

Chapter 15 Event-Driven Programming



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

Procedural vs. Event-Driven Programming

- *Procedural programming* is executed in procedural order.
- In event-driven programming, code is executed upon activation of events.

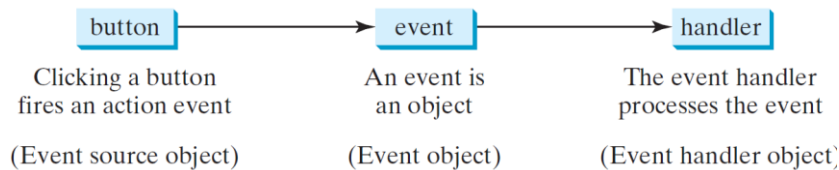


liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

Handling GUI Events

Source object (e.g., button)

Listener object contains a method for processing the event.

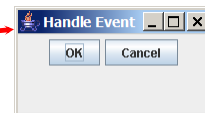


liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Nijoum

Trace Execution

```
public class HandleEvent extends Application {  
    public void start(Stage primaryStage) {  
        ...  
        OKHandlerClass handler1 = new OKHandlerClass();  
        btOK.setOnAction(handler1);  
        CancelHandlerClass handler2 = new CancelHandlerClass();  
        btCancel.setOnAction(handler2);  
        ...  
        primaryStage.show(); // Display the stage  
    }  
}
```

1. Start from the main method to create a window and display it



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



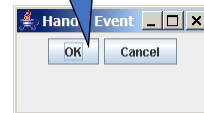
liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Nijoum

Trace Execution

```
public class HandleEvent extends Application {  
    public void start(Stage primaryStage) {  
        ...  
        OKHandlerClass handler1 = new OKHandlerClass();  
        btOK.setOnAction(handler1);  
        CancelHandlerClass handler2 = new CancelHandlerClass();  
        btCancel.setOnAction(handler2);  
        ...  
        primaryStage.show(); // Display the stage  
    }  
}
```

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

2. Click OK



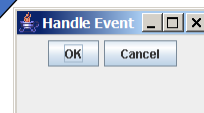
liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoun

Trace Execution

```
public class HandleEvent extends Application {  
    public void start(Stage primaryStage) {  
        ...  
        OKHandlerClass handler1 = new OKHandlerClass();  
        btOK.setOnAction(handler1);  
        CancelHandlerClass handler2 = new CancelHandlerClass();  
        btCancel.setOnAction(handler2);  
        ...  
        primaryStage.show(); // Display the stage  
    }  
}
```

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

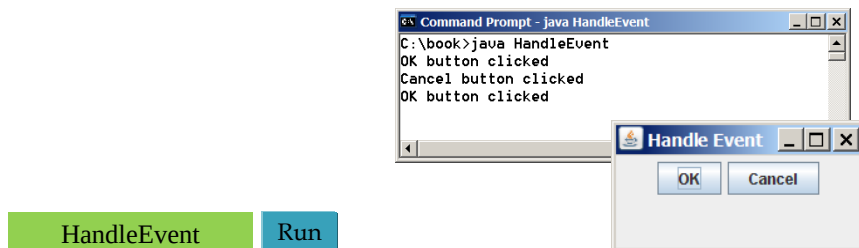
3. The JVM invokes the listener's handle method



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoun

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.



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn

```
public class HandelEvnts extends Application implements EventHandler<ActionEvent> {
    Button btOK, btCancel;
    @Override // Override the start method
    public void start(Stage primaryStage) {
        // Create a pane and set its properties
        HBox pane = new HBox(10);
        pane.setAlignment(Pos.CENTER);
        btOK = new Button("OK");
        btCancel = new Button("Cancel");

        btOK.setOnAction(this);
        btCancel.setOnAction(this);

        pane.getChildren().addAll(btOK, btCancel);

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane);
        primaryStage.setTitle("HandleEvent"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    public static void main(String[] args) {
        Launch(args);
    }
    @Override
    public void handle(ActionEvent event) {
        if(event.getSource() == btOK) {
            System.out.println("OK Button");
        }
        else if (event.getSource() == btCancel) {
            System.out.println("Cancle Button");
        }
    }
}
```



