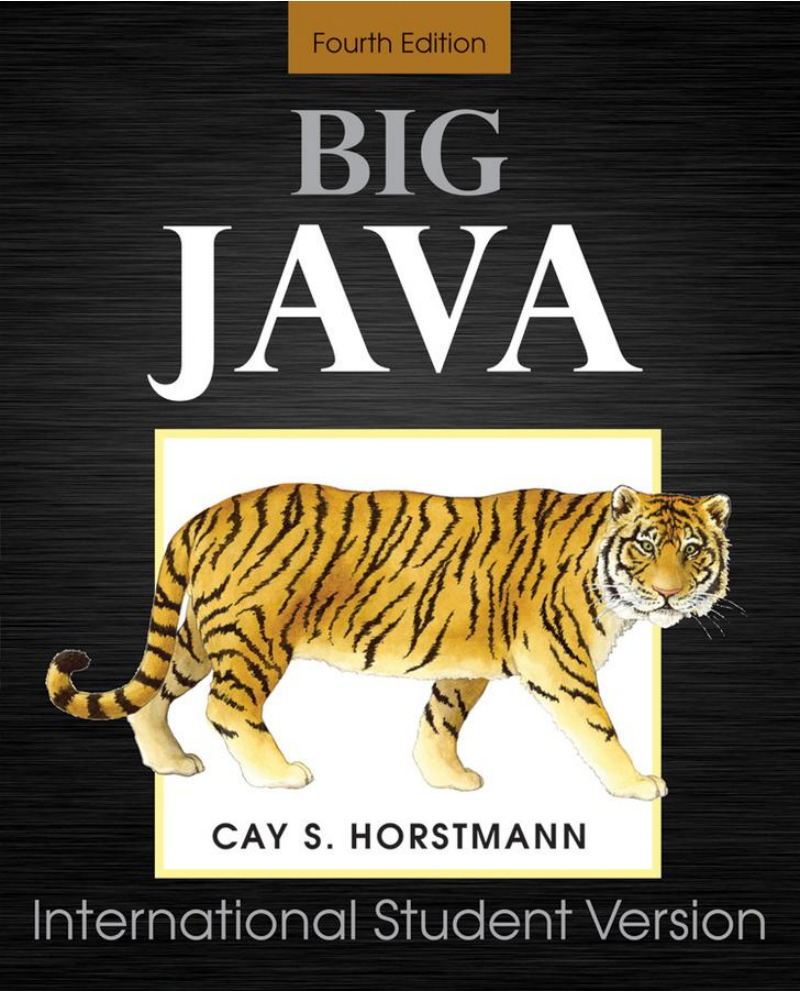


Chapter 2 – An Introduction to Objects and Classes

Chapter Goals

- To learn about variables
- To understand the concepts of classes and objects
- To be able to call methods
- To learn about parameters and return values
- To be able to browse the API documentation

T To implement test programs

- To understand the difference between objects and object references

G To write programs that display simple shapes

Types

- A **type** defines a set of values and the operations that can be carried out on the values
- Examples:
  - `13` *has type* `int`
  - `"Hello, World"` *has type* `String`
  - `System.out` *has type* `PrintStream`
- Java has separate types for **integers** and **floating-point numbers**
  - *The* `double` *type denotes floating-point numbers*
- A value such as `13` or `1.3` that occurs in a Java program is called a **number literal**

Number Literals

Table 1 Number Literals in Java

Number	Type	Comment
6	`int`	An integer has no fractional part.
–6	`int`	Integers can be negative.
0	`int`	Zero is an integer.
0.5	`double`	A number with a fractional part has type `double`.
1.0	`double`	An integer with a fractional part .0 has type `double`.
`1E6`	`double`	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type `double`.
`2.96E-2`	`double`	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
100,000		**Error:** Do not use a comma as a decimal separator.
3 1/2		**Error:** Do not use fractions; use decimal notation: 3.5.

Number Types

- A **type** defines a set of values and the operations that can be carried out on the values
- Number types are **primitive types**
  - *Numbers are not objects*
- Numbers can be combined by arithmetic operators such as `+`, `-`, and `*`

Self Check 2.1

What is the type of the values `0` and `"0"`?

Answer: `int` and `String`.

Self Check 2.2

Which number type would you use for storing the area of a circle?

Answer: `double`.

Self Check 2.3

Why is the expression `13.println()` an error?

Answer: An `int` is not an object, and you cannot call a method on it.

Self Check 2.4

Write an expression to compute the average of the values `x` and `y`.

Answer: `(x + y) * 0.5`

Variables

- Use a **variable** to store a value that you want to use at a later time
- A variable has a type, a name, and a value:

```
String greeting = "Hello, World!”
PrintStream printer = System.out;
int width = 13;
```

- Variables can be used in place of the values that they store:

```
printer.println(greeting);
// Same as System.out.println("Hello, World!”)
printer.println(width);
// Same asSystem.out.println(20)
```

Variables

- It is an error to store a value whose type does not match the type of the variable:

```
String greeting = 20; // ERROR: Types don't match
```

Variable Declarations

Table 2 Variable Declarations in Java

Variable Name	Comment
`int width = 10;`	Declares an integer variable and initializes it with 10.
`int area = width * height;`	The initial value can depend on other variables. (Of course, `width` and `height` must have been previously declared.)
`height = 5;`	**Error:** The type is missing. This statement is not a declaration but an assignment of a new value to an existing variable—see Section 2.3.
`int height = "5";`	**Error:** You cannot initialize a number with a string.
`int width, height;`	Declares two integer variables in a single statement. In this book, we will declare each variable in a separate statement.

Identifiers

- **Identifier:** name of a variable, method, or class
- Rules for identifiers in Java:
  - *Can be made up of letters, digits, and the underscore (_) and dollar sign ($) characters*
  - *Cannot start with a digit*
  - *Cannot use other symbols such as ? or %*
  - *Spaces are not permitted inside identifiers*
  - *You cannot use reserved words such as public*
  - *They are case sensitive*

Identifiers

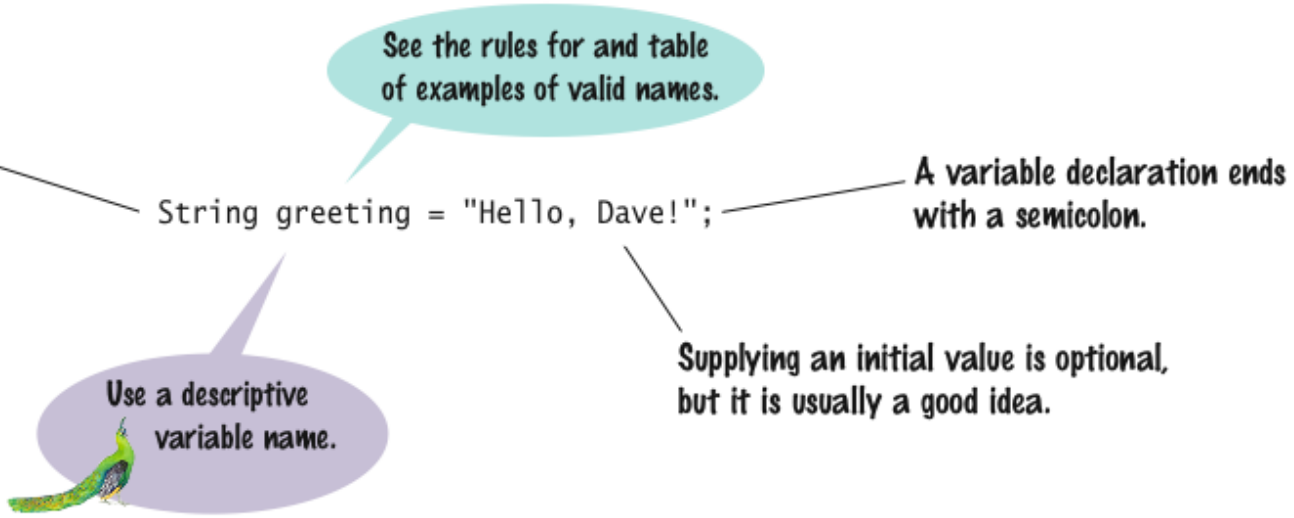
- By convention, variable names start with a lowercase letter
  - *"Camel case": Capitalize the first letter of a word in a compound word such as* `farewellMessage`
- By convention, class names start with an uppercase letter
- Do not use the `$` symbol in names — it is intended for names that are automatically generated by tools

Syntax 2.1 Variable Declaration

Syntax

typeName variableName = value;
or
typeName variableName;

Example



Variable Names

Table 3 Variable Names in Java

Variable Name	Comment
`farewellMessage`	Use "camel case" for variable names consisting of multiple words.
`x`	In mathematics, you use short variable names such as x or y. This is legal in Java, but not very common, because it can make programs harder to understand.
`Greeting`	**Caution:** Variable names are case-sensitive. This variable name is different from `greeting`.
`6pack`	**Error:** Variable names cannot start with a number.
`farewell message`	**Error:** Variable names cannot contain spaces.
`public`	**Error:** You cannot use a reserved word as a variable name.

Self Check 2.5

Which of the following are legal identifiers?

```
Greeting1
g
void
101dalmatians
Hello, World
<greeting>
```

Answer: Only the first two are legal identifiers.

Self Check 2.6

Define a variable to hold your name. Use camel case in the variable name.

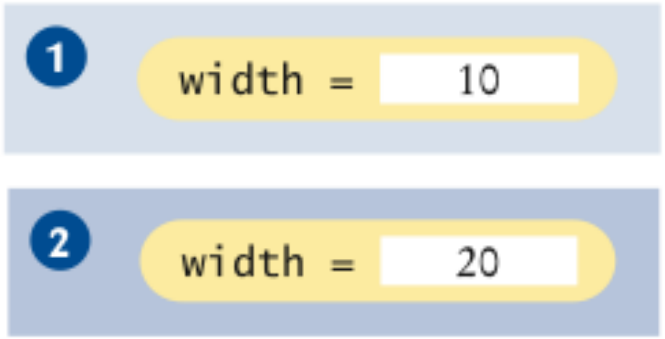
Answer:

```
String myName = "John Q. Public";
```

The Assignment Operator

- Assignment operator: =
- Used to change the value of a variable:

```
int width= 10; 1
width = 20;2
```



Uninitialized Variables

- It is an error to use a variable that has never had a value assigned to it:

```
int height;
width = height; // ERROR—uninitialized variable height
```

Figure 2
An Uninitialized Variable

- Remedy: assign a value to the variable before you use it:

```
int height = 30;
width = height; // OK
```

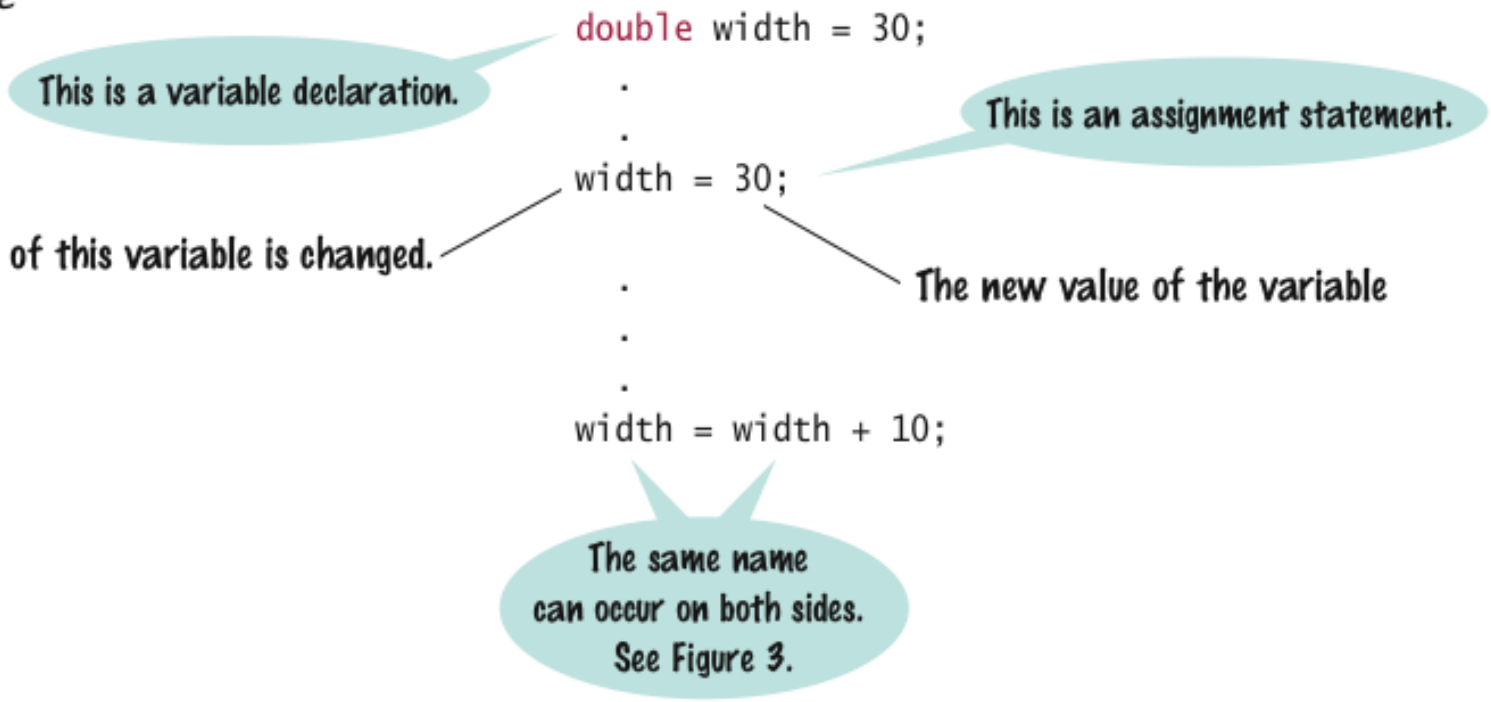
- Even better, initialize the variable when you declare it:

```
int height = 30;
int width = height; // OK
```

Syntax 2.2 Assignment

Syntax *variableName = value;*

Example



Assignment

- The right-hand side of the `=` symbol can be a mathematical expression:

```
width = height + 10;
```

- Means:

  1.compute the value of `width + 10`

  2.store that value in the variable `width`

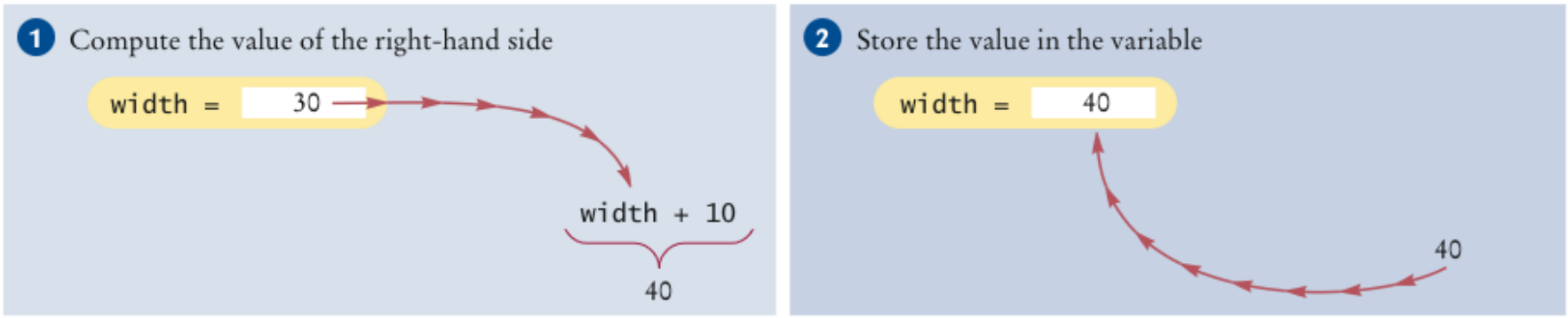


Figure 3 Executing the Statement `width = width + 10`

Animation 2.1: Variable Initialization and Assignment

```
int luckyNumber = 13;
luckyNumber = 12;
```

This animation demonstrates the process of variable initialization and assignment.

Self Check 2.7

Is `12 = 12` a valid expression in the Java language?

Answer: No, the left-hand side of the `=` operator must be a variable.

Self Check 2.8

How do you change the value of the `greeting` variable to `"Hello, Nina!"`?

Answer:

```
greeting = "Hello, Nina!";
```

Note that

```
String greeting = "Hello, Nina!";
```

is not the right answer – that statement defines a new variable.

Objects and Classes

- **Object:** entity that you can manipulate in your programs (by calling methods)
- Each object belongs to a **class**
- Example: `System.out` belongs to the class `PrintStream`

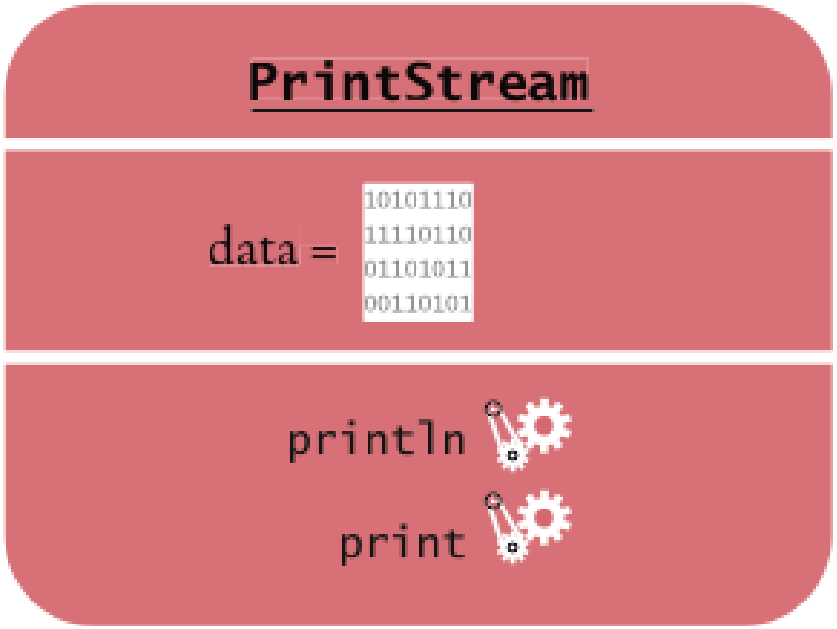


Figure 4 Representation of the `System.out` Object

Methods

- **Method:** sequence of instructions that accesses the data of an object
- You manipulate objects by calling its methods
- **Class:** declares the methods that you can apply to its objects
- Class determines legal methods:

```
String greeting = "Hello";
greeting.println() // Error
greeting.length() // OK
```

- **Public Interface:** specifies what you can do with the objects of a class

Overloaded Method

- **Overloaded method:** when a class declares two methods with the same name, but different parameters
- Example: the `PrintStream` class declares a second method, also called `println`, as

```
public void println(int output)
```

A Representation of Two String Objects

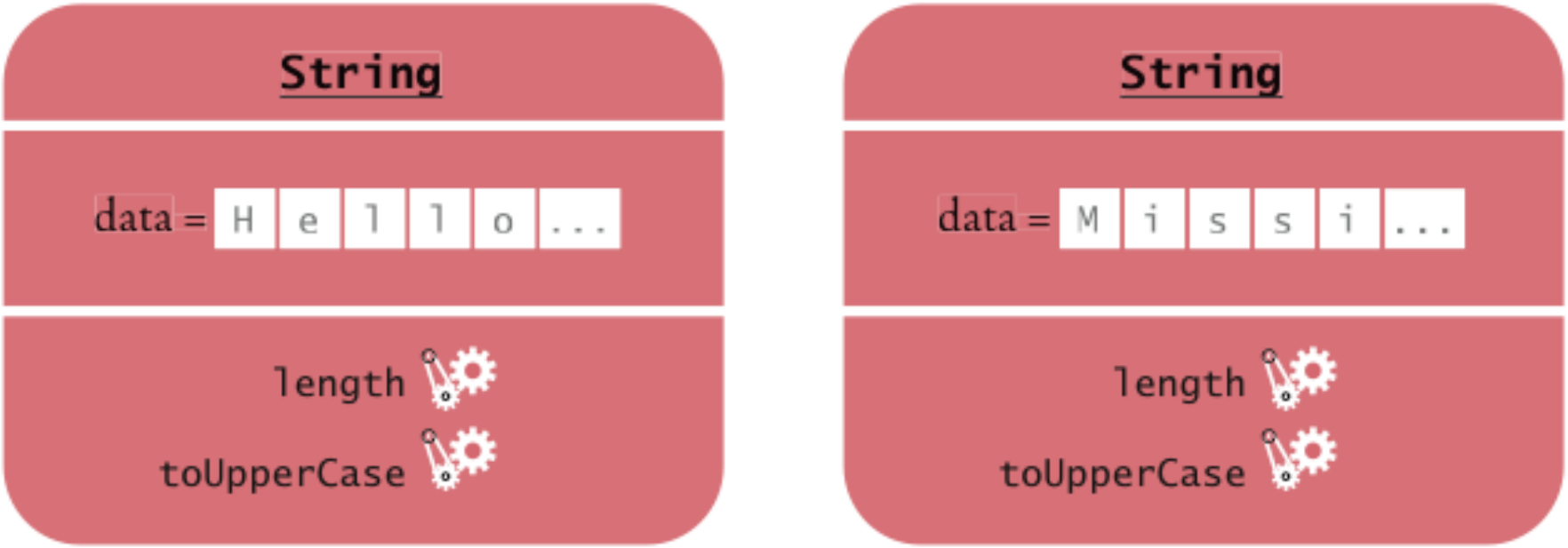


Figure 5 A Representation of Two String Objects

String Methods

- `length`: counts the number of characters in a string:

```
String greeting = "Hello, World!";
int n = greeting.length(); // sets n to 13
```

- `toUpperCase`: creates another String object that contains the characters of the original string, with lowercase letters converted to uppercase:

```
String river = "Mississippi";
String bigRiver = river.toUpperCase();
// sets bigRiver to "MISSISSIPPI"
```

- When applying a method to an object, make sure method is defined in the appropriate class:

```
System.out.length(); // This method call is an error
```

Self Check 2.9

How can you compute the length of the string `"Mississippi"`?

