

Chapter 15 Inner Classes and Lambda Expressions

Dr. Asem Kitana

Dr. Abdallah Karakra



Inner Classes



Inner Class Handlers

A handler class is designed specifically to create a handler object for a GUI component (e.g., a button). It will not be shared by other applications. So, it is appropriate to define the handler class inside the main class as an inner class.



Inner Classes

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

An inner class is a class defined within the scope of another class.

Inner classes are useful for defining handler classes.

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.

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)

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*.



```

14 public class ControlCircle extends Application {
15     private CirclePane circlePane = new CirclePane();
16
17     @Override // Override the start method in the Application class
18     public void start(Stage primaryStage) {
19         // Hold two buttons in an HBox
20         HBox hBox = new HBox();
21         hBox.setSpacing(10);
22         hBox.setAlignment(Pos.CENTER);
23         Button btEnlarge = new Button("Enlarge");
24         Button btShrink = new Button("Shrink");
25         hBox.getChildren().add(btEnlarge);
26         hBox.getChildren().add(btShrink);
27
28         // Create and register the handler
29         btEnlarge.setOnAction(new EnlargeHandler());
30
31         BorderPane borderPane = new BorderPane();
32         borderPane.setCenter(circlePane);
33         borderPane.setBottom(hBox);
34         BorderPane.setAlignment(hBox, Pos.CENTER);
35
36         // Create a scene and place it in the stage
37         Scene scene = new Scene(borderPane, 200, 150);
38         primaryStage.setTitle("ControlCircle"); // Set the stage title
39         primaryStage.setScene(scene); // Place the scene in the stage
40         primaryStage.show(); // Display the stage
41     }
42
43     class EnlargeHandler implements EventHandler<ActionEvent> {
44         @Override // Override the handle method
45         public void handle(ActionEvent e) {
46             circlePane.enlarge();
47         }
48     }
49 }

```

EnlargeHandler is an inner class
 And it's compiled into:
 ControlCircle\$EnlargeHandler.
 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




Anonymous Inner Classes

Inner class handlers 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  
}
```

Anonymous Inner Classes

```
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



Anonymous Inner Class Features

- ❑ 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.**



Example: ControlCircle with Anonymous Inner Classes

```
import javafx.application.Application;
import javafx.event.ActionEvent;
import javafx.event.EventHandler;
import javafx.geometry.Pos;
import javafx.scene.Scene;
import javafx.scene.control.Button;
import javafx.scene.layout.StackPane;
import javafx.scene.layout.HBox;
import javafx.scene.layout.BorderPane;
import javafx.scene.paint.Color;
import javafx.scene.shape.Circle;
import javafx.stage.Stage;
public class ControlCircle extends Application {
    @Override // Override the start method in the Application class
    public void start(Stage primaryStage) {
        StackPane pane = new StackPane();
        Circle circle = new Circle(50);
        circle.setStroke(Color.BLACK);
        circle.setFill(Color.WHITE);
        pane.getChildren().add(circle);

        HBox hBox = new HBox();
        hBox.setSpacing(10);
        hBox.setAlignment(Pos.CENTER);
        Button btEnlarge = new Button("Enlarge");
        btEnlarge.setOnAction(new EventHandler<ActionEvent>() {
            @Override
            public void handle(ActionEvent event) {
                circle.setRadius(circle.getRadius()+2);
            }
        });
    }
}

// Create and register the handler
btEnlarge.setOnAction(new EnlargeHandler());

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

Inner class

Anonymous
Inner class

```
Button btShrink = new Button("Shrink");
btShrink.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        circle.setRadius(circle.getRadius()-2);
    }
});
```

```
hBox.getChildren().add(btEnlarge);
hBox.getChildren().add(btShrink);
```

```
BorderPane borderPane = new BorderPane();
borderPane.setCenter(pane);
borderPane.setBottom(hBox);
borderPane.setAlignment(hBox, Pos.CENTER);
```

```
// Create a scene and place it in the stage
Scene scene = new Scene(borderPane, 200, 150);
primaryStage.setTitle("ControlCircle"); // Set the stage title
primaryStage.setScene(scene); // Place the scene in the stage
primaryStage.show(); // Display the stage
```

