Chapter 4
Control Statements: Part I

OBJECTIVES

In this chapter you'll learn:

- Basic problem-solving techniques.
- To develop algorithms through the process of top-down, stepwise refinement.
- To use the `if` and `if...else` selection statements to choose among alternative actions.
- To use the `while` repetition statement to execute statements in a program repeatedly.
- To use counter-controlled repetition and sentinel-controlled repetition.
- To use the compound assignment, increment and decrement operators.
- The portability of primitive data types.

4.1 Introduction

- This chapter introduces
  - The `if`, `if...else` and `while` statements
  - Compound assignment, increment and decrement operators
  - Portability of Java's primitive types

4.2 Pseudocode

- Particularly useful for developing algorithms that will be converted to structured portions of Java programs.

4.3 Control Structures

- Sequence structure

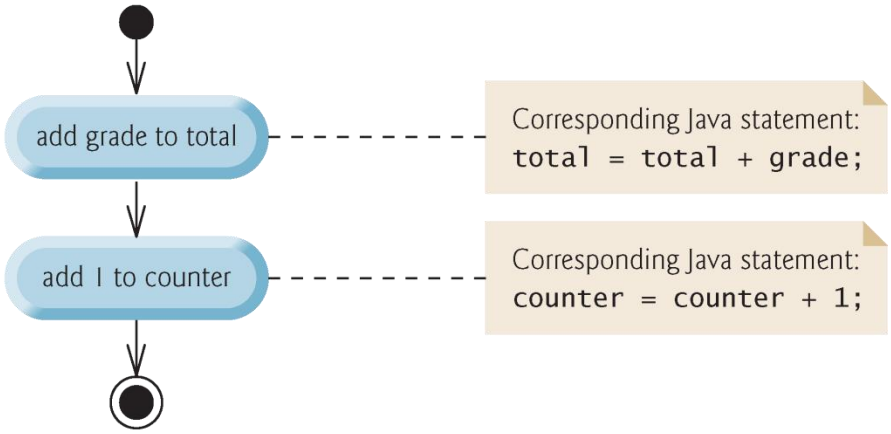


Fig. 4.1 | Sequence structure activity diagram.

4.3 Control Structures (Cont.)

- Three types of selection statements.
  - `if` statement:
    - Single-selection statement
  - `if...else` statement:
    - Double-selection statement
  - `switch` statement
    - Multiple-selection statement

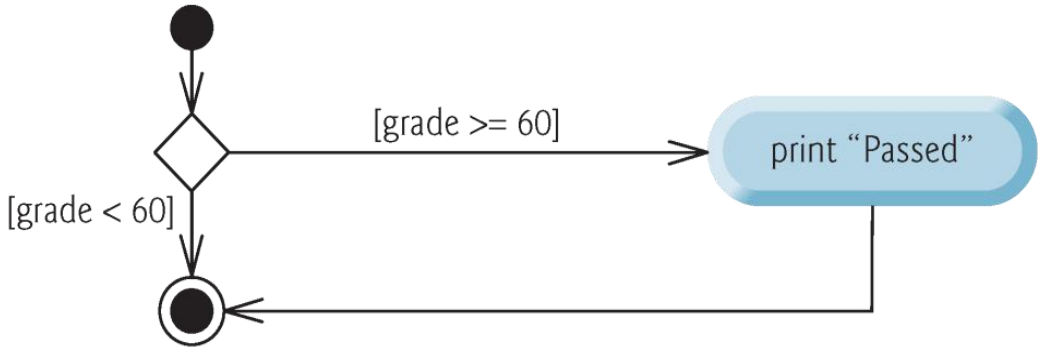


Fig. 4.2 | `if` single-selection statement UML activity diagram.

```
if (studentGrade >= 60)
   System.out.println("Passed");
```