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn

```

public class HandelEvnts extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a pane and set its properties
        HBox pane = new HBox(10);
        pane.setAlignment(Pos.CENTER);
        Button btOK = new Button("OK");
        Button btCancel = new Button("Cancel");

        btOK.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            //Anonymous Inner Class
            public void handle(ActionEvent e) {
                System.out.println("Cancel button clicked");
            }
        });

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

        pane.getChildren().addAll(btOK, btCancel);

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane);
        primaryStage.setTitle("HandleEvent"); // Set
        the stage title
        primaryStage.setScene(scene); // Place the
        scene in the stage
        primaryStage.show(); // Display the stage
    }

    public static void main(String[] args) {
        Launch(args);
    }
}

```

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



```

public class HandleEvent extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a pane and set its properties
        HBox pane = new HBox(10);
        pane.setAlignment(Pos.CENTER);
        Button btOK = new Button("OK");
        Button btCancel = new Button("Cancel");
        OKHandlerClass handler1 = new OKHandlerClass();
        btOK.setOnAction(handler1);
        CancelHandlerClass handler2 = new CancelHandlerClass();
        btCancel.setOnAction(handler2);
        pane.getChildren().addAll(btOK, btCancel);

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane);
        primaryStage.setTitle("HandleEvent"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }

    public static void main(String[] args) {
        Launch(args);
    }
}

```

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



```

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

```

```

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

```

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum



```

public class HandelEvnts extends Application {
    Button btOK, btCancel ;
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create a pane and set its properties
        HBox pane = new HBox(10);
        pane.setAlignment(Pos.CENTER);
        btOK = new Button("OK");
        btCancel = new Button("Cancel");

        btOK.setOnAction(e->{System.out.println("OK Button");});

        btCancel.setOnAction(e->{System.out.println("Cancle Button");});

        pane.getChildren().addAll(btOK, btCancel);

        // Create a scene and place it in the stage
        Scene scene = new Scene(pane);
        primaryStage.setTitle("HandleEvent"); // Set the stage title
        primaryStage.setScene(scene); // Place the scene in the stage
        primaryStage.show(); // Display the stage
    }
    public static void main(String[] args) {launch(args); } }

```

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum



Event Information

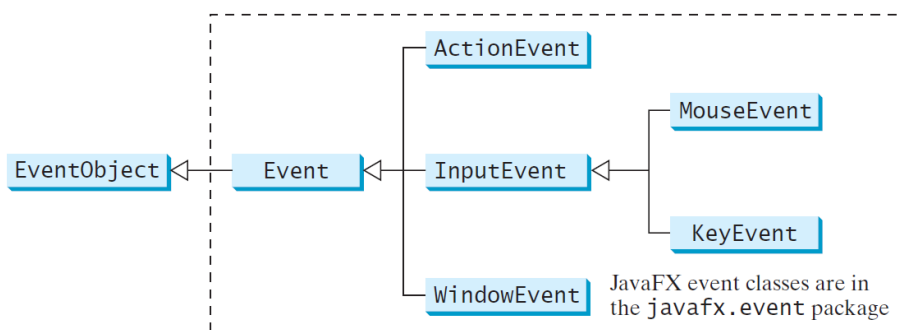
An event object contains whatever properties are pertinent to the event. You can identify the source object of the event using the `getSource()` instance method in the `EventObject` class.

The subclasses of `EventObject` deal with special types of events, such as **button actions**, **window events**, **mouse movements**, and **keystrokes**.

Table 15.1 lists external user actions, source objects, and event types generated.



Event Classes



Events

- ❑ An *event* can be defined as a type of signal to the program that something has happened.
- ❑ The event is generated by external user actions such as mouse movements, mouse clicks, or keystrokes.



