CSE 219
COMPUTER SCIENCE III

EVENT PROGRAMMING WITH JAVAFX

SLIDES COURTESY: PROF. RICHARD MCKENNA, STONY BROOK UNIVERSITY

EVENT PROGRAMMING

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

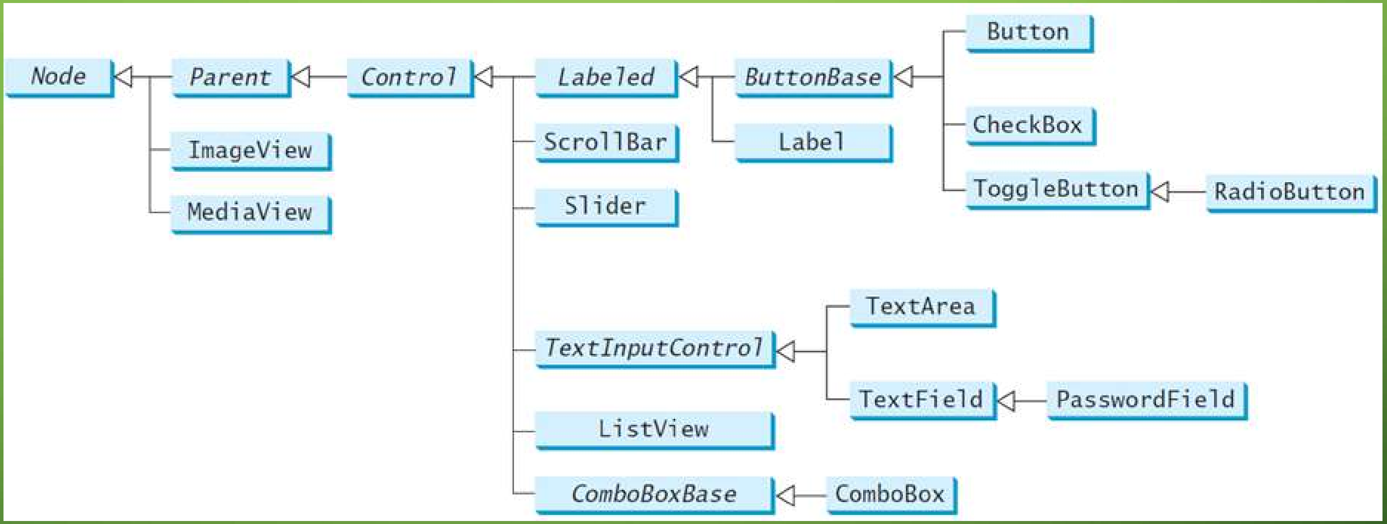
Event Handler?

- code with response to event
- a.k.a. event listener

JAVA'S EVENT HANDLING

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

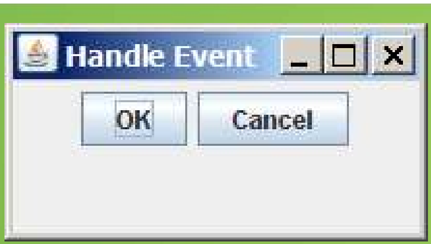
Contain information about the event

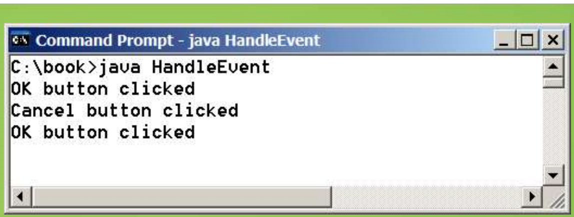
Like what?

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

Listeners use them to properly respond

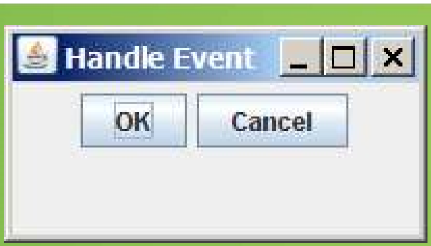
- different methods inside a listener object can react differently to different types of interactions

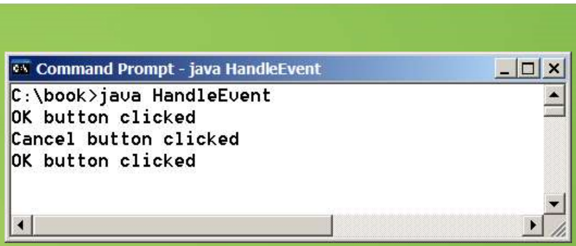




```
C:\book>java HandleEvent
OK button clicked
Cancel button clicked
OK button clicked
```

```
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();
 }…/*main*/}
```





```
C:\book>java HandleEvent
OK button clicked
Cancel button clicked
OK button clicked
```