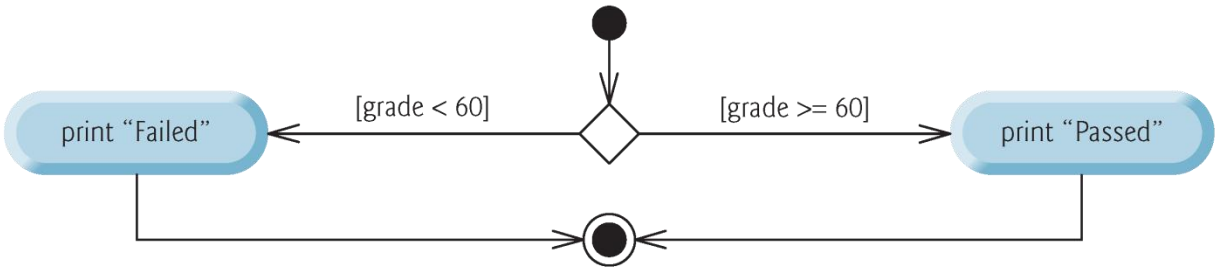


Fig. 4.3 | `if...else` double-selection statement UML activity diagram.

```
if (grade >= 60)
   System.out.println("Passed");
else
   System.out.println("Failed");
```

Nested if...else Statements

```
if (studentGrade >= 90)
   System.out.println("A");
else
   if (studentGrade >= 80)
      System.out.println("B");
   else
      if (studentGrade >= 70)
         System.out.println("C");
      else
         if (studentGrade >= 60)
            System.out.println("D");
         else
            System.out.println("F");
```

```
if (average >= 90.0)
   letterGrade = "A";
else if (average >= 80.0)
   letterGrade = "B";
else if (average >= 70.0)
   letterGrade = "C";
else if (average >= 60.0)
   letterGrade = "D";
else
   letterGrade = "F";
```

4.3 `if...else` Double-Selection Statement (Cont.)

- Conditional operator (?:)—shorthand `if...else`.
  - Ternary operator (三元运算符 takes three operands)
- Example:

```
System.out.println(
   studentGrade >= 60 ? "Passed" : "Failed" );
```

- Evaluates to the string `"Passed"` if the `boolean` expression `studentGrade >= 60` is true and to the string `"Failed"` if it is false.

Good Programming Practice 4.3

Conditional expressions are more difficult to read than `if...else` *statements and should be used to replace only simple* `if...else` *statements that choose between two values.*

4.3 Control Structures (Cont.)

- `while` and `for` statements perform the action(s) in their bodies zero or more times
- The `do…while` statement performs the action(s) in its body one or more times.

while

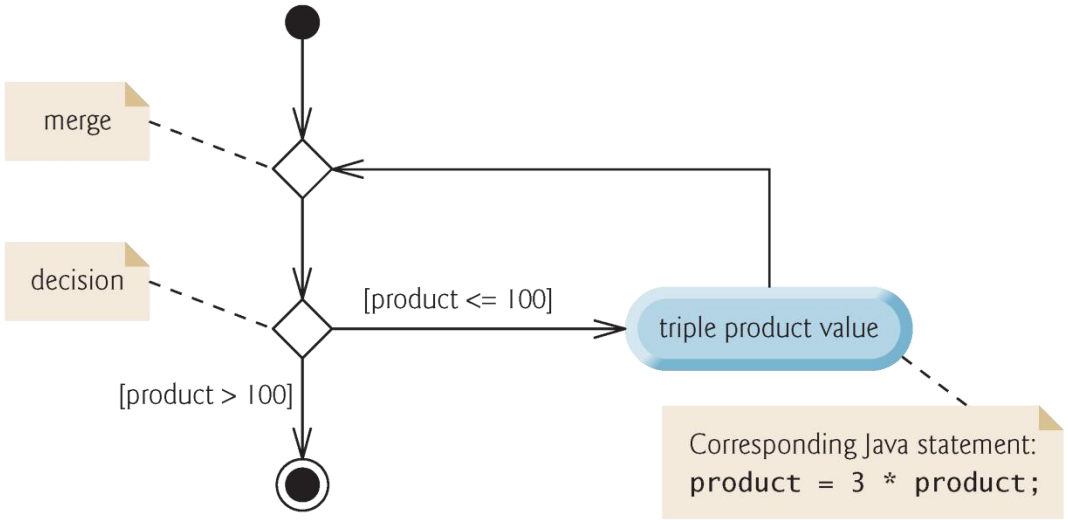


Fig. 4.4 | `while` repetition statement UML activity diagram.

```
while ( product <= 100 )
   product = 3 * product;
```