Answer: `river.length()` or `"Mississippi".length()`

Self Check 2.10

How can you print out the uppercase version of `"Hello, World!"`?

Answer:

```
System.out.println(greeting.toUpperCase());
```

Self Check 2.11

Is it legal to call `river.println()`? Why or why not?

Answer: It is not legal. The variable `river` has type `String`. The `println` method is not a method of the `String` class.

Parameters

- **Parameter:** an input to a method
- **Implicit parameter:** the object on which a method is invoked:

```
System.out.println(greeting)
```

- **Explicit parameters:** all parameters except the implicit parameter:

```
System.out.println(greeting)
```

- Not all methods have explicit parameters:

```
greeting.length() // has no explicit
  parameter
```

Passing a Parameter

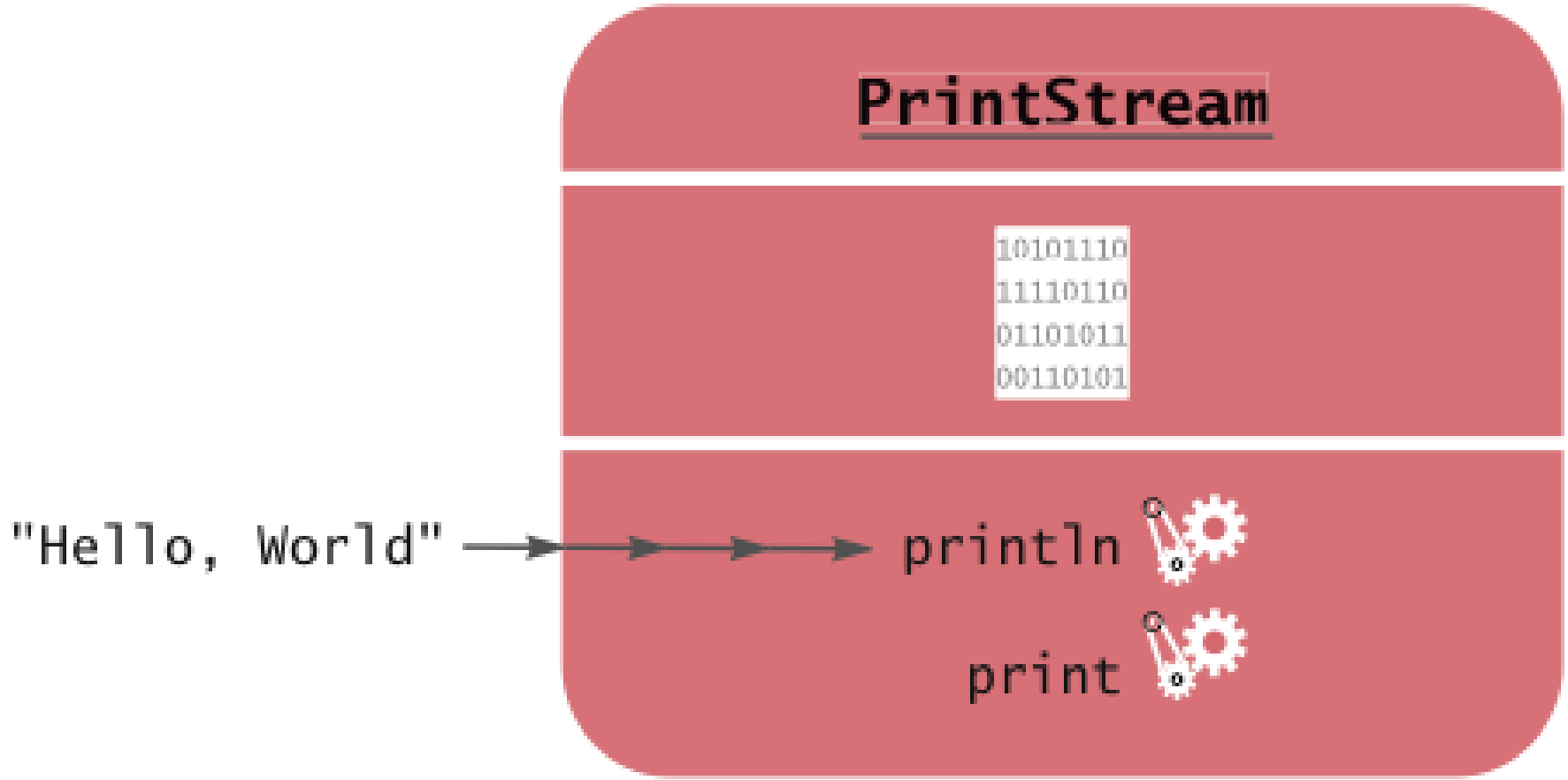


Figure 6 Passing a Parameter to the `println` Method

Return Values

- **Return value:** a result that the method has computed for use by the code that called it:

```
int n = greeting.length(); // return value stored in n
```

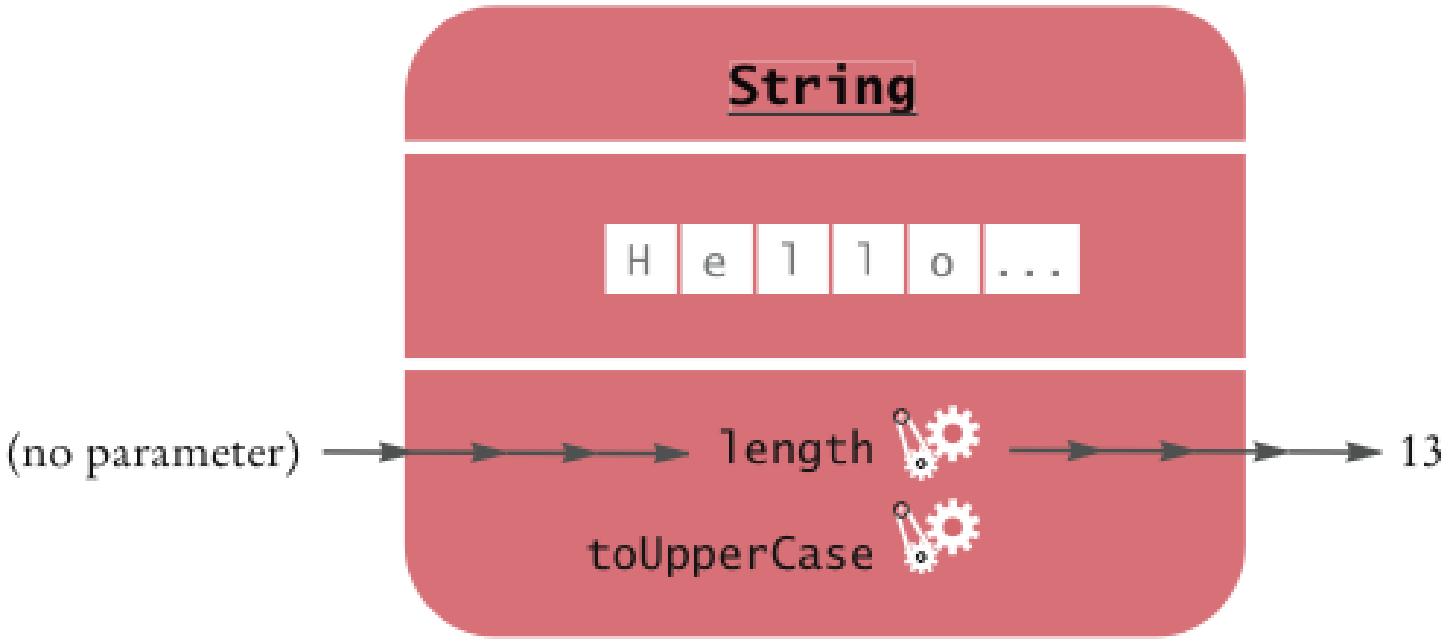


Figure 7 Invoking the `length` Method on a `String` Object

Passing Return Values

- You can also use the return value as a parameter of another method:

```
System.out.println(greeting.length());
```

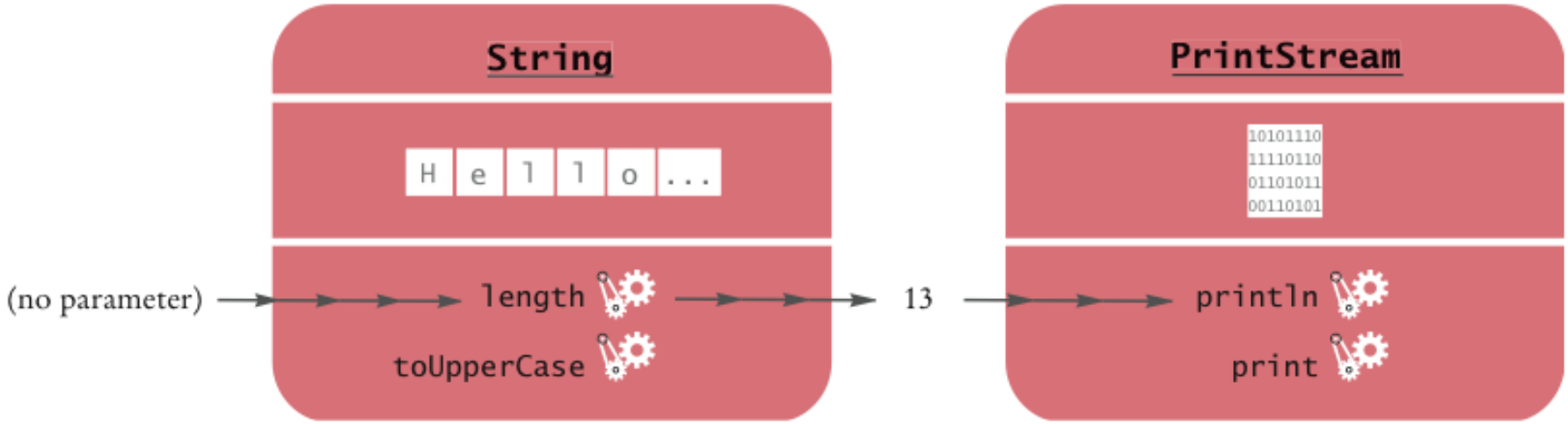


Figure 8 Passing the Result of a Method Call to Another Method

- Not all methods return values. Example: `println`

A More Complex Call

- `String` method `replace` carries out a search-and-replace operation:

```
river.replace("issipp", "our”)
// constructs a new string ("Missouri")
```

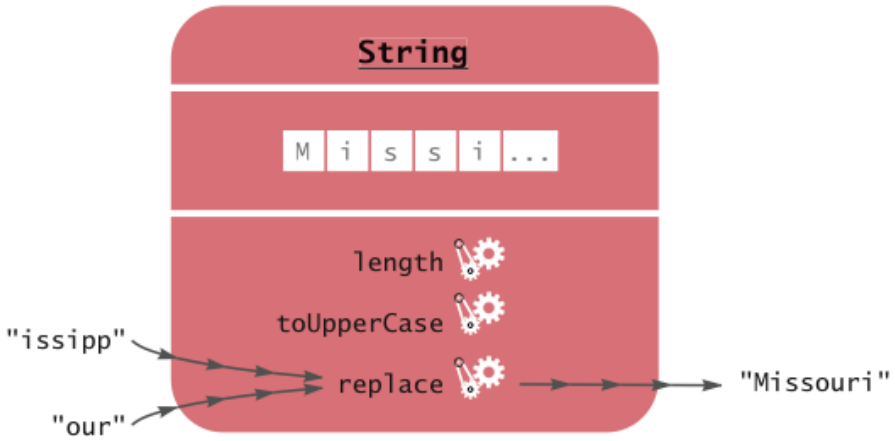


Figure 9 Calling the `replace` Method

- This method call has
  - *one implicit parameter: the string* `"Mississippi"`
  - *two explicit parameters: the strings* `"issipp"` *and* `"our"`
  - *a return value: the string* `"Missouri"`

Self Check 2.12

What are the implicit parameters, explicit parameters, and return values in the method call `river.length()`?

Answer: The implicit parameter is `river`. There is no explicit parameter. The return value is 11.

Self Check 2.13

What is the result of the call `river.replace("p", "s")`?

Answer: `"Missississi"`.

Self Check 2.14

What is the result of the call

```
greeting.replace("World", "Dave").length()
```
?

Answer: `12`.

Self Check 2.15

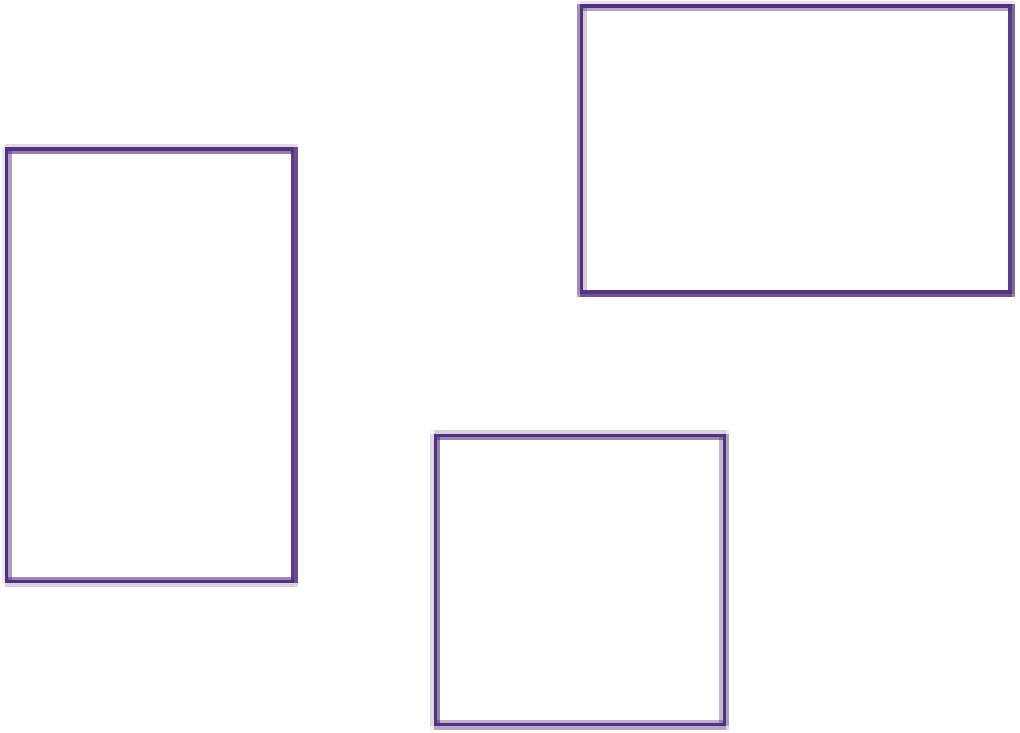
How is the `toUpperCase` method defined in the `String` class?

Answer: As `public String toUpperCase()`, with no explicit parameter and return type `String`.