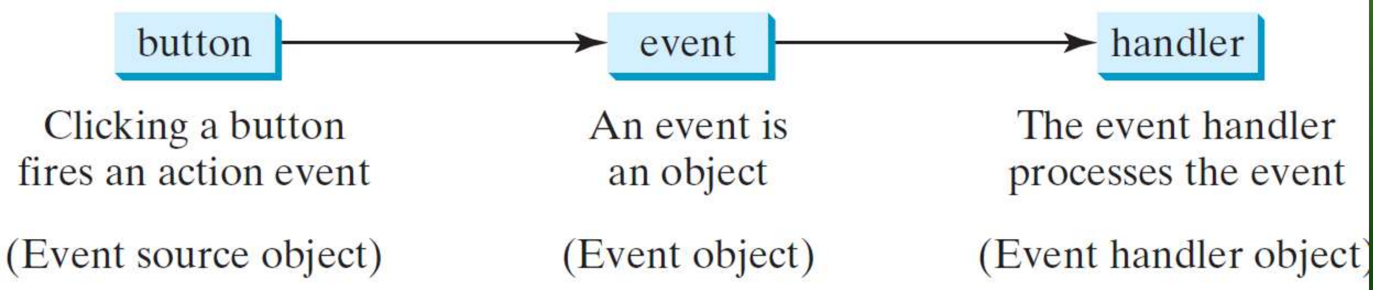
```
class OKHandler implements EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("OK button clicked");
    }
}
class CancelHandler implements
                    EventHandler<ActionEvent> {
    @Override
    public void handle(ActionEvent e) {
        System.out.println("Cancel button clicked");
    }
}
```

HANDLING GUI EVENTS

Source object: Button

Event object: ActionEvent

- Listener objects: OkHandler, CancelHandler



button

Clicking a button fires an action event

(Event source object)

event

An event is an object

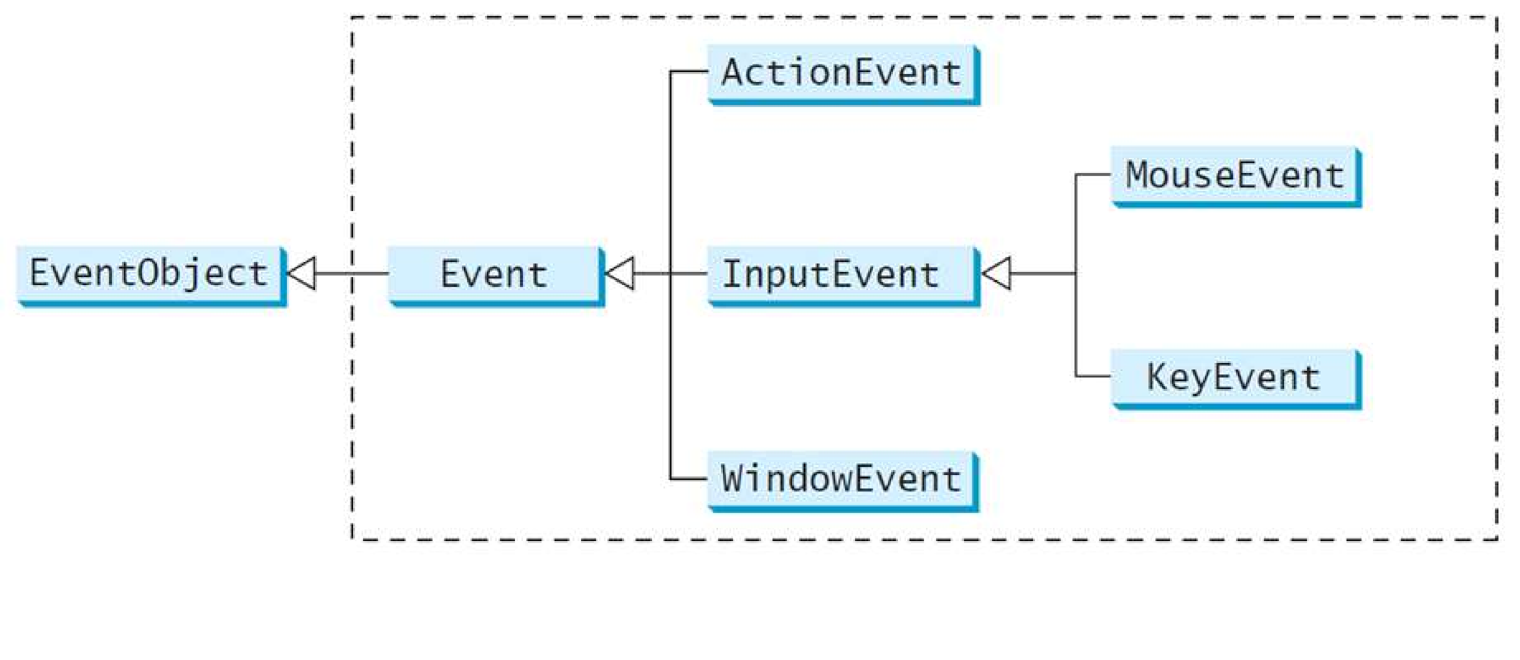
(Event object)