counter-controlled repetition

```
1  // Fig. 4.8: ClassAverage.java
2  // Solving the class-average problem using counter-controlled repetition.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class ClassAverage
6  {
7     public static void main(String[] args)
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner(System.in);
11
12       // initialization phase
13       int total = 0;  // initialize sum of grades entered by the user
14       int gradeCounter = 1; // initialize # of grade to be entered next
15
16       // processing phase uses counter-controlled repetition
17       while (gradeCounter <= 10) // loop 10 times
18       {
19          System.out.print("Enter grade: "); // prompt
20          int grade = input.nextInt(); // input next grade
21          total = total + grade; // add grade to total
22          gradeCounter = gradeCounter + 1; // increment counter by 1
23       }
24
25       // termination phase
26       int average = total / 10; // integer division yields integer result
27
28       // display total and average of grades
29       System.out.printf("%nTotal of all 10 grades is %d%n", total);
30       System.out.printf("Class average is %d%n", average);
31    }
32 } // end class ClassAverage
```

```
Enter grade: 67
Enter grade: 78
Enter grade: 89
Enter grade: 67
Enter grade: 87
Enter grade: 98
Enter grade: 93
Enter grade: 85
Enter grade: 82
Enter grade: 100

Total of all 10 grades is 846
Class average is 84
```

Sentinel-controlled repetition

```
1  // Fig. 4.10: ClassAverage.java
2  // Solving the class-average problem using sentinel-controlled repetition.
3  import java.util.Scanner; // program uses class Scanner
4
5  public class ClassAverage
6  {
7     public static void main(String[] args)
8     {
9        // create Scanner to obtain input from command window
10       Scanner input = new Scanner(System.in);
11
12       // initialization phase
13       int total = 0; // initialize sum of grades
14       int gradeCounter = 0; // initialize # of grades entered so far
15
16       // processing phase
17       // prompt for input and read grade from user
18       System.out.print("Enter grade or -1 to quit: ");
19       int grade = input.nextInt();
```

Sentinel-controlled repetition

```
21          // loop until sentinel value read from user
22          while (grade != -1)
23          {
24             total = total + grade; // add grade to total
25             gradeCounter = gradeCounter + 1; // increment counter
26
27             // prompt for input and read next grade from user
28             System.out.print("Enter grade or -1 to quit: ");
29             grade = input.nextInt();
30          }
31
32          // termination phase
33          // if user entered at least one grade...
34          if (gradeCounter != 0)
35          {
36             // use number with decimal point to calculate average of grades
37             double average = (double) total / gradeCounter;
38
39             // display total and average (with two digits of precision)
40             System.out.printf("%nTotal of the %d grades entered is %d%n",
41                gradeCounter, total);
42             System.out.printf("Class average is %.2f%n", average);
43          }
44          else // no grades were entered, so output appropriate message
45             System.out.println("No grades were entered");
46       }
47    } // end class ClassAverage
```

```
Enter grade or -1 to quit: 97
Enter grade or -1 to quit: 88
Enter grade or -1 to quit: 72
Enter grade or -1 to quit: -1

Total of the 3 grades entered is 257
Class average is 85.67
```

Nested Control Statements

- Control statements can be stacked on top of one another

```
18          while (studentCounter <= 10)
19          {
20             // prompt user for input and obtain value from user
21             System.out.print("Enter result (1 = pass, 2 = fail): ")
22             int result = input.nextInt();
23
24             // if...else is nested in the while statement
25             if (result == 1)
26                passes = passes + 1;
27             else
28                failures = failures + 1;
29
30             // increment studentCounter so loop eventually terminates
31             studentCounter = studentCounter + 1;
32          }
33
```

4.4 Compound Assignment Operators

Assignment operator	Sample expression	Explanation	Assigns
Assume: `int c = 3, d = 5, e = 4, f = 6, g = 12;`			
`+=`	`c += 7`	`c = c + 7`	10 to c
`-=`	`d -= 4`	`d = d - 4`	1 to d
`*=`	`e *= 5`	`e = e * 5`	20 to e
`/=`	`f /= 3`	`f = f / 3`	2 to f
`%=`	`g %= 9`	`g = g % 9`	3 to g

Fig. 4.13 | Arithmetic compound assignment operators.

