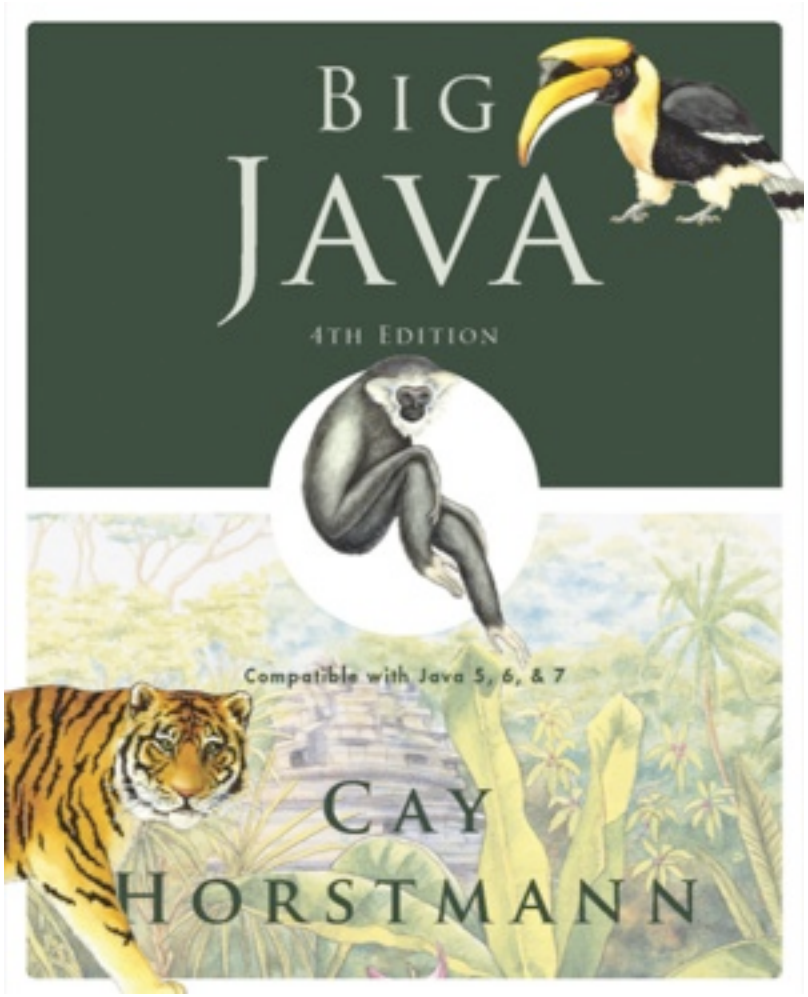


Chapter 2 – Using Objects

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"`?

Self Check 2.2

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

Self Check 2.3

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

Self Check 2.4

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

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

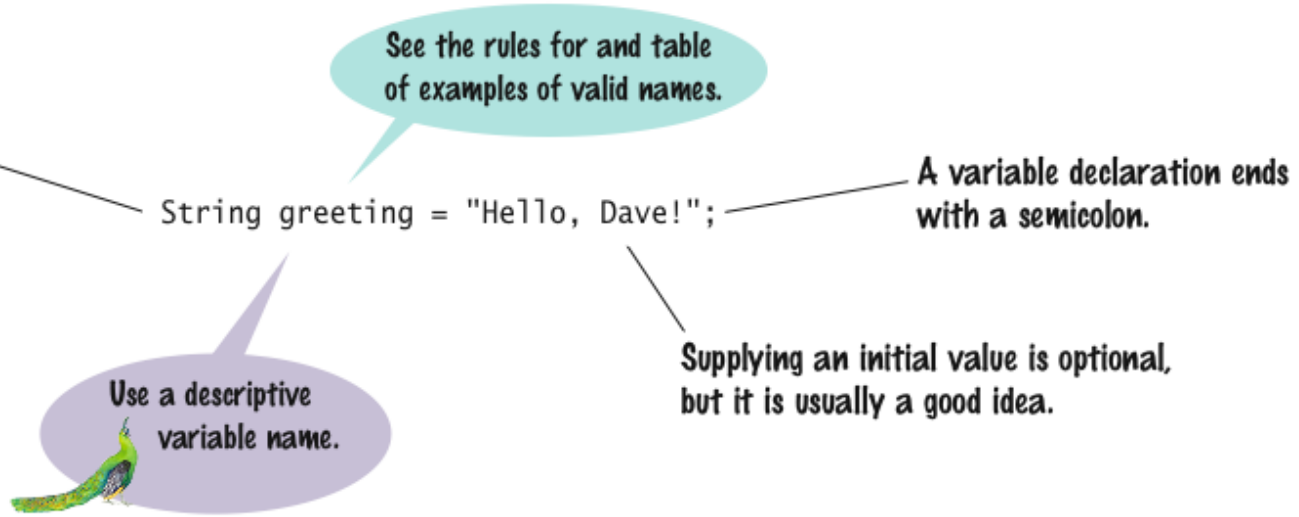
- By convention, variable names start with a lowercase letter
  - *"Camel case": Capitalize the first letter of a word in a compound word such as* `farewellMessage`
- By convention, class names start with an uppercase letter
- Do not use the `$` symbol in names — it is intended for names that are automatically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

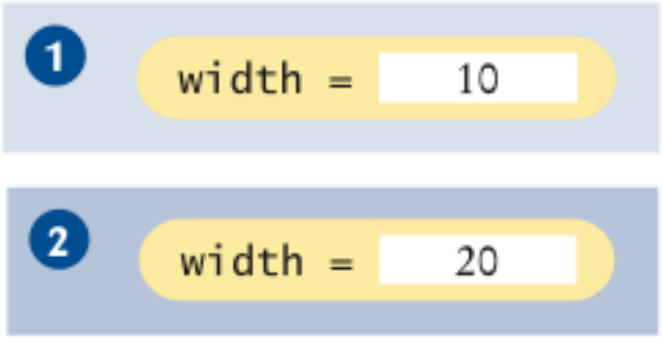
Self Check 2.6

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

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
```

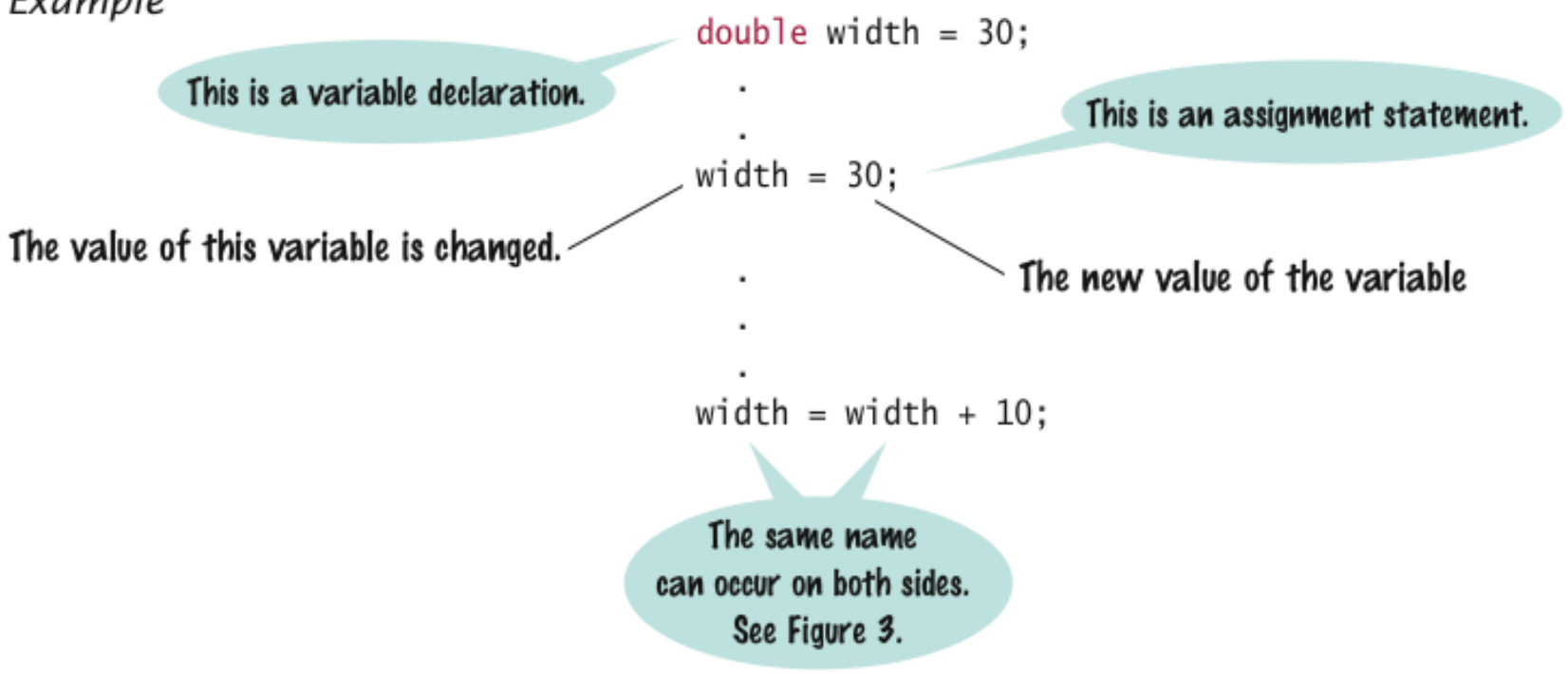
- Even better, initialize the variable when you declare it:

```
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`

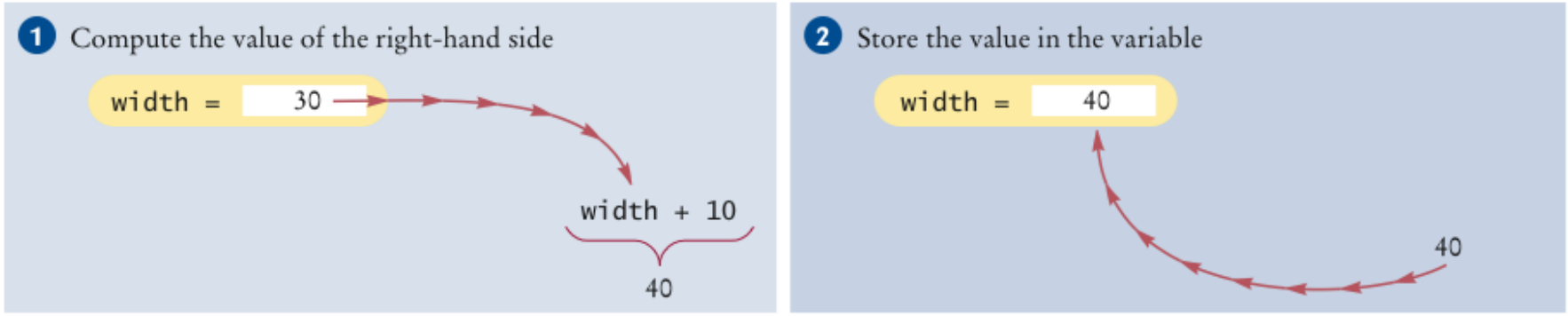
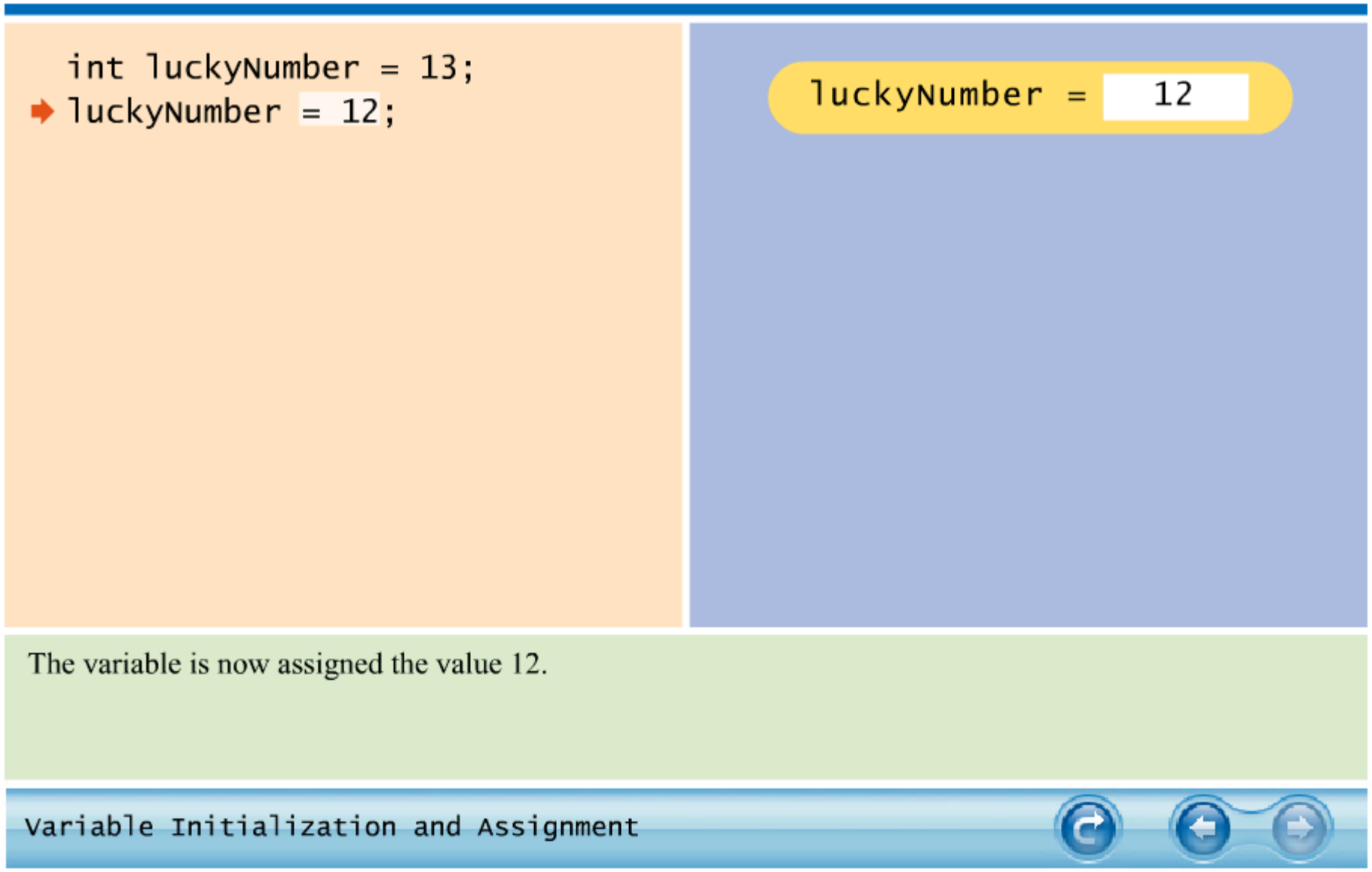


Figure 3 Executing the Statement `width = width + 10`

Animation 2.1: Variable Initialization and Assignment



Self Check 2.7

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

Self Check 2.8

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

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`

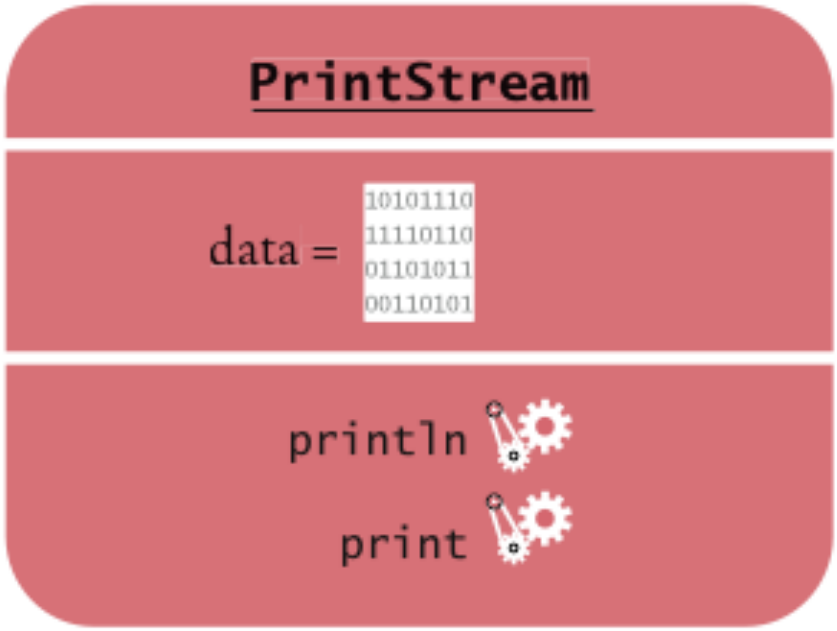


Figure 4 Representation of the `System.out` Object

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

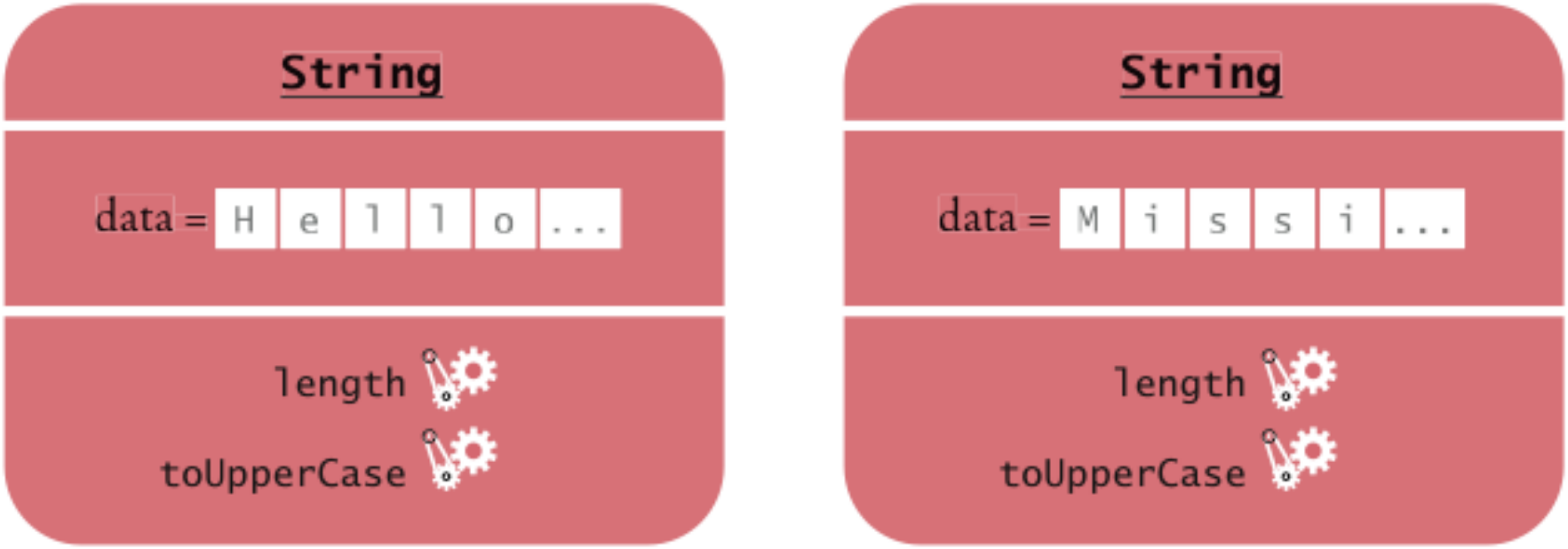


Figure 5 A Representation of Two String Objects

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"`?