handler

The event handler processes the event

(Event handler object)

EVENT CLASSES



EVENT INFORMATION

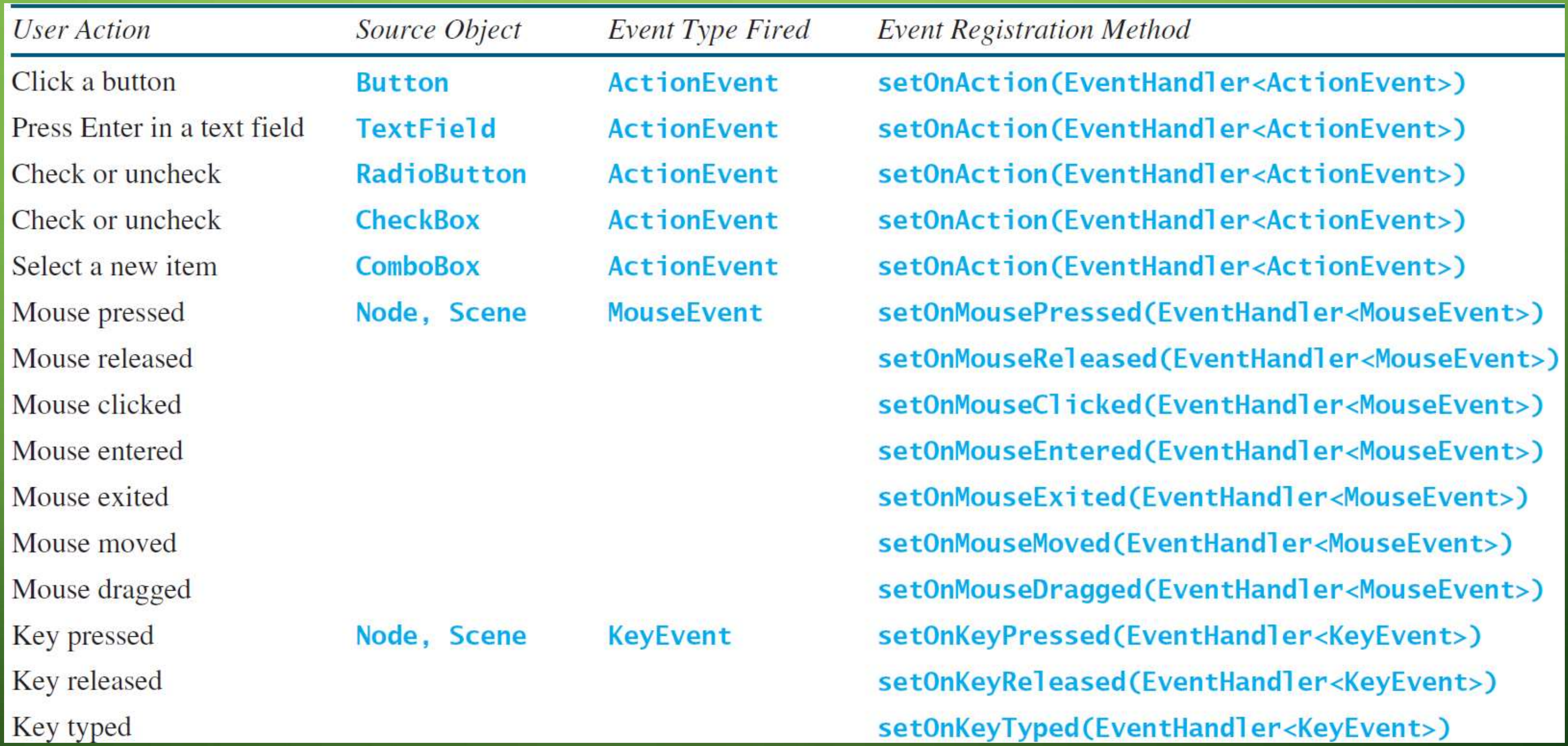
Event objects have info ***about*** the event:

– 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



User Action	*Source Object*	*Event Type Fired*	*Event Registration Method*
Click a button	`Button`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Press Enter in a text field	`TextField`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`RadioButton`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Check or uncheck	`CheckBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Select a new item	`ComboBox`	`ActionEvent`	`setOnAction(EventHandler<ActionEvent>)`
Mouse pressed	`Node, Scene`	`MouseEvent`	`setOnMousePressed(EventHandler<MouseEvent>)`
Mouse released			`setOnMouseReleased(EventHandler<MouseEvent>)`
Mouse clicked			`setOnMouseClicked(EventHandler<MouseEvent>)`
Mouse entered			`setOnMouseEntered(EventHandler<MouseEvent>)`
Mouse exited			`setOnMouseExited(EventHandler<MouseEvent>)`
Mouse moved			`setOnMouseMoved(EventHandler<MouseEvent>)`
Mouse dragged			`setOnMouseDragged(EventHandler<MouseEvent>)`
Key pressed	`Node, Scene`	`KeyEvent`	`setOnKeyPressed(EventHandler<KeyEvent>)`
Key released			`setOnKeyReleased(EventHandler<KeyEvent>)`
Key typed			`setOnKeyTyped(EventHandler<KeyEvent>)`

INNER CLASS LISTENERS

A listener class 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.**
- It will not be shared by other applications.

WHAT'S THE OUTPUT?

The **Inner** class is a class is a member of another class.

- class can reference the data and methods defined in the outer class
- is compiled as OuterClass$InnerClass.class

```
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--;
        }
    }
}
```

ANONYMOUS INNER CLASSES

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

```
new SuperClassName/InterfaceName() {
 // Implement or override methods in superclass/interface
 // Other methods if necessary
}
```

ANONYMOUS INNER CLASSES EXAMPLE

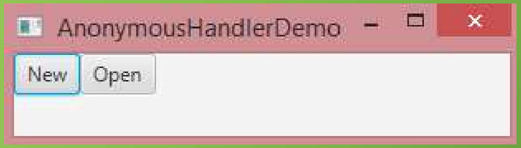
We could use this:

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

Instead of this:

```
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();
    } …}
```

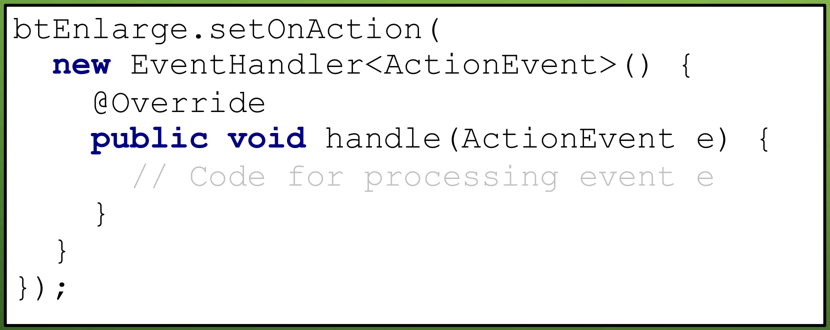


SIMPLIFYING EVENT HANDING USING LAMBDA EXPRESSIONS

Lambda expression is a new feature in Java 8.

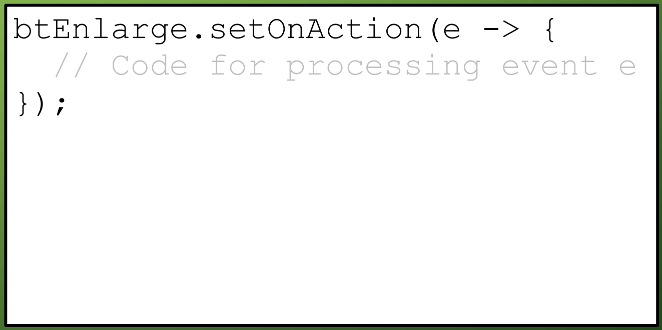
- Predefined functions for the type of the input.

Lambda expressions can be viewed as an anonymous method with a concise syntax.