Is "a+=b" same as "a =a + b" ?

```
short a = 3;
int b = 5;
a += b;
a = a + b;
```

√

Type mismatch: cannot convert from int to short

4.5 Increment and Decrement Operators

Operator	Operator name	Sample expression	Explanation
`++`	prefix increment	`++a`	Increment `a` by 1, then use the new value of `a` in the expression in which `a` resides.
`++`	postfix increment	`a++`	Use the current value of `a` in the expression in which `a` resides, then increment `a` by 1.
`--`	prefix decrement	`--b`	Decrement `b` by 1, then use the new value of `b` in the expression in which `b` resides.
`--`	postfix decrement	`b--`	Use the current value of `b` in the expression in which `b` resides, then decrement `b` by 1.

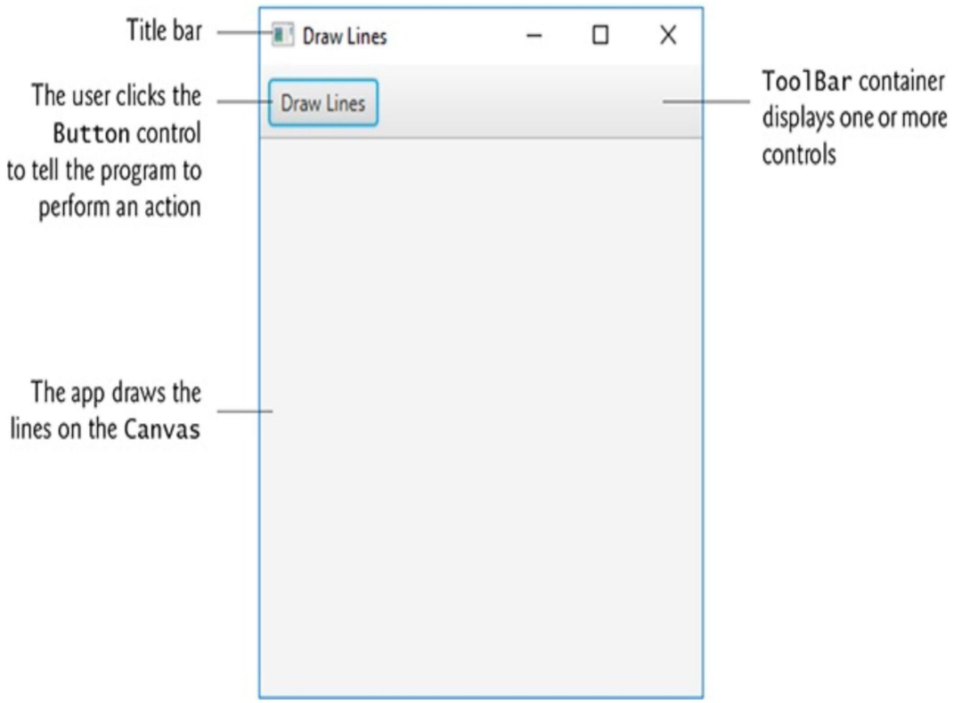
Fig. 4.14 | Increment and decrement operators.

4.6 Primitive Types

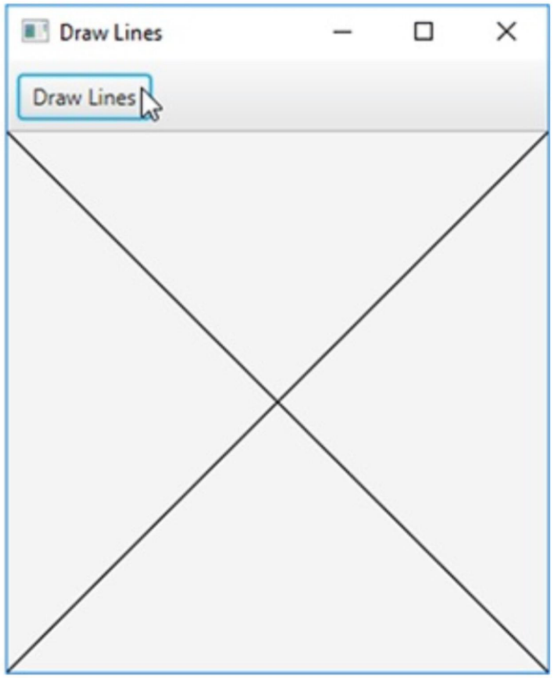
- Appendix D lists the eight primitive types in Java.
- Java requires all variables to have a type.
- Java is a strongly typed language.
- Primitive types in Java are portable across all platforms.
- Instance variables of types `char`, `byte`, `short`, `int`, `long`, `float` and `double` are all given the value `0` by default. Instance variables of type `boolean` are given the value false by default.
- Reference-type instance variables are initialized by default to the value null.