Self Check 2.10

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

Self Check 2.11

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

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

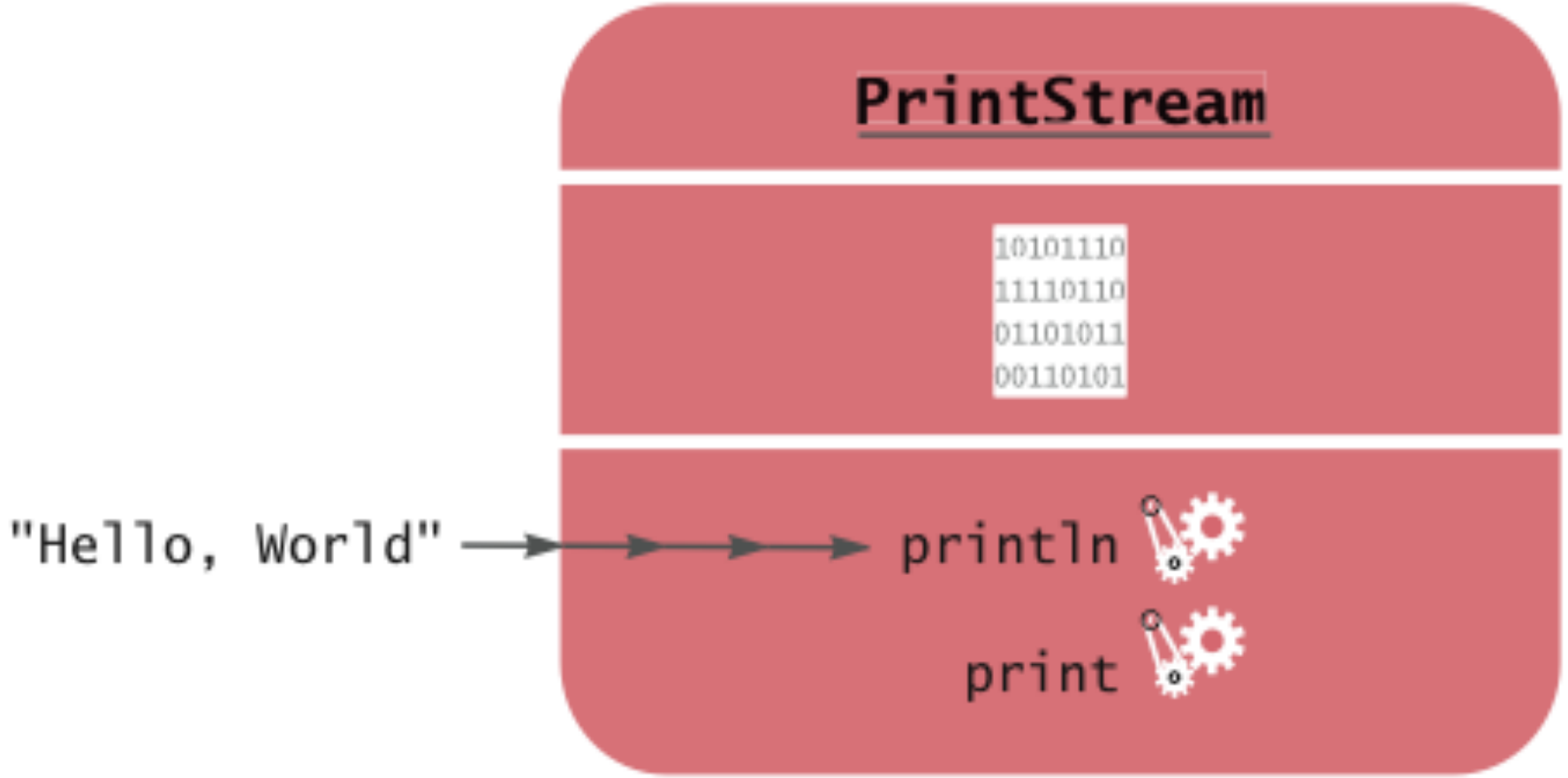


Figure 6 Passing a Parameter to the `println` Method

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
```

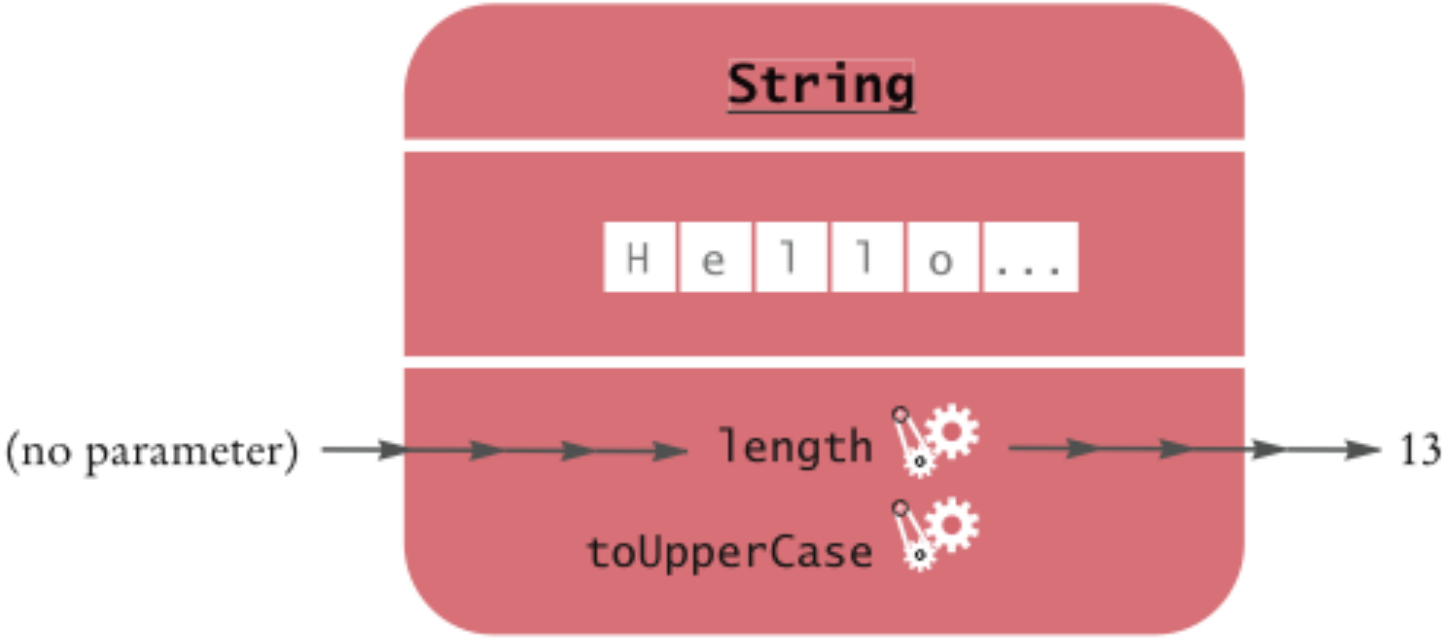


Figure 7 Invoking the `length` Method on a `String` Object

Passing Return Values

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

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

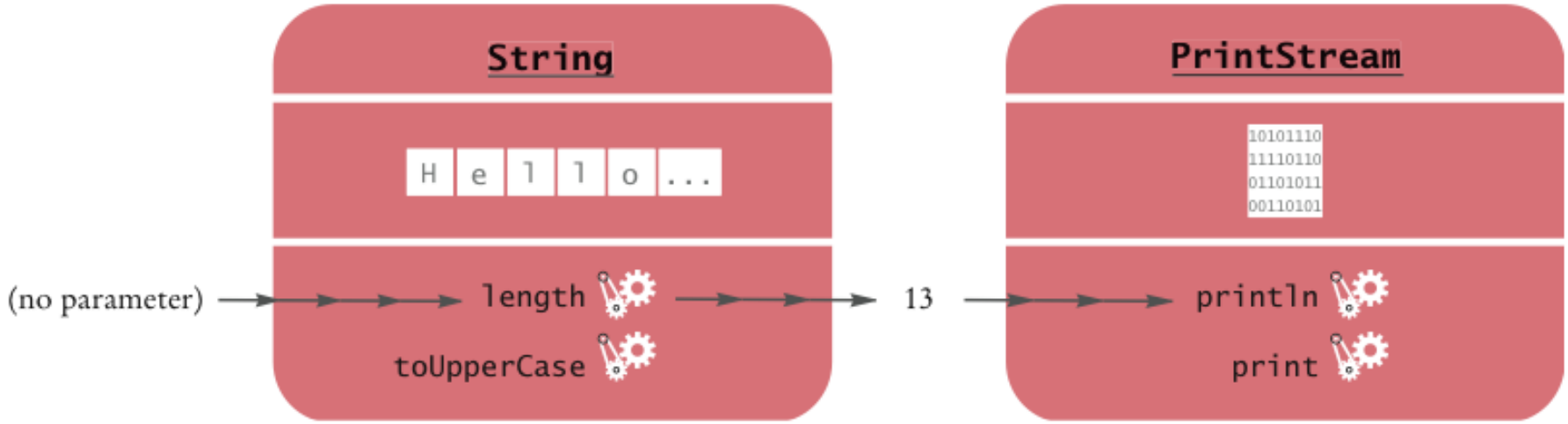


Figure 8 Passing the Result of a Method Call to Another Method

- 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")
```

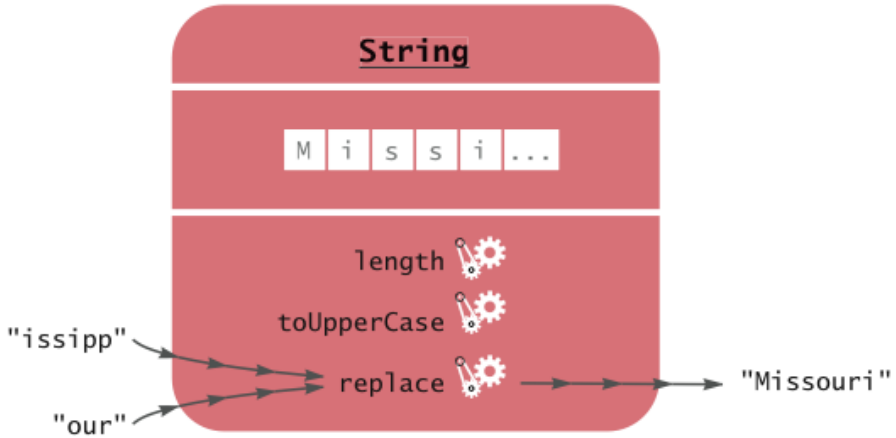
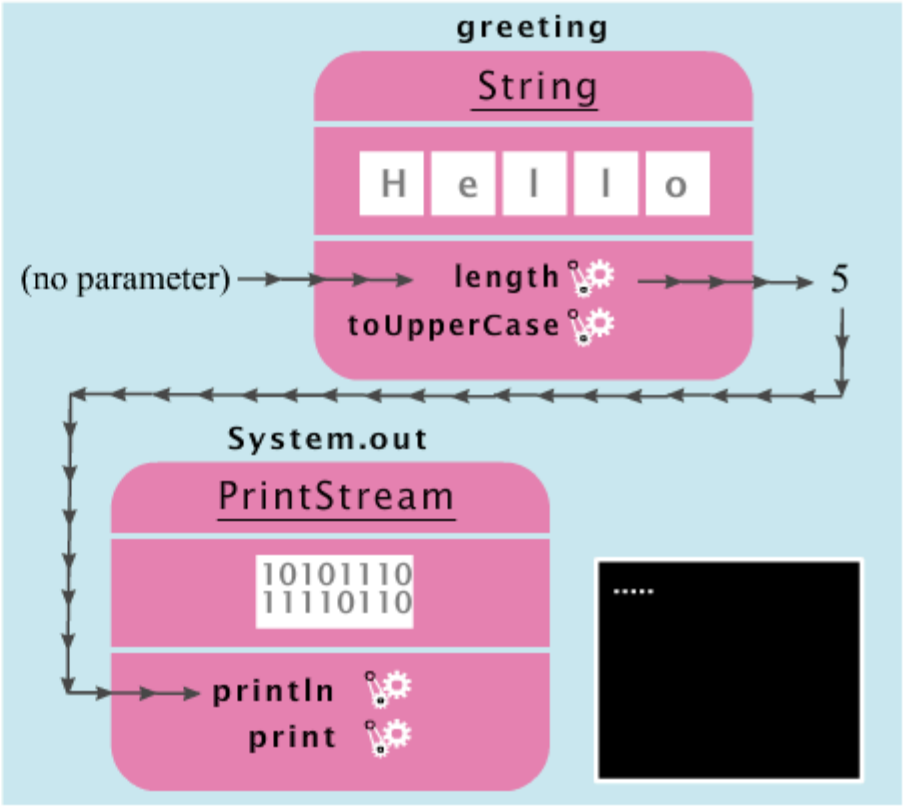


Figure 9 Calling the `replace` Method

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

Animation 2.2: Parameter Passing

```
System.out.println(5);
```



The `println` method causes the value 5 to appear in the console window. It does not return a value.

Parameter Passing

Self Check 2.12

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

Self Check 2.13

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

Self Check 2.14

What is the result of the call

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

?