Rectangular Shapes and Rectangle Objects

- Objects of type `Rectangle` *describe* rectangular shapes:

Figure 10
Rectangular Shapes



Rectangular Shapes and Rectangle Objects

- A `Rectangle` object isn't a rectangular shape – it is an object that contains a set of numbers that describe the rectangle:

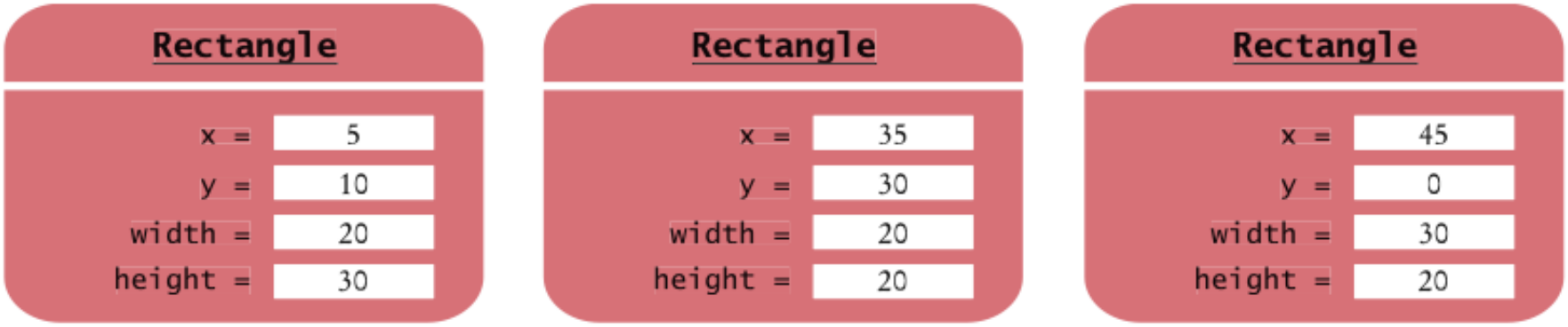


Figure 11 Rectangle Objects

Constructing Objects

```
new Rectangle(5, 10, 20, 30)
```

- Detail:
  1. *The* `new` *operator makes a* `Rectangle` *object*
  2. *It uses the parameters (in this case,* `5`*,* `10`*,* `20`*, and* `30`*) to initialize the data of the object*
  3. *It returns the object*
- Usually the output of the new operator is stored in a variable:

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

Constructing Objects

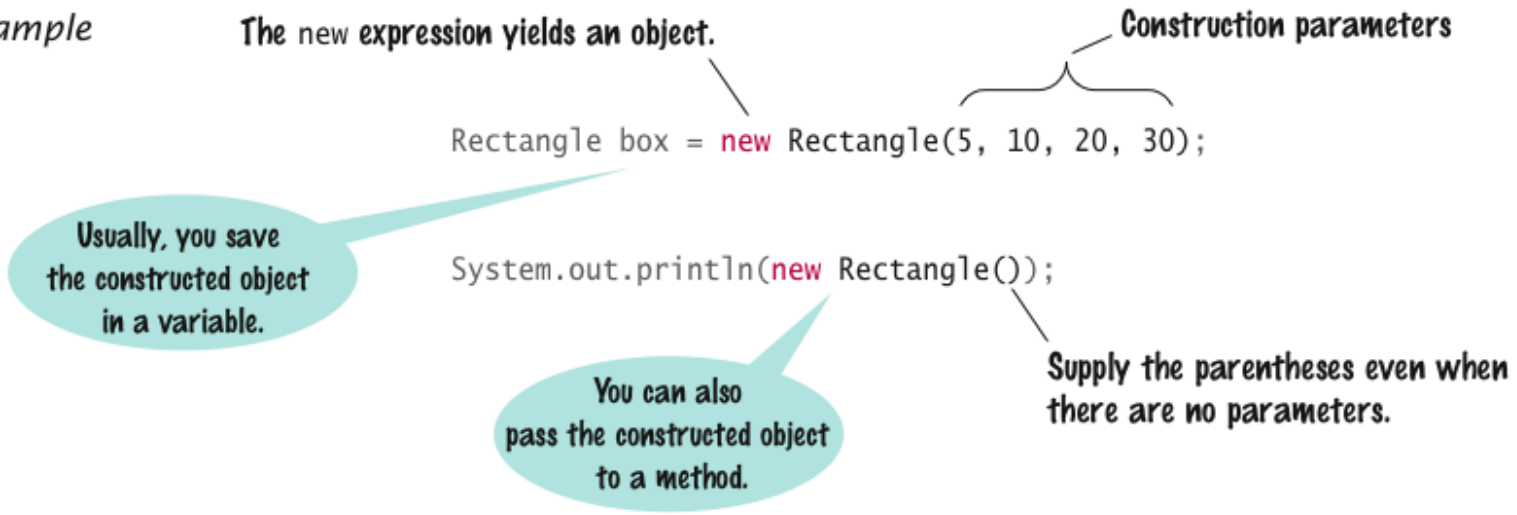
- **Construction:** the process of creating a new object
- The four values `5`, `10`, `20`, and `30` are called the *construction parameters*
- Some classes let you construct objects in multiple ways:

```
new Rectangle()
// constructs a rectangle with its top-left corner
// at the origin (0, 0), width 0, and height 0
```

Syntax 2.3 Object Construction

Syntax `new ClassName(parameters)`

Example



Self Check 2.16

How do you construct a square with center (100, 100) and side length 20?

Answer:

```
new Rectangle(90, 90, 20, 20)
```

Self Check 2.17

The `getWidth` method returns the width of a `Rectangle` object. What does the following statement print?

```
System.out.println(new Rectangle().getWidth());
```

Answer:

```
0
```

Accessor and Mutator Methods

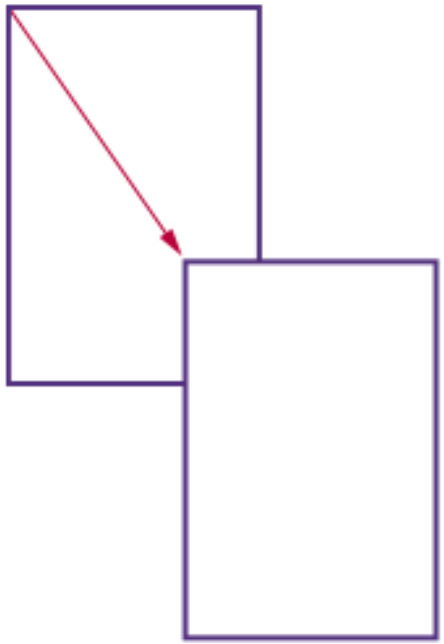
- **Accessor method**: does not change the state of its implicit parameter:

```
double width = box.getWidth();
```

- **Mutator method**: changes the state of its implicit parameter:

```
box.translate(15, 25);
```

Figure 12
Using the `translate` Method to Move a Rectangle



Self Check 2.18

Is the `toUpperCase` method of the `String` class an accessor or a mutator?

Answer: An accessor – it doesn't modify the original string but returns a new string with uppercase letters.

Self Check 2.19

Which call to `translate` is needed to move the `box` rectangle so that its top-left corner is the origin (0, 0)?

Answer: `box.translate(-5, -10)`, provided the method is called immediately after storing the new rectangle into `box`.

The API Documentation

- **API:** Application Programming Interface
- **API documentation:** lists classes and methods in the Java library
- http://java.sun.com/javase/7/docs/api/index.html

The API Documentation of the Standard Java Library

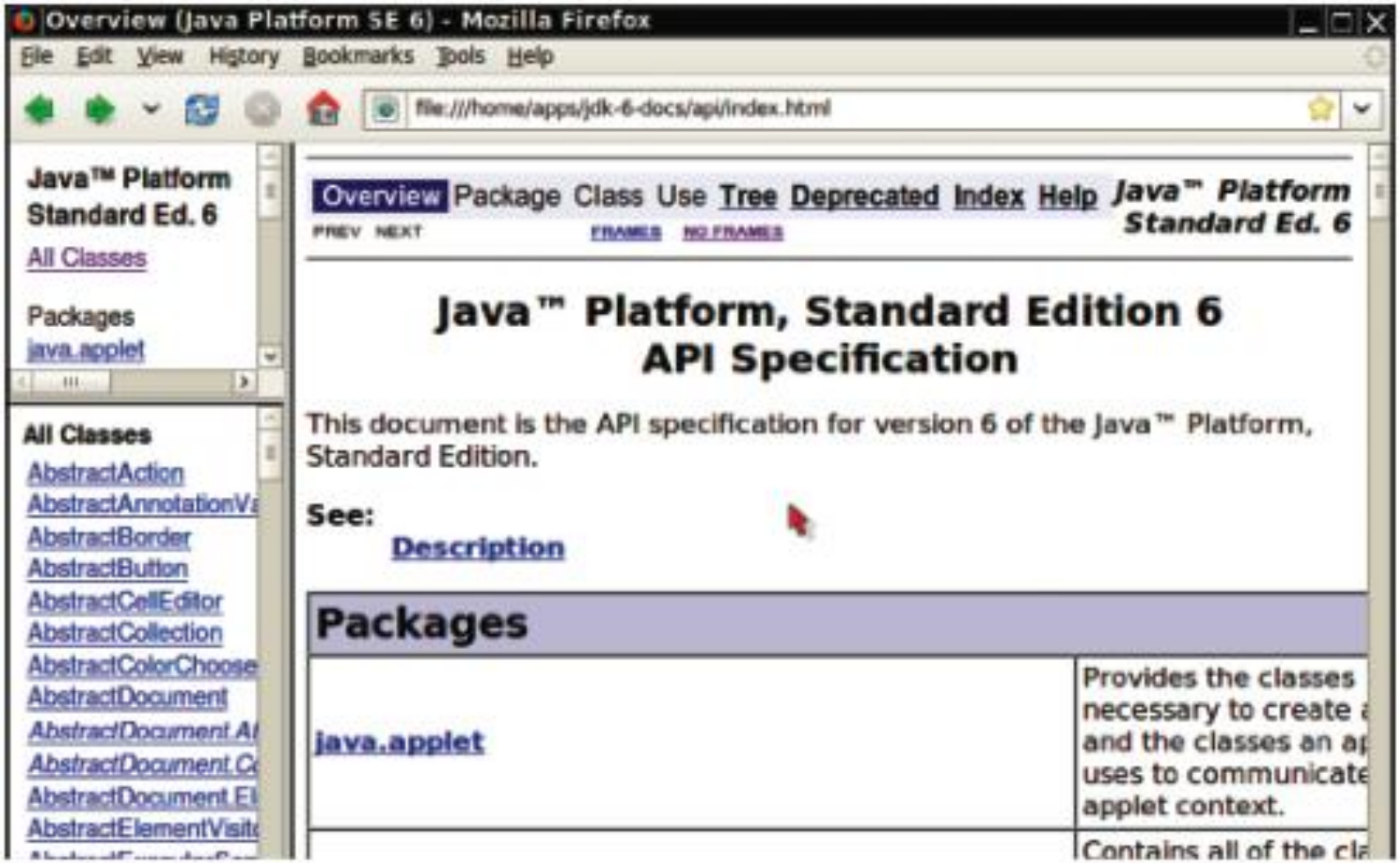


Figure 13 The API Documentation of the Standard Java Library

The API Documentation for the Rectangle Class

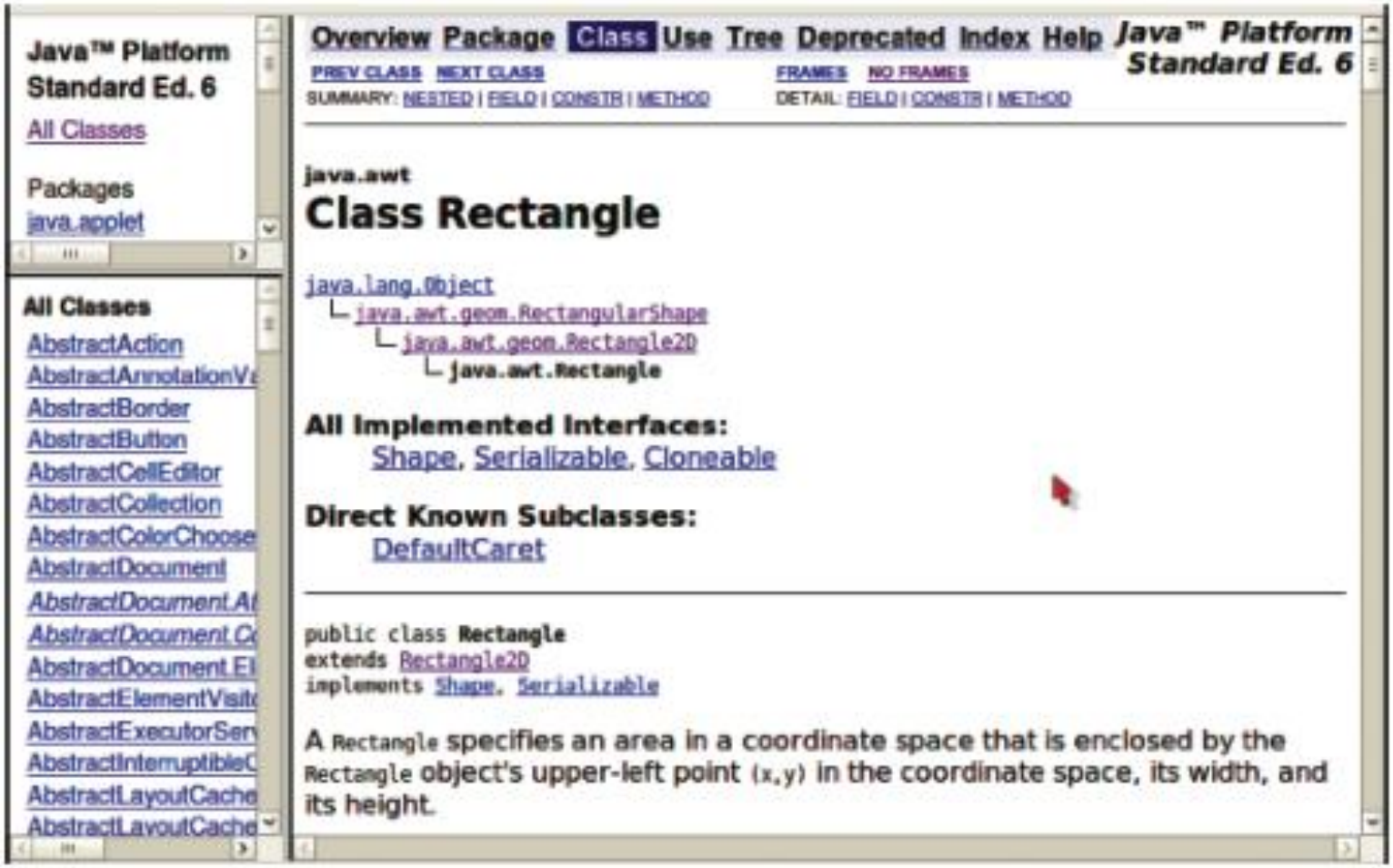


Figure 14 The API Documentation for the Rectangle Class

Method Summary

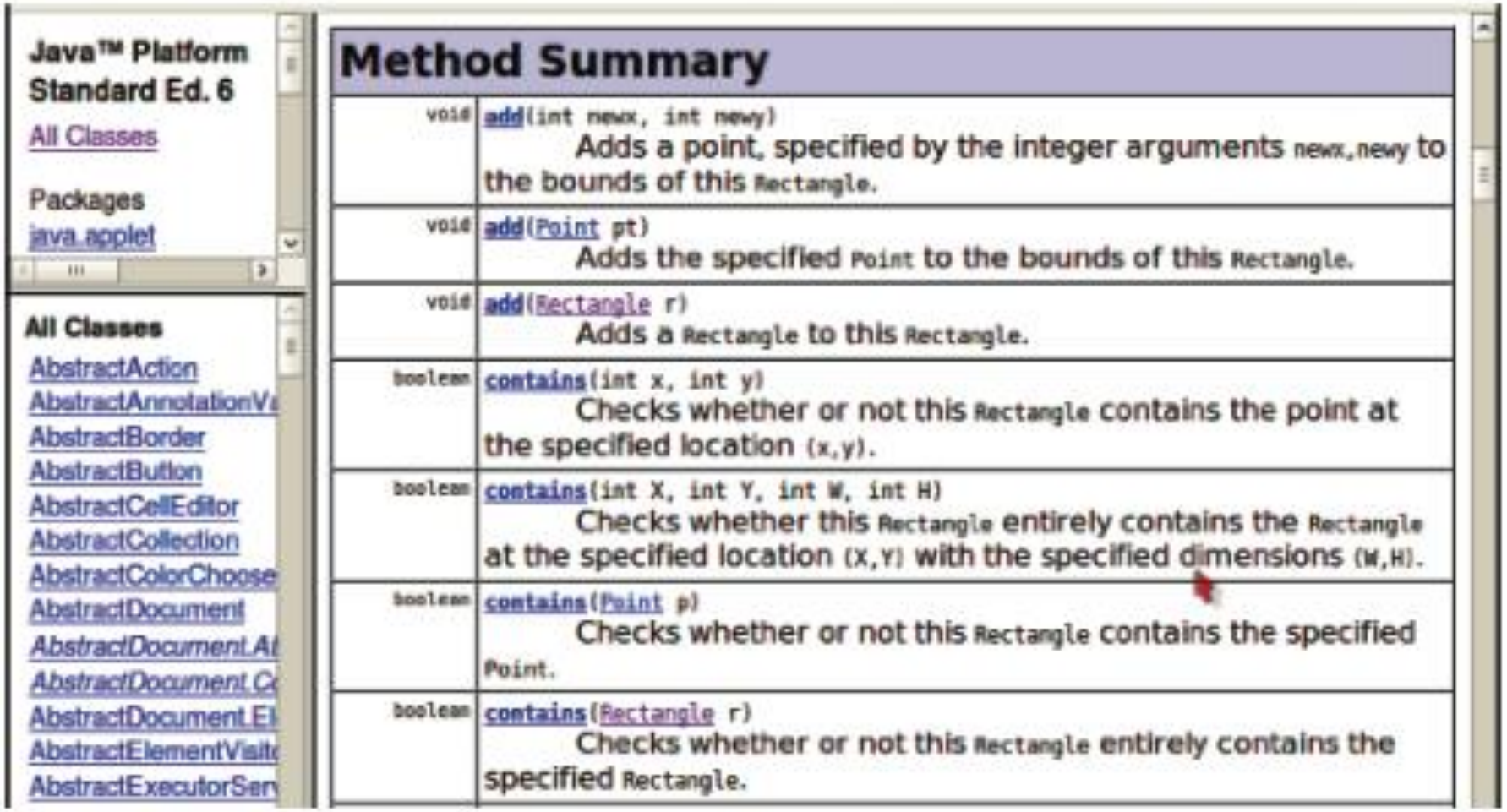


Figure 15 The Method Summary for the Rectangle Class

Detailed Method Description

The detailed description of a method shows:

- The action that the method carries out
- The parameters that the method receives
- The value that it returns (or the reserved word void if the method doesn't return any value)

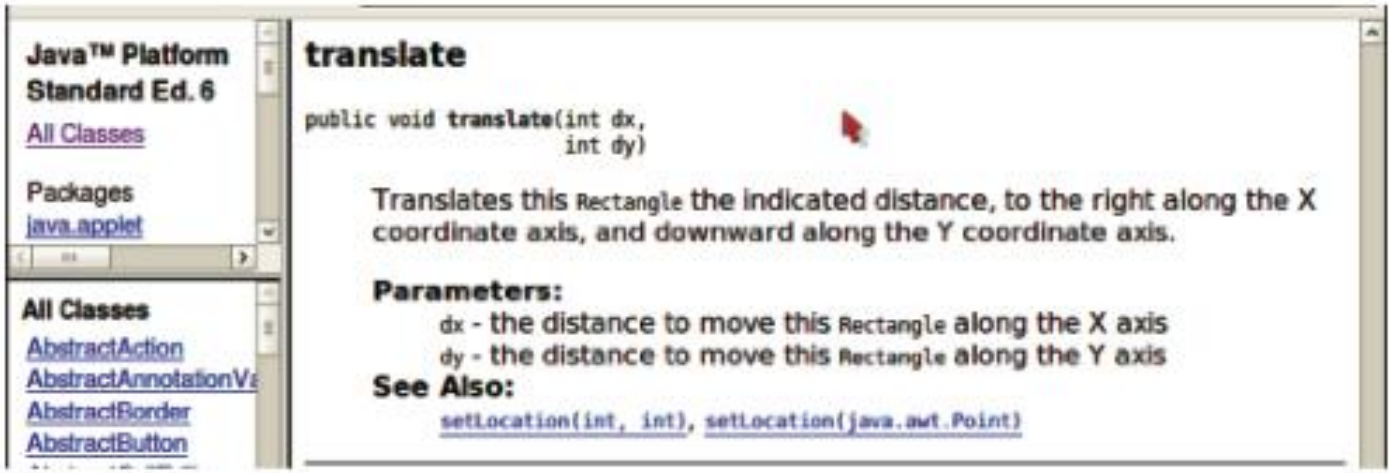


Figure 16 The API Documentation of the `translate` Method

Packages

- **Package**: a collection of classes with a related purpose
- Import library classes by specifying the package and class name:

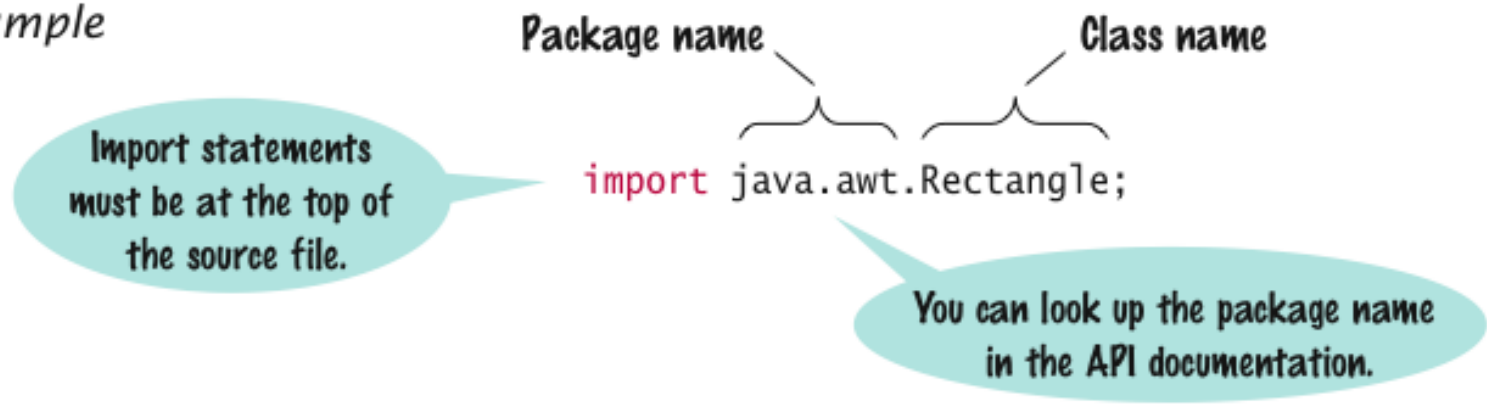
```
import java.awt.Rectangle;
```

- You don't need to import classes in the `java.lang` package such as `String` and `System`

Syntax 2.4 Importing a Class from a Package

Syntax `import` *packageName.ClassName*;

Example



Self Check 2.20

Look at the API documentation of the `String` class. Which method would you use to obtain the string `"hello, world!"` from the string `"Hello, World!"`?

