambda Expression Event Handling (Page 2)

- Format:

```
ClassName object = new ConstructorName(…);
object.setOnAction( (e) ->
{
   statements
}
);
```

  – The parameter variable e (or other programmer-defined variables) may be explicitly declared by type or the type inferred by the compiler)

34 □ Lambda Expression Event Handling (Page 3)

- Example:

```
Button buttonOK = new Button("OK");
buttonOK.setOnAction( (e) ->
{
   System.out.println("OK clicked");
}
);
```

37 □ Lambda Expression Event Handling (Page 4)

- The Lambda expression may point directly to a *method call*
- Also the parameter variable e does not have to be wrapped inside (parentheses)

38 □ Lambda Expression Event Handling (Page 5)

- Format:

```
ClassName object = new ConstructorName(…);
object.setOnAction( e -> methodCall() );
```

- Examples:

```
Button buttonOK = new Button("OK");
buttonOK.setOnAction( e -> JOptionPane.showMessageDialog(null, "OK button was
     clicked") );

Button buttonMale = new Button("Male");
buttonMale.setOnAction( e -> maleUser() );
```

41 □ The PathTransition Class (Page 1)

- Used to create a “path” which is the “border” of one shape node along which another node travels, e.g.:
  – A Rectangle node object traverses along the outer border of a Circle node object
  – An ImageView node object displaying an image traverses along a Line node object
- Imported from javafx.animation package:

import javafx.animation.PathTransition;

42 □ The PathTransition Class (Page 2)

- Format to instantiate a PathTransition object:

  PathTransition *object* = new PathTransition();
- Example:

  PathTransition path = new PathTransition();

43 □ The setDuration Method (Page 1)

- For a PathTransition object, sets the amount of time that it takes the node object to traverse the "path" one time
- Amount of time is measure in milliseconds (1000 milliseconds is one second)

  – Default is 400 milliseconds (0.4 seconds)

44 □ The setDuration Method (Page 2)

- The setDuration() method takes an argument from one of the methods of class Duration:
- These methods include:

  Duration.millis(*double*) // milliseconds

  Duration.seconds(*double*)

  Duration.minutes(*double*)

  Duration.hours(*double*)
- Class is imported from javafx.util package:

  import javafx.util.Duration;

45 □ The setDuration Method (Page 3)

- Format:

  pathTransitionObject.setDuration(Duration.*methodName*(*double*));
- Example:

  path.setDuration(Duration.millis(5000));

  – 5000 milliseconds is five seconds

  path.setDuration(Duration.seconds(5));

  – Same as previous

  –

46 □ The setPath Method

- For a PathTransition object, sets (names) the node (e.g. Circle, Rectangle, Line, etc.) object that is the "path" for another node object to follow
- Format:

  pathTransitionObject.setPath(*nodeObject*);

  – *nodeObject* becomes the "path"
- Example:

  path.setPath(circle);

47 □ The setNode Method

- For a PathTransition object, sets (names) the animated node (e.g. Circle, Rectangle, etc.) that follows the "path"
- Format:

  pathTransitionObject.setNode(*nodeObject*);

  – *nodeObject* is the node that follows the "path"
- Example:

  path.setNode(rectangle);

48 The setOrientation Method (Page 1)

- For a PathTransition object, sets the "upright orientation" of the node object along path
- The method takes an enum constants from class PathTransition.OrientationType:

  PathTransition.OrientationType.NONE

  - The node stays upright (default)

  PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT

  - The node rotates to keep perpendicular with the path

49 The setOrientation Method (Page 2)

- Format:

  pathTransitionObject.setOrientation(*orientationType*);
- Examples:

  path.setOrientation(PathTransition.OrientationType.ORTHOGONAL_TO_TANGENT);

  path.setOrientation(PathTransition.OrientationType.NONE);

50 The setCycleCount Method (Page 1)

- For a PathTransition object, sets the number of times traversal of the "path" will be repeated

  – Default is 1
- Method is inherited from superclass Animation
- Format:

  pathTransitionObject.setCycleCount(*int*);

  – *int* is the number of repetitions
- Examples:

  path.setCycleCount(5);

51 The setCycleCount Method (Page 2)

- The INDEFINITE constant from class Timeline specifies that an animation repeats indefinitely
- Class imported from javafx.animation package:

  import javafx.animation.Timeline;
- Format:

  pathTransitionObject.setCycleCount(Timeline.INDEFINITE);
- Example:

  path.setCycleCount(Timeline.INDEFINITE);

52 □ The setAutoReverse Method

- For a PathTransition object, sets boolean property which determines whether the animation reverses direction on each alternating cycle
  - Default is false (in which case the animation loops)
- Method is inherited from superclass Animation
- Format:

  pathTransitionObject.setAutoReverse(true / false);
- Examples:

  path.setAutoReverse(true);

53 □ The play Method

- For a PathTransition object, starts animation running (has no effect if already running)
- Method is inherited from superclass Animation
- Format:

  pathTransitionObject.play();
- Examples:

  path.play();

54 □ The pause Method

- For a PathTransition object, pauses running animation (has no effect if not currently running)
- Continues *from same point* when it runs again
- Method is inherited from superclass Animation
- Format:

  pathTransitionObject.pause();
- Examples:

  path.pause();