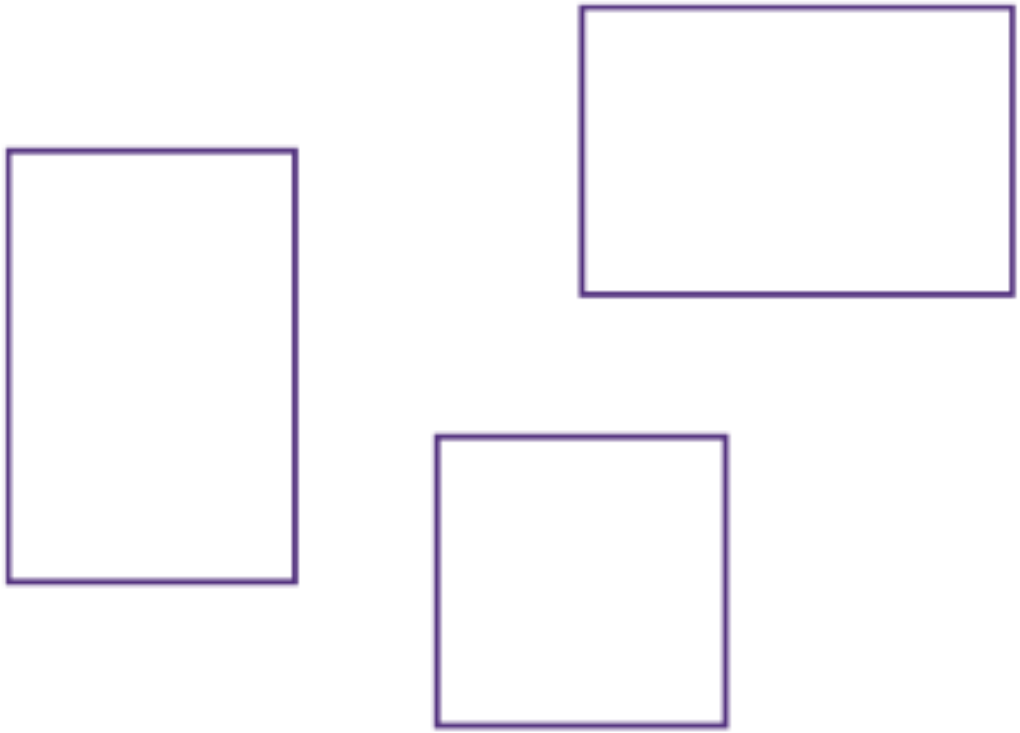
Self Check 2.15

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

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:

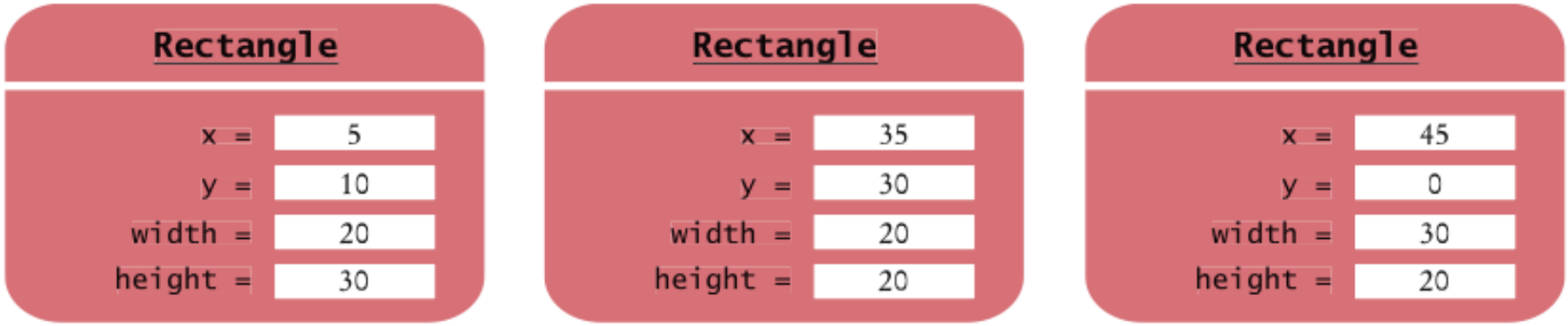


Figure 11 Rectangle Objects

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:

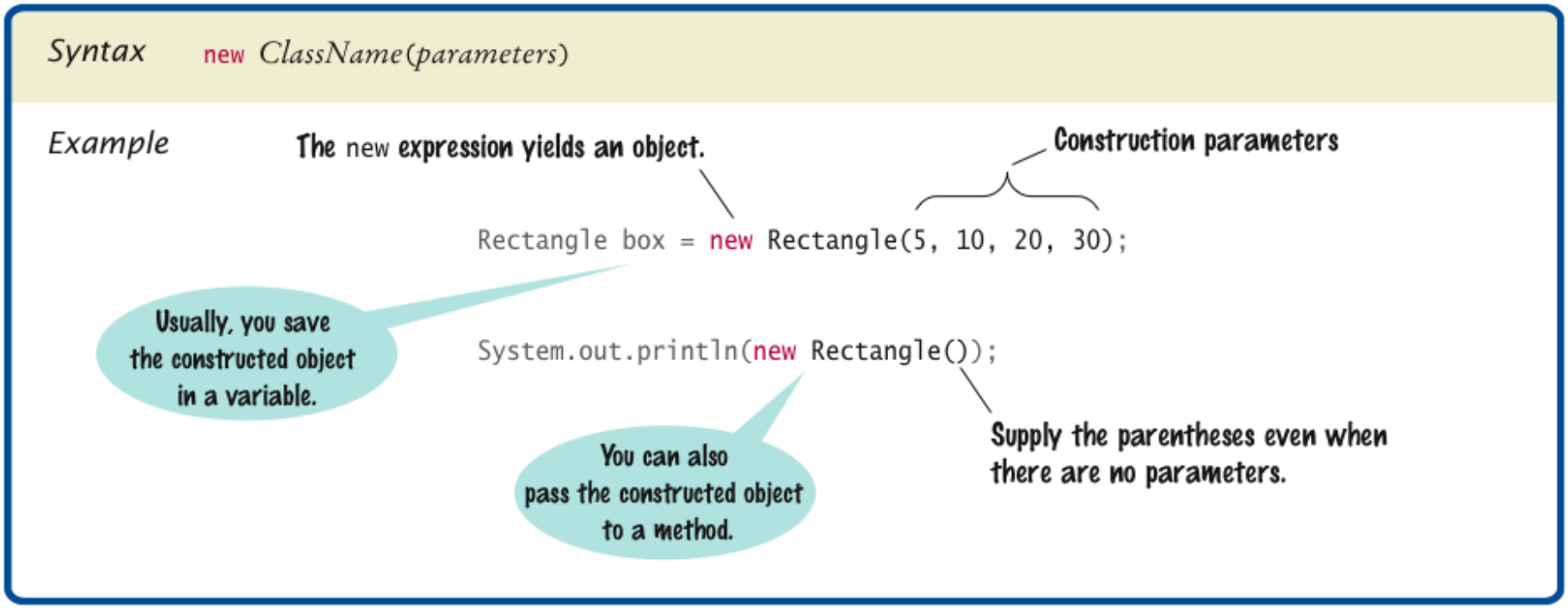
```
Rectangle box = new Rectangle(5, 10, 20, 30);
```

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



Self Check 2.16

How do you construct a square with center (100, 100) and side length 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());
```

Accessor and Mutator Methods

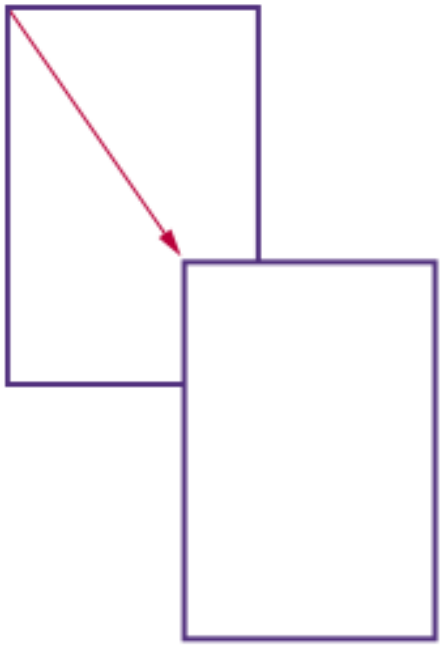
- **Accessor method**: does not change the state of its implicit parameter:

```
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?

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)?