4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines

a) Initial **Draw Lines** app window

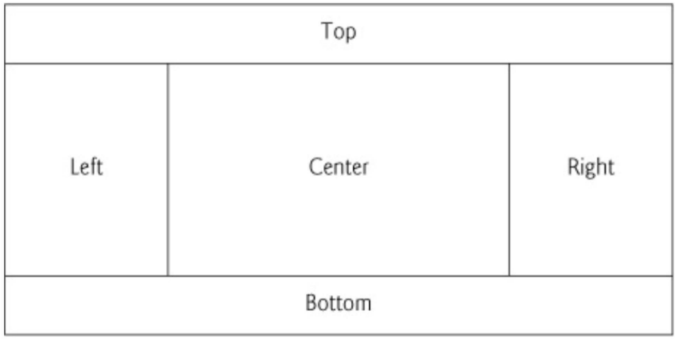


b) **Draw Lines** app window after the user clicks the **Draw Lines** `Button`



4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Controls
  - Canvas control—a rectangular area in which you can draw.
  - BorderPane layout container arranges controls into one or more of the five regions.



4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Controls
  - **ToolBar** layout container
    - ToolBars typically organize multiple controls at a layout's edges, such as in a BorderPane's top, right,bottom or left areas.
    - **ToolBar** arranges controls horizontally by defult.
    - Initial ToolBar containing a Button – by default the Tool Bar is the full width of the BorderPane's top area.

4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- **Design the GUI**
  - Drag-and-drop a BorderPane from the Library window's Containers section onto Scene Builder's content panel.
  - Drag-and-drop a **ToolBar** layout container to the BorderPane's top region.
    - double-click the Button to edit its text
  - Drag-and-Drop a Canvas from the Scene Builder Library's Miscellaneous section to the BorderPan's center area.
    - Width ,Height, Pref Width, Pref Height
  - File > Save

4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- For the ToolBar, set the following properties:
  - text——Draw Lines
- For the Canvas, set the following properties:
  - Width——300
  - Height——300
- For the BorderPane, set the following properties:
  - Pref Width, Pref Height——USE_COMPUTED_SIZE

4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Preparing to Interact with the GUI Programmatically
  - JavaFX GUI controls also are objects that you can interact with programmatically. For GUIs created in Scene Builder, *JavaFX creates the controls for you* when the app begins executing.