Answer: `toLowerCase`

Self Check 2.21

In the API documentation of the `String` class, look at the description of the `trim` method. What is the result of applying trim to the string `" Hello, Space ! "`? (Note the spaces in the string.)

Answer: `"Hello, Space !"` – only the leading and trailing spaces are trimmed.

Self Check 2.22

The `Random` class is defined in the `java.util` package. What do you need to do in order to use that class in your program?

Answer: Add the statement

```
import java.util.Random;
```

at the top of your program.

Implementing a Test Program

1. Provide a tester class.
2. Supply a `main` method.
3. Inside the `main` method, construct one or more objects.
4. Apply methods to the objects.
5. Display the results of the method calls.
6. Display the values that you expect to get.

ch02/rectangle/MoveTester.java

```
1  import java.awt.Rectangle;
2
3  public class MoveTester
4  {
5     public static void main(String[] args)
6     {
7        Rectangle box = new Rectangle(5, 10, 20, 30);
8
9        // Move the rectangle
10       box.translate(15, 25);
11
12       // Print information about the moved rectangle
13       System.out.print("x: ");
14       System.out.println(box.getX());
15       System.out.println("Expected: 20");
16
17       System.out.print("y: ");
18       System.out.println(box.getY());
19       System.out.println("Expected: 35");
20    }
21 }
```

ch02/rectangle/MoveTester.java (cont.)

Program Run:

```
x: 20
Expected: 20
y: 35
Expected: 35
```

Self Check 2.23

Suppose we had called `box.translate(25, 15)` instead of `box.translate(15, 25)`. What are the expected outputs?

Answer:

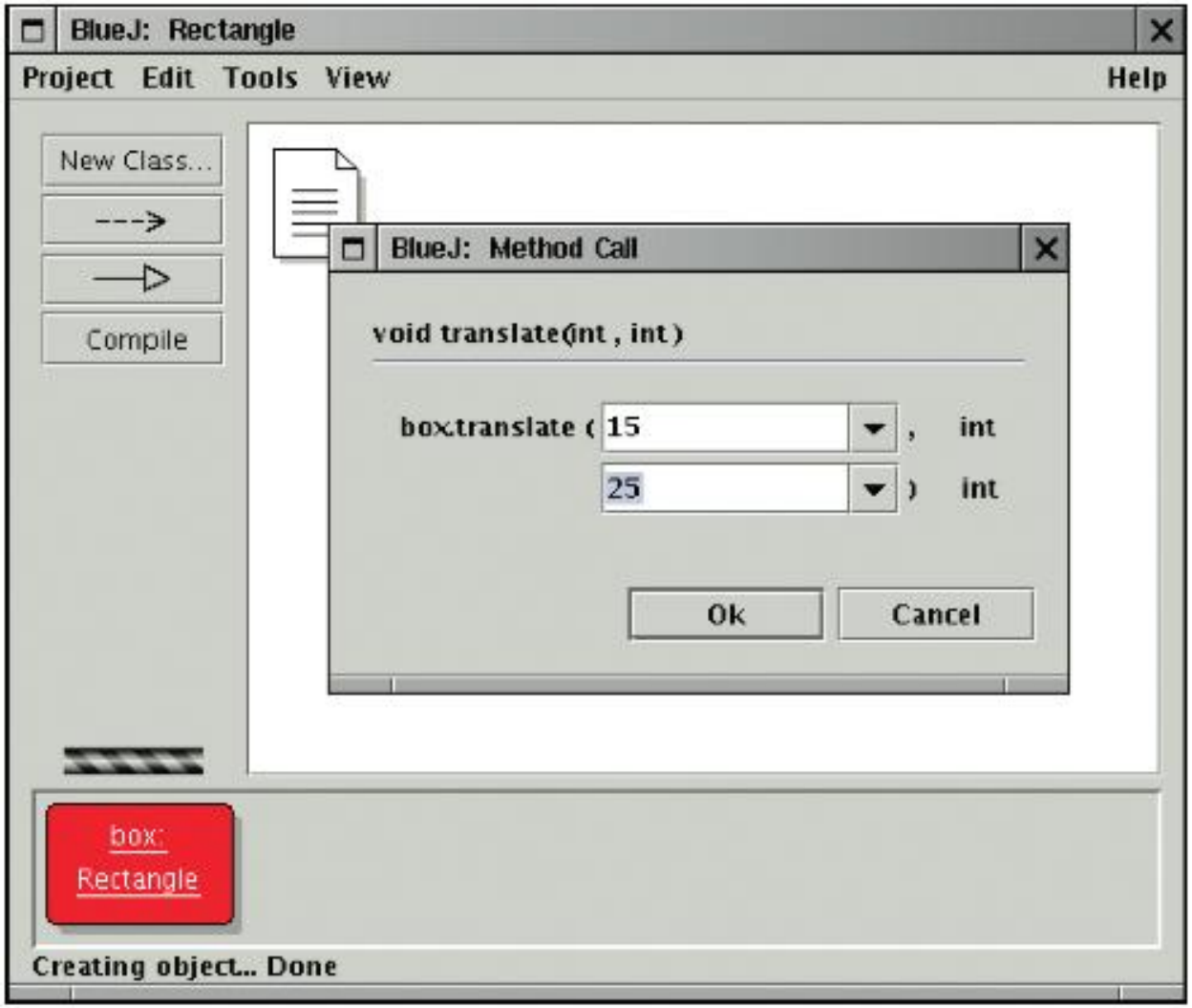
```
x: 30, y: 25
```

Self Check 2.24

Why doesn't the `MoveTester` program print the width and height of the rectangle?

Answer: Because the `translate` method doesn't modify the shape of the rectangle.

Testing Classes in an Interactive Environment



Testing a Method Call in BlueJ

Object References

- **Object reference:** describes the location of an object
- The `new` operator returns a reference to a new object:

```
Rectangle box = new Rectangle();
```

- Multiple object variables can refer to the same object:

```
Rectangle box = new Rectangle(5, 10, 20, 30);
Rectangle box2 = box;
box2.translate(15, 25);
```

- Primitive type variables ≠ object variables

Object Variables and Number Variables

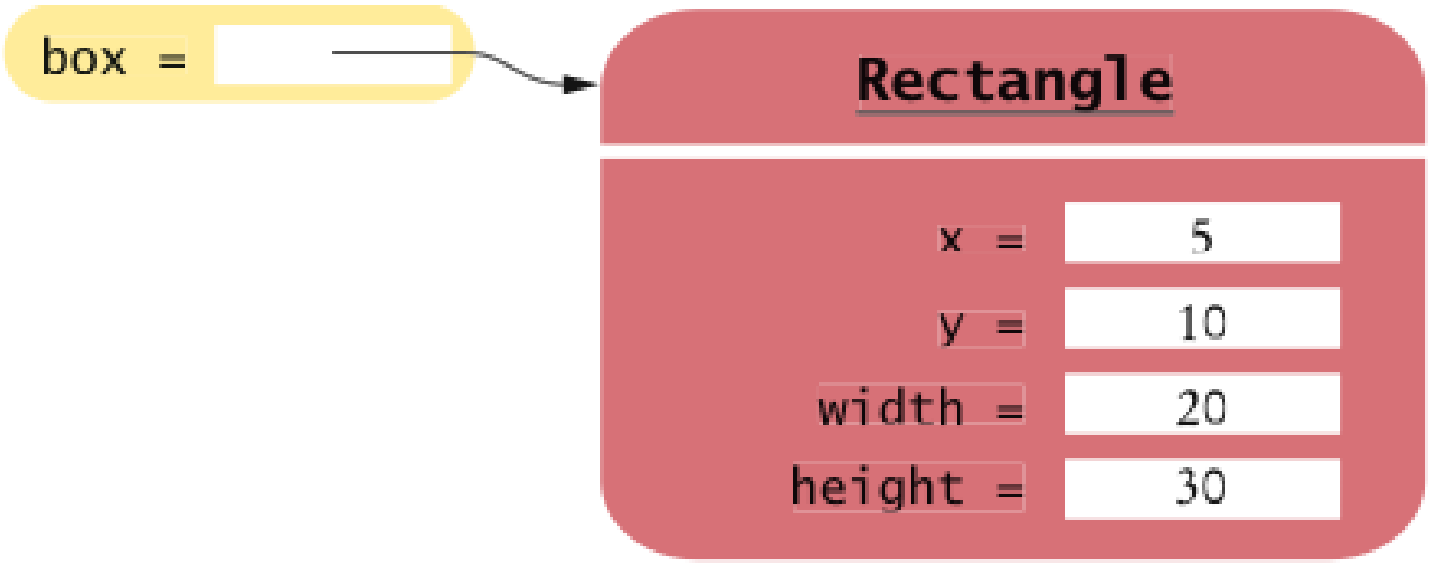


Figure 17 An Object Variable Containing an Object Reference

Object Variables and Number Variables

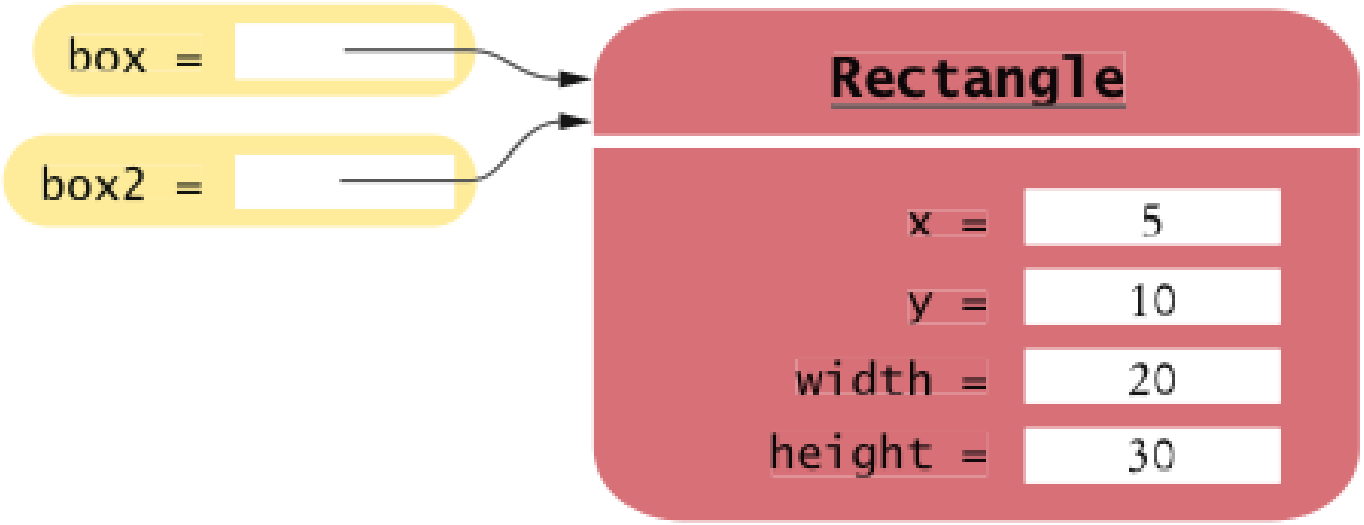


Figure 18 Two Object Variables Referring to the Same Object

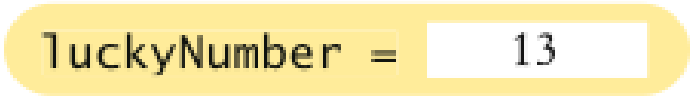
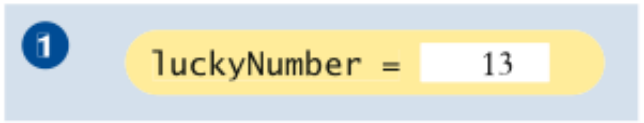


Figure 19 A Number Variable Stores a Number

Copying Numbers

```
int luckyNumber = 13;
```

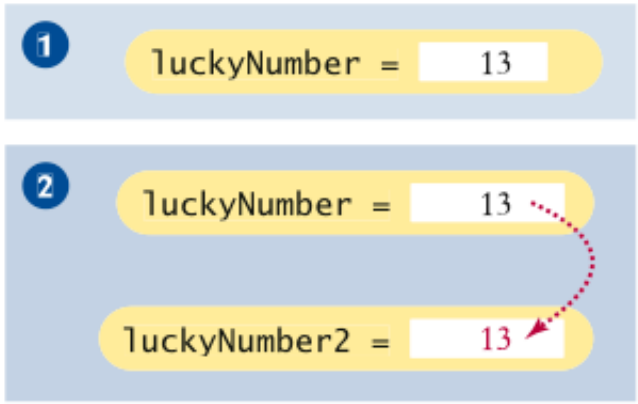
Figure 20
Copying Numbers



Copying Numbers (cont.)

```
int luckyNumber = 13;
int luckyNumber2 = luckyNumber;
```

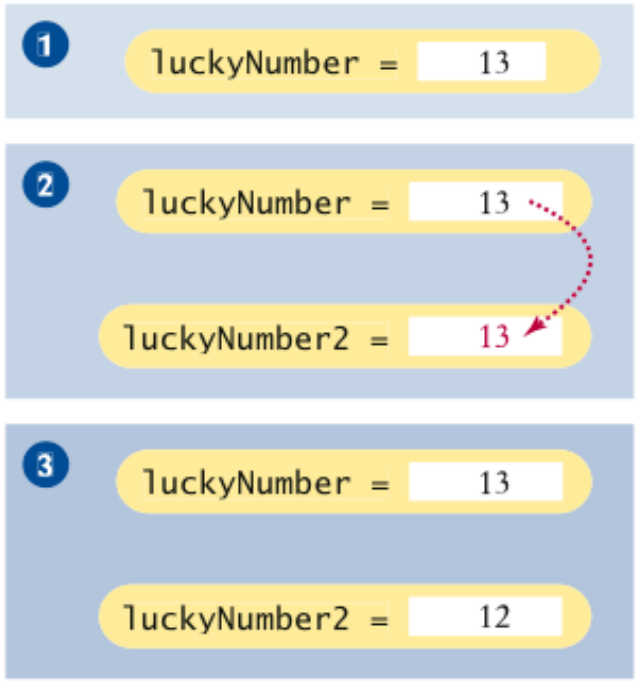
Figure 20
Copying Numbers



Copying Numbers (cont.)

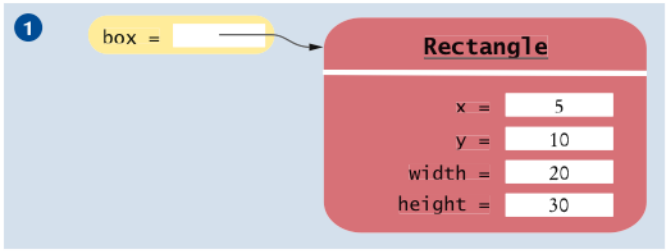
```
int luckyNumber = 13; 1
int luckyNumber2 = luckyNumber; 2
luckyNumber2 = 12; 3
```

Figure 20
Copying Numbers



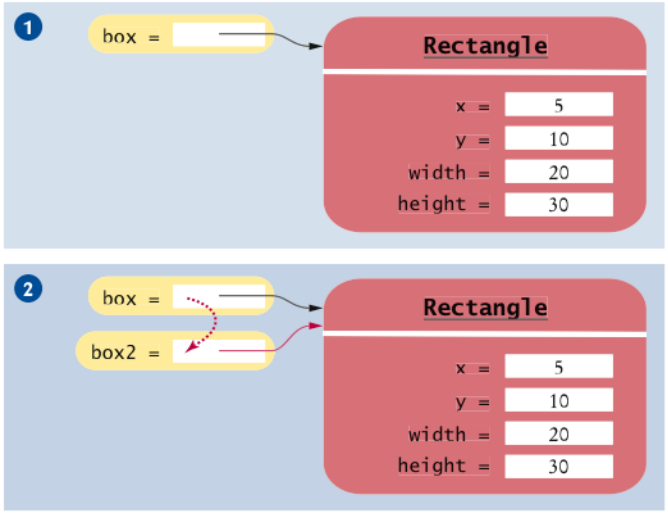
Copying Object References

```
Rectangle box = new Rectangle(5, 10, 20, 30);
```



Copying Object References (cont.)

```
Rectangle box = new Rectangle(5, 10, 20, 30); 1
Rectangle box2 = box; 2
```



Copying Object References (cont.)

```
Rectangle box = new Rectangle(5, 10, 20, 30); 1
Rectangle box2 = box; 2
Box2.translate(15, 25); 3
```

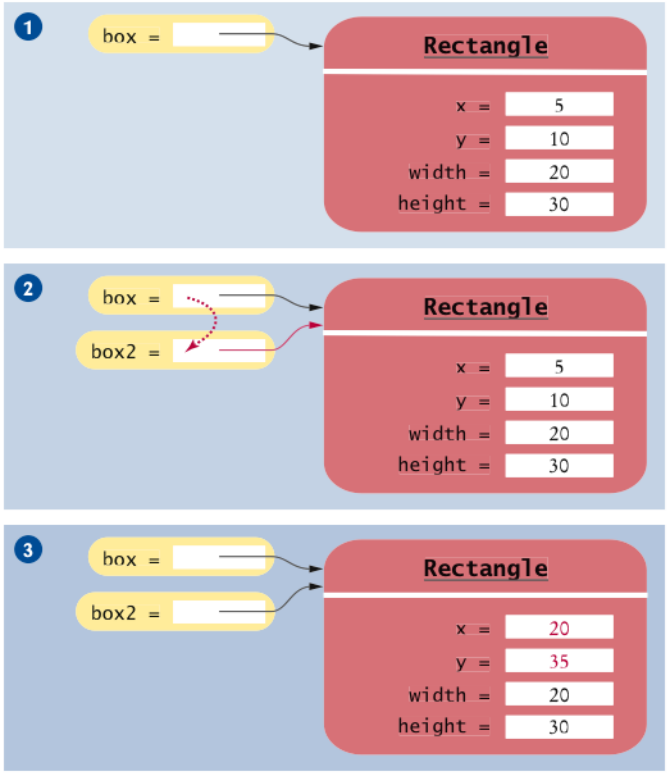


Figure 21 Copying Object References

Self Check 2.25

What is the effect of the assignment `greeting2 = greeting`?

