

# ICOM 4015: Advanced Programming

## Lecture 2

**Reading: Chapter Two: Using Objects**



## Chapter Two: Using Objects

ICOM 4015 Fall 2008

*Big Java* by Cay Horstmann  
Copyright © 2008 by John Wiley & Sons. All rights reserved.

## 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 implement test programs
- To be able to browse the API documentation
- To realize the difference between objects and object references
- To write programs that display simple shapes

# Types and Variables

---

- Every value has a type
- Variable declaration examples:

```
String greeting = "Hello, World!";  
PrintStream printer = System.out;  
int luckyNumber = 13;
```
- Variables
  - *Store values*
  - *Can be used in place of the objects they store*

## Syntax 2.1 Variable Definition

```
typeName variableName = value;
```

or

```
typeName variableName;
```

### Example:

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

### Purpose:

To define a new variable of a particular type and optionally supply an initial value.

# Identifiers

---

- Identifier: name of a variable, method, or class
- Rules for identifiers in Java:
  - *Can be made up of letters, digits, and the underscore ( \_ ) character*
  - *Cannot start with a digit*
  - *Cannot use other symbols such as ? or %*
  - *Spaces are not permitted inside identifiers*
  - *You cannot use reserved words*
  - *They are case sensitive*
- By convention, variable names start with a lowercase letter
- By convention, class names start with an uppercase letter

## Self Check 2.1

---

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

**Answer:** `int` and `String`.

## Self Check 2.2

---

Which of the following are legal identifiers?

Greeting1

g

void

101dalmatians

Hello, World

<greeting>

**Answer:** Only the first two are legal identifiers.



## Self Check 2.3

---

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: =
- Not used as a statement about equality
- Used to change the value of a variable

# The Assignment Operator

---

```
int luckyNumber = 13; ①
```

①

luckyNumber = 13

# The Assignment Operator

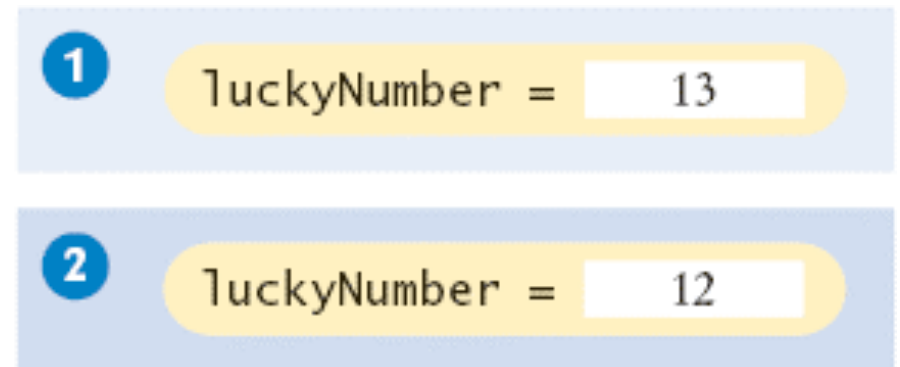
---

```
int luckyNumber = 13; ①
```

```
luckyNumber = 12; ②
```

## Figure 1

Assigning a New Value to a Variable



# Uninitialized Variables

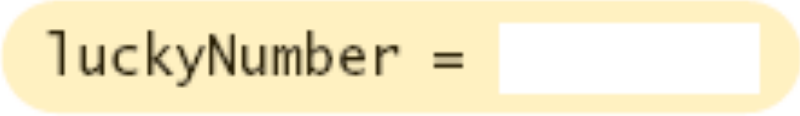
---

- Error:

```
Int luckyNumber;  
  
System.out.println(luckyNumber);  
  
// ERROR - uninitialized variable
```

## Figure 2

An Uninitialized Object Variable



luckyNumber =

## Syntax 2.2 Assignment

---

```
variableName = value;
```

### **Example:**

```
luckyNumber = 12;
```

### **Purpose:**

To assign a new value to a previously defined variable.

## Animation 2.1

---

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

This animation demonstrates the process of variable initialization and assignment.

2-01 variable Initialization and Assignment



## Self Check 2.4

---

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 2.5

---

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

### Answer:

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

### Note that

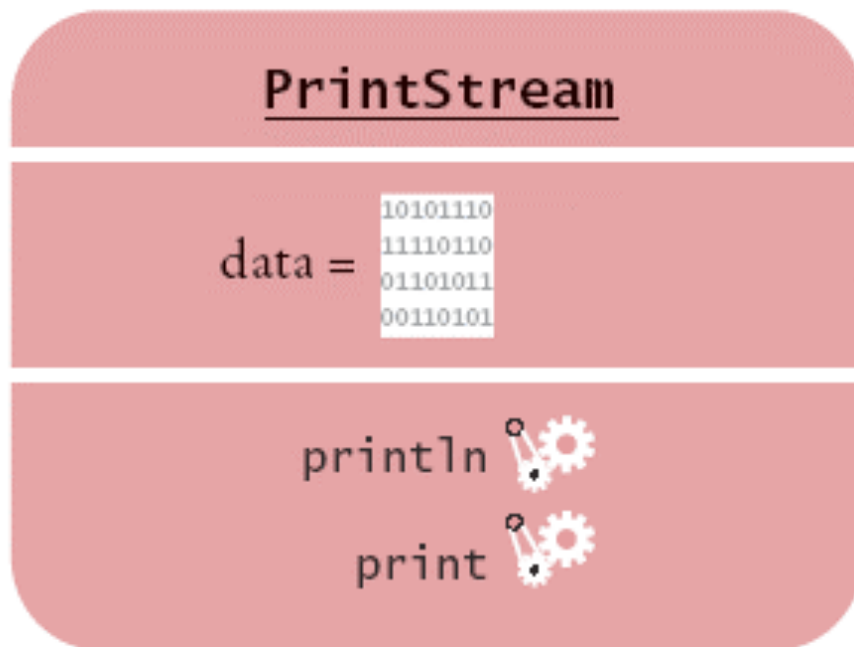
```
String greeting = "Hello, Nina!";
```

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. For example, `System.out` belongs to the class `PrintStream`