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

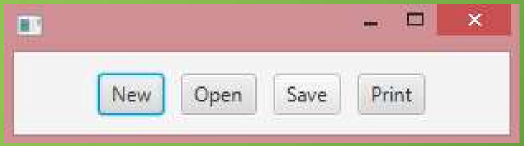
(a) Anonymous inner class event handler



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

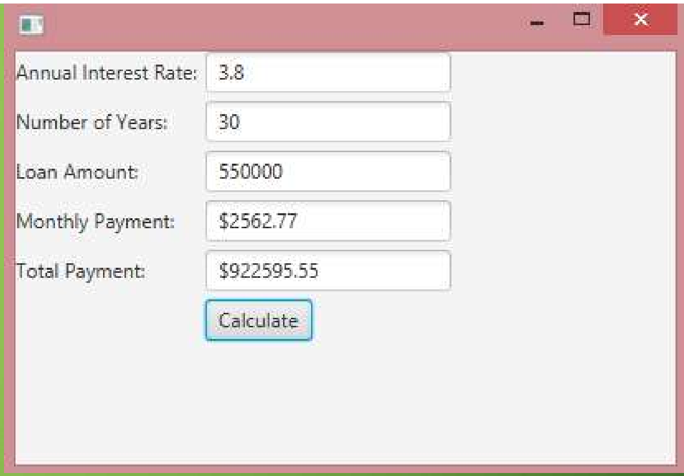
(b) Lambda expression event handler

```
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");
        });
        …
```



LOAN CALCULATOR

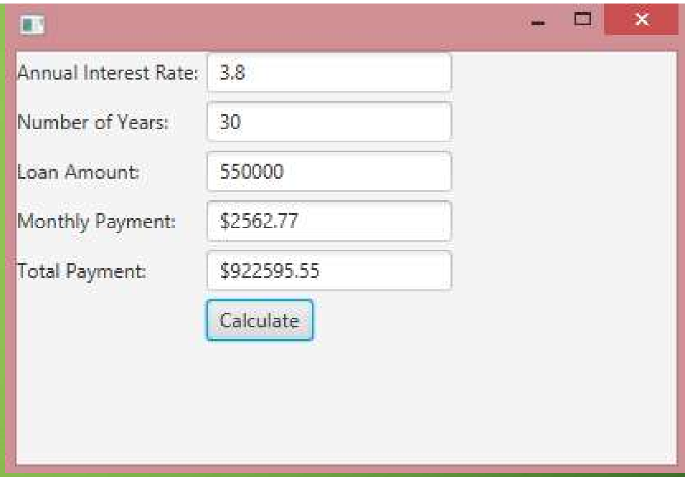
```
public class LoanCalculator extends Application {
    private Stage primaryStage;
    private TextField tfAnnualInterestRate;
    private TextField tfNumberOfYears;
    private TextField tfLoanAmount;
    private TextField tfMonthlyPayment;
    private TextField tfTotalPayment;
    private Button btCalculate;
    private Scene scene;

    @Override
    public void start(Stage initPrimaryStage) {
        primaryStage = initPrimaryStage;
        layoutGUI();
        initHandlers();
    }
...
```



Annual Interest Rate:	3.8
Number of Years:	30
Loan Amount:	550000
Monthly Payment:	$2562.77
Total Payment:	$922595.55

LOAN CALCULATOR

```
...
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();
...
```