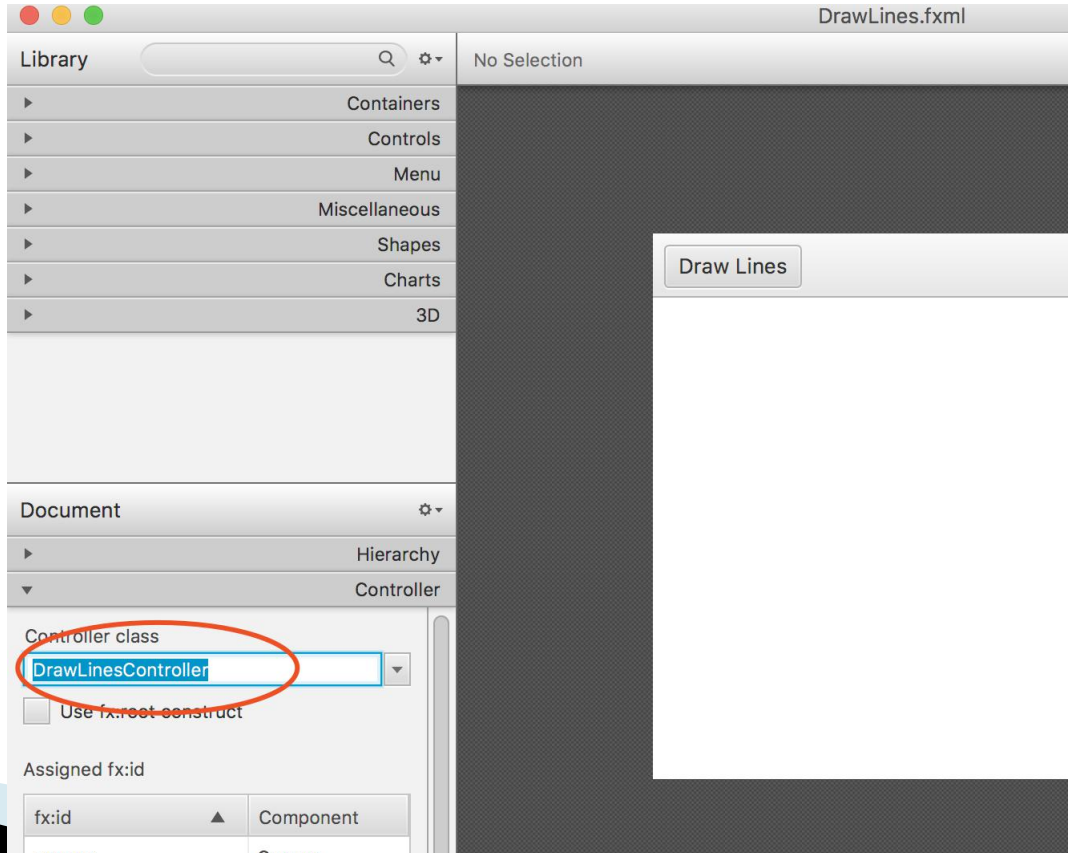
4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Preparing to Interact with the GUI Programmatically
  1. Specifying the App's Controller Class
     - When you launch a JavaFX app with an FXML GUI, the JavaFX runtime:
       - creates the GUI,
       - creates an object of the controller class that you specified in Scene Builder,
       - initializes any of the controller object's instance variables that you specified in Scene Builder, so they refer to the corresponding control objects, and
       - configures any of the controller object's event handlers that you [illegible] Builder, so that they'll be called when the user [illegible] corresponding control(s).

4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Preparing to Interact with the GUI Programmatically
  1. Specifying the App's Controller Class

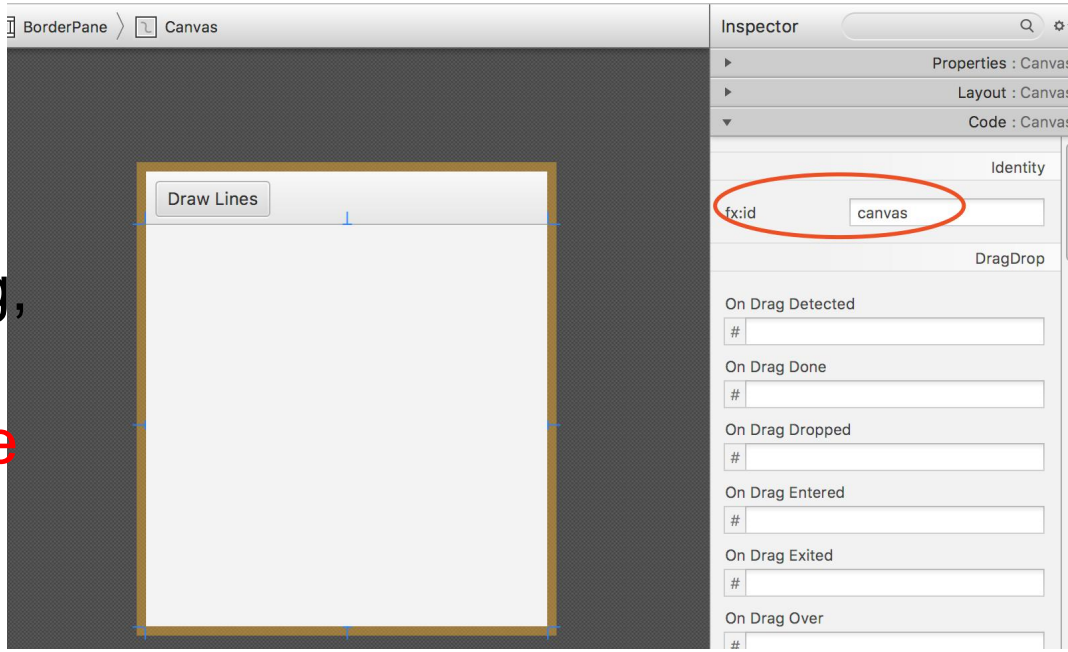
To specify the controller class's name



4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Preparing to Interact with the GUI Programmatically
  - 2. Specifying the Canvas's Instance Variable Name