**Figure 3** Representation of the `System.out` Object

## Methods

---

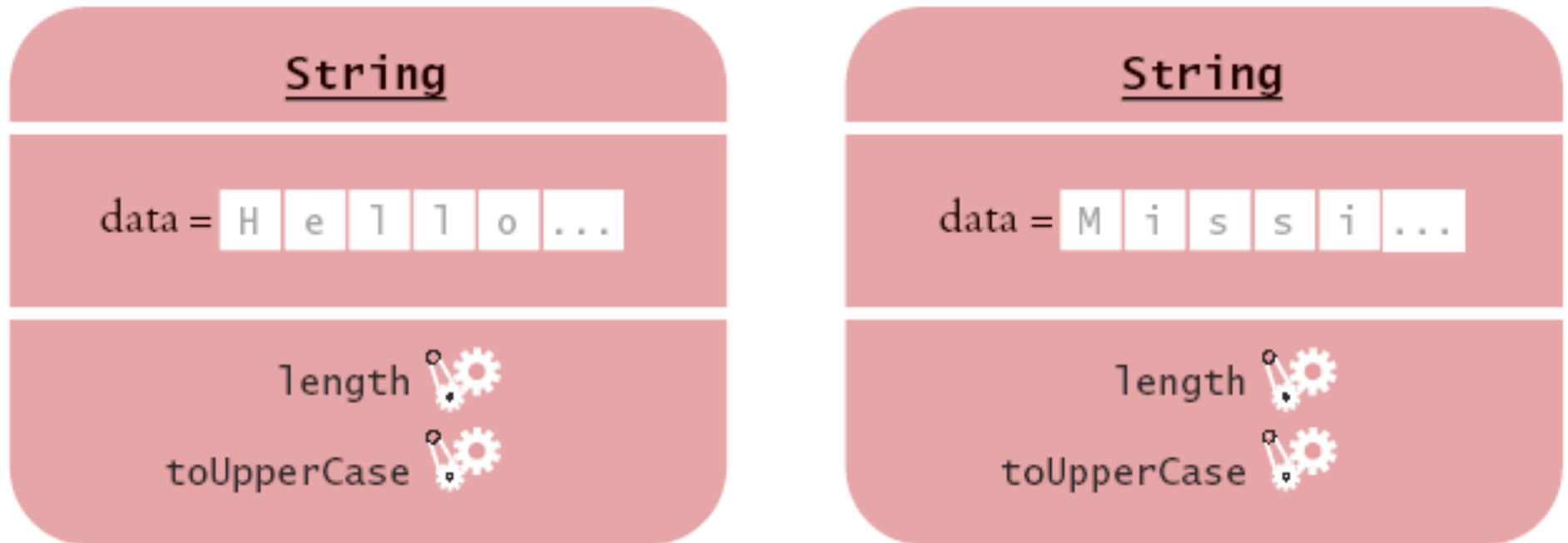
- Method: Sequence of instructions that accesses the data of an object
- You manipulate objects by calling its methods
- Class: Set of objects with the same behavior
- Class determines legal methods

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

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

# A Representation of Two String Objects

---



**Figure 4** 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 2.6

---

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

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

## Self Check 2.7

---

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

**Answer:** `System.out.println(greeting.toUpperCase());`

## Self Check 2.8

---

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

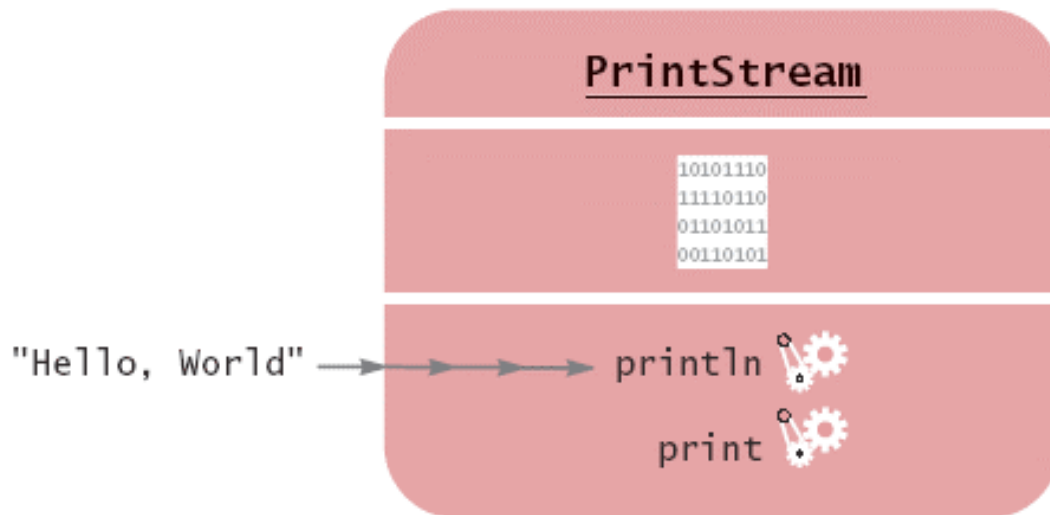
**Answer:** It is not legal. The variable `river` has type `String`. The `println` method is not a method of the `String` class.



# Implicit and Explicit Parameters

- Parameter (explicit parameter): Input to a method. Not all methods have explicit parameters.

```
System.out.println(greeting)  
greeting.length() // has no explicit parameter
```



**Figure 5** Passing a Parameter to the println Method

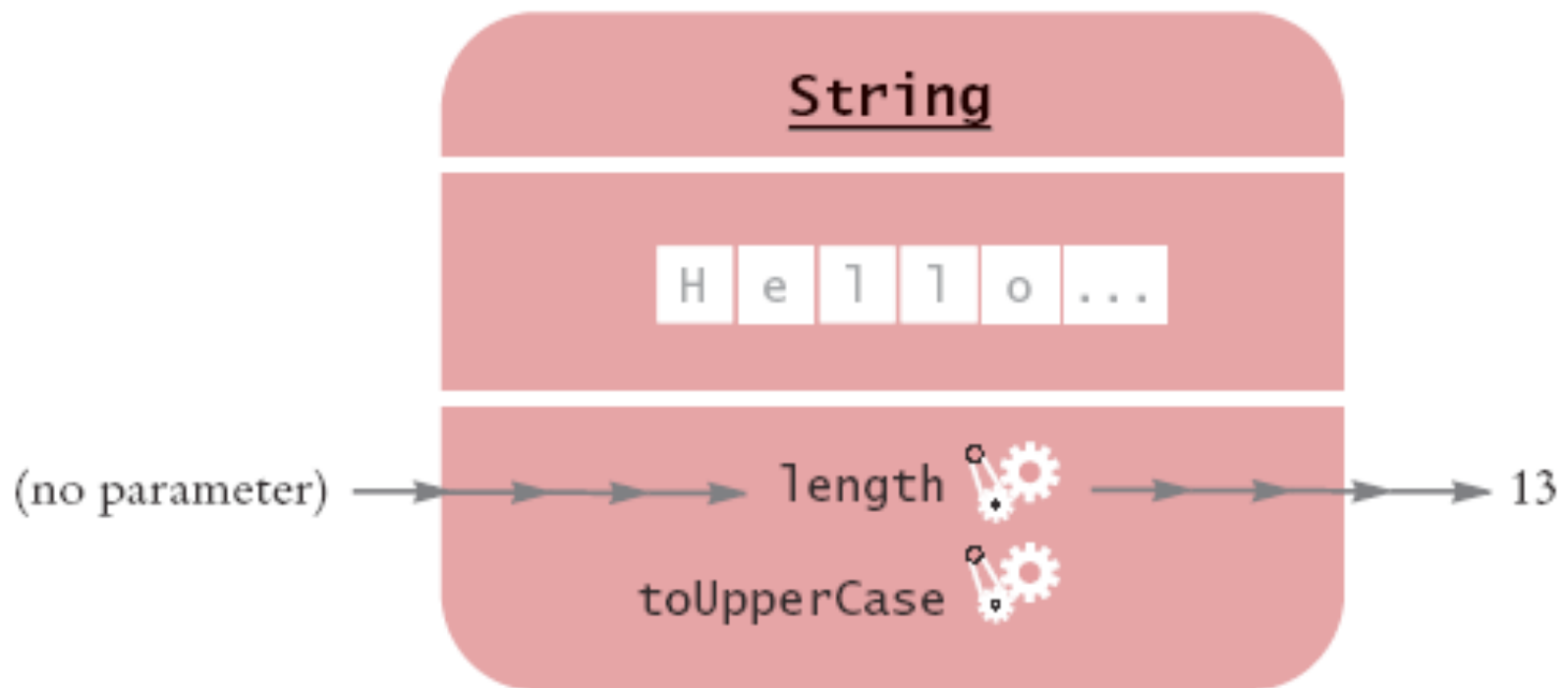
- Implicit parameter: The object on which a method is invoked

```
system.out.println(greeting)
```