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

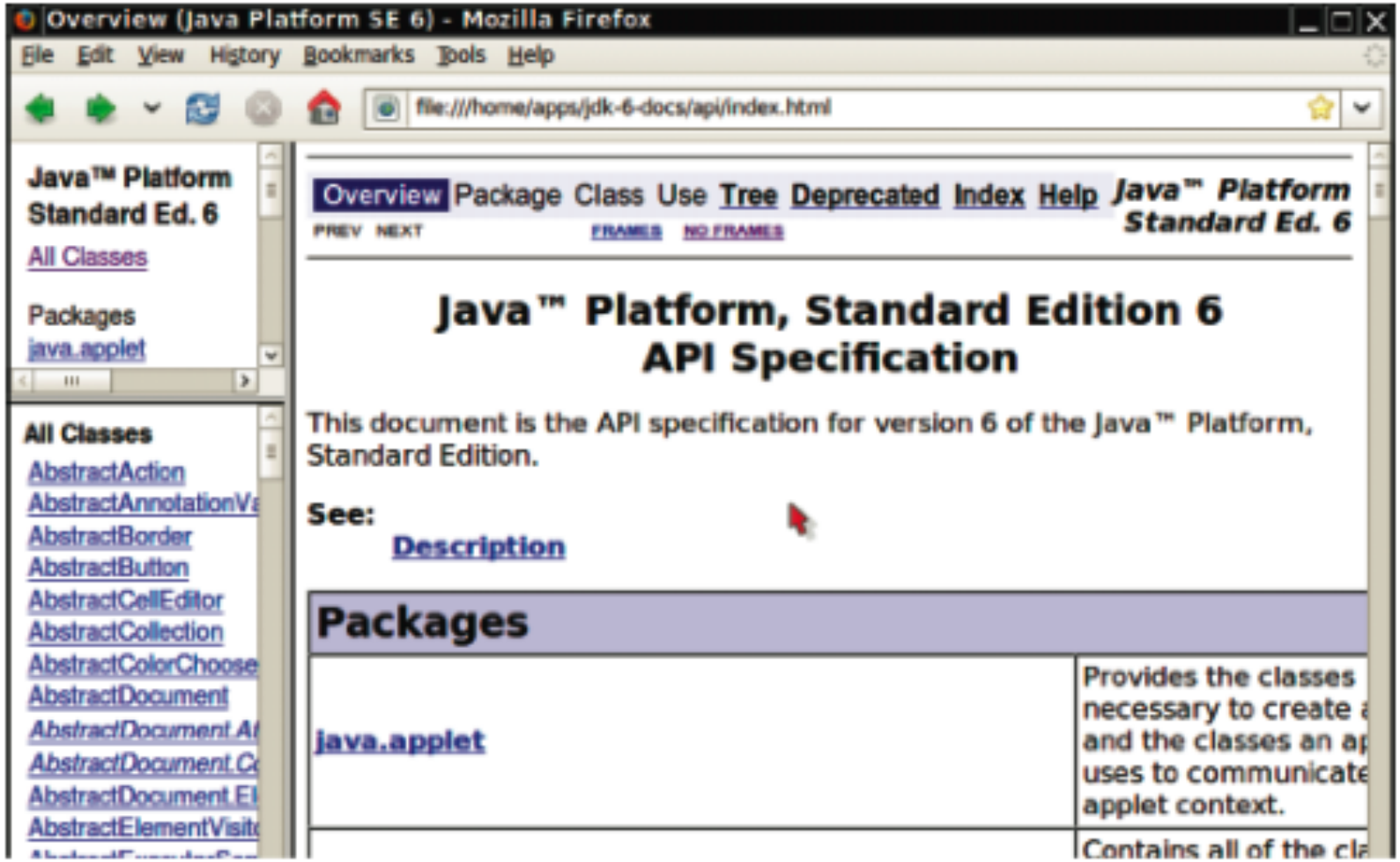


Figure 13 The API Documentation of the Standard Java Library

The API Documentation for the Rectangle Class

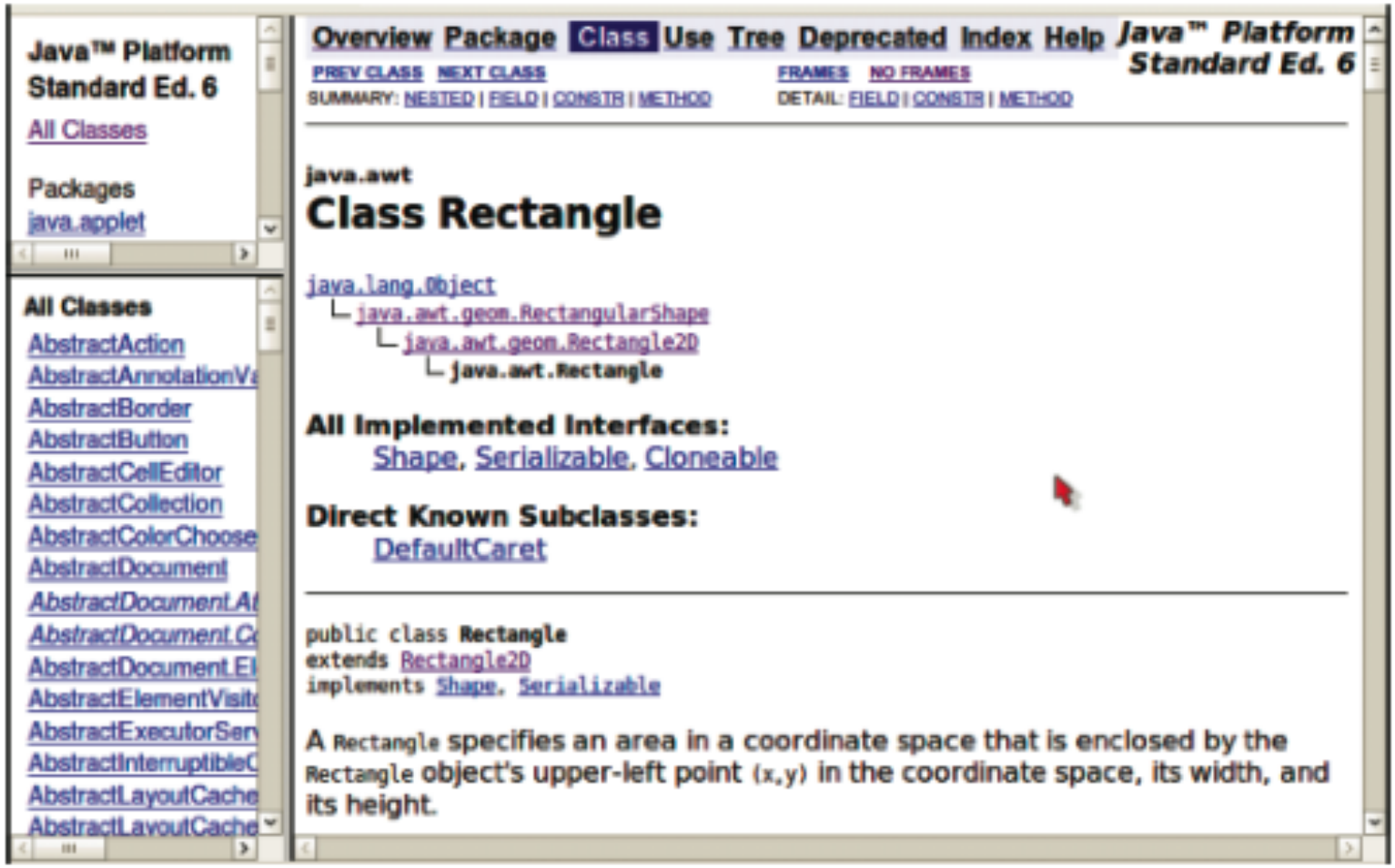


Figure 14 The API Documentation for the Rectangle Class

Method Summary

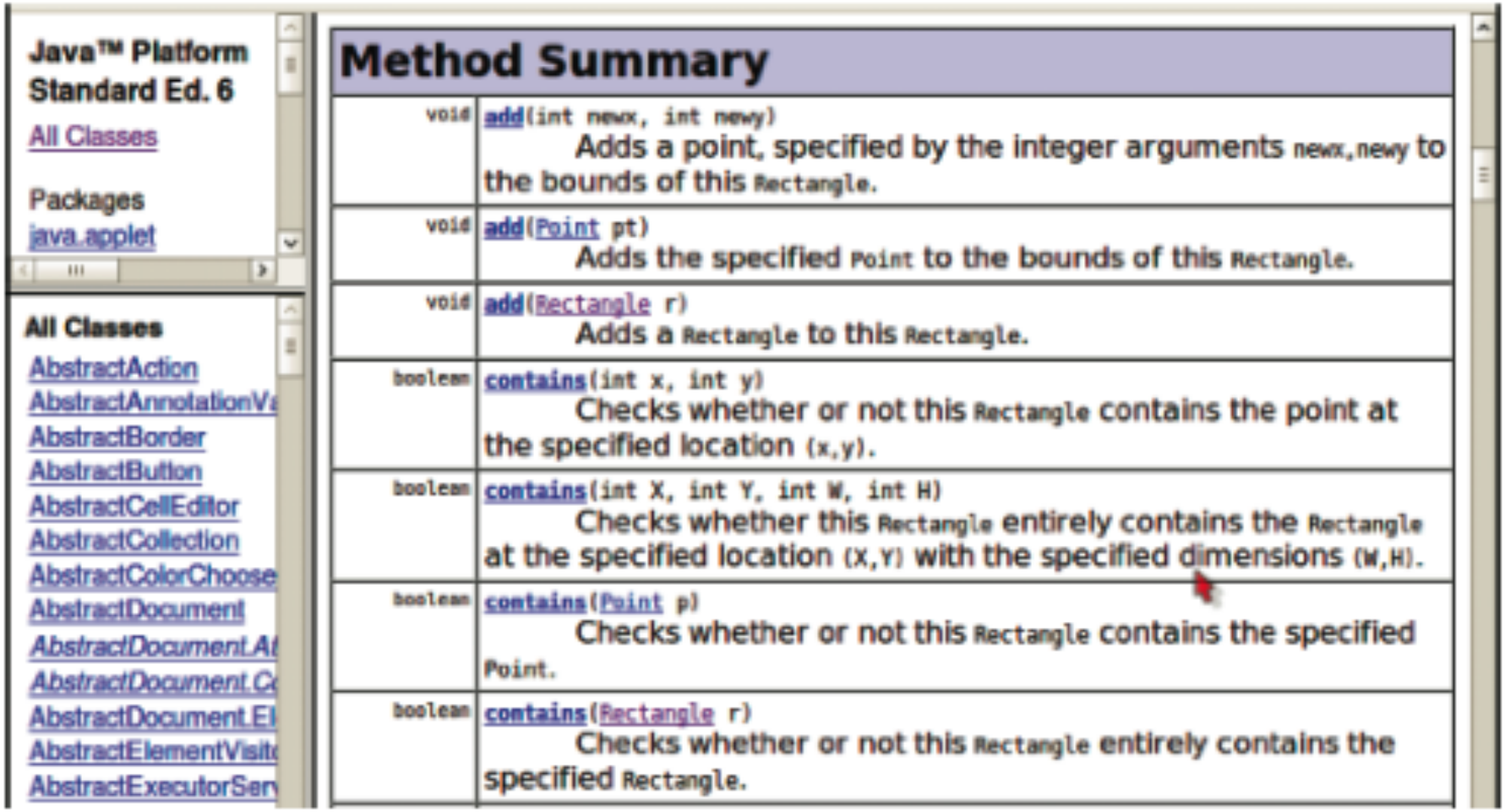


Figure 15 The Method Summary for the Rectangle Class

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)

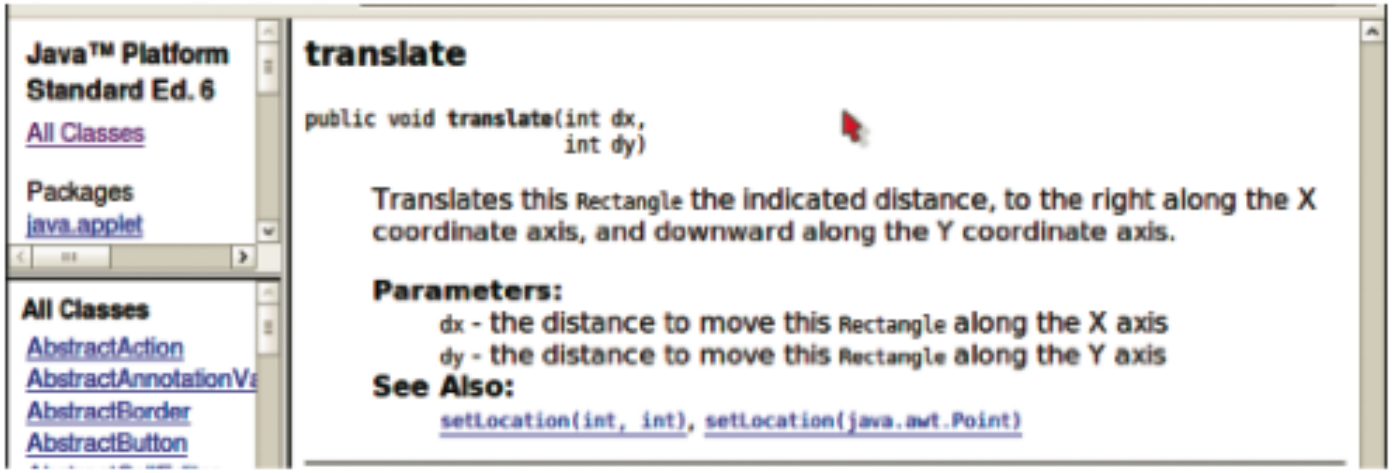


Figure 16 The API Documentation of the `translate` Method

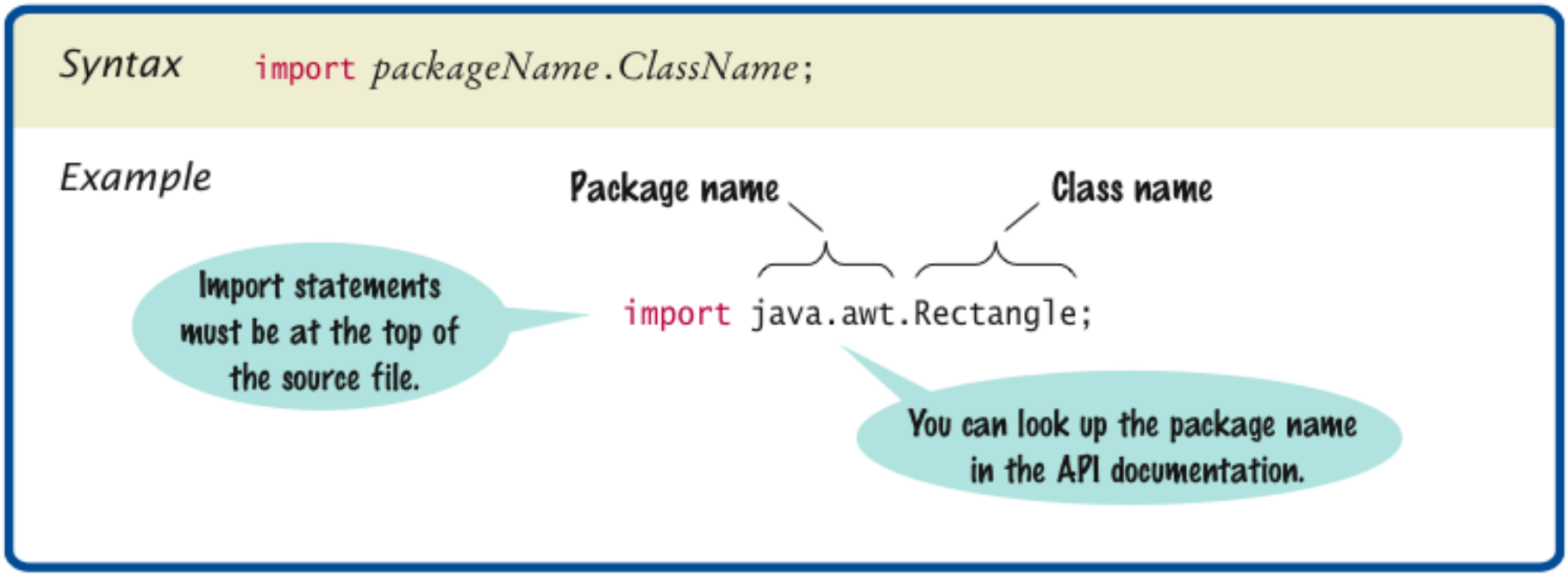
Packages

- **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



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!”`?

