In event-driven programming, code is executed upon activation of events.

Operating Systems constantly monitor events
- Ex: keystrokes, mouse clicks, etc...

The OS:
- sorts out these events
- reports them to the appropriate programs
WHERE DO WE COME IN?

For each control (button, combo box, etc.):
  ─ define an event handler
  ─ construct an instance of event handler
  ─ tell the control who its event handler is

Event Handler?
  ─ code with response to event
  ─ a.k.a. event listener
An event source is a GUI control

- JavaFX: Button, ChoiceBox, ListView, etc.

- different types of sources:
  - can detect different types of events
  - can register different types of listeners (handlers)
JAVA’S EVENT HANDLING

When the user interacts with a control (source):

— an event object is constructed

— the event object is sent to all registered listener objects

— the listener object (handler) responds as you defined it to
EVENT LISTENERS (EVENT HANDLER)

Defined by you, the application programmer

- you customize the response
- How?
  - Inheritance & Polymorphism

You define your own listener class

- implement the appropriate interface
- define responses in all necessary methods
EVENT OBJECTS

Contain information about the event

Like what?
- location of mouse click
- event source that was interacted with
- etc.

Listeners use them to properly respond
- different methods inside a listener object can react differently to different types of interactions
public class HandleEvent extends Application {
    public void start(Stage primaryStage) {
        HBox pane = new HBox(10);
        Button btOK = new Button("OK");
        Button btCancel = new Button("Cancel");
        OKHandler handler1 = new OKHandler();
        btOK.setOnAction(handler1);
        CancelHandler handler2 =
            new CancelHandler();
        btCancel.setOnAction(handler2);
        pane.getChildren().addAll(btOK, btCancel);
        Scene scene = new Scene(pane);
        primaryStage.setScene(scene);
        primaryStage.show();
    ...
}
```java
class OKHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}
class CancelHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Cancel button clicked");
    }
}
```
HANDLING GUI EVENTS

Source object: Button

Event object: ActionEvent

- Listener objects: OkHandler, CancelHandler
Event objects have info **about** the event:
— e.g. the **source object** (via `getSource()`)

EventObject subclasses are for special events:

- such as button actions
- window events
- component events
- mouse movements
- keystrokes
# SELECTED USER ACTIONS AND HANDLERS

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

A listener class is typically for a particular GUI component (e.g., one button).

- Any object instance of the inner handler class has access to all GUI fields of the outer class.

- How can the (non-static) inner class access the outer class?

- It will not be shared by other applications.
public class OuterClass {
    private int outerData = 0;
    private InnerClass iC1;
    private InnerClass iC2;

    public OuterClass() {
        iC1 = new InnerClass();
        iC2 = new InnerClass();
    }

    public void update() {
        iC1.updateFromInner();
        iC2.updateFromInner();
        iC2.updateFromInner();
    }

    public void print() {
        System.out.println(outerData);
        System.out.println(iC1.innerData);
        System.out.println(iC2.innerData);
    }

    public static void main(String[] args) {
        OuterClass x = new OuterClass();
        System.out.println(x.outerData);
    }
}

class InnerClass {
    private int innerData = 0;
    public void updateFromInner() {
        OuterClass.this.outerData++;
        this.innerData--;
    }
}
Inner class listeners can be shortened using anonymous inner classes

- inner classes without a name.
- combines declaring an inner class and creating an instance of the class in one step

```java
new SuperClassName/InterfaceName() {
    // Implement or override methods in superclass/interface
    // Other methods if necessary
}
```
We could use this:

```java
btOK.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }
});
```

Instead of this:

```java
OKHandler handler1 = new OKHandler();
btOK.setOnAction(handler1);
```
public class AnonymousHandlerDemo extends Application {
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        Button btNew = new Button("New");
        Button btOpen = new Button("Open");
        hBox.getChildren().addAll(btNew, btOpen);
        btNew.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                System.out.println("Process New");
            }
        });
        btOpen.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent e) {
                System.out.println("Process Open");
            }
        });
        Scene scene = new Scene(hBox, 300, 50);
        primaryStage.setTitle("AnonymousHandlerDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    } ...
}
LAMBDA EXPRESSIONS

Lambda expression is a new feature in Java 8.