## 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 6**  
Invoking the Length Method on a String Object

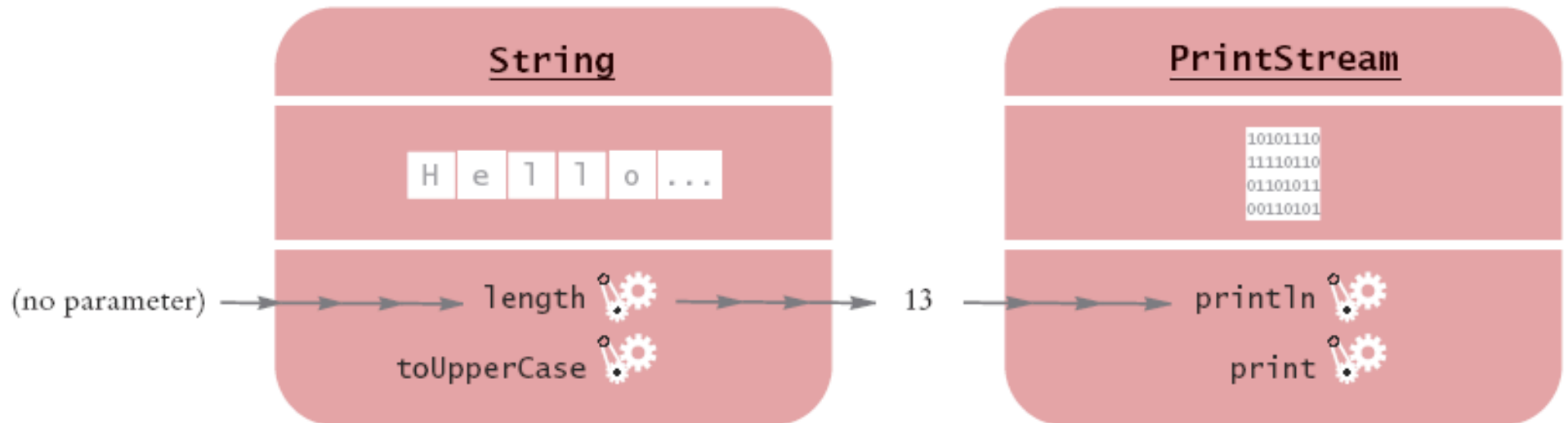
Big Java by Cay Horstmann

Copyright © 2008 by John Wiley & Sons. All rights reserved.

## Passing Return Values

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

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



**Figure 7** Passing the Result of a Method Call to Another Method

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

ICOM 4015 Fall 2008

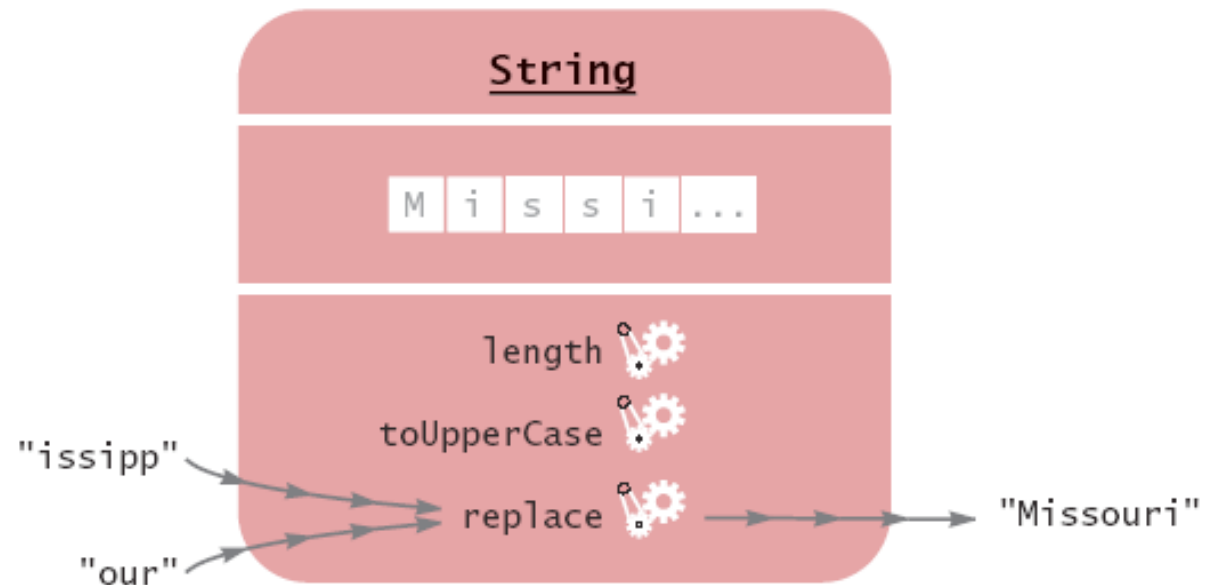
# Animation 2.2

---

## A More Complex Call

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

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



**Figure 8** 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"*

## Method Definitions

---

- Method definition specifies types of explicit parameters and return value
- Type of implicit parameter = current class; not mentioned in method definition
- Example: Class `String` defines

```
public int length()  
    // return type: int  
    // no explicit parameter  
  
public String replace(String target, String replacement)  
    // return type: String;  
    // two explicit parameters of type String
```

## Method Definitions

---

- If method returns no value, the return type is declared as `void`

```
public void println(String output) // in class  
    PrintStream
```

- A method name is overloaded if a class has more than one method with the same name (but different parameter types)

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

## Self Check 2.9

---

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 2.10

---

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

**Answer:** `"Missississi"`.

## Self Check 2.11

---

What is the result of the call `greeting.replace("World", "Dave").length()`?

**Answer:** 12.

## Self Check 2.12

---

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

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

# Number Types

---

- **Integers:** `short`, `int`, `long`  
13

- **Floating point numbers:** `float`, `double`  
1.3  
0.00013

- When a floating-point number is multiplied or divided by 10, only the position of the decimal point changes; it "floats". This representation is related to the "scientific" notation

$1.3 \times 10^{-4}$ .