55 □ The stop Method

- For a PathTransition object, stops a running animation and *resets* play to *back initial position* (has no effect if not currently running)
- Method is inherited from superclass Animation
- Format:

  pathTransitionObject.stop();
- Examples:

  path.stop();

56 □ The setOnMousePressed Method

- For shape nodes (Circle, Rectangle, etc.) defines an event handler that responds when a user clicks and holds down the mouse on that object
- Format using a *lambda expression*:

  node.setOnMousePressed(e -> *method*());
  - Could be any *method*, even programmer-defined class

- Example:
  circle.setOnMousePressed(e -> path.pause());

57 □ The setOnMouseReleased Method

- For shape nodes (Circle, Rectangle, etc.) defines an event handler that responds when a user release the mouse from that object
- Format using a *lambda expression*:
  node.setOnMouseReleased(e -> *method*());
  – Could be any *method*, even programmer-defined class
- Example:
  circle.setOnMouseReleased(e -> path.play());

59 □ Subclasses of Pane (Page 1)

- Objects instantiated from a class that extends class Pane contain JavaFX node objects and can be placed directly into a Scene
- Format:
  public class *ClassName* extends Pane { … }
- Example:
  public class StickMan extends Pane { … }

60 □ Subclasses of Pane (Page 2)

- Example to instantiate the object:
  StickMan stickman = new StickMan();
- Example to place Pane object into Scene:
  Scene scene = new Scene(stickMan, 300, 300);

61 □ The KeyEvent Class (Page 1)

- The KeyEvent class is a generic *subtype* that provides functionality for JavaFX applications to respond to keyboard events
  – Alternative to ActionEvent class for mouse events
- Imported from javafx.scene.input package:
  import javafx.scene.input.KeyEvent;

62 □ The KeyEvent Class (Page 2)

- The method setOnKeyPressed() "attaches" an event handler for the keyboard to a JavaFX object
- Format with a *lambda expression*:
  object.setOnKeyPressed(*e* -> *keyEventHandlerMethod* (*e*));
  – *e* is the KeyEvent parameter
- Example:
  scene.setOnKeyPressed(e -> moveStickMan(e));

63 □ The KeyEvent Class (Page 3)

- For keyboard events, class KeyEvent is the object variable type for the "event" parameter in method handler's header

- Format:
  public void *keyEventHandlerMethod*(KeyEvent e)
  { … }
- Example:
  public void moveStickMan(KeyEvent e)
  { … }

64 □ The getCode Method (Page 1)

- For the ActionEvent parameter of method handle(), the getCode() method returns a code for non-displaying keyboard keys, e.g.:
  - DOWN, UP, ALT, CONTROL, etc.
- Format:
  e.getCode()

65 □ The getCode Method (Page 2)

- Example:

```
if (e.getCode() == DOWN)
{
   y += 10;
}
else if (e.getCode() == UP)
{
   y -= 10;
}
else if (e.getCode() == LEFT)
{
   x -= 10;
}
else if (e.getCode() == RIGHT)
{
   x += 10;
}
```

67 □ The switch Statement (Page 1)

- A Java structure that can be used to implement a *linear nested* function
  - In place of: (if … else if … else if …)
- The value of a *single* variable or expression can be tested for multiple “equal to” values

68 □ The switch Statement (Page 2)

- The keyword break terminates execution of the switch structure when a true code block finishes executing
  - Otherwise program execution will “crash” into subsequent cases
- A final optional default case may be specified and executes if all the previous cases are false

69 □ **Format of switch Structure**

```
switch (testExpression)
{
   case value:
      statement(s) to be executed when
        this case is true go here;
      break;
   case value:
      statement(s) to be executed when
        this case is true go here;
      break;
  [case … ]

  [default:
      statement(s) to be executed when
        no case is true go here;]
}
```

70 □ **Example of switch Structure**

```
switch ( e.getCode() )
{
   case DOWN:
      y += 10;
      break;
   case UP:
      y -= 10;
      break;
   case LEFT:
      x -= 10;
      break;
   case RIGHT:
      x += 10;
      break;
}
```

71 □ **Equivalent of switch**

```
if (e.getCode() == DOWN)
{
   y += 10;
}
else if (e.getCode() == UP)
{
   y -= 10;
}
else if (e.getCode() == LEFT)
{
   x -= 10;
}
else if (e.getCode() == RIGHT)
{
   x += 10;
}
```

72 □ Testing for More than One true case in a switch

- Two or more true cases may evaluate as being equivalent as follows:

```
switch ( e.getCode() )
{
   case LEFT:
   case BACKSPACE:
      x -= 10;
      break;
   ...
}
```

  – Evaluates as true if e.getCode() returns either LEFT *or* BACKSPACE

74 □ The Ternary Operator (Page 1)

- The ternary operator (?) returns one of two values depending upon the value of a *booleanExpression*
- It can be used as an alternative to Java if/else syntax, but it actually goes beyond that
  - It can be used on the right side of Java assignment statements as well as in other operations
- Format:

  booleanExpression ? *valueIfTrue* : *valueIfFalse*

75 □ The Ternary Operator (Page 2)

- Example 1:

```
int x
x = (x > 400) ? 0 : x + 5;
```

- Equivalent:

```
if (x > 400)
{
   x = 0;
}
else
{
    x = x + 5;
}
```

76 The Ternary Operator (Page 3)

- Example 2:

```
System.out.println("The x-coordinate is " + (x > 400) ? 0 : x + 5;
```