```
}
```

```
/**
```

```
* 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);
```

```
}
```

```
}
```

The two anonymous inner classes in this example are compiled into:

ControlCircle\$1.class
ControlCircle\$2.class



Lambda Expressions



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

Example of Lambda Expression

```
1 package application;
2
3 import javafx.application.Application;
4 import javafx.event.ActionEvent;
5 import javafx.event.EventHandler;
6 import javafx.geometry.Pos;
7 import javafx.scene.Scene;
8 import javafx.scene.control.Button;
9 import javafx.scene.layout.StackPane;
10 import javafx.scene.layout.HBox;
11 import javafx.scene.layout.BorderPane;
12 import javafx.scene.paint.Color;
13 import javafx.scene.shape.Circle;
14 import javafx.stage.Stage;
15
16 public class Main extends Application {
17     @Override
18     public void start(Stage primaryStage) {
19
```



```
20 StackPane pane = new StackPane();
21 Circle circle = new Circle(50);
22 circle.setStroke(Color.BLACK);
23 circle.setFill(Color.WHITE);
24 pane.getChildren().add(circle);
25
26 // Hold two buttons in an HBox
27 HBox hBox = new HBox();
28 hBox.setSpacing(10);
29 hBox.setAlignment(Pos.CENTER);
30 Button btEnlarge = new Button("Enlarge");
31 Button btShrink = new Button("Shrink");
32 hBox.getChildren().add(btEnlarge);
33 hBox.getChildren().add(btShrink);
34
```



```
35 // Create and register the handler
36 btEnlarge.setOnAction(e -> {
37     circle.setRadius(circle.getRadius() + 2);
38 });
39
40 btShrink.setOnAction(e -> {
41     circle.setRadius(circle.getRadius() > 2 ?
42         circle.getRadius() - 2 : circle.getRadius());
43 });
44
45 BorderPane borderPane = new BorderPane();
46 borderPane.setCenter(pane);
47 borderPane.setBottom(hBox);
48 BorderPane.setAlignment(hBox, Pos.CENTER);
49
```

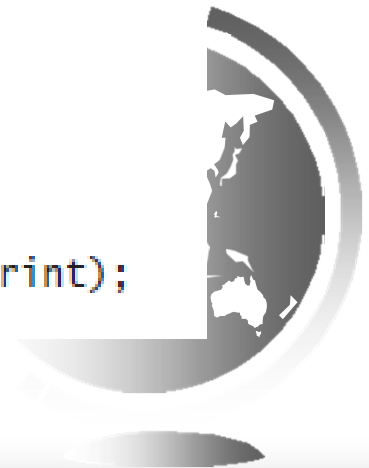


```
50 // Create a scene and place it in the stage
51 Scene scene = new Scene(borderPane, 200, 150);
52 primaryStage.setTitle("ControlCircle"); // Set the stage title
53 primaryStage.setScene(scene); // Place the scene in the stage
54 primaryStage.show(); // Display the stage
55 }
56 public static void main(String[] args) {
57     launch(args);
58 }
59 }
```



Example2 of Lambda Expression