Answer: Now `greeting` and `greeting2` both refer to the same `String` object.

Self Check 2.26

After calling `greeting2.toUpperCase()`, what are the contents of `greeting` and `greeting2`?

Answer: Both variables still refer to the same string, and the string has not been modified. Recall that the `toUpperCase` method constructs a new string that contains uppercase characters, leaving the original string unchanged.

Mainframes – When Dinosaurs Ruled the Earth

A Mainframe Computer

Graphical Applications and Frame Windows

To show a frame:

1. Construct an object of the `JFrame` class:

```
JFrame frame = new JFrame();
```

2. Set the size of the frame:

```
frame.setSize(300, 400);
```

3. If you'd like, set the title of the frame:

```
frame.setTitle("An Empty Frame");
```

4. Set the "default close operation":

```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

5. Make the frame visible:

```
frame.setVisible(true);
```

A Frame Window

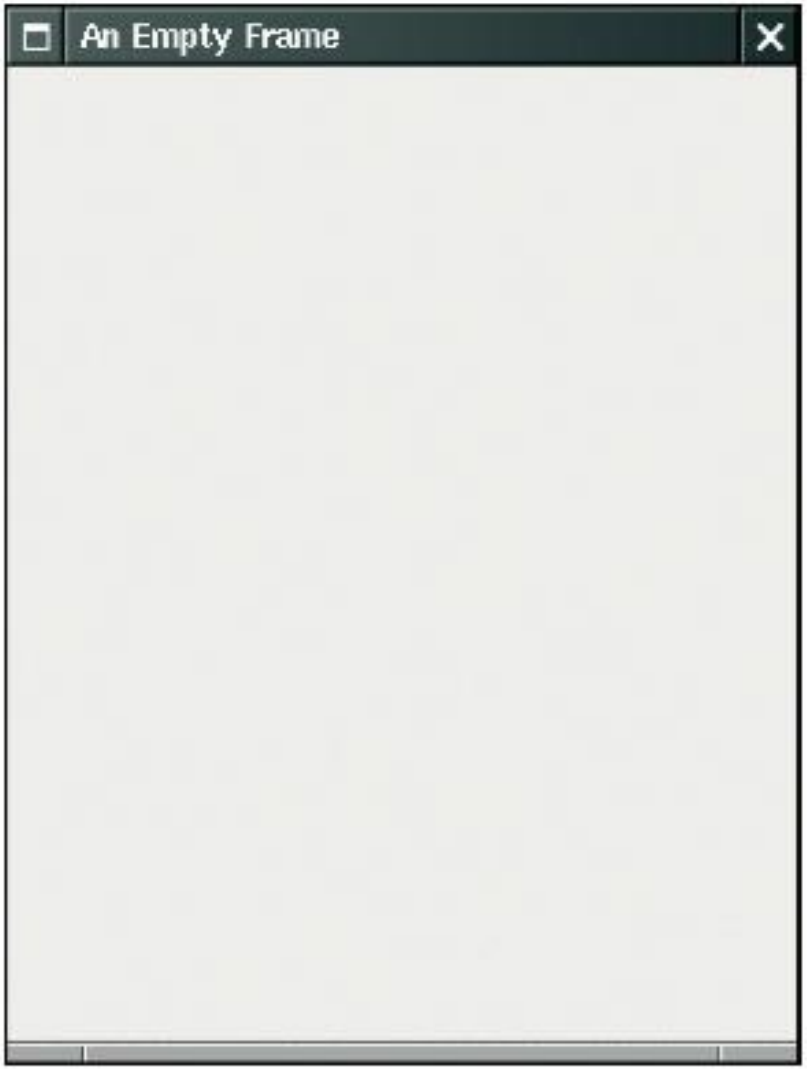


Figure 22
A Frame Window

ch02/emptyframe/EmptyFrameViewer.java

```
 1  import javax.swing.JFrame;
 2
 3  public class EmptyFrameViewer
 4  {
 5     public static void main(String[] args)
 6     {
 7        JFrame frame = new JFrame();
 8
 9        frame.setSize(300, 400);
10        frame.setTitle("An Empty Frame");
11        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13        frame.setVisible(true);
14     }
15  }
```

Self Check 2.27

How do you display a square frame with a title bar that reads `"Hello, World!”`?

Answer: Modify the `EmptyFrameViewer` program as follows:

```
frame.setSize(300, 300);
frame.setTitle("Hello, World!");
```

Self Check 2.28

How can a program display two frames at once?

Answer: Construct two `JFrame` objects, set each of their sizes, and call `setVisible(true)` on each of them.

Drawing on a Component

- In order to display a drawing in a frame, define a class that extends the `JComponent` class
- Place drawing instructions inside the `paintComponent` method. That method is called whenever the component needs to be repainted:

```
public class RectangleComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      Drawing instructions go here
   }
}
```

Classes `Graphics` and `Graphics2D`

- `Graphics` class lets you manipulate the graphics state (such as current color)
- `Graphics2D` class has methods to draw shape objects
- Use a cast to recover the `Graphics2D` object from the `Graphics` parameter:

```
public class RectangleComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;
      . . .
   }
}
```

Classes `Graphics` and `Graphics2D`

- Call method `draw` of the `Graphics2D` class to draw shapes, such as rectangles, ellipses, line segments, polygons, and arcs:

```
public class RectangleComponent extends JComponent
{
   public void paintComponent(Graphics g)
   {
      . . .
      Rectangle box = new Rectangle(5, 10, 20, 30);
      g2.draw(box);
      . . .
   }
}
```

Drawing Rectangles

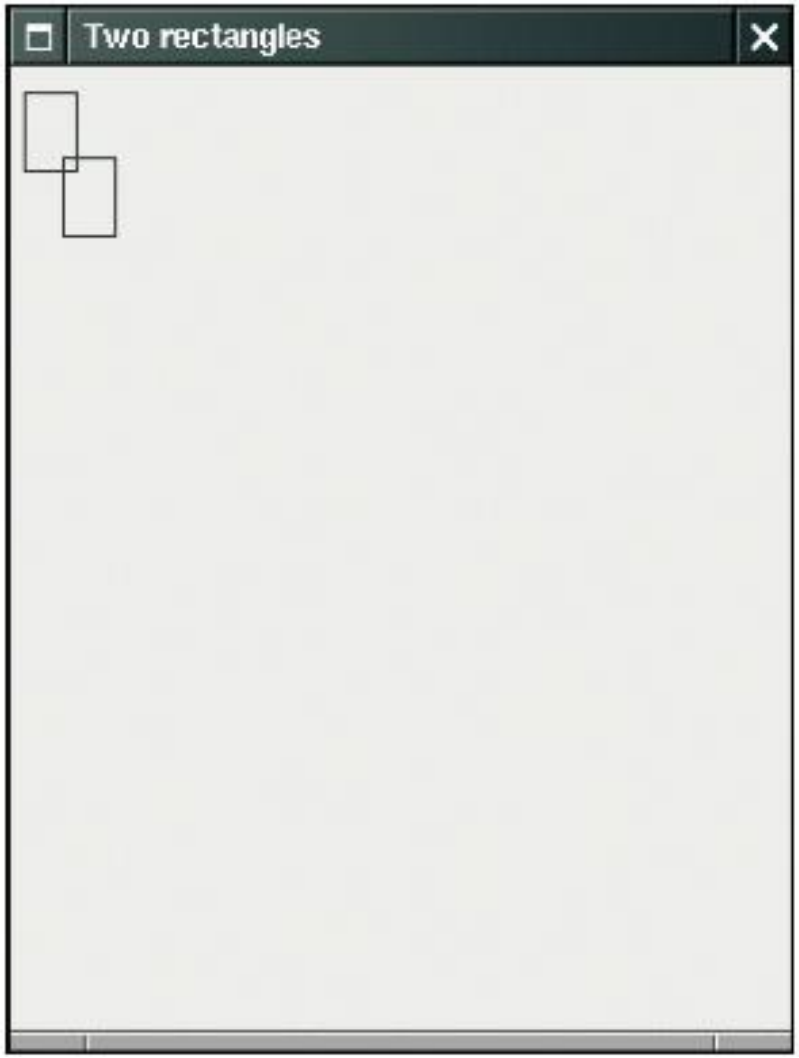


Figure 23
Drawing Rectangles

ch02/rectangles/RectangleComponent.java

```
 1  import java.awt.Graphics;
 2  import java.awt.Graphics2D;
 3  import java.awt.Rectangle;
 4  import javax.swing.JComponent;
 5
 6  /**
 7     A component that draws two rectangles.
 8  */
 9  public class RectangleComponent extends JComponent
10  {
11     public void paintComponent(Graphics g)
12     {
13        // Recover Graphics2D
14        Graphics2D g2 = (Graphics2D) g;
15
16        // Construct a rectangle and draw it
17        Rectangle box = new Rectangle(5, 10, 20, 30);
18        g2.draw(box);
19
```

Continued

ch02/rectangles/RectangleComponent.java (cont.)

```
20          // Move rectangle 15 units to the right and 25 units down
21          box.translate(15, 25);
22
23          // Draw moved rectangle
24          g2.draw(box);
25       }
26    }
```

Using a Component

1. Construct a frame.
2. Construct an object of your component class:
   ```
   RectangleComponent component = new RectangleComponent();
   ```
3. Add the component to the frame:
   ```
   frame.add(component);
   ```
4. Make the frame visible.

ch02/rectangles/RectangleViewer.java

```
1  import javax.swing.JFrame;
2
3  public class RectangleViewer
4  {
5     public static void main(String[] args)
6     {
7        JFrame frame = new JFrame();
8
9        frame.setSize(300, 400);
10       frame.setTitle("Two rectangles");
11       frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13       RectangleComponent component = new RectangleComponent();
14       frame.add(component);
15
16       frame.setVisible(true);
17    }
18 }
```

Self Check 2.29

How do you modify the program to draw two squares?

Answer:

```
Rectangle box = new Rectangle(5, 10, 20, 20);
```

Self Check 2.30

How do you modify the program to draw one rectangle and one square?

Answer: Replace the call to `box.translate(15, 25)` with

```
box = new Rectangle(20, 35, 20, 20);
```

Self Check 2.31

What happens if you call `g.draw(box)` instead of `g2.draw(box)`?

Answer: The compiler complains that `g` doesn't have a `draw` method.

Applets

- **Applet:** program that runs inside a web browser
- To implement an applet, use this code outline:

```
public class MyApplet extends JApplet
{
   public void paint(Graphics g)
   {
      // Recover Graphics2D
      Graphics2D g2 = (Graphics2D) g;
      // Drawing instructions go here
      . . .
   }
}
```

Applets

- This is almost the same outline as for a component, with two minor differences:
  1. *You extend* `JApplet`*, not* `JComponent`
  2. *You place the drawing code inside the* `paint` *method, not inside* `paintComponent`
- To run an applet, you need an HTML file with the `applet` tag
- An HTML file can have multiple applets; add a separate `applet` tag for each applet
- You view applets with the applet viewer or a Java enabled browser:

```
appletviewer RectangleApplet.html
```

ch02/applet/RectangleApplet.java

```
 1  import java.awt.Graphics;
 2  import java.awt.Graphics2D;
 3  import java.awt.Rectangle;
 4  import javax.swing.JApplet;
 5
 6  /**
 7     An applet that draws two rectangles.
 8  */
 9  public class RectangleApplet extends JApplet
10  {
11     public void paint(Graphics g)
12     {
13        // Prepare for extended graphics
14        Graphics2D g2 = (Graphics2D) g;
15
16        // Construct a rectangle and draw it
17        Rectangle box = new Rectangle(5, 10, 20, 30);
18        g2.draw(box);
19
```

Continued

ch02/applet/RectangleApplet.java (cont.)

```
20          // Move rectangle 15 units to the right and 25 units down
21          box.translate(15, 25);
22
23          // Draw moved rectangle
24          g2.draw(box);
25       }
26    }
27
```

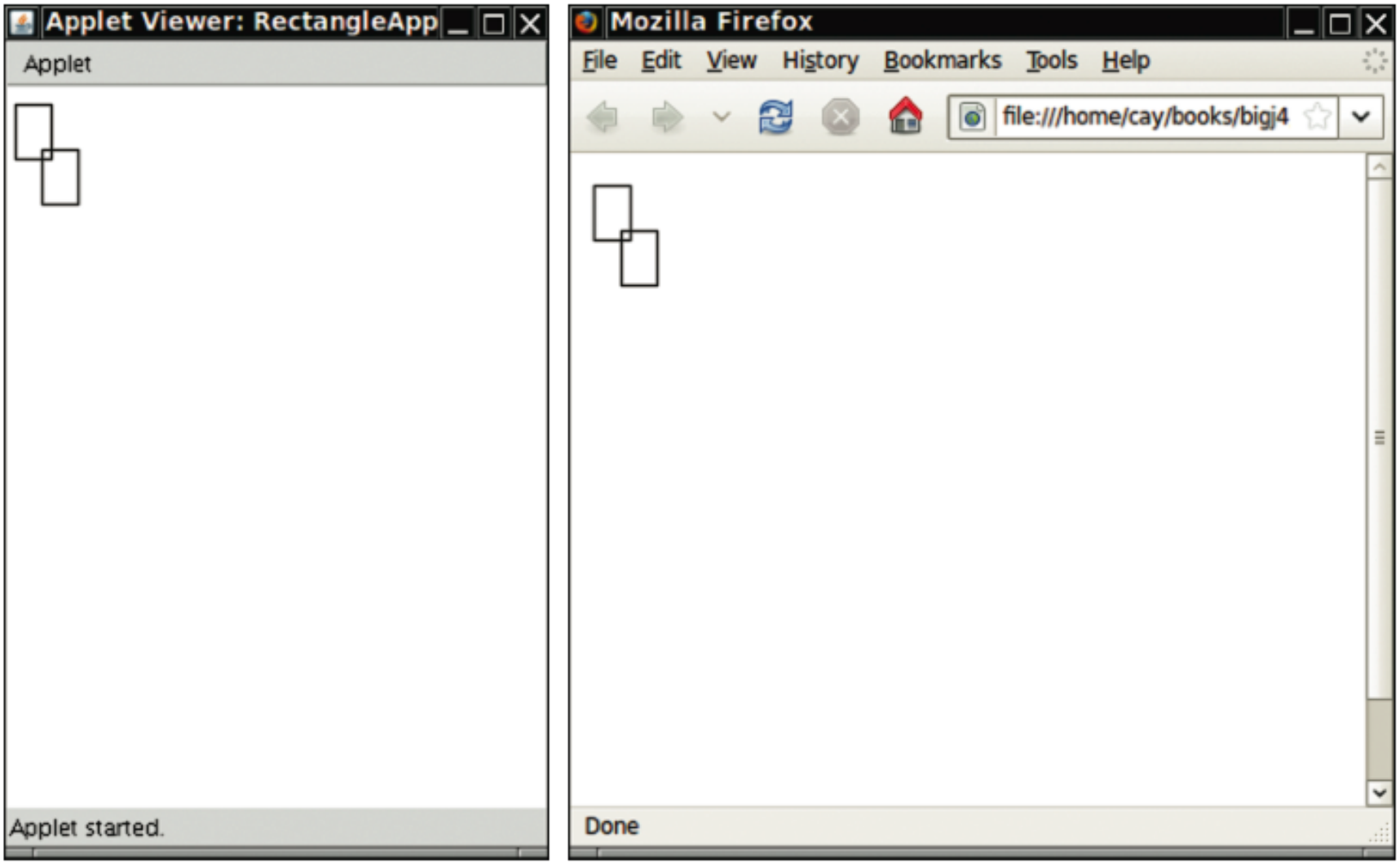
ch02/applet/RectangleApplet.html

```
1 <applet code="RectangleApplet.class" width="300" height="400">
2 </applet>
```

ch02/applet/RectangleAppletExplained.html

```
1  <html>
2     <head>
3        <title>Two rectangles</title>
4     </head>
5     <body>
6        <p>Here is my <i>first applet</i>:</p>
7        <applet code="RectangleApplet.class" width="300" height="400">
8        </applet>
9     </body>
10 </html>
```

Applets



An Applet in the Applet Viewer

An Applet in a Web Browser

Ellipses

- `Ellipse2D.Double` describes an ellipse
- This class is an inner class – doesn't matter to us except for the `import` statement:

```
import java.awt.geom.Ellipse2D; // no .Double
```

- Must construct *and draw* the shape:

```
Ellipse2D.Double ellipse =
   new Ellipse2D.Double(x, y, width, height);
g2.draw(ellipse);
```

An Ellipse

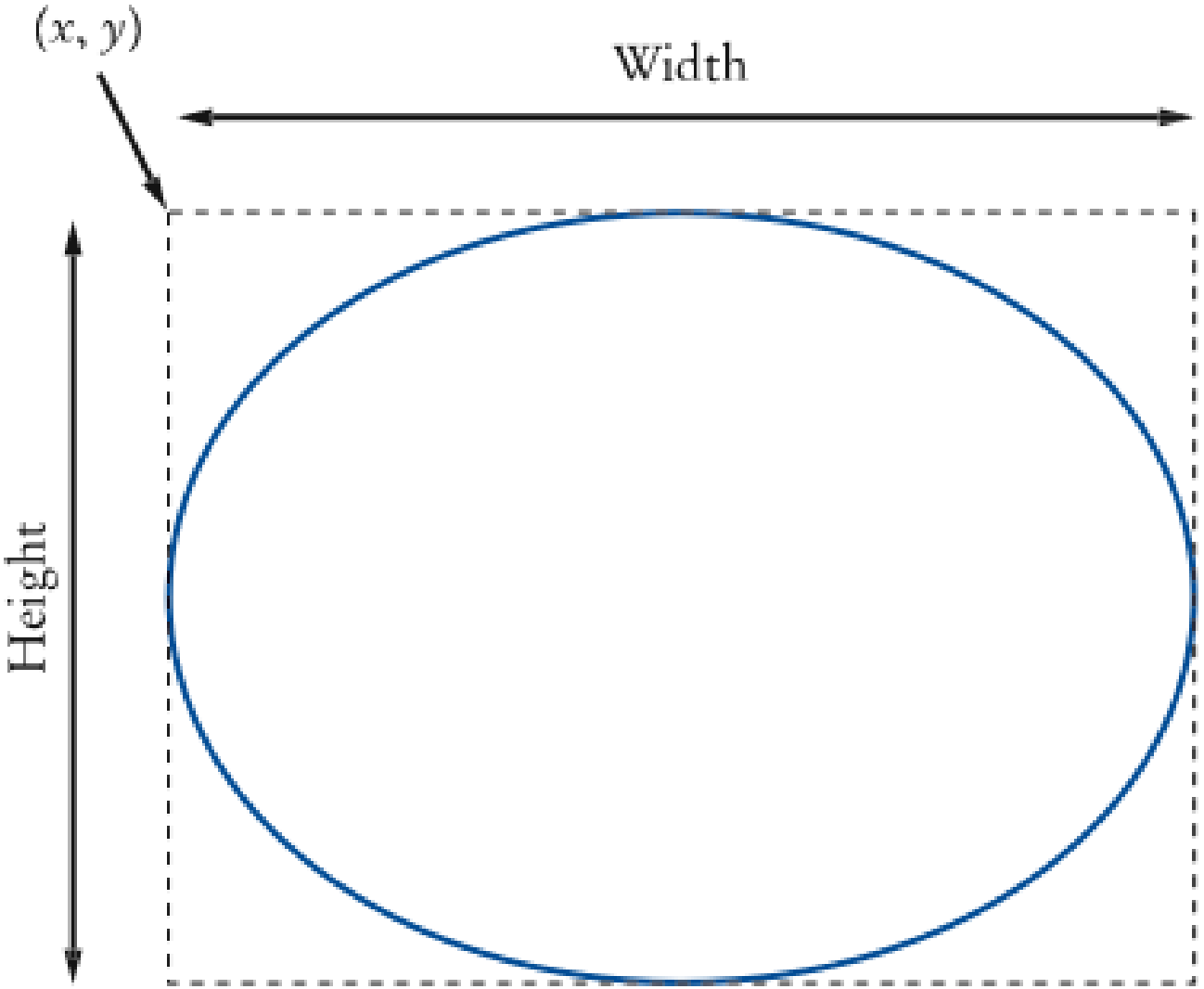


Figure 24 An Ellipse and Its Bounding Box

Drawing Lines

- To draw a line:

```
Line2D.Double segment =
   new Line2D.Double(x1, y1, x2, y2);
g2.draw(segment);
```

or,

```
Point2D.Double from = new Point2D.Double(x1, y1);
Point2D.Double to = new Point2D.Double(x2, y2);
Line2D.Double segment = new Line2D.Double(from, to);
g2.draw(segment);
```

Drawing Text

```
g2.drawString("Message", 50, 100);
```

Figure 25 Basepoint and Baseline

Colors

- Standard colors `Color.BLUE`, `Color.RED`, `Color.PINK`, etc.
- Specify red, green, blue between `0` and `255`:

```
Color magenta = new Color(255, 0, 255);
```

- Set color in graphics context:

```
g2.setColor(magenta);
```

- Color is used when drawing and filling shapes:

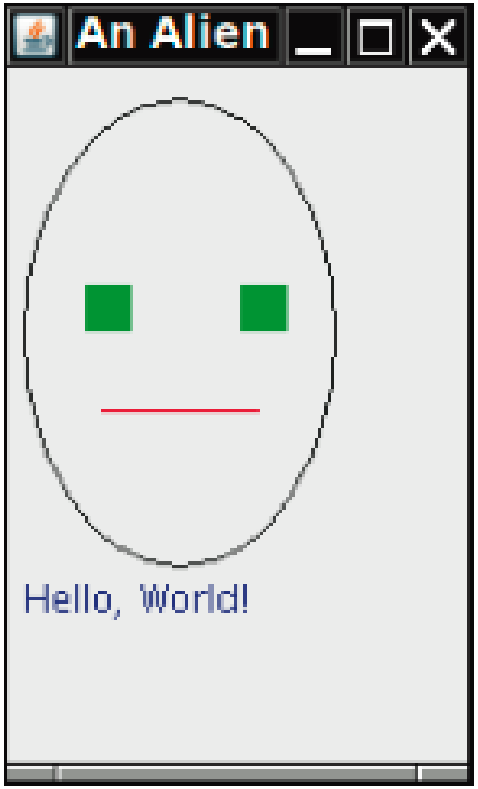
```
g2.fill(rectangle); // filled with current color
```

Predefined Colors and Their RGB Values

Color	RGB Value	
Color.BLACK	0, 0, 0	
Color.BLUE	0, 0, 255	
Color.CYAN	0, 255, 255	
Color.GRAY	128, 128, 128	
Color.DARKGRAY	64, 64, 64	
Color.LIGHTGRAY	192, 192, 192	
Color.GREEN	0, 255, 0	
Color.MAGENTA	255, 0, 255	
Color.ORANGE	255, 200, 0	
Color.PINK	255, 175, 175	
Color.RED	255, 0, 0	
Color.WHITE	255, 255, 255	
Color.YELLOW	255, 255, 0	

Alien Face

Figure 26
An Alien Face



ch02/face/FaceComponent.java

```
1  import java.awt.Color;
2  import java.awt.Graphics;
3  import java.awt.Graphics2D;
4  import java.awt.Rectangle;
5  import java.awt.geom.Ellipse2D;
6  import java.awt.geom.Line2D;
7  import javax.swing.JComponent;
8
9  /**
10    A component that draws an alien face
11 */
12 public class FaceComponent extends JComponent
13 {
14    public void paintComponent(Graphics g)
15    {
16       // Recover Graphics2D
17       Graphics2D g2 = (Graphics2D) g;
18
```

Continued

ch02/face/FaceComponent.java (cont.)

```
19          // Draw the head
20          Ellipse2D.Double head = new Ellipse2D.Double(5, 10, 100, 150);
21          g2.draw(head);
22
23          // Draw the eyes
24          g2.setColor(Color.GREEN);
25          Rectangle eye = new Rectangle(25, 70, 15, 15);
26          g2.fill(eye);
27          eye.translate(50, 0);
28          g2.fill(eye);
29
30          // Draw the mouth
31          Line2D.Double mouth = new Line2D.Double(30, 110, 80, 110);
32          g2.setColor(Color.RED);
33          g2.draw(mouth);
34
35          // Draw the greeting
36          g2.setColor(Color.BLUE);
37          g2.drawString("Hello, World!", 5, 175);
38       }
39    }
```

ch02/face/FaceViewer.java

```
 1  import javax.swing.JFrame;
 2
 3  public class FaceViewer
 4  {
 5     public static void main(String[] args)
 6     {
 7        JFrame frame = new JFrame();
 8        frame.setSize(150, 250);
 9        frame.setTitle("An Alien Face");
10        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
11
12        FaceComponent component = new FaceComponent();
13        frame.add(component);
14
15        frame.setVisible(true);
16     }
17  }
```

Self Check 2.32

Give instructions to draw a circle with center (100, 100) and radius 25.

Answer:

```
g2.draw(new Ellipse2D.Double(75, 75, 50, 50));
```

Self Check 2.33

Give instructions to draw a letter "V" by drawing two line segments.

Answer:

```
Line2D.Double segment1 = new Line2D.Double(0, 0, 10, 30);
g2.draw(segment1);
Line2D.Double segment2 = new Line2D.Double(10, 30, 20, 0);
g2.draw(segment2);
```

Self Check 2.34

Give instructions to draw a string consisting of the letter "V”.

Answer:

```
g2.drawString("V", 0, 30);
```

Self Check 2.35

What are the RGB color values of `Color.BLUE`?

Answer: `0, 0,` and `255`

Self Check 2.36

How do you draw a yellow square on a red background?

Answer: First fill a big red square, then fill a small yellow square inside:

```
g2.setColor(Color.RED);
g2.fill(new Rectangle(0, 0, 200, 200));
g2.setColor(Color.YELLOW);
g2.fill(new Rectangle(50, 50, 100, 100));
```

Chapter Goals

- To become familiar with the process of implementing classes
- To be able to implement simple methods
- To understand the purpose and use of constructors
- To understand how to access instance variables and local variables
- To be able to write javadoc comments

G To implement classes for drawing graphical shapes

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Instance Variables

- **Example:** tally counter
- Simulator statements:

```
Counter tally = new Counter();
tally.count();
tally.count();
int result = tally.getValue(); // Sets result to 2
```

Figure 1 A Tally Counter

- Each counter needs to store a variable that keeps track of how many times the counter has been advanced

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Instance Variables

- **Instance variables** store the data of an object
- **Instance of a class:** an object of the class
- The class declaration specifies the instance variables:

```
public class Counter
{
   private int value;
   …
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Instance Variables

- An instance variable declaration consists of the following parts:
  - *access specifier (*`private`*)*
  - *type of variable (such as* `int`*)*
  - *name of variable (such as* `value`*)*
- Each object of a class has its own set of instance variables
- You should declare all instance variables as private

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Instance Variables

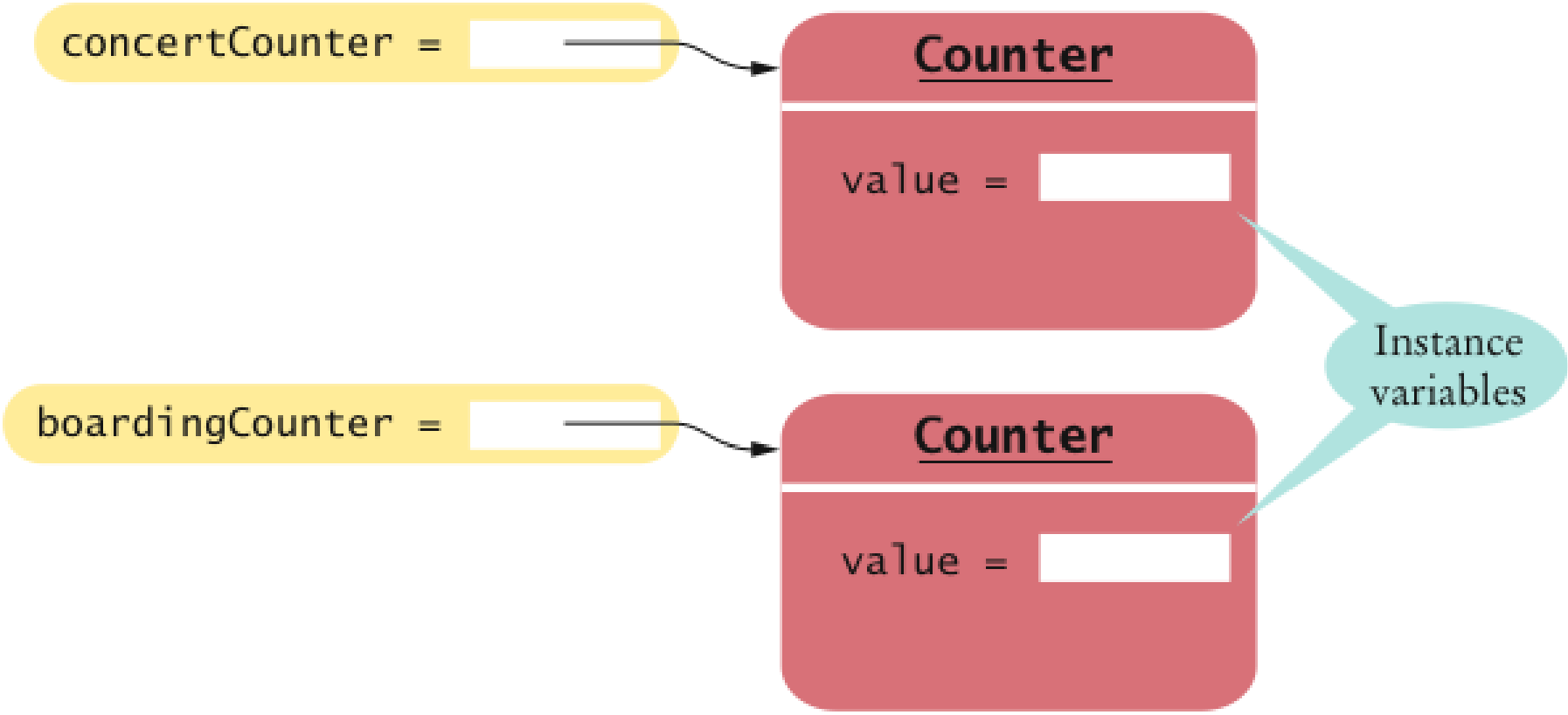


Figure 2 Instance Variables

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Syntax 3.1 Instance Variable Declaration

Syntax

```
accessSpecifier class ClassName
{
   accessSpecifier typeName variableName;
   . . .
}
```

Example

```
public class Counter
{
   private int value;
   . . .
}
```

Instance variables should always be private.

Each object of this class has a separate copy of this instance variable.

Type of the variable

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Accessing Instance Variables

- The `count` method advances the counter value by 1:

```
public void count()
{
   value = value + 1;
}
```

- The `getValue` method returns the current value:

```
public int getValue()
{
   return value;
}
```

- Private instance variables can only be accessed by methods of the same class

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.1

Supply the body of a method `public void reset()` that resets the counter back to zero.

Answer:

```
public void reset()
{
   value = 0;
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.2

Suppose you use a class `Clock` with private instance variables `hours` and `minutes`. How can you access these variables in your program?

Answer: You can only access them by invoking the methods of the `Clock` class.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Instance Variables

- **Encapsulation** is the process of hiding object data and providing methods for data access
- To encapsulate data, declare instance variables as `private` and declare public methods that access the variables
- Encapsulation allows a programmer to use a class without having to know its implementation
- Information hiding makes it simpler for the implementor of a class to locate errors and change implementations

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.3

Consider the `Counter` class. A counter's value starts at 0 and is advanced by the `count` method, so it should never be negative. Suppose you found a negative `value` variable during testing. Where would you look for the error?

Answer: In one of the methods of the `Counter` class.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.4

In Chapters 1 and 2, you used `System.out` as a black box to cause output to appear on the screen. Who designed and implemented `System.out`?

Answer: The programmers who designed and implemented the Java library.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.5

Suppose you are working in a company that produces personal finance software. You are asked to design and implement a class for representing bank accounts. Who will be the users of your class?

Answer: Other programmers who work on the personal finance application.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Specifying the Public Interface of a Class

Behavior of bank account (abstraction):

- deposit money
- withdraw money
- get balance

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Specifying the Public Interface of a Class: Methods

- Methods of `BankAccount` class:
  - `deposit`
  - `withdraw`
  - `getBalance`

- We want to support method calls such as the following:

```
harrysChecking.deposit(2000);
harrysChecking.withdraw(500);
System.out.println(harrysChecking.getBalance());
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Specifying the Public Interface of a Class: Method Declaration

access specifier (such as `public`)

- return type (such as `String` or `void`)
- method name (such as `deposit`)
- list of parameters (`double amount` for `deposit`)
- method body in `{ }`

Examples:

- `public void deposit(double amount) { . . . }`
- `public void withdraw(double amount) { . . . }`
- `public double getBalance() { . . . }`

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Specifying the Public Interface of a Class: Method Header

- access specifier (such as `public`)
- return type (such as `void` or `double`)
- method name (such as `deposit`)
- list of parameter variables (such as `double amount`)

Examples:

- `public void deposit(double amount)`
- `public void withdraw(double amount)`
- `public double getBalance()`

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Specifying the Public Interface of a Class: Constructor Declaration

- A constructor initializes the instance variables
- Constructor name = class name

```
public BankAccount()
{
   // body--filled in later
}
```

- Constructor body is executed when new object is created
- Statements in constructor body will set the internal data of the object that is being constructed
- All constructors of a class have the same name
- Compiler can tell constructors apart because they take different parameters

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

`BankAccount` Public Interface

The public constructors and methods of a class form the *public interface* of the class:

```
public class BankAccount
{
   // private variables--filled in later

   // Constructors public BankAccount()
   {
      // body--filled in later
   }

   public BankAccount(double initialBalance)
   {
      // body--filled in later
   }
```

Continued

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

`BankAccount` Public Interface (cont.)

```
   // Methods
   public void deposit(double amount)
   {
      // body--filled in later
   }
   public void withdraw(double amount)
   {
      // body--filled in later
   }
   public double getBalance()
   {
      // body--filled in later
   }
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Syntax 3.2 Class Declaration

Syntax

```
accessSpecifier class ClassName
{
   instance variables
   constructors
   methods
}
```

Example

```
public class Counter
{
   private int value;

   public Counter(double initialValue) { value = initialValue; }

   public void count() { value = value + 1; }
   public int getValue() { return value; }
}
```

Public interface

Private implementation

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.6

How can you use the methods of the public interface to *empty* the `harrysChecking` bank account?

Answer:

```
harrysChecking.withdraw(harrysChecking.getBalance())
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.7

What is wrong with this sequence of statements?

```
BankAccount harrysChecking = new BankAccount(10000);
System.out.println(harrysChecking.withdraw(500));
```

Answer: The `withdraw` method has return type `void`. It doesn't return a value. Use the `getBalance` method to obtain the balance after the withdrawal.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.8

Suppose you want a more powerful bank account abstraction that keeps track of an *account number* in addition to the balance. How would you change the public interface to accommodate this enhancement?

Answer: Add an `accountNumber` parameter to the constructors, and add a `getAccountNumber` method. There is no need for a `setAccountNumber` method – the account number never changes after construction.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Commenting the Public Interface

```
/**
   Withdraws money from the bank account.
   @param amount the amount to withdraw
*/
public void withdraw(double amount)
{
   //implementation filled in later
}
/**
   Gets the current balance of the bank account.
   @return the current balance
*/
public double getBalance()
{
   //implementation filled in later
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Class Comment