`1.3E-4`            `// 1.3 × 10-4 written in Java`

- Numbers are not objects; numbers types are primitive types

# Arithmetic Operations

---

- Operators: + - \*

10 + n

n - 1

10 \* n // 10 × n

- As in mathematics, the \* operator binds more strongly than the + operator

x + y \* 2 // means the sum of x and y \* 2

(x + y) \* 2 // multiplies the sum of x and y with 2

## Self Check 2.13

---

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

**Answer:** `double`.

## Self Check 2.14

---

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 2.15

---

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

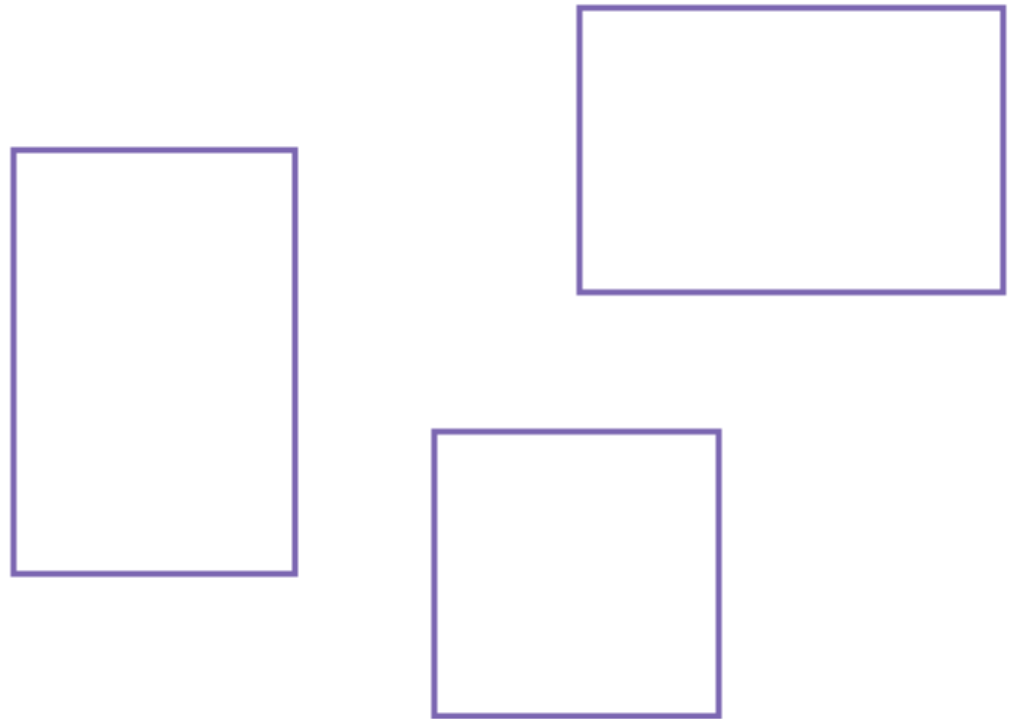
**Answer:**  $(x + y) * 0.5$



# Rectangular Shapes and Rectangle Objects

---

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

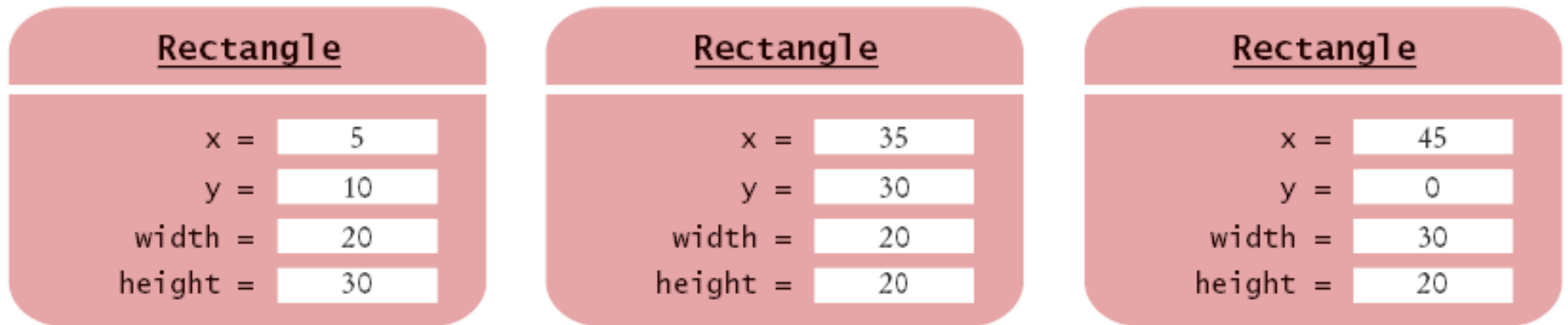


**Figure 9**  
Rectangular Shapes

## 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 10**

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

---

- The process of creating a new object is called *construction*
- 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

---

```
new ClassName(parameters)
```

### **Example:**

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

### **Purpose:**

To construct a new object, initialize it with the construction parameters, and return a reference to the constructed object.

## 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

---

What does the following statement print?

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

**Answer:**

0

## Accessor and Mutator Methods

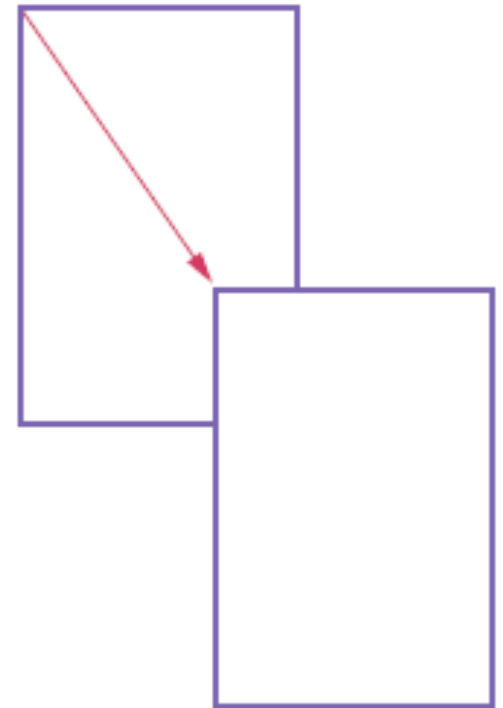
---

- 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 11**

Using the translate Method to Move a Rectangle

*Big Java* by Cay Horstmann

Copyright © 2008 by John Wiley & Sons. All rights reserved.



## 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`.

## 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

---

```
01: import java.awt.Rectangle;
02:
03: public class MoveTester
04: {
05:     public static void main(String[] args)
06:     {
07:         Rectangle box = new Rectangle(5, 10, 20, 30);
08:
09:         // 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: }
```

## ch02/rectangle/MoveTester.java (cont.)

---

### Output:

x: 20

Expected: 20

y: 35

Expected: 35

## Importing Packages

---

- Don't forget to include appropriate packages:
- Java classes are grouped into packages
- 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

---

```
import packageName.ClassName;
```

### **Example:**

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

### **Purpose:**

To import a class from a package for use in a program.

## Self Check 2.20

---

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 2.21

---

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.