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.)

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?

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
```

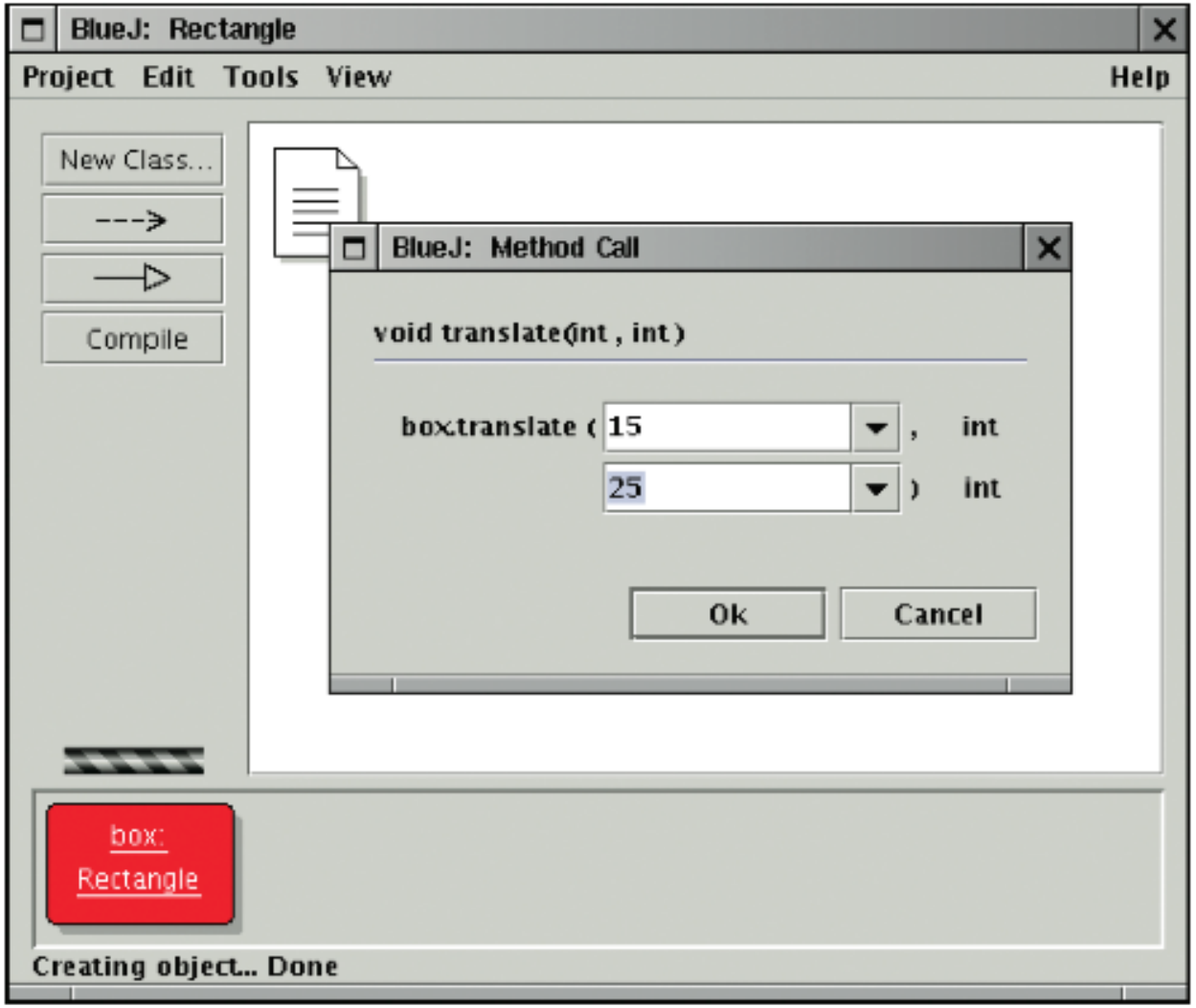
Self Check 2.23

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

Self Check 2.24

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

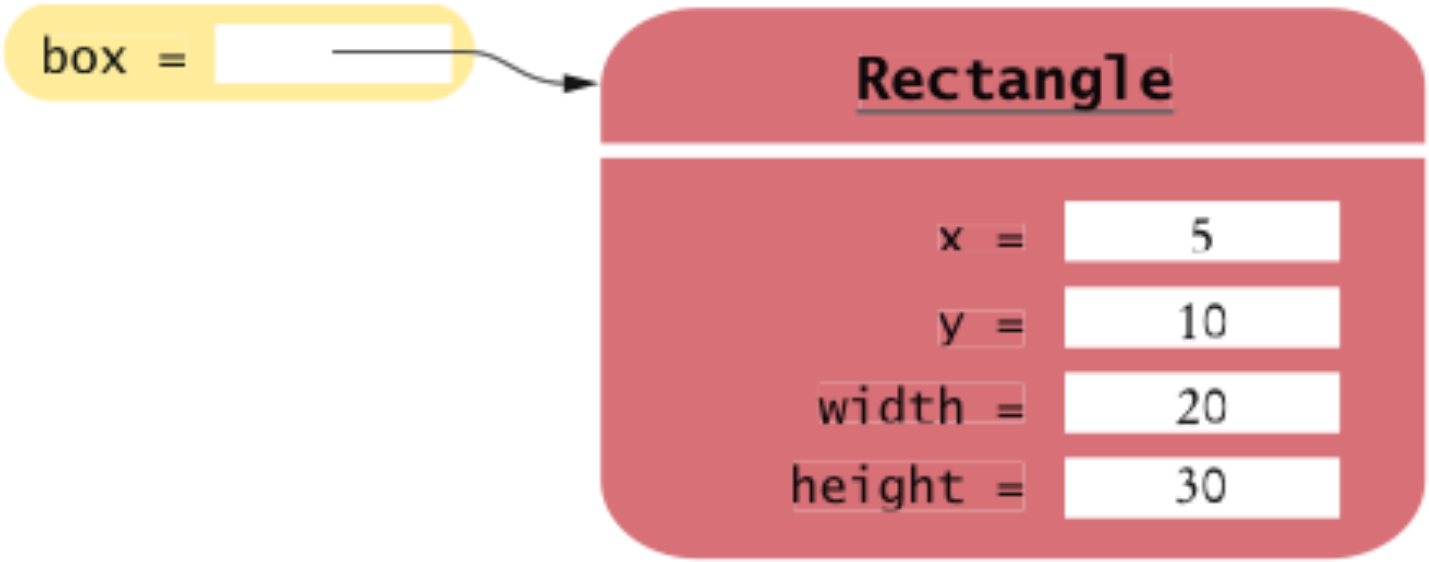


Figure 17 An Object Variable Containing an Object Reference

Object Variables and Number Variables

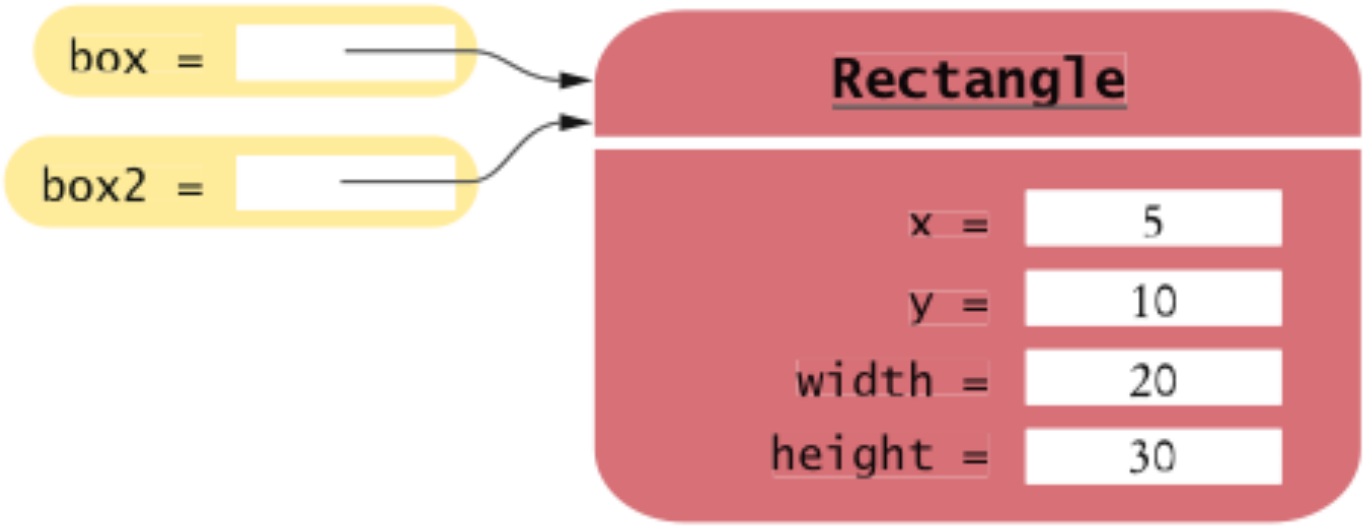


Figure 18 Two Object Variables Referring to the Same Object

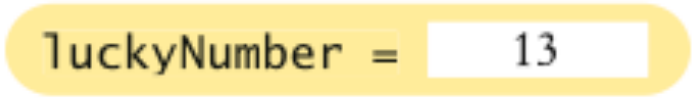
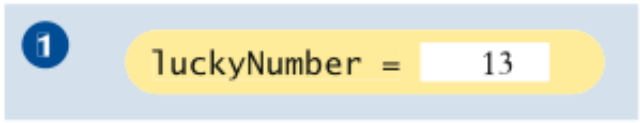


Figure 19 A Number Variable Stores a Number

Copying Numbers

```
int luckyNumber = 13;
```
1

Figure 20
Copying Numbers

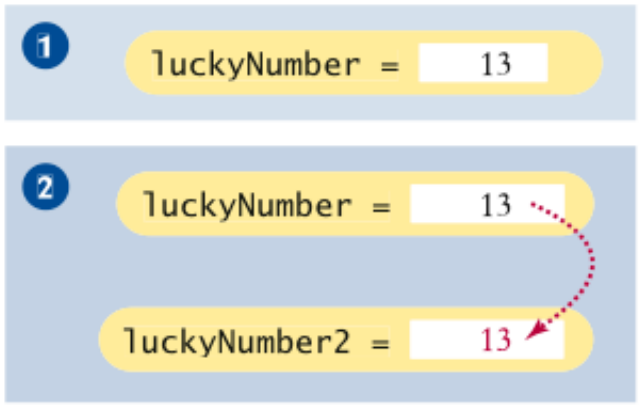


Copying Numbers (cont.)