```
1 import javafx.application.Application;
2 import javafx.event.ActionEvent;
3 import javafx.geometry.Pos;
4 import javafx.scene.Scene;
5 import javafx.scene.control.Button;
6 import javafx.scene.layout.HBox;
7 import javafx.stage.Stage;
8
9 public class LambdaHandlerDemo extends Application {
10     @Override // Override the start method in the Application class
11     public void start(Stage primaryStage) {
12         // Hold two buttons in an HBox
13         HBox hBox = new HBox();
14         hBox.setSpacing(10);
15         hBox.setAlignment(Pos.CENTER);
16         Button btNew = new Button("New");
17         Button btOpen = new Button("Open");
18         Button btSave = new Button("Save");
19         Button btPrint = new Button("Print");
20         hBox.getChildren().addAll(btNew, btOpen, btSave, btPrint);
21     }
```

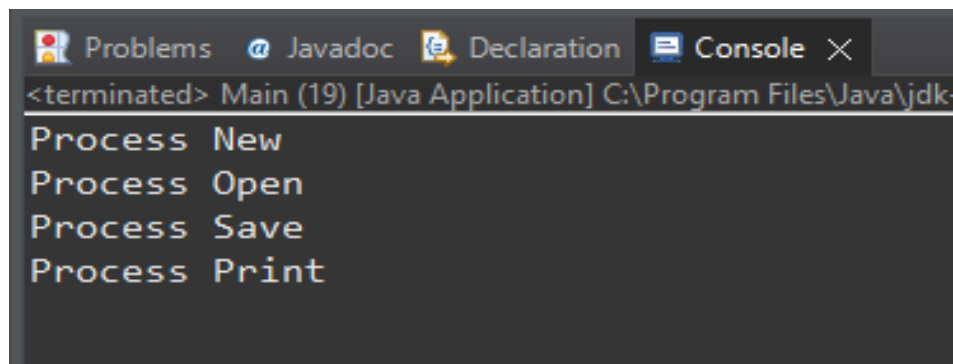
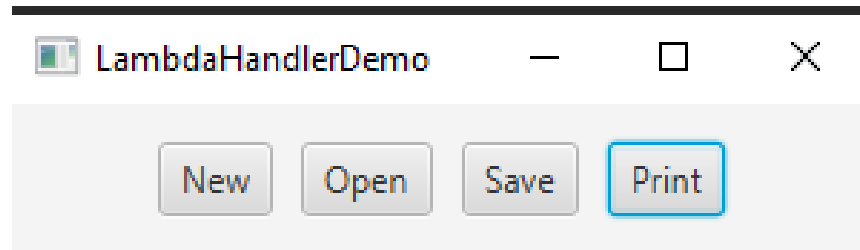


```
22 // Create and register the handler
23 btNew.setAction((ActionEvent e) -> {
24     System.out.println("Process New");
25 });
26
27 btOpen.setAction((e) -> {
28     System.out.println("Process Open");
29 });
30
31 btSave.setAction(e -> {
32     System.out.println("Process Save");
33 });
34
35 btPrint.setAction(e -> System.out.println("Process Print"));
```

4 different
variations of
Lambda
handlers



```
36
37 // Create a scene and place it in the stage
38 Scene scene = new Scene(hBox, 300, 50);
39 primaryStage.setTitle("LambdaHandlerDemo"); // Set title
40 primaryStage.setScene(scene); // Place the scene in the stage
41 primaryStage.show(); // Display the stage
42 }
43 }
```



Single Abstract Method Interface (SAM)

The compiler treats a lambda expression as if it is an object created from an anonymous inner class. In this case, the compiler understands that the object must be an instance of **EventHandler<ActionEvent>**. Since the **EventHandler** interface defines the **handle** method with a parameter of the **ActionEvent** type, the compiler automatically recognizes that **e** is a parameter of the **ActionEvent** type, and the statements are for the body of the **handle** method. The **EventHandler** interface contains just one method. The statements in the lambda expression are 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.



Inner Classes vs. Anonymous Inner Classes vs. Lambda Expressions

Inner class

```
// Create and register the handler
btEnlarge.setOnAction(new EnlargeHandler());

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

Anonymous Inner class

```
btEnlarge.setOnAction(new EventHandler<ActionEvent>() {
    @Override
    public void handle(ActionEvent event) {
        circle.setRadius(circle.getRadius()+2);
    }
});
```

Lambda Expression

```
// Create and register the handler
btEnlarge.setOnAction(e -> {
    circle.setRadius(circle.getRadius() + 2);
});
```



Inner Classes vs. Anonymous Inner Classes vs. Lambda Expressions

You can handle events by defining **handler classes** using **inner classes**, **anonymous inner classes**, or **lambda expressions**. We recommend that you use lambda expressions because it produces a shorter, clearer, and cleaner code.