## Self Check 2.22

---

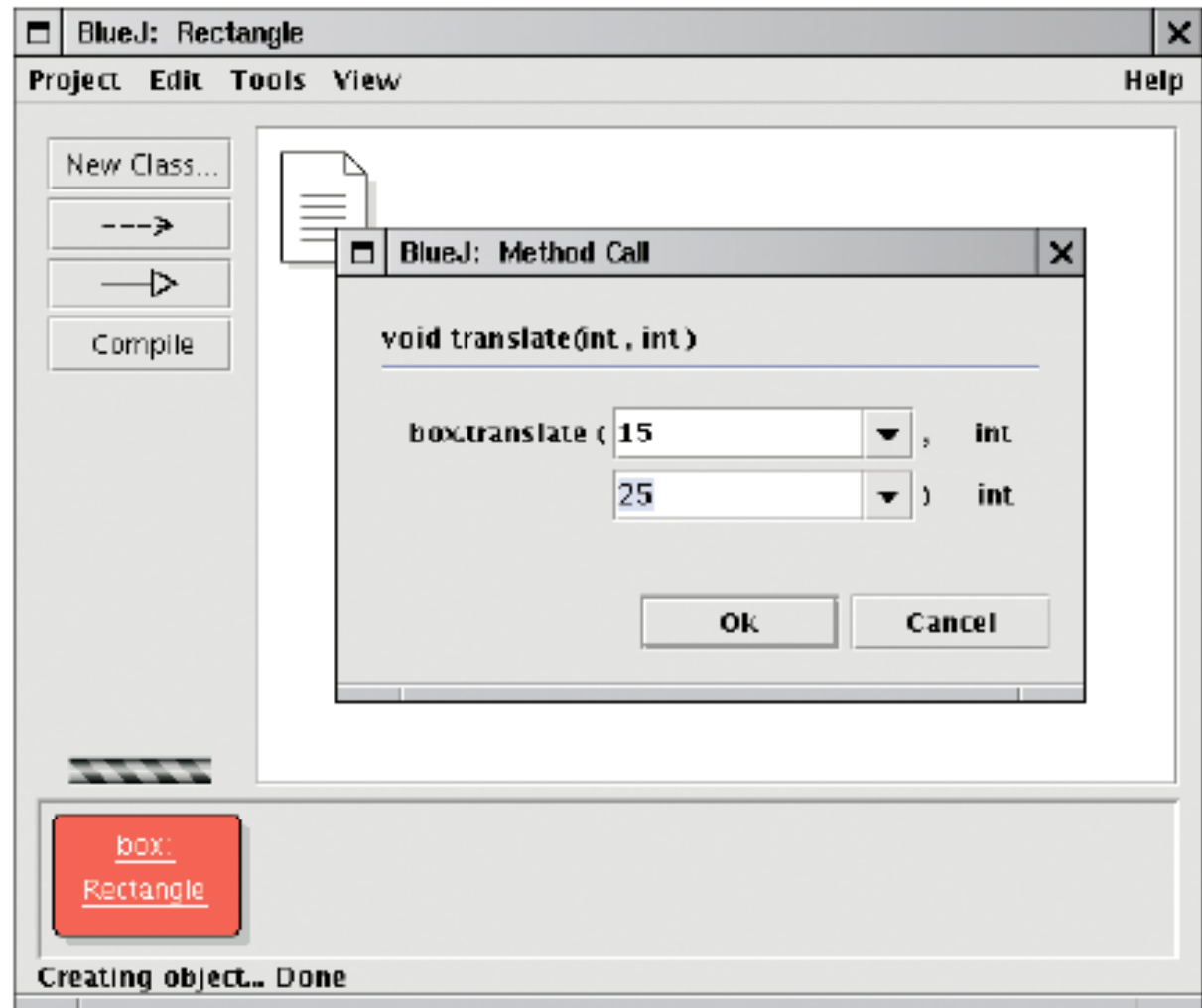
The `Random` class is defined in the `java.util` package. What do you need to do in order to use that class in your program?

**Answer:** Add the statement

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

at the top of your program.

# Testing Classes in an Interactive Environment

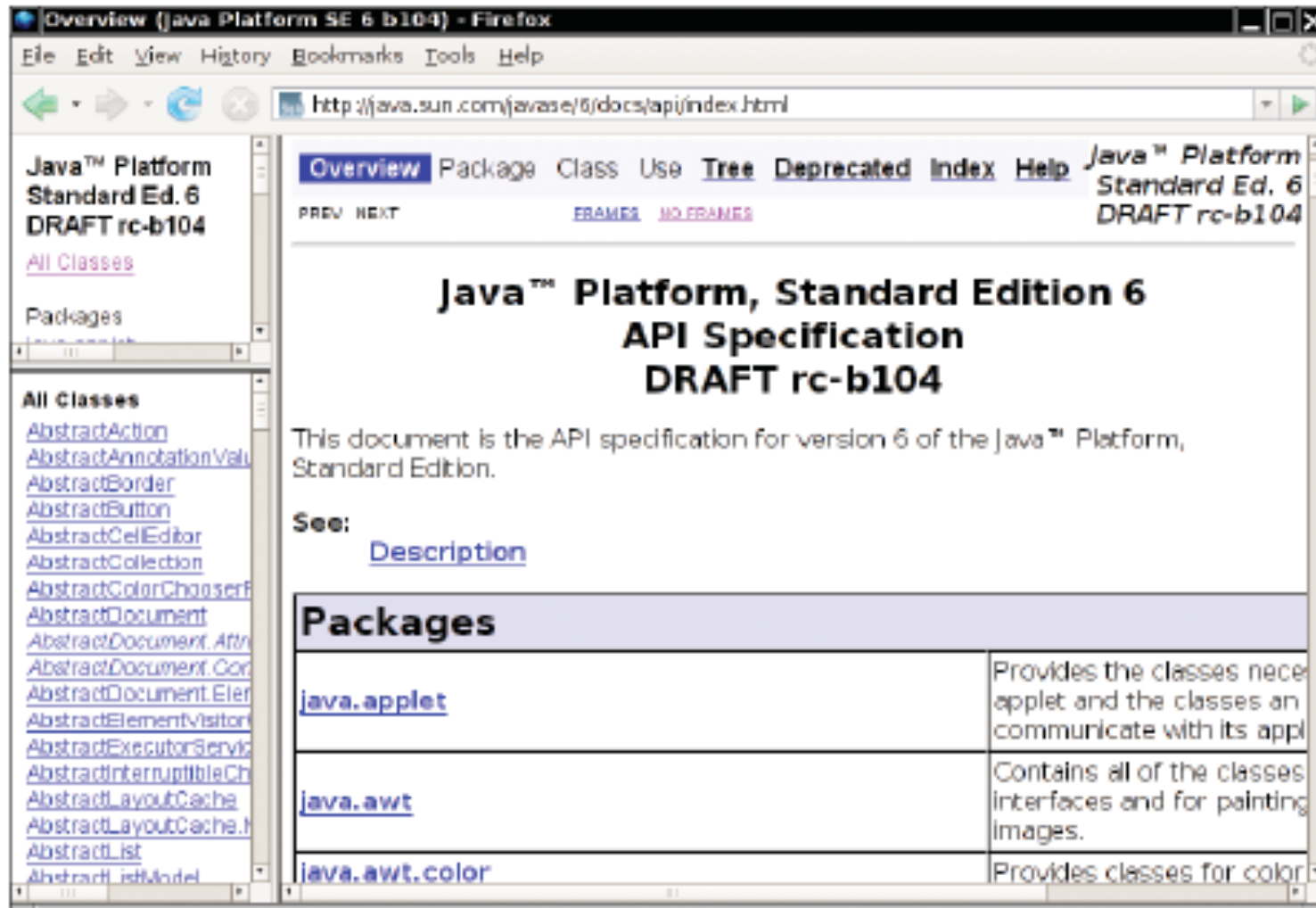


# The API Documentation

---

- API: Application Programming Interface
- Lists classes and methods in the Java library
- <http://java.sun.com/javase/6/docs/api/index.html>