```
/**
   A bank account has a balance that can be changed by
   deposits and withdrawals.
*/
public class BankAccount
{
   . . .
}
```

- Provide documentation comments for
  - *every class*
  - *every method*
  - *every parameter*
  - *every return value*

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Javadoc Method Summary

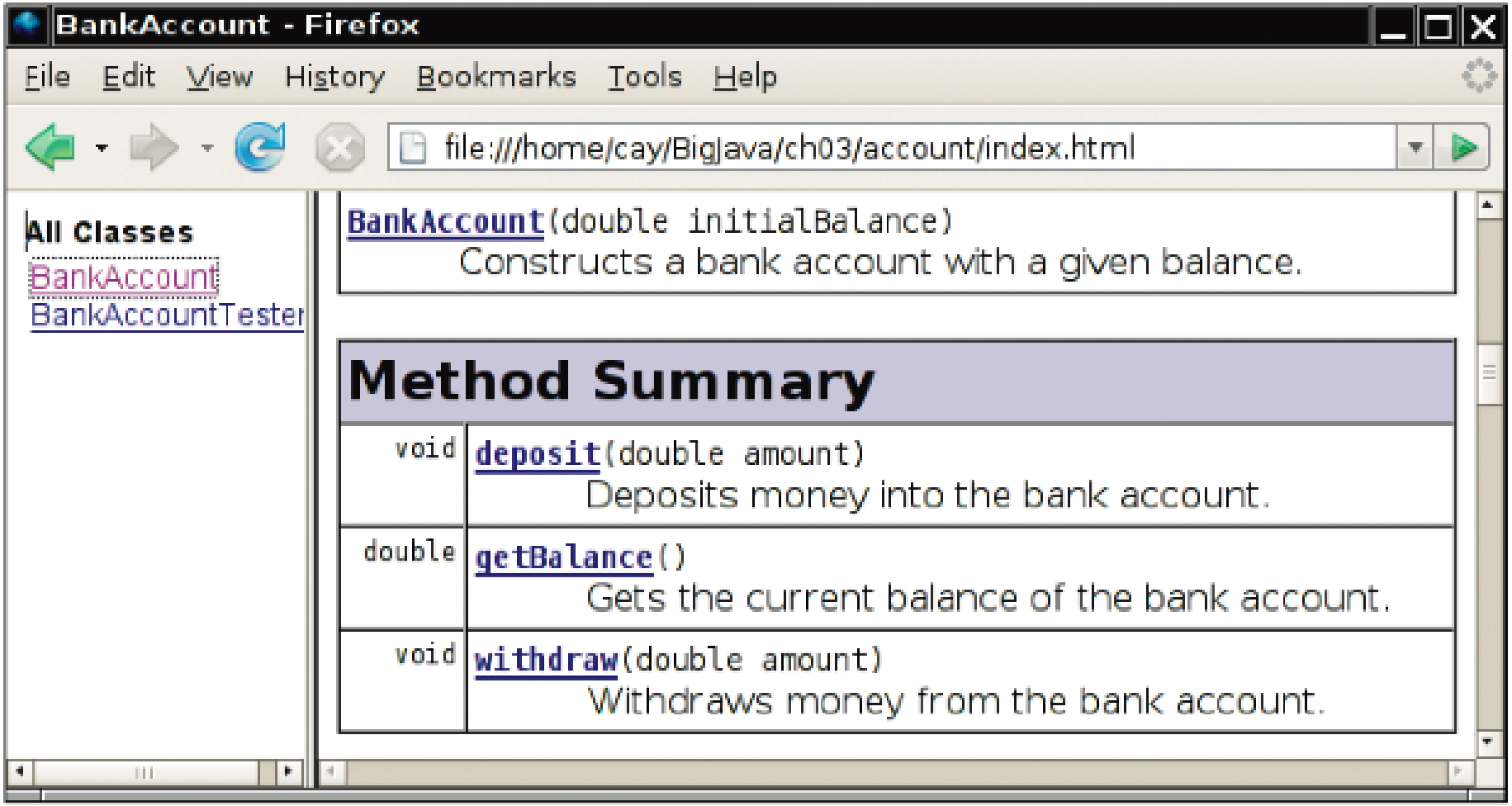


Figure 3 A Method Summary Generated by javadoc

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Javadoc Method Detail

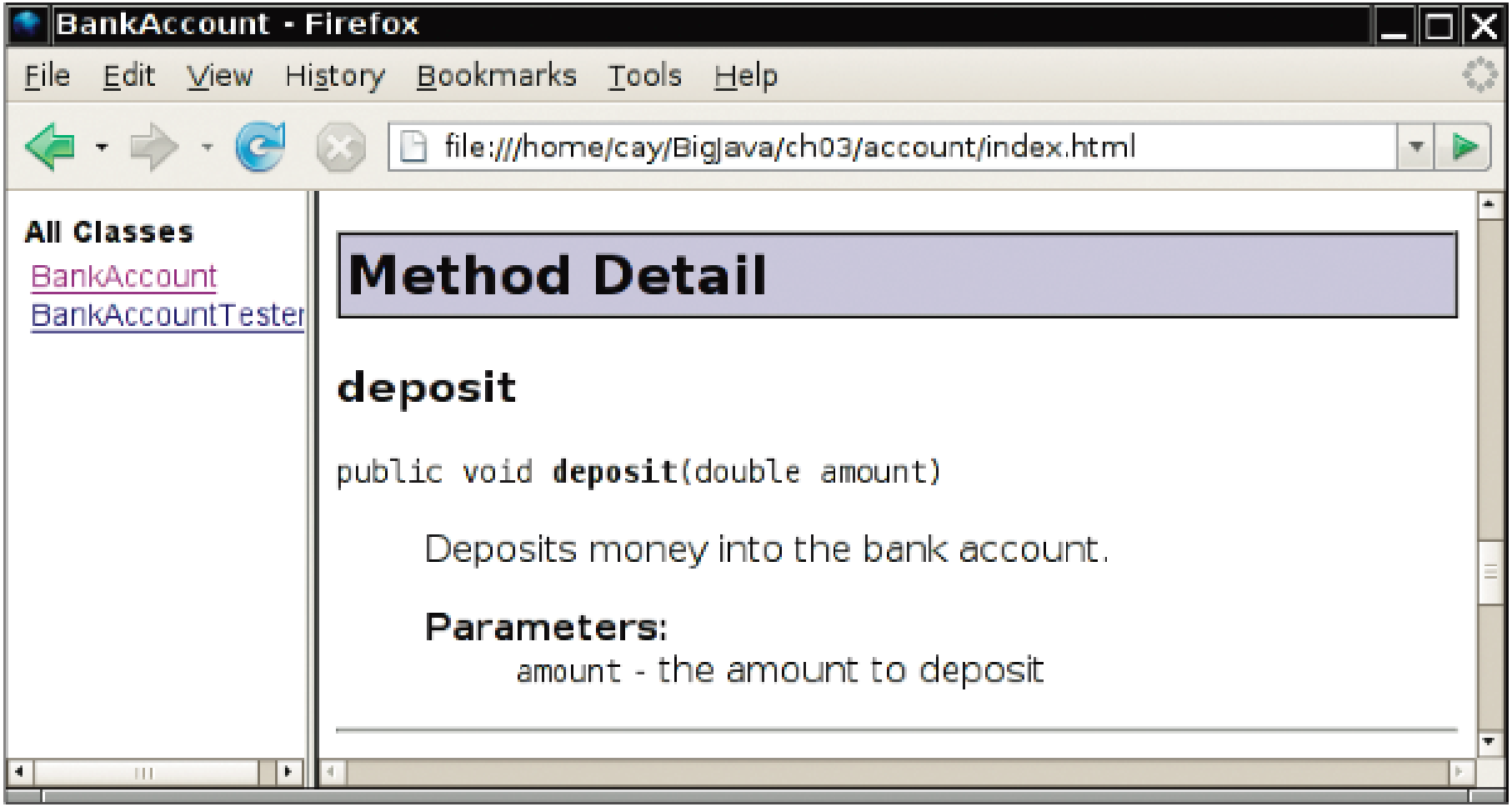


Figure 4 Method Detail Generated by javadoc

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.9

Provide documentation comments for the `Counter` class of Section 3.1.

Answer:

```
/**
   This class models a tally counter.
*/
public class Counter
{
   private int value;
   /**
      Gets the current value of this counter.
      @return the current value
   */
   public int getValue()
   {
      return value;
   }
```

Continued

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.9 (cont.)

```
   /**
      Advances the value of this counter by 1.
   */
   public void count()
   {
      value = value + 1;
   }
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.10

Suppose we enhance the `BankAccount` class so that each account has an account number. Supply a documentation comment for the constructor

```
public BankAccount(int accountNumber, double initialBalance)
```

Answer:

```
/**
   Constructs a new bank account with a given initial balance.
   @param accountNumber the account number for this account
   @param initialBalance the initial balance for this account
*/
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.11

Why is the following documentation comment questionable?

```
/**
   Each account has an account number.
   @return the account number of this account
*/
public int getAccountNumber()
```

Answer: The first sentence of the method description should describe the method – it is displayed in isolation in the summary table.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implementing Constructors

- Constructors contain instructions to initialize the instance variables of an object:

```
public BankAccount()
{
   balance = 0;
}

public BankAccount(double initialBalance)
{
   balance = initialBalance;
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Constructor Call Example

- Statement:

```
BankAccount harrysChecking = new BankAccount(1000);
```

  - *Create a new object of type* `BankAccount`
  - *Call the second constructor (because a construction parameter is supplied in the constructor call)*
  - *Set the parameter variable* `initialBalance` *to* `1000`
  - *Set the* `balance` *instance variable of the newly created object to* `initialBalance`
  - *Return an object reference, that is, the memory location of the object, as the value of the* `new` *expression*
  - *Store that object reference in the* `harrysChecking` *variable*

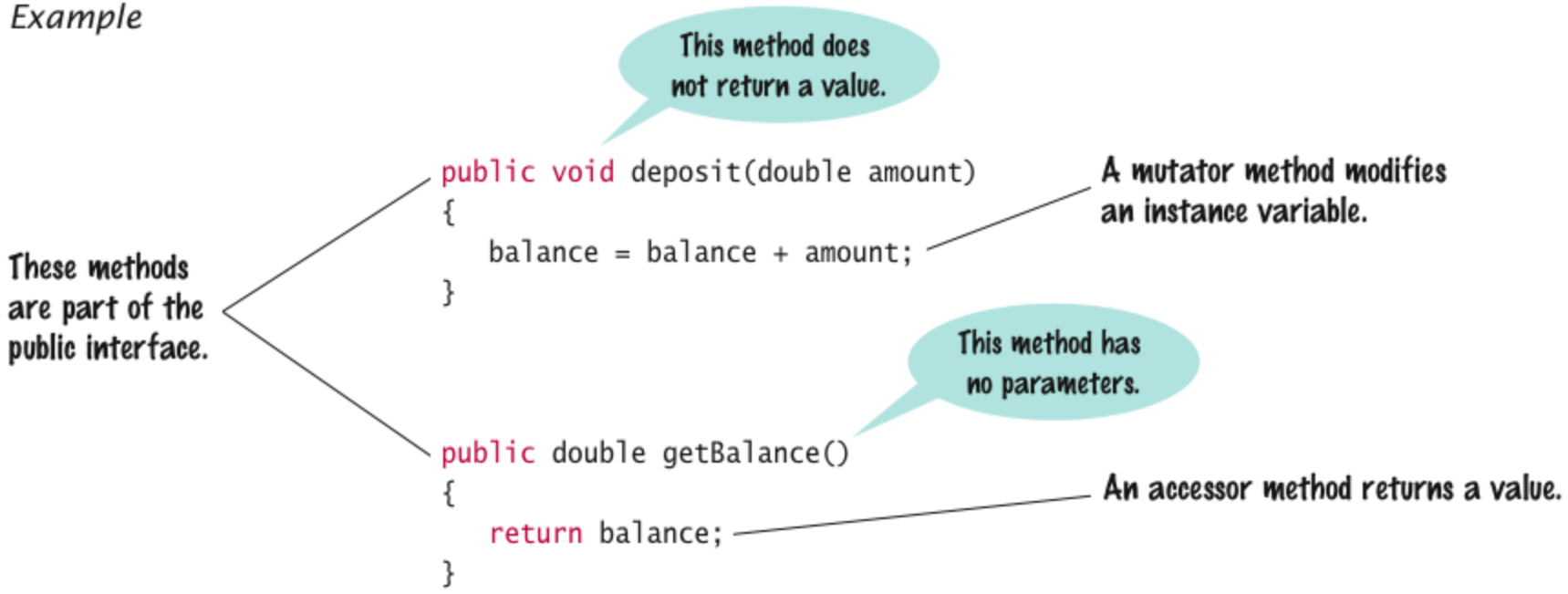
Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Syntax 3.3 Method Declaration

Syntax

```
accessSpecifier returnType methodName(parameterType parameterName, . . . )
{
   method body
}
```

Example



Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implementing Methods

- `deposit` method:

```
public void deposit(double amount)
{
   balance = balance + amount;
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Method Call Example

- Statement:

```
harrysChecking.deposit(500);
```

  - *Set the parameter variable* `amount` *to 500*
  - *Fetch the* `balance` *variable of the object whose location is stored in* `harrysChecking`
  - *Add the value of* `amount` *to* `balance`
  - *Store the sum in the* `balance` *instance variable, overwriting the old value*

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implementing Methods

- ```
  public void withdraw(double amount)
  {
     balance = balance - amount;
  }
  ```
- ```
  public double getBalance()
  {
     return balance;
  }
  ```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

ch03/account/BankAccount.java

```
1  /**
2     A bank account has a balance that can be changed by
3     deposits and withdrawals.
4  */
5  public class BankAccount
6  {
7     private double balance;
8
9     /**
10       Constructs a bank account with a zero balance.
11    */
12    public BankAccount()
13    {
14       balance = 0;
15    }
16
17    /**
18       Constructs a bank account with a given balance.
19       @param initialBalance the initial balance
20    */
21    public BankAccount(double initialBalance)
22    {
23       balance = initialBalance;
24    }
```


Continued

ch03/account/BankAccount.java (cont.)

```
25
26      /**
27         Deposits money into the bank account.
28         @param amount the amount to deposit
29      */
30      public void deposit(double amount)
31      {
32         balance = balance + amount;
33      }
34
35      /**
36         Withdraws money from the bank account.
37         @param amount the amount to withdraw
38      */
39      public void withdraw(double amount)
40      {
41         balance = balance - amount;
42      }
43
```


Continued

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

ch03/account/BankAccount.java (cont.)

```
44      /**
45         Gets the current balance of the bank account.
46         @return the current balance
47      */
48      public double getBalance()
49      {
50         return balance;
51      }
52   }
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.12

Suppose we modify the `BankAccount` class so that each bank account has an account number. How does this change affect the instance variables?

Answer:

An instance variable

```
private int accountNumber;
```

needs to be added to the class.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.13

Why does the following code not succeed in robbing mom's bank account?

```
public class BankRobber
{
   public static void main(String[] args)
   {
      BankAccount momsSavings = new BankAccount(1000);
      momsSavings.balance = 0;
   }
}
```

Answer: Because the `balance` instance variable is accessed from the `main` method of `BankRobber`. The compiler will report an error because `balance` has private access in `BankAccount`.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.14

The `Rectangle` class has four instance variables: `x`, `y`, `width`, and `height`. Give a possible implementation of the `getWidth` method.

Answer:

```
public int getWidth()
{
   return width;
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.15

Give a possible implementation of the `translate` method of the `Rectangle` class.

Answer: There is more than one correct answer. One possible implementation is as follows:

```
public void translate(int dx, int dy)
{
   int newx = x + dx;
   x = newx;
   int newy = y + dy;
   y = newy;
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Unit Testing

- *Unit test*: Verifies that a class works correctly in isolation, outside a complete program
- To test a class, use an environment for interactive testing, or write a tester class
- *Tester class*: A class with a main method that contains statements to test another class
- Typically carries out the following steps:
  1. *Construct one or more objects of the class that is being tested*
  2. *Invoke one or more methods*
  3. *Print out one or more results*
  4. *Print the expected results*

Continued

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

ch03/account/BankAccountTester.java

```
1  /**
2     A class to test the BankAccount class.
3  */
4  public class BankAccountTester
5  {
6     /**
7        Tests the methods of the BankAccount class.
8        @param args not used
9     */
10    public static void main(String[] args)
11    {
12       BankAccount harrysChecking = new BankAccount();
13       harrysChecking.deposit(2000);
14       harrysChecking.withdraw(500);
15       System.out.println(harrysChecking.getBalance());
16       System.out.println("Expected: 1500");
17    }
18 }
```

Program Run:

```
1500
Expected: 1500
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

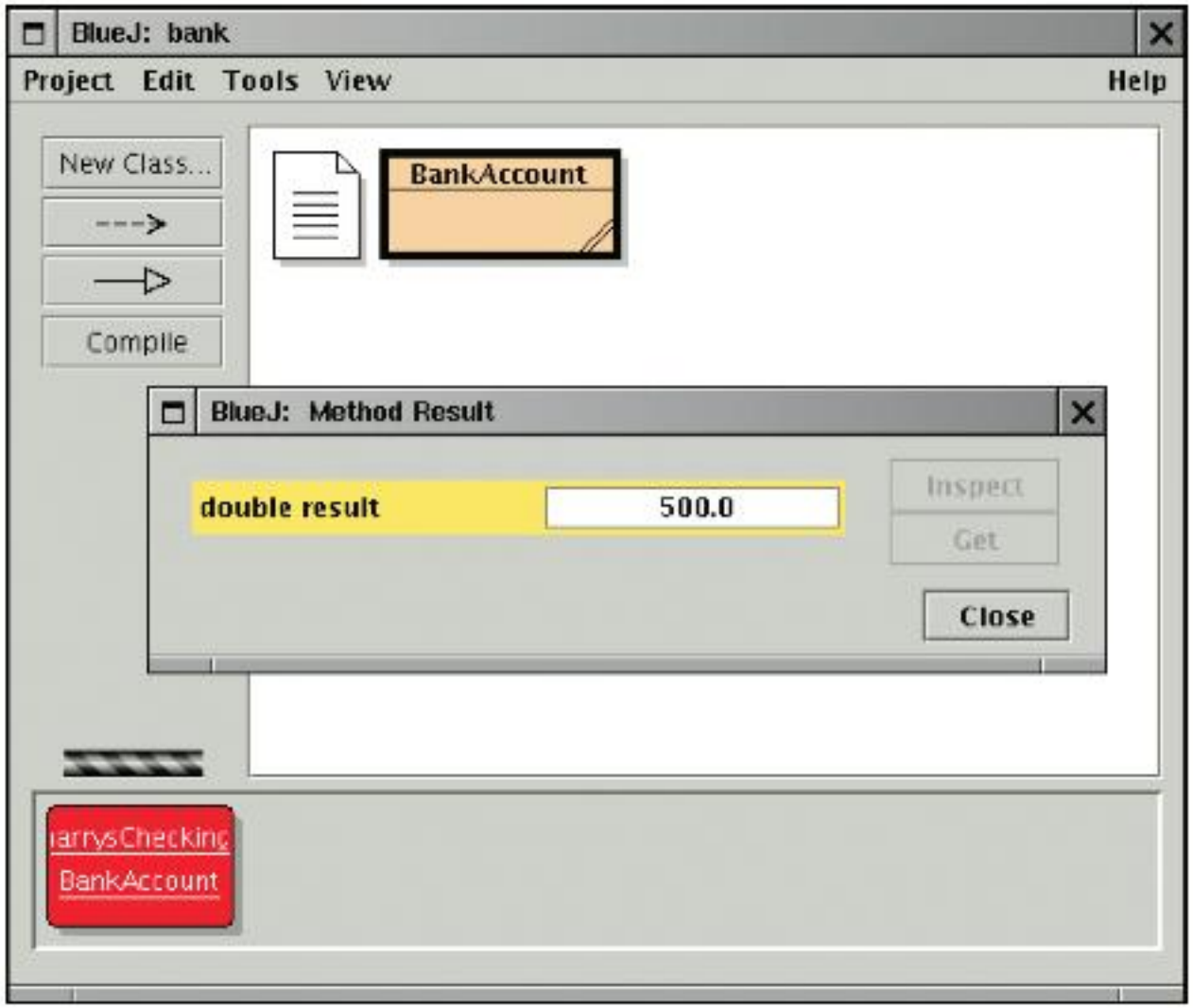
Unit Testing (cont.)

- Details for building the program vary. In most environments, you need to carry out these steps:
  1. *Make a new subfolder for your program*
  2. *Make two files, one for each class*
  3. *Compile both files*
  4. *Run the test program*

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Testing With BlueJ

Figure 5
The Return Value of the `getBalance` Method in BlueJ



Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.16

When you run the `BankAccountTester` program, how many objects of class `BankAccount` are constructed? How many objects of type `BankAccountTester`?

Answer: One `BankAccount` object, no `BankAccountTester` object. The purpose of the `BankAccountTester` class is merely to hold the `main` method.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.17

Why is the `BankAccountTester` class unnecessary in development environments that allow interactive testing, such as BlueJ?

Answer: In those environments, you can issue interactive commands to construct `BankAccount` objects, invoke methods, and display their return values.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Local Variables

- Local and parameter variables belong to a method
  - *When a method or constructor runs, its local and parameter variables come to life*
  - *When the method or constructor exits, they are removed immediately*
- Instance variables belongs to an objects, not methods
  - *When an object is constructed, its instance variables are created*
  - *The instance variables stay alive until no method uses the object any longer*

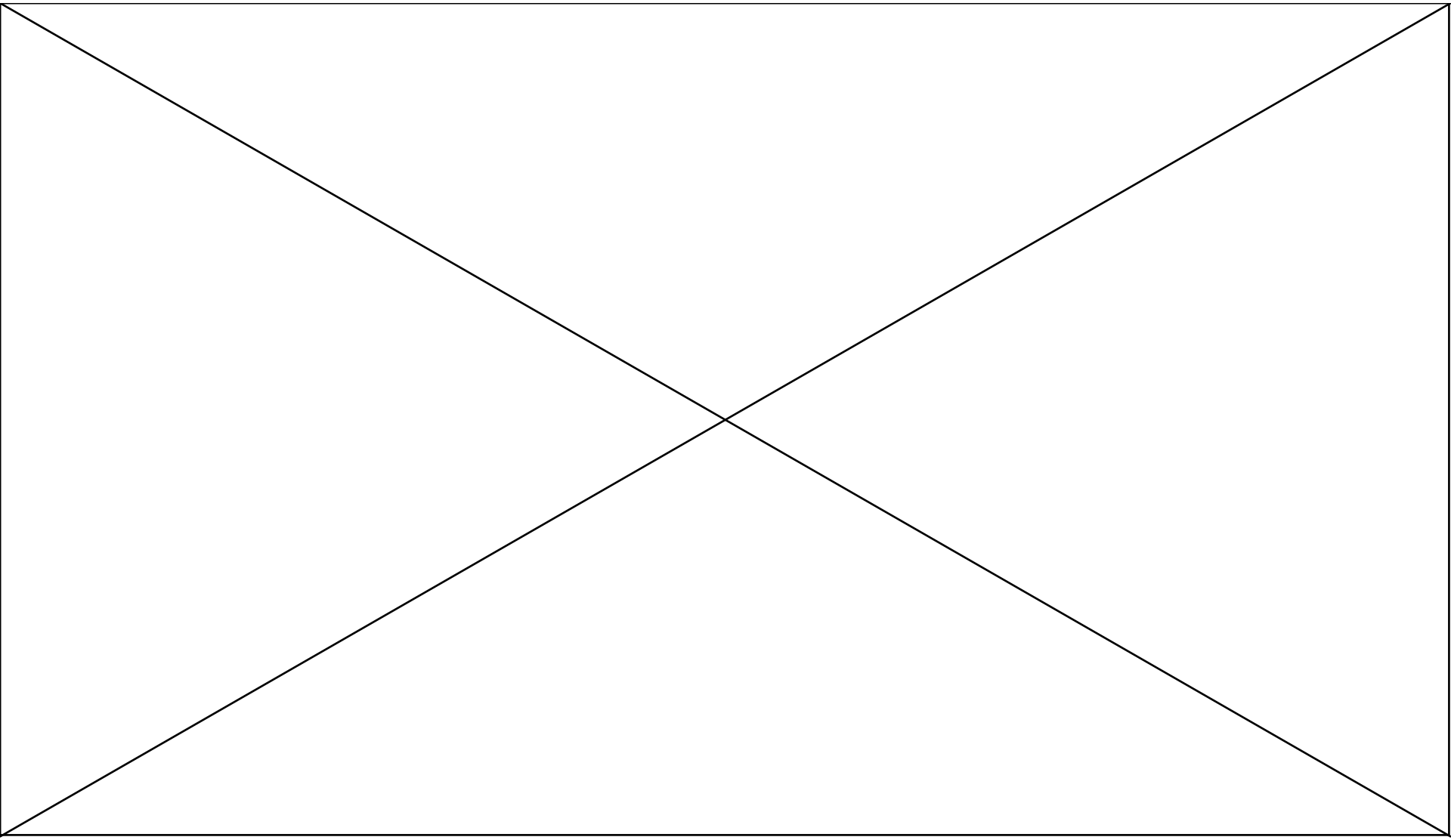
Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Local Variables

- In Java, the *garbage collector* periodically reclaims objects when they are no longer used
- Instance variables are initialized to a default value, but you must initialize local variables

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Animation 3.1: Lifetime of Variables



Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.18

What do local variables and parameter variables have in common? In which essential aspect do they differ?

Answer: Variables of both categories belong to methods – they come alive when the method is called, and they die when the method exits. They differ in their initialization. Parameter variables are initialized with the call values; local variables must be explicitly initialized.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.19

Why was it necessary to introduce the local variable `change` in the `giveChange` method? That is, why didn't the method simply end with the statement