```
int luckyNumber = 13;
```
1

```
int luckyNumber2 = luckyNumber;
```
2

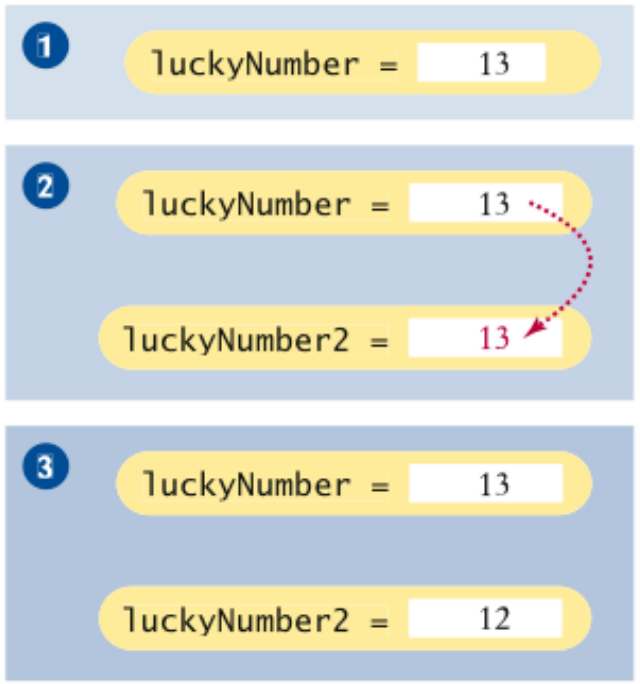
Figure 20
Copying Numbers



Copying Numbers (cont.)

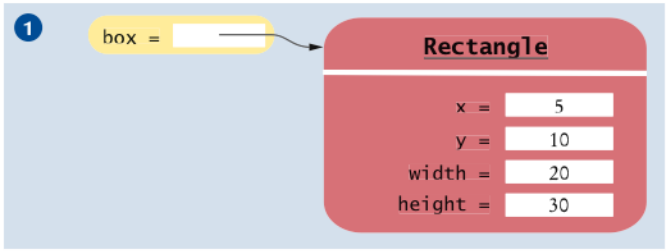
```
int luckyNumber = 13; 1
int luckyNumber2 = luckyNumber; 2
luckyNumber2 = 12;3
```

Figure 20
Copying Numbers



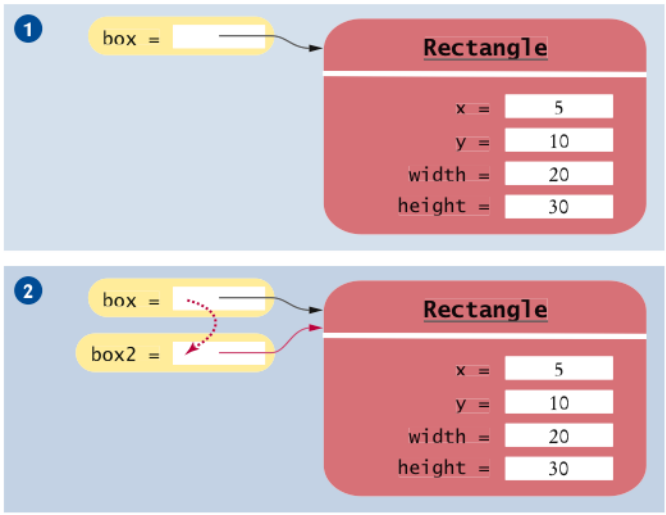
Copying Object References

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



Copying Object References (cont.)

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



Copying Object References (cont.)

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

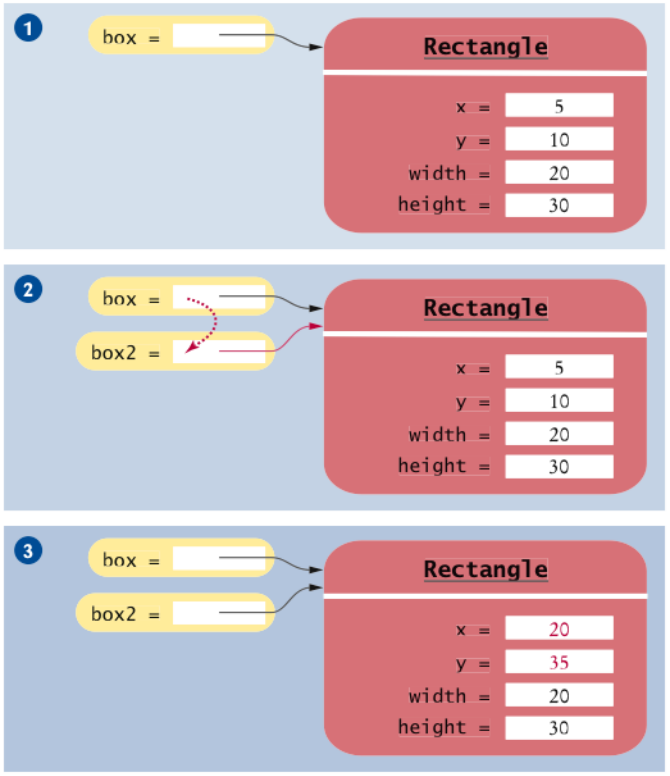


Figure 21 Copying Object References

Self Check 2.25

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

Self Check 2.26

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

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

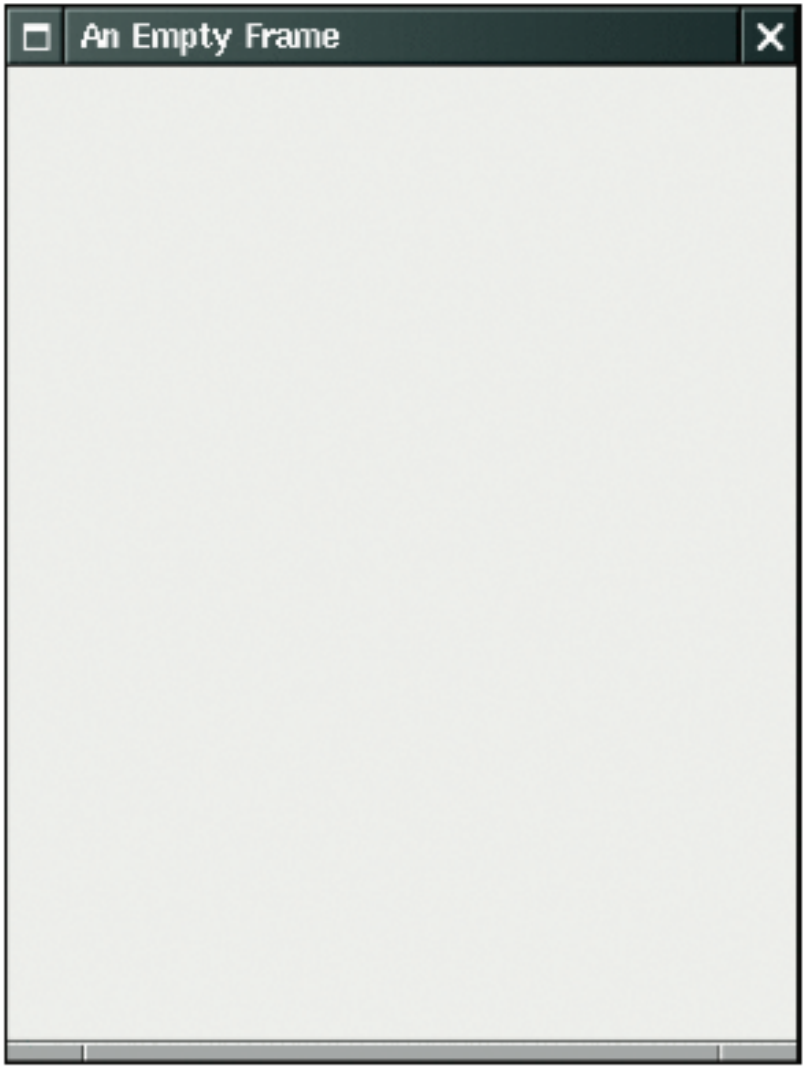


Figure 22
A Frame Window

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!”`?

Self Check 2.28

How can a program display two frames at once?

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

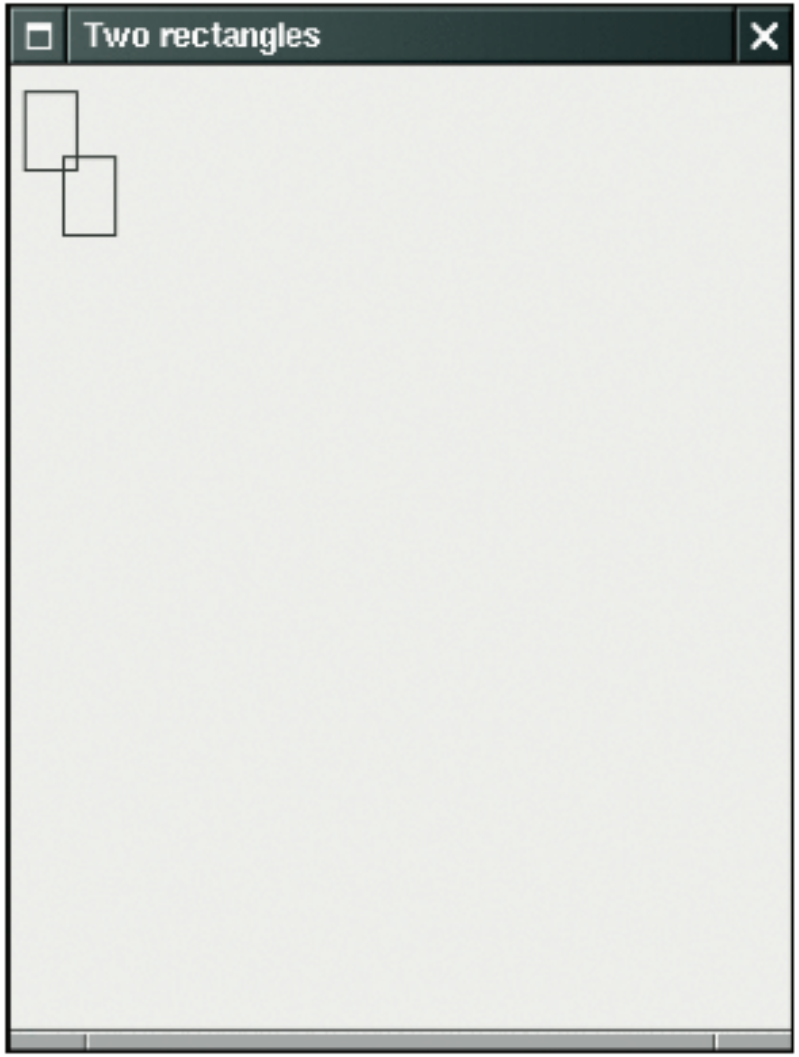


Figure 23
Drawing Rectangles

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?

Self Check 2.30

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

Self Check 2.31

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

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
```