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

Selected User Actions and Handlers

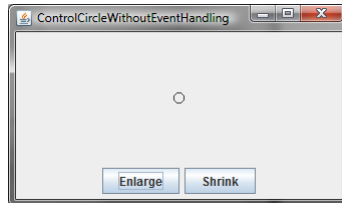
User Action	Source Object	Event Type Fired	Event Registration Method
Click a button	Button	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Press Enter in a text field	TextField	ActionEvent	setOnAction(EventHandler<ActionEvent>)
Check or uncheck	RadioButton	ActionEvent	setOnAction(EventHandler<ActionEvent>)
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	Node, Scene	MouseEvent	setOnMouseMoved(EventHandler<MouseEvent>)
Mouse dragged			setOnMouseDragged(EventHandler<MouseEvent>)
Key pressed		KeyEvent	setOnKeyPressed(EventHandler<KeyEvent>)
Key released			setOnKeyReleased(EventHandler<KeyEvent>)
Key typed			setOnKeyTyped(EventHandler<KeyEvent>)



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

Example: First Version for ControlCircle (no listeners)

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



ControlCircleWithoutEventHandling

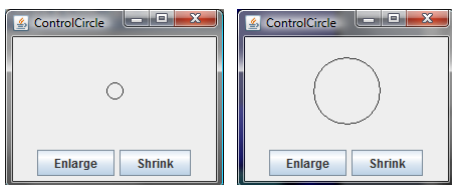
Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



Example: Second Version for ControlCircle (with listener for Enlarge)

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



```
public void enlarge()  
{ circle.setRadius(circle.getRadius() + 2);  
}  
public void shrink()  
{ circle.setRadius(circle.getRadius() > 2 ? circle.getRadius() - 2 : circle.getRadius()); }
```

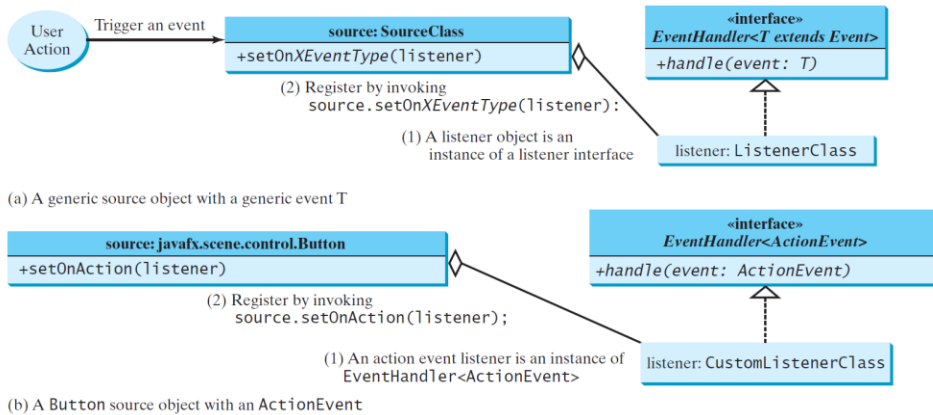
ControlCircle

Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



The Delegation Model



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

19

Inner Class Listeners

A listener class is designed specifically to create a listener object for a GUI component (e.g., a button).

It will **not be shared by other applications**. So, it is appropriate to define the listener class inside the frame class **as an inner class**.



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

20

Inner Classes

Inner class: A class is a member of another class.

Advantages: In some applications, you can use an inner class to make programs simple.

An inner class can reference the data and methods defined in the outer class in which it nests, **so you do not need to pass the reference** of the outer class to the constructor of the inner class.

ShowInnerClass



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

Inner Classes, cont.

```
public class Test {  
    ...  
    public class A {  
        ...  
    }  
}
```

(a)

```
public class Test {  
    ...  
    // Inner class  
    public class A {  
        ...  
    }  
}
```

(b)

```
// OuterClass.java: inner class demo  
public class OuterClass {  
    private int data;  
  
    /** A method in the outer class */  
    public void m() {  
        // Do something  
    }  
  
    // An inner class  
    class InnerClass {  
        /** A method in the inner class */  
        public void mi() {  
            // Directly reference data and method  
            // defined in its outer class  
            data++;  
            m();  
        }  
    }  
}
```

(c)



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

Inner Classes (cont.)