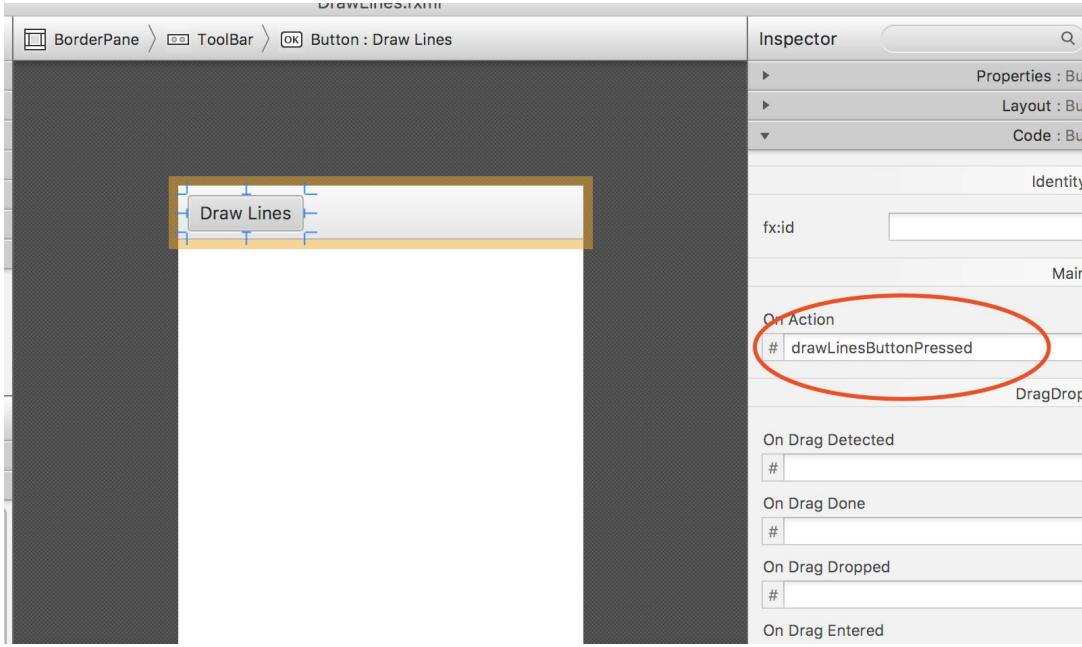
A control's fx:id property specifies the instance variable's name—when the app begins executing, JavaFX initializes the instance variable with the corresponding control object



4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Preparing to Interact with the GUI Programmatically
  - 3. Specifying the Button's Event-Handler Method

An event handler is a method that's called in response to a user interaction.
The method is called by the JavaFX runtime on an object of the controller class