- Predefined functions for the type of the input.

Lambda expressions

- can be viewed as anonymous methods without the syntactic burden!
- allow us to pass “functionality” as a method argument

```java
tbEnlarge.setOnAction(
    new EventHandler<ActionEvent>() {
        @Override
        public void handle(ActionEvent e) {
            // Code for processing event e
        }
    });

(b) Lambda expression event handler
```

```java
btEnlarge.setOnAction(
    e -> {
        // Code for processing event e
    });
```

(a) Anonymous inner class event handler
public class LambdaHandlerDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        HBox hBox = new HBox();
        hBox.setSpacing(10);
        hBox.setAlignment(Pos.CENTER);
        Button btNew = new Button("New");
        Button btOpen = new Button("Open");
        Button btSave = new Button("Save");
        Button btPrint = new Button("Print");
        hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);
        btNew.setOnAction(e -> System.out.println("Process New"));
        btOpen.setOnAction(e -> System.out.println("Process Open"));
        btSave.setOnAction(e -> System.out.println("Process Save"));
        btPrint.setOnAction(e -> System.out.println("Process Print"));
        ...
    }
}
The statements in the lambda expression is all for that method.

- If it contains multiple methods, the compiler will not be able to compile the lambda expression.
- So, for the compiler to understand lambda expressions, the interface must contain exactly one abstract method.
- Such an interface is known as a functional interface, or a Single Abstract Method (SAM) interface.
public class LoanCalculator extends Application {
    private Stage primaryStage;
    private TextField tfAnnualInterestRate;
    private TextField tfNumberOfYears;
    private TextField tfLoanAmount;
    private TextField tfMonthlyPayment;
    private TextField tfTotalPayment;
    private Button btCalculate;
    private Scene scene;

    @Override
    public void start(Stage initPrimaryStage) {
        primaryStage = initPrimaryStage;
        layoutGUI();
        initHandlers();
    }
    ...
}
public void layoutGUI() {
    tfAnnualInterestRate = new TextField();
    tfNumberOfYears = new TextField();
    tfLoanAmount = new TextField();
    tfMonthlyPayment = new TextField();
    tfTotalPayment = new TextField();
    btCalculate = new Button("Calculate");
    GridPane gridPane = new GridPane();
    scene = new Scene(gridPane, 400, 250);
    primaryStage.setScene(scene);
    primaryStage.show();
}
LOAN CALCULATOR

gridPane.setHgap(5);
gridPane.setVgap(5);
gridPane.add(new Label("Annual Interest Rate:"), 0, 0);
gridPane.add(tfAnnualInterestRate, 1, 0);
gridPane.add(new Label("Number of Years:"), 0, 1);
gridPane.add(tfNumberOfYears, 1, 1);
gridPane.add(new Label("Loan Amount:"), 0, 2);
gridPane.add(tfLoanAmount, 1, 2);
gridPane.add(new Label("Monthly Payment:"), 0, 3);
gridPane.add(tfMonthlyPayment, 1, 3);
gridPane.add(new Label("Total Payment:"), 0, 4);
gridPane.add(tfTotalPayment, 1, 4);
gridPane.add(btCalculate, 1, 5);
public void initHandlers() {
    btCalculate.setOnAction(e -> calculateLoanPayment());
}

private void calculateLoanPayment() {
    double interest = Double.parseDouble(tfAnnualInterestRate.getText());
    int year = Integer.parseInt(tfNumberOfYears.getText());
    double loanAmount = Double.parseDouble(tfLoanAmount.getText());
    Loan loan = new Loan(interest, year, loanAmount);
    tfMonthlyPayment.setText(String.format("$%.2f", loan.getMonthlyPayment()));
    tfTotalPayment.setText(String.format("$%.2f", loan.getTotalPayment()));
}