Inner classes can make programs simple and concise.

An inner class supports the work of its containing outer class and is compiled into a class named

`OuterClassName$InnerClassName.class`. For example, the inner class `InnerClass` in `OuterClass` is compiled into `OuterClass$InnerClass.class`.



Inner Classes (cont.)

- ☐ An inner class can be declared `public`, `protected`, or `private` subject to the same visibility rules applied to a member of the class.
- ☐ An inner class can be declared `static`. A static inner class can be accessed using the outer class name. A static inner class cannot access nonstatic members of the outer class.



Anonymous Inner Classes

- ❑ An anonymous inner class **must always extend a superclass or implement an interface**, but it cannot have an **explicit extends or implements clause**.
- ❑ An anonymous inner class **must implement all the abstract methods** in the superclass or in the interface.
- ❑ An anonymous inner class **always uses the no-arg constructor** from its superclass to create an instance. If an anonymous inner class implements an interface, **the constructor is `Object()`**.
- ❑ An anonymous inner class is compiled into a class named **`OuterClassName$n.class`**. For example, if the outer class `Test` has two anonymous inner classes, these two classes are compiled into **`Test$1.class` and `Test$2.class`**.

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



Anonymous Inner Classes (cont.)

Inner class listeners can be shortened using anonymous inner classes.

*An **anonymous inner class** is an inner class without a name.*

It combines declaring an inner class and creating an instance of the class in one step. An anonymous inner class is declared as follows:

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

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



Anonymous Inner Classes (cont.)

```
public void start(Stage primaryStage) {
    // Omitted

    btEnlarge.setOnAction(
        new EnlargeHandler());
}

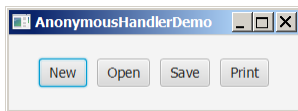
class EnlargeHandler
    implements EventHandler<ActionEvent> {
    public void handle(ActionEvent e) {
        circlePane.enlarge();
    }
}
```

(a) Inner class EnlargeListener

```
public void start(Stage primaryStage) {
    // Omitted

    btEnlarge.setOnAction(
        new class EnlargeHandler
            implements EventHandler<ActionEvent>() {
                public void handle(ActionEvent e) {
                    circlePane.enlarge();
                }
            });
}
```

(b) Anonymous inner class



AnonymousHandlerDemo

Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

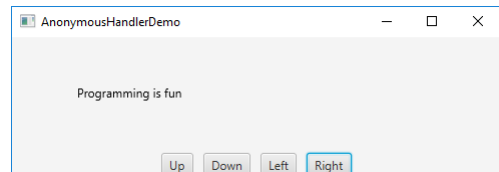


```
public class AnonymousHandlerDemo extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        Text text = new Text(40, 40, "Programming is fun");
        Pane pane = new Pane(text);

        // Hold four buttons in an HBox
        Button btUp = new Button("Up");
        Button btDown = new Button("Down");
        Button btLeft = new Button("Left");
        Button btRight = new Button("Right");
        HBox hBox = new HBox(btUp, btDown, btLeft, btRight);
        hBox.setSpacing(10);
        hBox.setAlignment(Pos.CENTER);

        BorderPane borderPane = new BorderPane(pane);
        borderPane.setBottom(hBox);

        // Create and register the handler
        btUp.setOnAction(new EventHandler<ActionEvent>() {
            @Override // Override the handle method
            public void handle(ActionEvent e) {
                text.setY(text.getY() > 10 ? text.getY() - 5 : 10); } });
    }
}
```



```
BorderPane borderPane = new BorderPane(pane);
borderPane.setBottom(hBox);
```

```
// Create and register the handler
btUp.setOnAction(new EventHandler<ActionEvent>() {
    @Override // Override the handle method
    public void handle(ActionEvent e) {
        text.setY(text.getY() > 10 ? text.getY() - 5 : 10); } });
```