# The API Documentation of the Standard Java Library=



**Figure 12** The API Documentation of the Standard Java Library

TCOM 4013 Fall 2008

# The API Documentation for the Rectangle Class

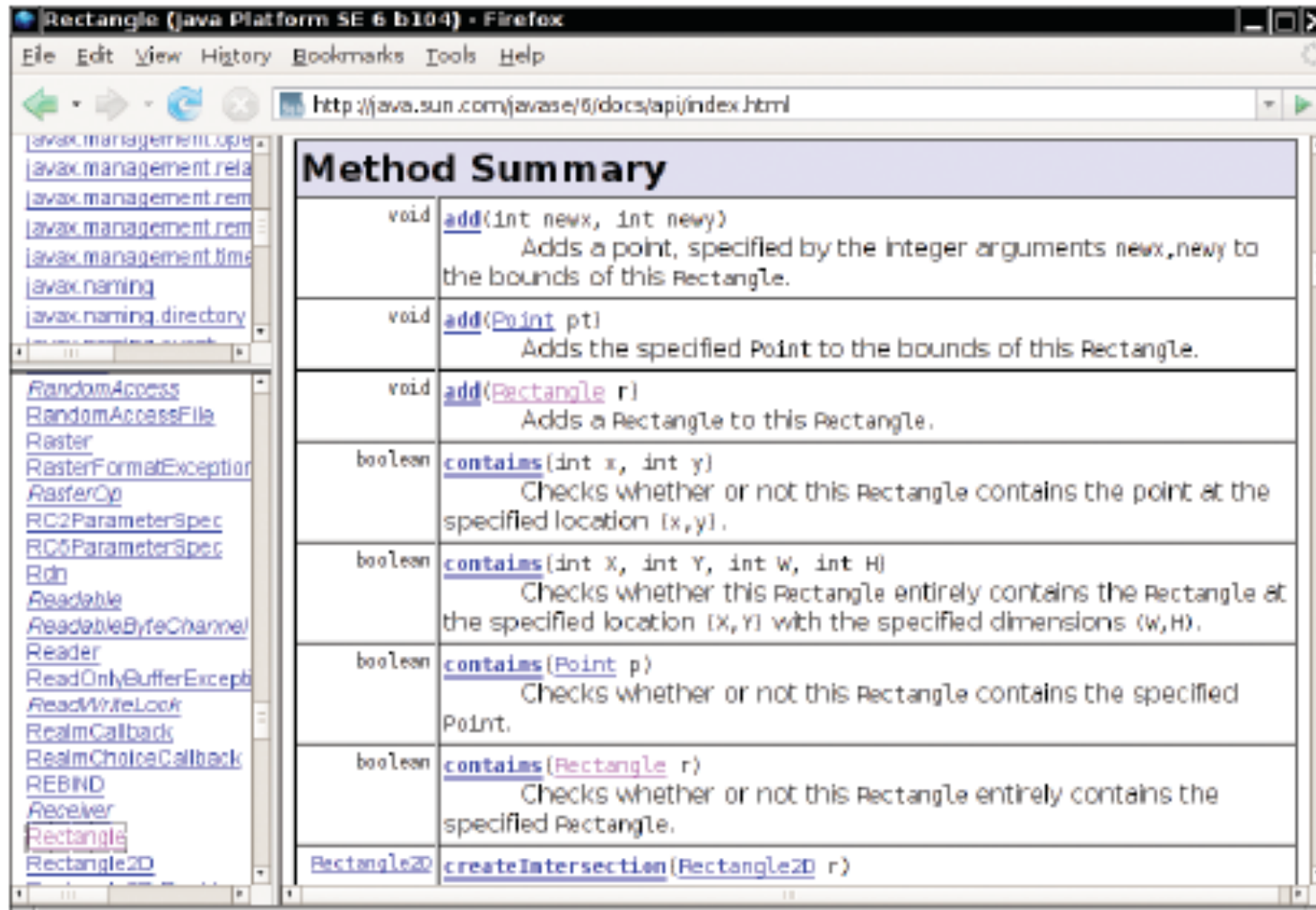


The screenshot shows a Firefox browser window displaying the Java API documentation for the `Rectangle` class. The browser's address bar shows the URL `http://java.sun.com/javase/6/docs/api/index.html`. The page title is "Rectangle (Java Platform SE 6 b104) - Firefox". The navigation menu includes "Overview", "Package", "Class", "Use", "Tree", "Deprecated", "Index", and "Help". The "Class" tab is selected, and the "Class" sub-tab is also selected. The page content shows the class hierarchy for `Rectangle`, which extends `Rectangle2D` and implements `Shape` and `Serializable`. The class is defined as `public class Rectangle`. The documentation also lists "All Implemented Interfaces" as `Shape`, `Serializable`, and `Cloneable`, and "Direct Known Subclasses" as `DefaultCaret`. The class description states: "A Rectangle specifies an area in a coordinate space that is enclosed by the Rectangle object's upper-left point (x, y) in the coordinate space, its width, and its height." The browser's left sidebar shows a list of classes, with `Rectangle` and `Rectangle2D` highlighted.

**Figure 13** The API Documentation for the Rectangle Class

© 2008 John Wiley & Sons, Inc.

# Javadoc Method Summary



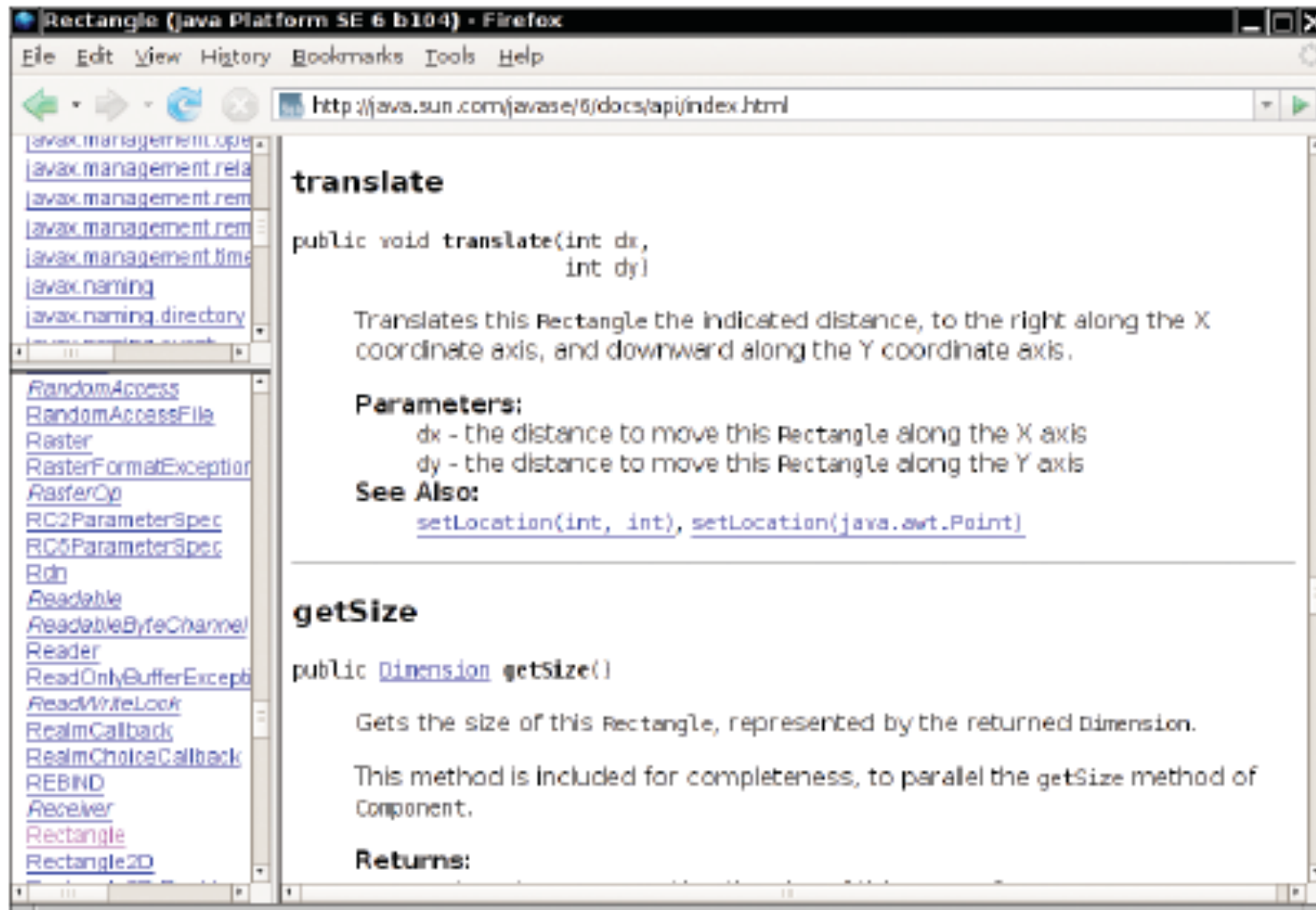
The screenshot shows a Firefox browser window displaying the Javadoc Method Summary for the `Rectangle` class. The browser's address bar shows the URL `http://java.sun.com/javase/6/docs/api/index.html`. The page title is "Method Summary". The left sidebar contains a list of class names, with `Rectangle` selected. The main content area displays the following methods:

Return Type	Method Name	Signature	Description
void	<a href="#">add</a>	<code>(int newX, int newY)</code>	Adds a point, specified by the integer arguments <code>newX</code> , <code>newY</code> to the bounds of this <code>Rectangle</code> .
void	<a href="#">add</a>	<code>(Point pt)</code>	Adds the specified <code>Point</code> to the bounds of this <code>Rectangle</code> .
void	<a href="#">add</a>	<code>(Rectangle r)</code>	Adds a <code>Rectangle</code> to this <code>Rectangle</code> .
boolean	<a href="#">contains</a>	<code>(int x, int y)</code>	Checks whether or not this <code>Rectangle</code> contains the point at the specified location <code>(x,y)</code> .
boolean	<a href="#">contains</a>	<code>(int x, int y, int w, int h)</code>	Checks whether this <code>Rectangle</code> entirely contains the <code>Rectangle</code> at the specified location <code>(x,y)</code> with the specified dimensions <code>(w,h)</code> .
boolean	<a href="#">contains</a>	<code>(Point p)</code>	Checks whether or not this <code>Rectangle</code> contains the specified <code>Point</code> .
boolean	<a href="#">contains</a>	<code>(Rectangle r)</code>	Checks whether or not this <code>Rectangle</code> entirely contains the specified <code>Rectangle</code> .
<code>Rectangle2D</code>	<a href="#">createIntersection</a>	<code>(Rectangle2D r)</code>	

**Figure 14** The Method Summary for the `Rectangle` Class

© 2008 John Wiley & Sons, Inc.

# translate Method Documentation



**Figure 15** The API Documentation of the `translate` Method

ICOM 4015 Fall 2008



## Self Check 2.23

---

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 2.24

---

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.

# Object References

---

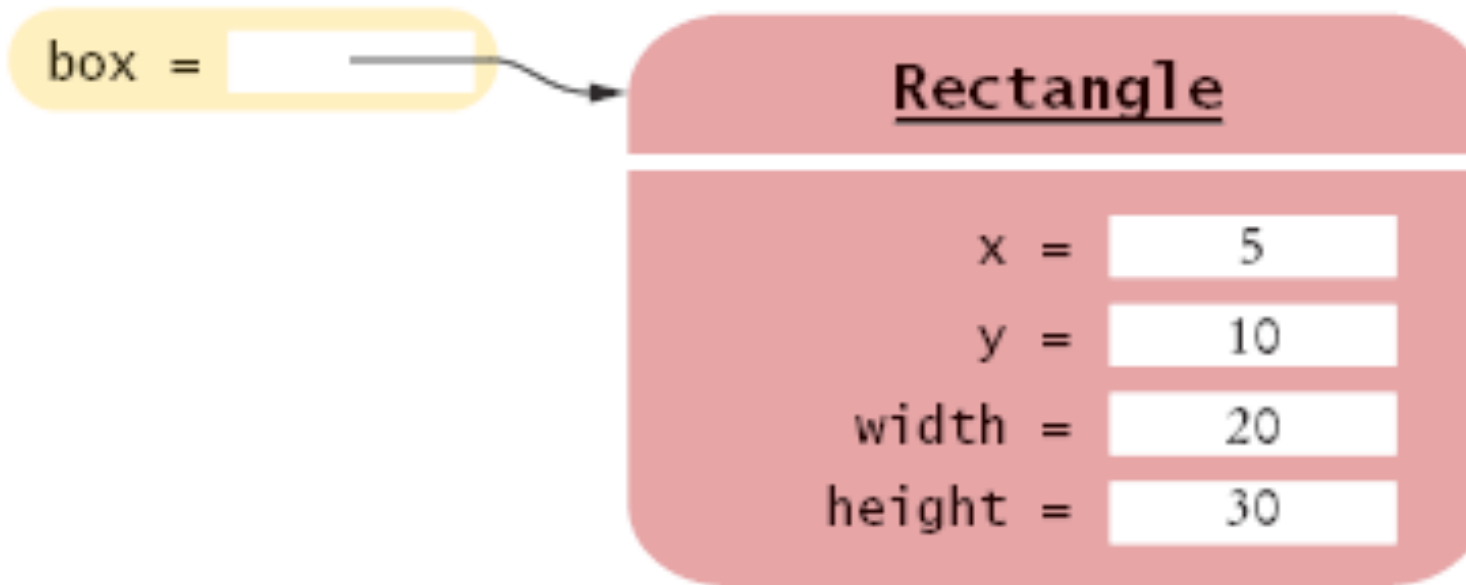
- Object reference describes the location of an object
- The `new` operator returns a reference to a new object  

```
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  $\neq$  object variables

# Object Variables and Number Variables

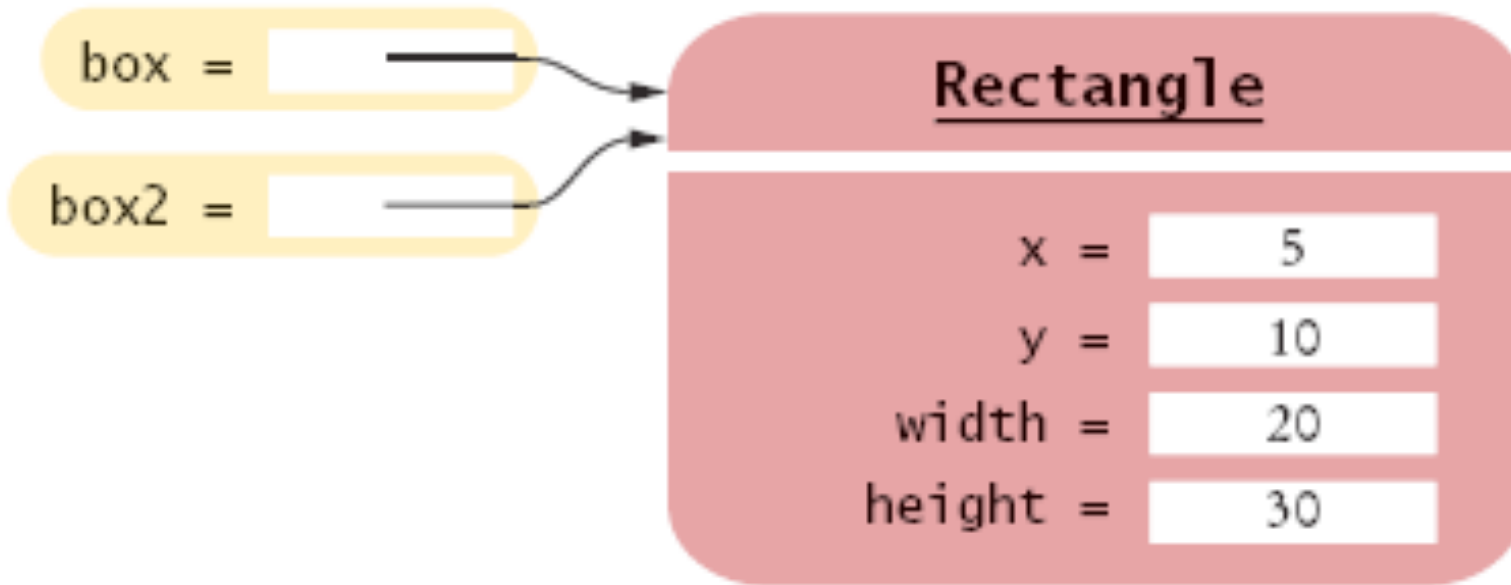
---



**Figure 16** An Object Variable Containing an Object Reference

## Object Variables and Number Variables (Cont.)

---



**Figure 17** Two Object Variables Referring to the Same Object

`LuckyNumber = 13`

**Figure 18** A Number Variable Stores a Number

# Copying Numbers

---