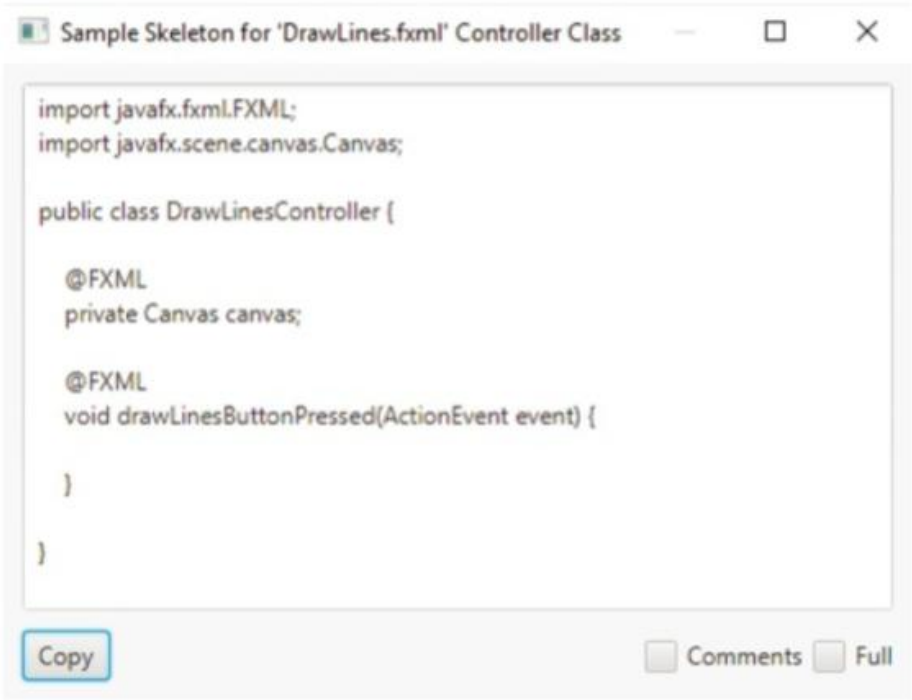


4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Preparing to Interact with the GUI Programmatically
  - 4. Generating the Initial Controller Class

Select View > Show Sample Controller Skeleton to display the class

- create a Java source-code file named DrawLinesController.java in the same folder as DrawLines.fxml
- paste the code into that file and save the file. You can add the program's logic into the file.



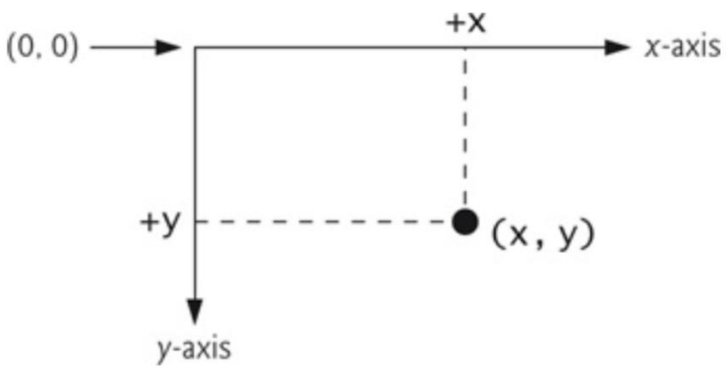
```java
// Fig. 4.27: DrawLinesController.java
import javafx.event.ActionEvent;

public class DrawLinesController {
   @FXML private Canvas canvas; // used to get the GraphicsContext

   // when user presses Draw Lines button, draw two Lines in the Canvas
   @FXML
   void drawLinesButtonPressed(ActionEvent event) {
      // get the GraphicsContext, which is used to draw on the Canvas
      GraphicsContext gc = canvas.getGraphicsContext2D();

      // draw line from upper-left to lower-right corner
      gc.strokeLine(0, 0, canvas.getWidth(), canvas.getHeight());

      // draw line from upper-right to lower-left corner
      gc.strokeLine(canvas.getWidth(), 0, 0, canvas.getHeight());
   }
}
```



4.7 (Optional) GUI and Graphics Case Study: Event handling; Drawing Lines (Cont.)

- Preparing to Interact with the GUI Programmatically
  - 5. Generating the main-application class
    - The main-applicaiton class loads the app's FXML file, creates the GUI then creates and configures the corresponding controller-class.
    - By convention, this class has the same base name as the FXML file (DrawLines in DrawLines.fxml).

```
// Fig. 4.28: DrawLines.java
import javafx.application.Application;

public class DrawLines extends Application {
   @Override
   public void start(Stage stage) throws Exception {
      // loads DrawLines.fxml and configures the DrawLinesController
      Parent root =
         FXMLLoader.load(getClass().getResource("DrawLines.fxml"));

      Scene scene = new Scene(root); // attach scene graph to scene
      stage.setTitle("Draw Lines"); // displayed in window's title bar
      stage.setScene(scene); // attach scene to stage
      stage.show(); // display the stage
   }

   public static void main(String[] args) {
      launch(args); // create a DrawLines object and call its start method
   }
}
```