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum



```

btDown.setOnAction(new EventHandler<ActionEvent>() {
    @Override // Override the handle method
    public void handle(ActionEvent e) {
        text.setY(text.getY() < pane.getHeight() ?
            text.getY() + 5 : pane.getHeight());
    }
});

btLeft.setOnAction(new EventHandler<ActionEvent>() {
    @Override // Override the handle method
    public void handle(ActionEvent e) {
        text.setX(text.getX() > 0 ? text.getX() - 5 : 0);
    }
});

btRight.setOnAction(new EventHandler<ActionEvent>() {
    @Override // Override the handle method
    public void handle(ActionEvent e) {
        text.setX(text.getX() < pane.getWidth() - 100?
            text.getX() + 5 : pane.getWidth() - 100);
    }
});

```

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



Simplifying Event Handling Using Lambda Expressions

Lambda expression is a new feature in Java 8. Lambda expressions can be viewed as an anonymous method with a concise syntax. For example, the following code in (a) can be greatly simplified using a lambda expression in (b) in three lines.

```

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

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



Basic Syntax for a Lambda Expression

The basic syntax for a lambda expression is either

`(type1 param1, type2 param2, ...) -> expression`

or

`(type1 param1, type2 param2, ...) -> { statements; }`

The data type for a parameter may be explicitly declared or implicitly inferred by the compiler. The parentheses can be omitted if there is only one parameter without an explicit data type.



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoun

```
(p1, p2) -> System.out.println("Multiple parameters: " + p1 + ", " + p2);
```

```
(Car car) -> System.out.println("The car is: " + car.getName());
```



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoun

Single Abstract Method Interface (SAM)

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.

AnonymousHandlerDemo

Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum



The MouseEvent Class

javafx.scene.input.MouseEvent

```
+getButton(): MouseButton  
+getClickCount(): int  
+getX(): double  
+getY(): double  
+getSceneX(): double  
+getSceneY(): double  
+getScreenX(): double  
+getScreenY(): double  
+isAltDown(): boolean  
+isControlDown(): boolean  
+isMetaDown(): boolean  
+isShiftDown(): boolean
```

Indicates which mouse button has been clicked.
Returns the number of mouse clicks associated with this event.
Returns the x-coordinate of the mouse point in the event source node.
Returns the y-coordinate of the mouse point in the event source node.
Returns the x-coordinate of the mouse point in the scene.
Returns the y-coordinate of the mouse point in the scene.
Returns the x-coordinate of the mouse point in the screen.
Returns the y-coordinate of the mouse point in the screen.
Returns true if the **Alt** key is pressed on this event.
Returns true if the **Control** key is pressed on this event.
Returns true if the mouse **Meta** button is pressed on this event.
Returns true if the **Shift** key is pressed on this event.

MouseEventDemo

Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum



The KeyEvent Class

javafx.scene.input.KeyEvent

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

Returns the character associated with the key in this event.
Returns the key code associated with the key in this event.
Returns a string describing the key code.
Returns true if the **Alt** key is pressed on this event.
Returns true if the **Control** key is pressed on this event.
Returns true if the mouse **Meta** button is pressed on this event.
Returns true if the **Shift** key is pressed on this event.

KeyEventDemo

Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



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

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn



Problem: Loan Calculator

LoanCalculator

Run



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn

```
public class LoanCalculator extends Application {
    private TextField tfAnnualInterestRate = new TextField();
    private TextField tfNumberOfYears = new TextField();
    private TextField tfLoanAmount = new TextField();
    private TextField tfMonthlyPayment = new TextField();
    private TextField tfTotalPayment = new TextField();
    private Button btCalculate = new Button("Calculate");