public static void main(String[] args) {
    launch(args);
}
<table>
<thead>
<tr>
<th>Method Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getButton()</code></td>
<td>MouseButton</td>
</tr>
<tr>
<td><code>getClickCount()</code></td>
<td>Returns the number of mouse clicks associated with this event.</td>
</tr>
<tr>
<td><code>getX()</code></td>
<td>Returns the x-coordinate of the mouse point in the event source node.</td>
</tr>
<tr>
<td><code>getY()</code></td>
<td>Returns the y-coordinate of the mouse point in the event source node.</td>
</tr>
<tr>
<td><code>getSceneX()</code></td>
<td>Returns the x-coordinate of the mouse point in the scene.</td>
</tr>
<tr>
<td><code>getSceneY()</code></td>
<td>Returns the y-coordinate of the mouse point in the scene.</td>
</tr>
<tr>
<td><code>getScreenX()</code></td>
<td>Returns the x-coordinate of the mouse point in the screen.</td>
</tr>
<tr>
<td><code>getScreenY()</code></td>
<td>Returns the y-coordinate of the mouse point in the screen.</td>
</tr>
<tr>
<td><code>isAltDown()</code></td>
<td>Returns true if the Alt key is pressed on this event.</td>
</tr>
<tr>
<td><code>isControlDown()</code></td>
<td>Returns true if the Control key is pressed on this event.</td>
</tr>
<tr>
<td><code>isMetaDown()</code></td>
<td>Returns true if the mouse Meta button is pressed on this event.</td>
</tr>
<tr>
<td><code>isShiftDown()</code></td>
<td>Returns true if the Shift key is pressed on this event.</td>
</tr>
</tbody>
</table>
public class MouseEventDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Text text = new Text(20, 20, "Programming is fun");
        pane.getChildren().addAll(text);
        text.setOnMouseDragged(e -> {
            text.setX(e.getX());
            text.setY(e.getY());
        });

        Scene scene = new Scene(pane, 300, 100);
        primaryStage.setTitle("MouseEventDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }

    public static void main(String[] args) {
        launch(args);
    }
}
# The KeyEvent Class

<table>
<thead>
<tr>
<th>Method</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>getCharacter()</code>: String</td>
<td>Returns the character associated with the key in this event.</td>
</tr>
<tr>
<td><code>getCode()</code>: KeyCode</td>
<td>Returns the key code associated with the key in this event.</td>
</tr>
<tr>
<td><code>getText()</code>: String</td>
<td>Returns a string describing the key code.</td>
</tr>
<tr>
<td><code>isAltDown()</code>: boolean</td>
<td>Returns true if the Alt key is pressed on this event.</td>
</tr>
<tr>
<td><code>isControlDown()</code>: boolean</td>
<td>Returns true if the Control key is pressed on this event.</td>
</tr>
<tr>
<td><code>isMetaDown()</code>: boolean</td>
<td>Returns true if the mouse Meta button is pressed on this event.</td>
</tr>
<tr>
<td><code>isShiftDown()</code>: boolean</td>
<td>Returns true if the Shift key is pressed on this event.</td>
</tr>
</tbody>
</table>
public class KeyEventDemo extends Application {
    @Override
    public void start(Stage primaryStage) {
        Pane pane = new Pane();
        Text text = new Text(20, 20, "A");
        text.setFocused(true);
        pane.getChildren().add(text);
        text.setOnKeyPressed(e -> {
            switch (e.getCode()) {
                case DOWN: text.setY(text.getY() + 10); break;
                case UP: text.setY(text.getY() - 10); break;
                case LEFT: text.setX(text.getX() - 10); break;
                case RIGHT: text.setX(text.getX() + 10); break;
                default:
                    if (Character.isLetterOrDigit(e.getText().charAt(0)))
                        text.setText(e.getText());
            }
        });
        Scene scene = new Scene(pane);
        primaryStage.setTitle("KeyEventDemo");
        primaryStage.setScene(scene);
        primaryStage.show();
    }
}
# 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 <code>keyCode</code> 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>
JavaFX has event programming support for mobile devices:

`javafx.scene.input.SwipeEvent`,
`javafx.scene.input.TouchEvent`,
`javafx.scene.input.ZoomEvent`.

Example:

http://docs.oracle.com/javase/8/javafx/events-tutorial/gestureeventsjava.htm

http://docs.oracle.com/javase/8/javafx/events-tutorial/toucheventsjava.htm