```
return payment - purchase;
```

Answer: After computing the change due, `payment` and `purchase` were set to zero. If the method returned `payment - purchase`, it would always return zero.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implicit Parameter

- The **implicit parameter** of a method is the object on which the method is invoked

```
public void deposit(double amount)
{
   balance = balance + amount;
}
```

- In the call

```
momsSavings.deposit(500)
```

  The implicit parameter is `momsSavings` and the explicit parameter is `500`

- When you refer to an instance variable inside a method, it means the instance variable of the implicit parameter

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implicit Parameters and `this`

- The `this` reference denotes the implicit parameter
- `balance = balance + amount;`

  actually means

  `this.balance = this.balance + amount;`
- When you refer to an instance variable in a method, the compiler automatically applies it to the `this` reference

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implicit Parameters and `this`

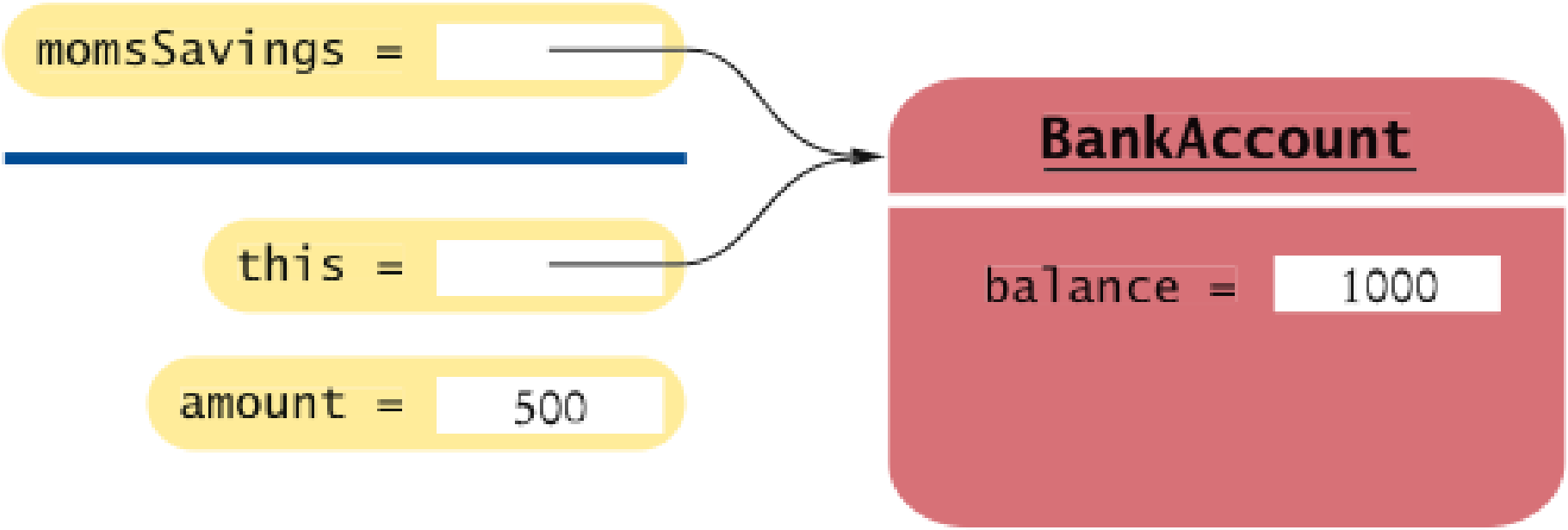


Figure 6 The Implicit Parameter of a Method Call

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implicit Parameters and `this`

- Some programmers feel that manually inserting the `this` reference before every instance variable reference makes the code clearer:

```
public BankAccount(double initialBalance)
{
   this.balance = initialBalance;
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implicit Parameters and `this`

- A method call without an implicit parameter is applied to the same object
- Example:

```
public class BankAccount
{
   . . .
   public void monthlyFee()
   {
      withdraw(10); // Withdraw $10 from this account
   }
}
```

- The implicit parameter of the `withdraw` method is the (invisible) implicit parameter of the `monthlyFee` method

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Implicit Parameters and `this`

- You can use the `this` reference to make the method easier to read:

```
public class BankAccount
{
   . . .
   public void monthlyFee()
   {
      this.withdraw(10); // Withdraw $10 from
  this account
   }
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.20

How many implicit and explicit parameters does the `withdraw` method of the `BankAccount` class have, and what are their names and types?

Answer: One implicit parameter, called `this`, of type `BankAccount`, and one explicit parameter, called `amount`, of type `double`.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.21

In the `deposit` method, what is the meaning of `this.amount`? Or, if the expression has no meaning, why not?

Answer: It is not a legal expression. `this` is of type `BankAccount` and the `BankAccount` class has no variable named `amount`. s

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.22

How many implicit and explicit parameters does the `main` method of the `BankAccountTester` class have, and what are they called?

Answer: No implicit parameter – the main method is not ivoked on any object – and one explicit parameter, called `args`.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Shape Classes

- Good practice: Make a class for each graphical shape

```
public class Car
{
   public Car(int x, int y)
   {
      // Remember position
      . . .
   }
      public void draw(Graphics2D g2)
   {
      // Drawing instructions
      . . .
   }
}
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Drawing Cars

- Draw two cars: one in top-left corner of window, and another in the bottom right
- Compute bottom right position, inside `paintComponent` method:

```
int x = getWidth() - 60;
int y = getHeight() - 30;
Car car2 = new Car(x, y);
```

- `getWidth` and `getHeight` are applied to object that executes `paintComponent`
- If window is resized `paintComponent` is called and car position recomputed

Continued

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Drawing Cars (cont.)

Figure 7
The Car Component Draws Two Car Shapes

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Plan Complex Shapes on Graph Paper

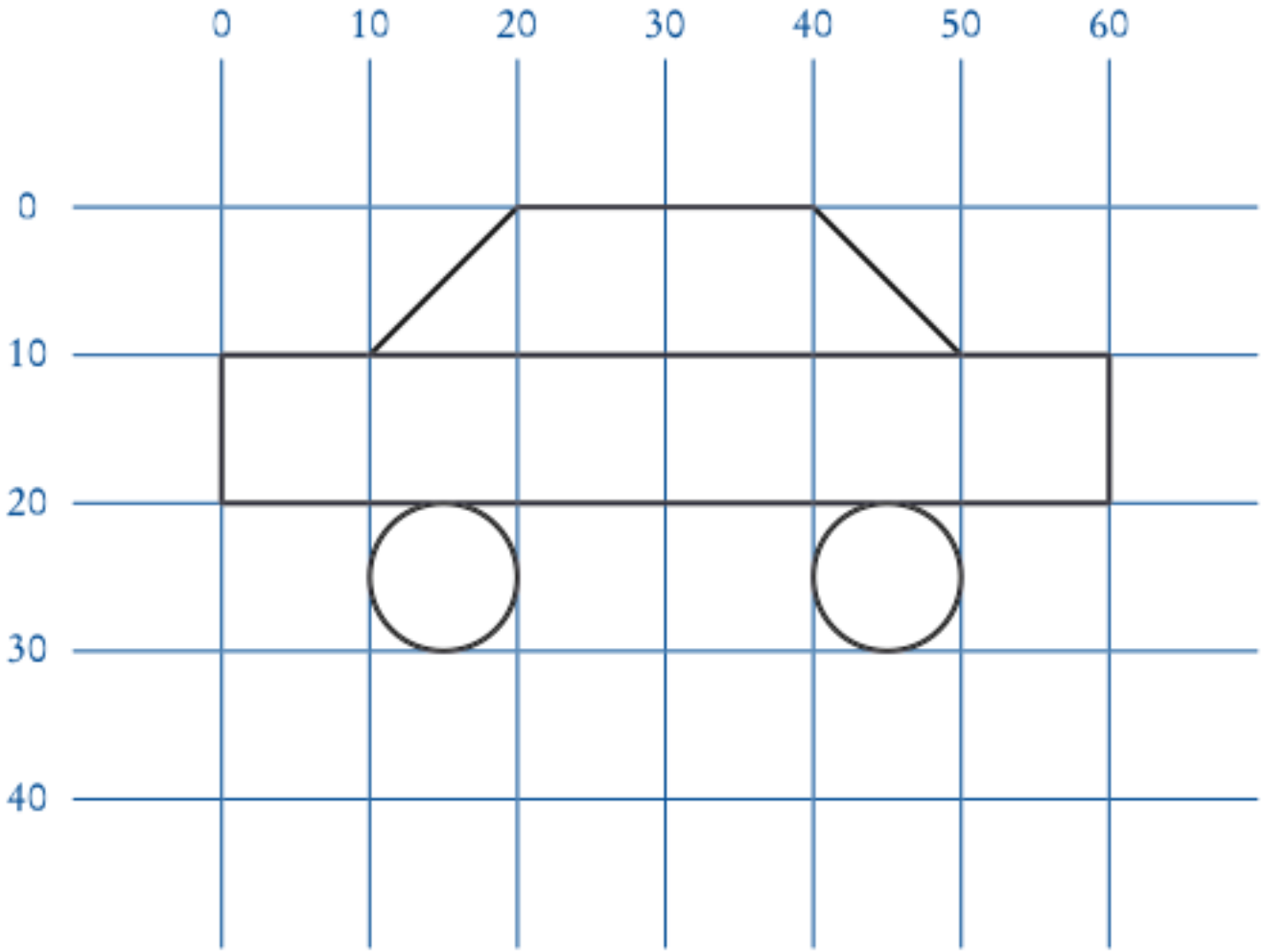


Figure 8 Using Graph Paper to Find Shape Coordinates

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Classes of Car Drawing Program

- `Car`: responsible for drawing a single car
  - *Two objects of this class are constructed, one for each car*
- `CarComponent`: displays the drawing
- `CarViewer`: shows a frame that contains a `CarComponent`

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

ch03/car/Car.java

```
1  import java.awt.Graphics2D;
2  import java.awt.Rectangle;
3  import java.awt.geom.Ellipse2D;
4  import java.awt.geom.Line2D;
5  import java.awt.geom.Point2D;
6
7  /**
8     A car shape that can be positioned anywhere on the screen.
9  */
10 public class Car
11 {
12    private int xLeft;
13    private int yTop;
14
15    /**
16       Constructs a car with a given top left corner.
17       @param x the x coordinate of the top left corner
18       @param y the y coordinate of the top left corner
19    */
20    public Car(int x, int y)
21    {
22       xLeft = x;
23       yTop = y;
24    }
```


Continued

ch03/car/Car.java (cont.)

```
25
26      /**
27         Draws the car.
28         @param g2 the graphics context
29      */
30      public void draw(Graphics2D g2)
31      {
32         Rectangle body
33               = new Rectangle(xLeft, yTop + 10, 60, 10);
34         Ellipse2D.Double frontTire
35               = new Ellipse2D.Double(xLeft + 10, yTop + 20, 10, 10);
36         Ellipse2D.Double rearTire
37               = new Ellipse2D.Double(xLeft + 40, yTop + 20, 10, 10);
38
39         // The bottom of the front windshield
40         Point2D.Double r1
41               = new Point2D.Double(xLeft + 10, yTop + 10);
42         // The front of the roof
43         Point2D.Double r2
44               = new Point2D.Double(xLeft + 20, yTop);
45         // The rear of the roof
46         Point2D.Double r3
47               = new Point2D.Double(xLeft + 40, yTop);
```

Continued

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

ch03/car/Car.java (cont.)

```
48          // The bottom of the rear windshield
49          Point2D.Double r4
50                = new Point2D.Double(xLeft + 50, yTop + 10);
51
52          Line2D.Double frontWindshield
53                = new Line2D.Double(r1, r2);
54          Line2D.Double roofTop
55                = new Line2D.Double(r2, r3);
56          Line2D.Double rearWindshield
57                = new Line2D.Double(r3, r4);
58
59          g2.draw(body);
60          g2.draw(frontTire);
61          g2.draw(rearTire);
62          g2.draw(frontWindshield);
63          g2.draw(roofTop);
64          g2.draw(rearWindshield);
65       }
66    }
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

ch03/car/CarComponent.java

```
1  import java.awt.Graphics;
2  import java.awt.Graphics2D;
3  import javax.swing.JComponent;
4
5  /**
6     This component draws two car shapes.
7  */
8  public class CarComponent extends JComponent
9  {
10    public void paintComponent(Graphics g)
11    {
12       Graphics2D g2 = (Graphics2D) g;
13
14       Car car1 = new Car(0, 0);
15
16       int x = getWidth() - 60;
17       int y = getHeight() - 30;
18
19       Car car2 = new Car(x, y);
20
21       car1.draw(g2);
22       car2.draw(g2);
23    }
24 }
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

ch03/car/CarViewer.java

```
 1  import javax.swing.JFrame;
 2
 3  public class CarViewer
 4  {
 5     public static void main(String[] args)
 6     {
 7        JFrame frame = new JFrame();
 8
 9        frame.setSize(300, 400);
10        frame.setTitle("Two cars");
11        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
12
13        CarComponent component = new CarComponent();
14        frame.add(component);
15
16        frame.setVisible(true);
17     }
18  }
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.23

Which class needs to be modified to have the two cars positioned next to each other?

Answer: `CarComponent`

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Self Check 3.24

Which class needs to be modified to have the car tires painted in black, and what modification do you need to make?

Answer: In the `draw` method of the `Car` class, call

```
g2.fill(frontTire);
g2.fill(rearTire);
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

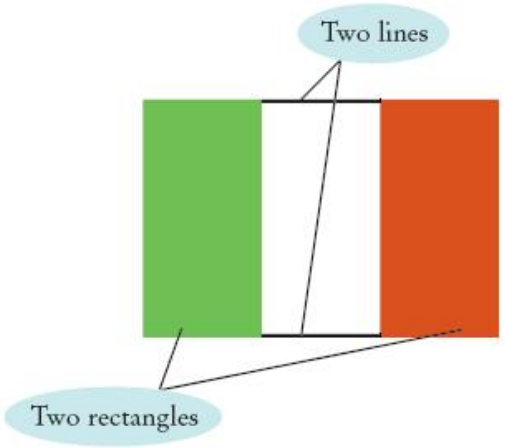
Self Check 3.25

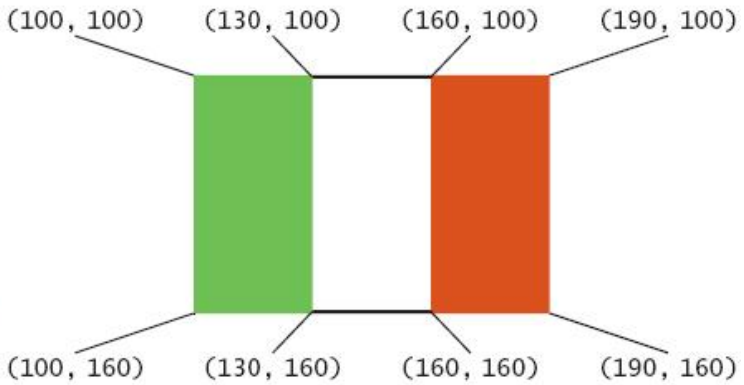
How do you make the cars twice as big?

Answer: Double all measurements in the `draw` method of the `Car` class.

Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.

Drawing Graphical Shapes






```
Rectangle leftRectangle = new Rectangle(100, 100, 30, 60);
Rectangle rightRectangle = new Rectangle(160, 100, 30, 60);
Line2D.Double topLine = new Line2D.Double(130, 100, 160, 100);
Line2D.Double bottomLine = new Line2D.Double(130, 160, 160, 160);
```


Big Java by Cay Horstmann
Copyright © 2009 by John Wiley & Sons. All rights reserved.