    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        // Create UI
        GridPane gridPane = new GridPane();
        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:	4.5
Number of Years:	4
Loan Amount:	5000
Monthly Payment:	\$114.02
Total Payment:	\$5472.84

Calculator



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njourn

```
// Set properties for UI
gridPane.setAlignment(Pos.CENTER);
tfAnnualInterestRate.setAlignment(Pos.BOTTOM_RIGHT);
tfNumberOfYears.setAlignment(Pos.BOTTOM_RIGHT);
tfLoanAmount.setAlignment(Pos.BOTTOM_RIGHT);
tfMonthlyPayment.setAlignment(Pos.BOTTOM_RIGHT);
tfTotalPayment.setAlignment(Pos.BOTTOM_RIGHT);
tfMonthlyPayment.setEditable(false);
tfTotalPayment.setEditable(false);
GridPane.setHalignment(btCalculate, HPos.RIGHT);

// Process events
btCalculate.setOnAction(e -> calculateLoanPayment());

// Create a scene and place it in the stage
Scene scene = new Scene(gridPane, 400, 250);
primaryStage.setTitle("LoanCalculator"); // Set title
primaryStage.setScene(scene); // Place the scene in the stage
primaryStage.show(); // Display the stage
}
```



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

```
private void calculateLoanPayment() {
    // Get values from text fields
    double interest =
        Double.parseDouble(tfAnnualInterestRate.getText());
    int year = Integer.parseInt(tfNumberOfYears.getText());
    double loanAmount = Double.parseDouble(tfLoanAmount.getText());

    // Create a loan object. Loan defined in Listing 10.2
    Loan loan = new Loan(interest, year, loanAmount);

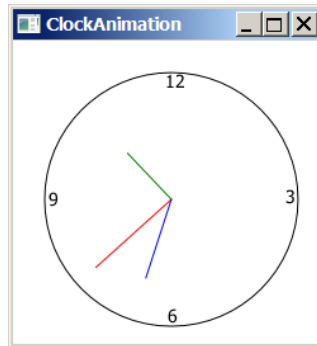
    // Display monthly payment and total payment
    tfMonthlyPayment.setText(String.format("%.2f",
        loan.getMonthlyPayment()));
    tfTotalPayment.setText(String.format("%.2f",
        loan.getTotalPayment()));
}

/**
 * The main method is only needed for the IDE with limited
 * JavaFX support. Not needed for running from the command line.
 */
public static void main(String[] args) {
    launch(args);
}
}
```



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum

Clock Animation



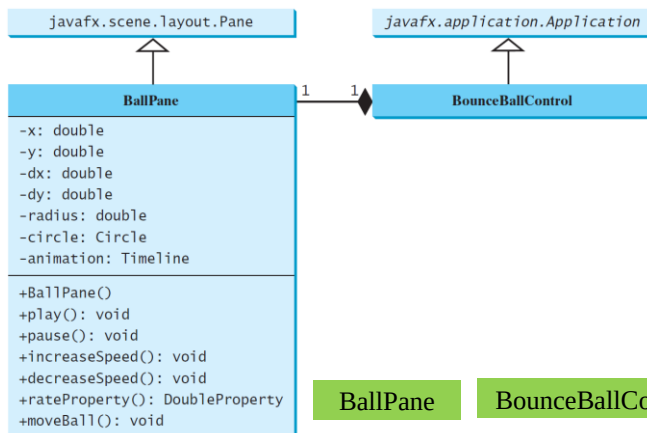
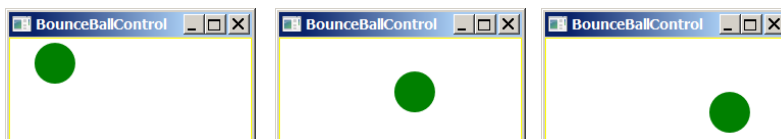
ClockAnimation

Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum



Case Study: Bouncing Ball



BallPane

BounceBallControl

Run

liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum



Show Coc



liang introduction to java programming 11th edition ,2019 , Edit By : Mr.Murad Njoum