Annual Interest Rate: 3.8

Number of Years: 30

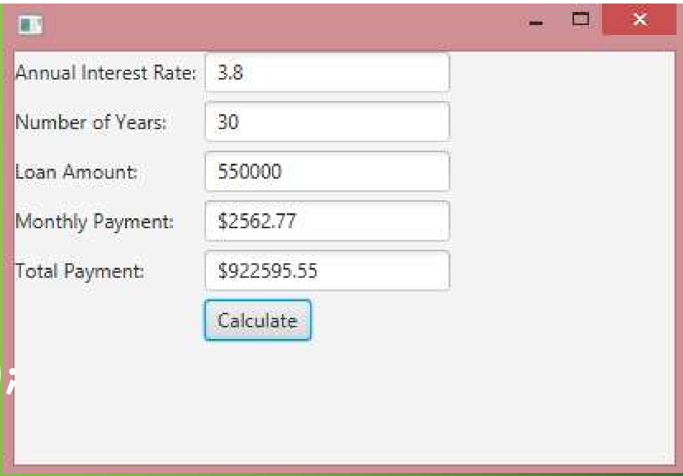
Loan Amount: 550000

Monthly Payment: $2562.77

Total Payment: $922595.55

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);
  }
...
```



Annual Interest Rate: 3.8

Number of Years: 30

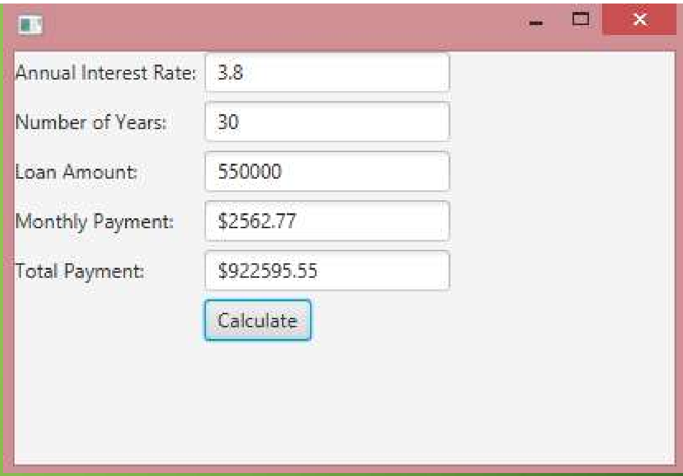
Loan Amount: 550000

Monthly Payment: $2562.77

Total Payment: $922595.55

LOAN CALCULATOR