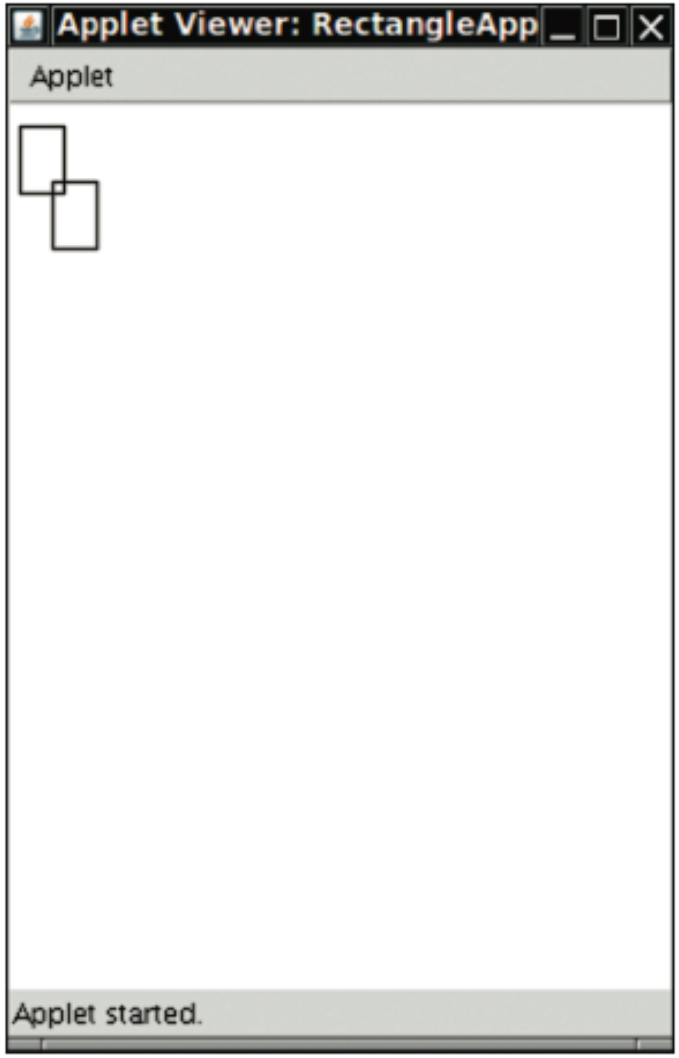
ch02/applet/RectangleApplet.html

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

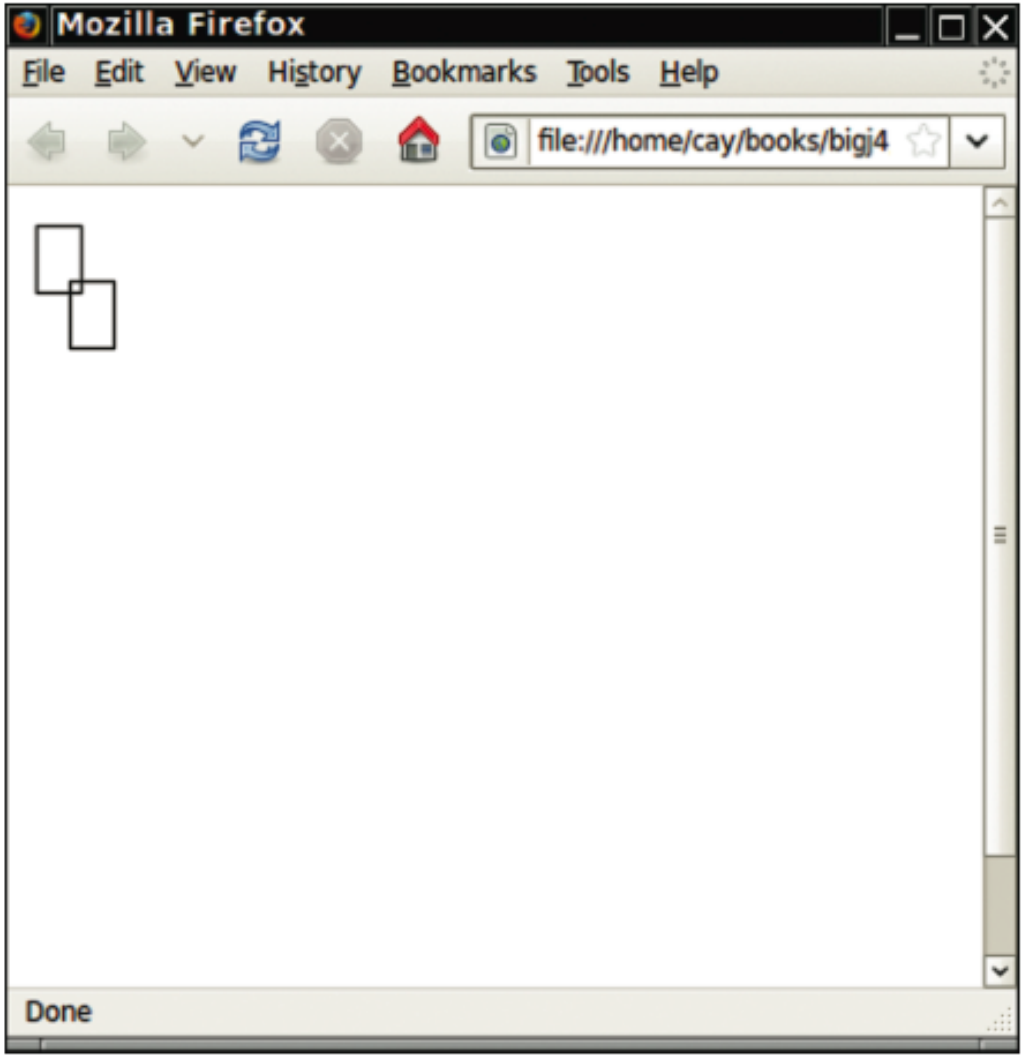
ch02/applet/RectangleAppletExplained.html

```
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

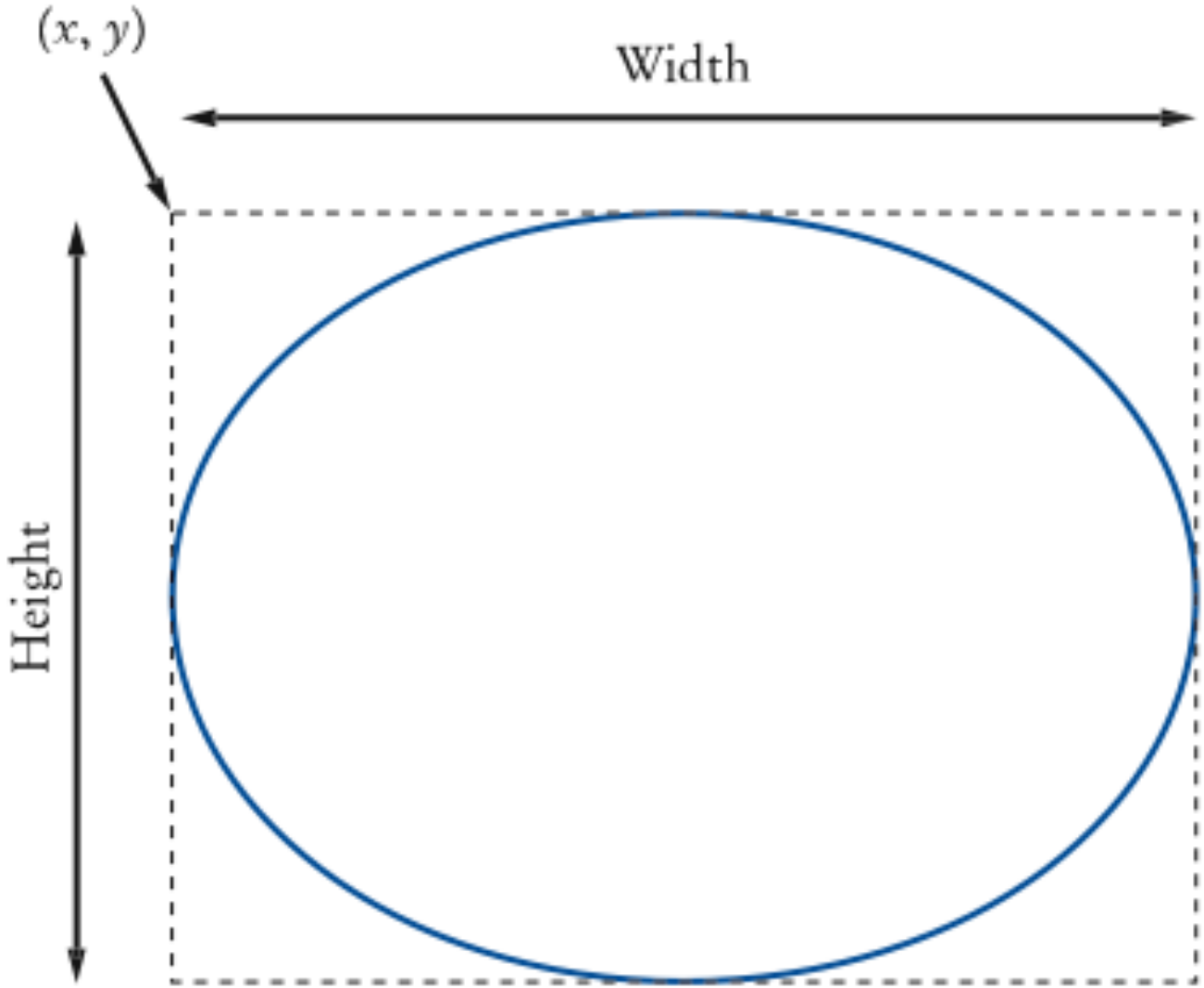


Figure 24 An Ellipse and Its Bounding Box

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:

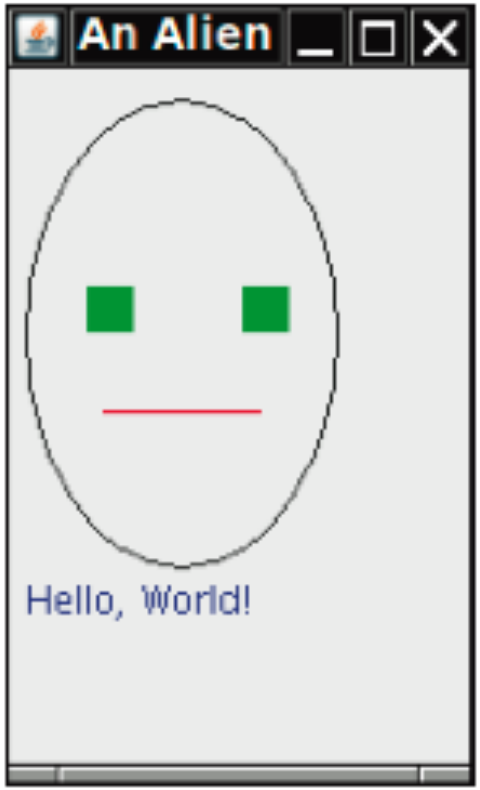
```
g2.fill(rectangle); // filled with current color
```

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.

Self Check 2.33

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

Self Check 2.34

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

Self Check 2.35

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

Self Check 2.36

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