Problem: Loan Calculator

The screenshot shows a Java Swing window titled "LoanCalculator" with a standard title bar (minimize, maximize, close buttons). The window contains a grid of components. On the left side, there are five labels: "Annual Interest Rate:", "Number of Years:", "Loan Amount:", "Monthly Payment:", and "Total Payment:". To the right of each label is a text field. The text fields contain the values "4.5", "4", "5000", "\$114.02", and "\$5472.84" respectively. At the bottom right of the window is a "Calculate" button. Three annotations with arrows point to specific components: "GridPane" points to the top right corner of the window; "Text field is right aligned" points to the text field containing "5000"; and "Button is right aligned" points to the "Calculate" button.

| Label | Value |
|-----------------------|-----------|
| Annual Interest Rate: | 4.5 |
| Number of Years: | 4 |
| Loan Amount: | 5000 |
| Monthly Payment: | \$114.02 |
| Total Payment: | \$5472.84 |

Annotations:

- GridPane
- Text field is right aligned
- Button is right aligned

.9 The program computes loan payments.

Problem: Loan Calculator

```
1  import javafx.application.Application;
2  import javafx.geometry.Pos;
3  import javafx.geometry.HPos;
4  import javafx.scene.Scene;
5  import javafx.scene.control.Button;
6  import javafx.scene.control.Label;
7  import javafx.scene.control.TextField;
8  import javafx.scene.layout.GridPane;
9  import javafx.stage.Stage;
10
11 public class LoanCalculator extends Application {
12     private TextField tfAnnualInterestRate = new TextField();
13     private TextField tfNumberOfYears = new TextField();
14     private TextField tfLoanAmount = new TextField();
15     private TextField tfMonthlyPayment = new TextField();
16     private TextField tfTotalPayment = new TextField();
17     private Button btCalculate = new Button("Calculate");
18
19     @Override // Override the start method in the Application class
20     public void start(Stage primaryStage) {
21         // Create UI
22         GridPane gridPane = new GridPane();
```

```
23     gridPane.setHgap(5);
24     gridPane.setVgap(5);
25     gridPane.add(new Label("Annual Interest Rate:"), 0, 0);
26     gridPane.add(tfAnnualInterestRate, 1, 0);
27     gridPane.add(new Label("Number of Years:"), 0, 1);
28     gridPane.add(tfNumberOfYears, 1, 1);
29     gridPane.add(new Label("Loan Amount:"), 0, 2);
30     gridPane.add(tfLoanAmount, 1, 2);
31     gridPane.add(new Label("Monthly Payment:"), 0, 3);
32     gridPane.add(tfMonthlyPayment, 1, 3);
33     gridPane.add(new Label("Total Payment:"), 0, 4);
34     gridPane.add(tfTotalPayment, 1, 4);
35     gridPane.add(btCalculate, 1, 5);
36
37     // Set properties for UI
38     gridPane.setAlignment(Pos.CENTER);
39     tfAnnualInterestRate.setAlignment(Pos.BOTTOM_RIGHT);
40     tfNumberOfYears.setAlignment(Pos.BOTTOM_RIGHT);
41     tfLoanAmount.setAlignment(Pos.BOTTOM_RIGHT);
42     tfMonthlyPayment.setAlignment(Pos.BOTTOM_RIGHT);
43     tfTotalPayment.setAlignment(Pos.BOTTOM_RIGHT);
44     tfMonthlyPayment.setEditable(false);
45     tfTotalPayment.setEditable(false);
46     GridPane.setHalignment(btCalculate, HPos.RIGHT);
47
```

```
49     btCalculate.setOnAction(e -> calculateLoanPayment());
50
51     // Create a scene and place it in the stage
52     Scene scene = new Scene(gridPane, 400, 250);
53     primaryStage.setTitle("LoanCalculator"); // Set title
54     primaryStage.setScene(scene); // Place the scene in the stage
55     primaryStage.show(); // Display the stage
56 }
57
58 private void calculateLoanPayment() {
59     // Get values from text fields
60     double interest =
61         Double.parseDouble(tfAnnualInterestRate.getText());
62     int year = Integer.parseInt(tfNumberOfYears.getText());
63     double loanAmount =
64         Double.parseDouble(tfLoanAmount.getText());
65
66     // Create a loan object. Loan defined in Listing 10.2
67     Loan loan = new Loan(interest, year, loanAmount);
68
69     // Display monthly payment and total payment
70     tfMonthlyPayment.setText(String.format("%.2f",
71         loan.getMonthlyPayment()));
72     tfTotalPayment.setText(String.format("%.2f",
73         loan.getTotalPayment()));
74 }
75 }
```