```
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);
}
}
```



Annual Interest Rate: 3.8

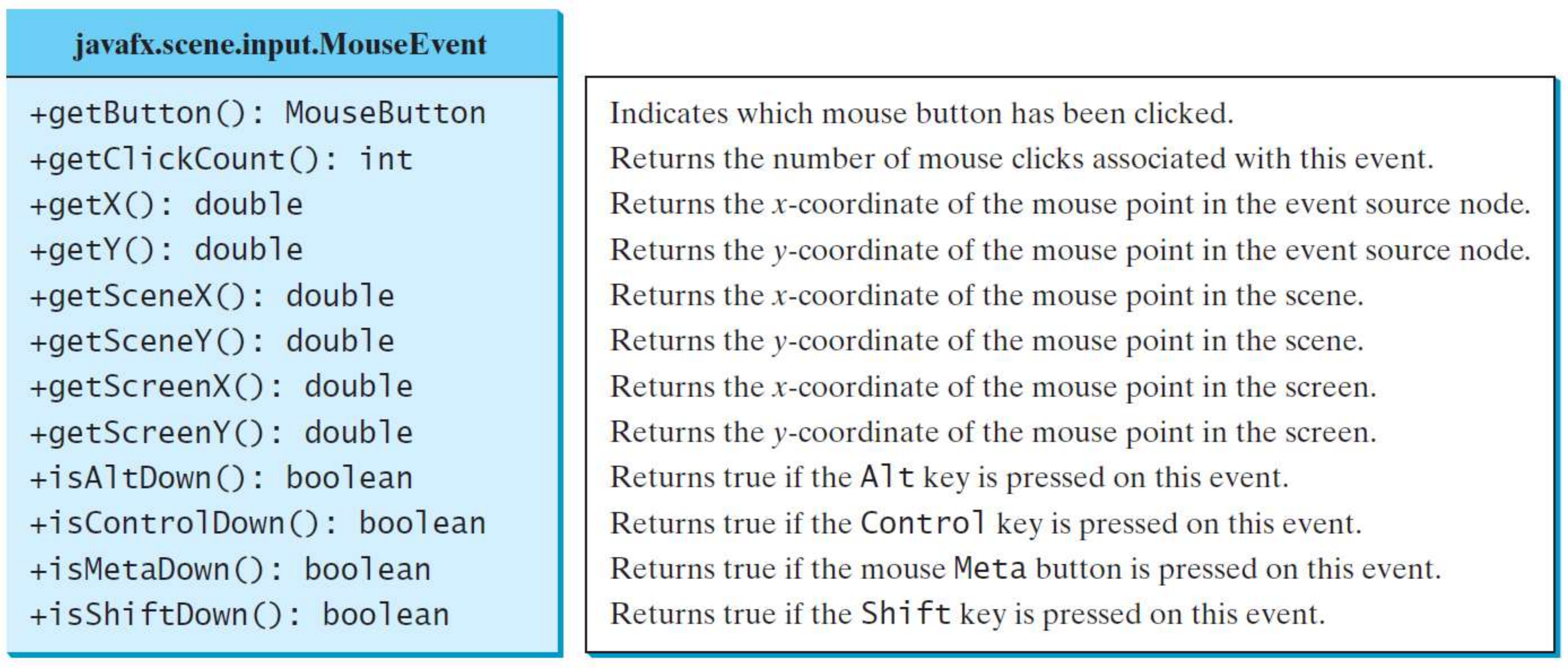
Number of Years: 30

Loan Amount: 550000

Monthly Payment: $2562.77

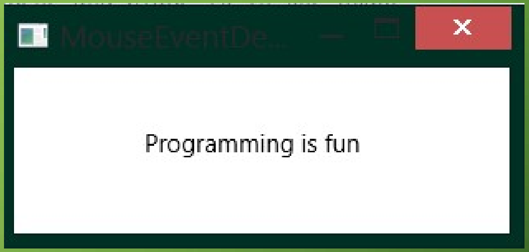
Total Payment: $922595.55

MOUSEEVENT



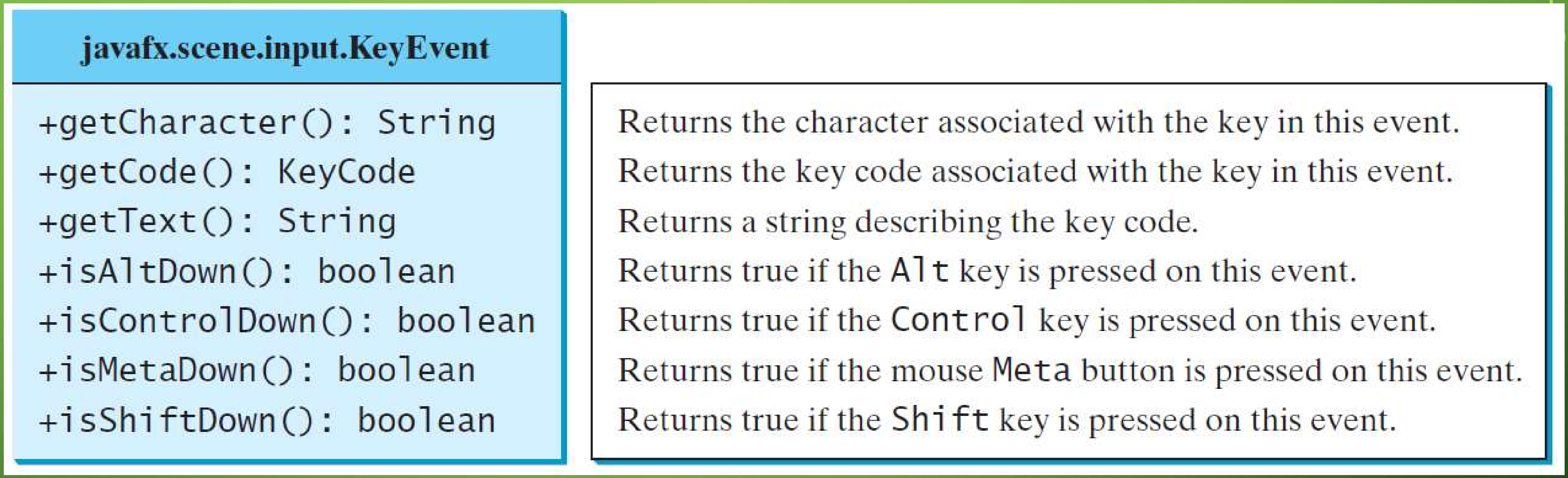
javafx.scene.input.MouseEvent	
`+getButton(): MouseButton`	Indicates which mouse button has been clicked.
`+getClickCount(): int`	Returns the number of mouse clicks associated with this event.
`+getX(): double`	Returns the *x*-coordinate of the mouse point in the event source node.
`+getY(): double`	Returns the *y*-coordinate of the mouse point in the event source node.
`+getSceneX(): double`	Returns the *x*-coordinate of the mouse point in the scene.
`+getSceneY(): double`	Returns the *y*-coordinate of the mouse point in the scene.
`+getScreenX(): double`	Returns the *x*-coordinate of the mouse point in the screen.
`+getScreenY(): double`	Returns the *y*-coordinate of the mouse point in the screen.
`+isAltDown(): boolean`	Returns true if the `Alt` key is pressed on this event.
`+isControlDown(): boolean`	Returns true if the `Control` key is pressed on this event.
`+isMetaDown(): boolean`	Returns true if the mouse `Meta` button is pressed on this event.
`+isShiftDown(): boolean`	Returns true if the `Shift` key is pressed on this event.

```
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);
    }
}
```



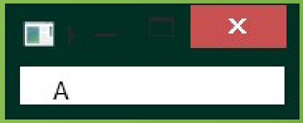
Programming is fun

THE KEYEVENT CLASS

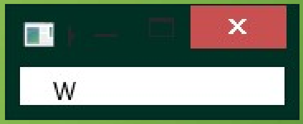


javafx.scene.input.KeyEvent	
+getCharacter(): String	Returns the character associated with the key in this event.
+getCode(): KeyCode	Returns the key code associated with the key in this event.
+getText(): String	Returns a string describing the key code.
+isAltDown(): boolean	Returns true if the Alt key is pressed on this event.
+isControlDown(): boolean	Returns true if the Control key is pressed on this event.
+isMetaDown(): boolean	Returns true if the mouse Meta button is pressed on this event.
+isShiftDown(): boolean	Returns true if the Shift key is pressed on this event.

```
public class KeyEventDemo extends Application {
  @Override
  public void start(Stage primaryStage) {
    Pane pane = new Pane();
    Text text = new Text(20, 20, "A");
    text.setFocusTraversable(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();
  }
…
```

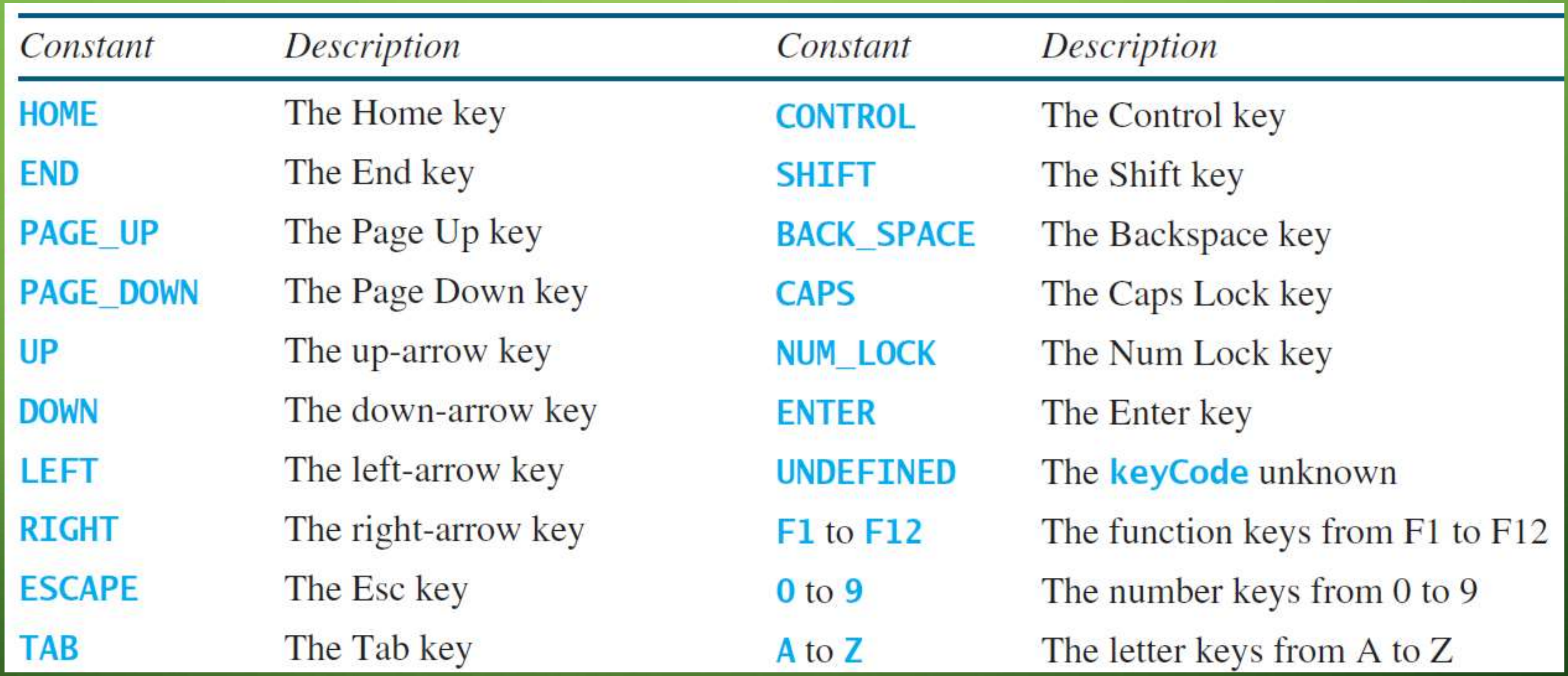


A



W

THE KEYCODE CONSTANTS



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

JAVAFX SUPPORT FOR MOBILE DEVICES

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