```
int luckyNumber = 13; ①
```

①

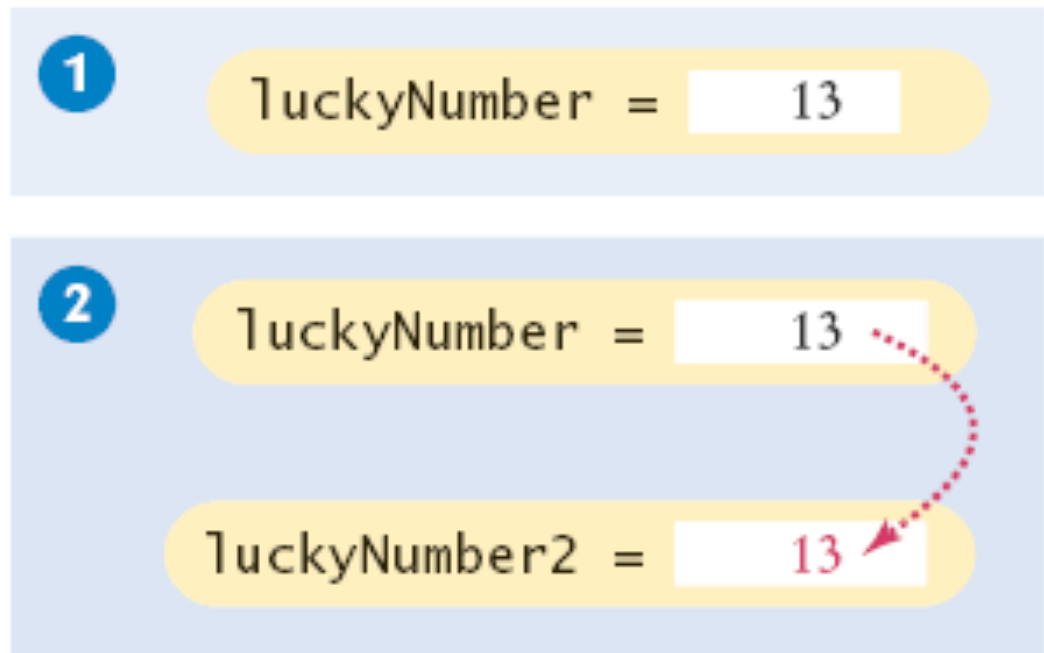
luckyNumber = 13

# Copying Numbers

---

```
int luckyNumber = 13; ①
```

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



# Animation 2.3

---

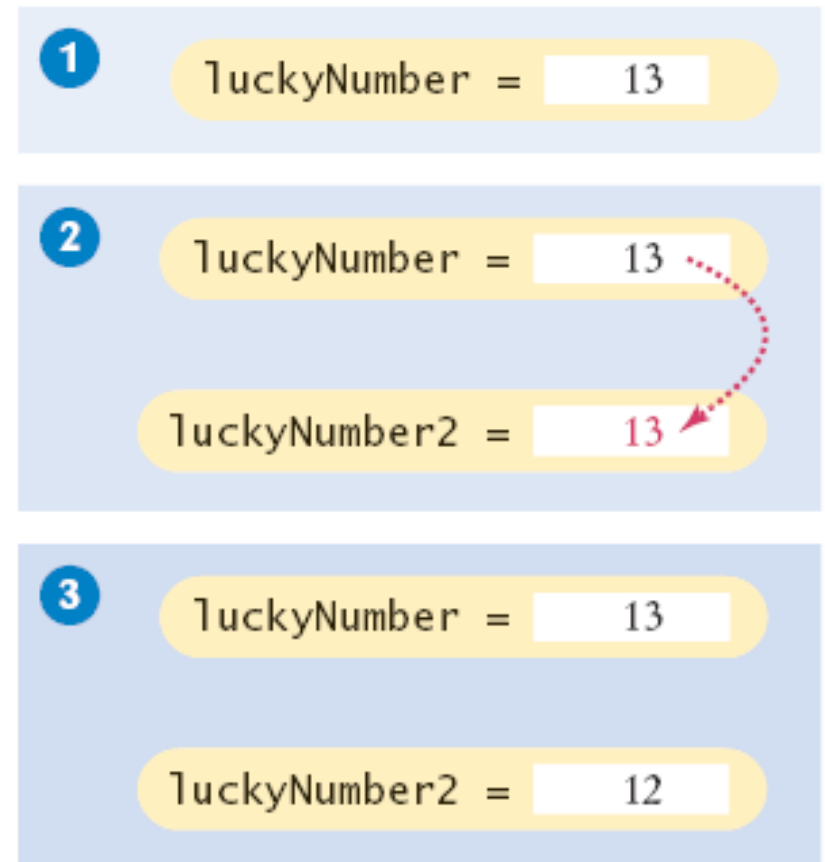


# Copying Numbers

```
int luckyNumber = 13; ①
```

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

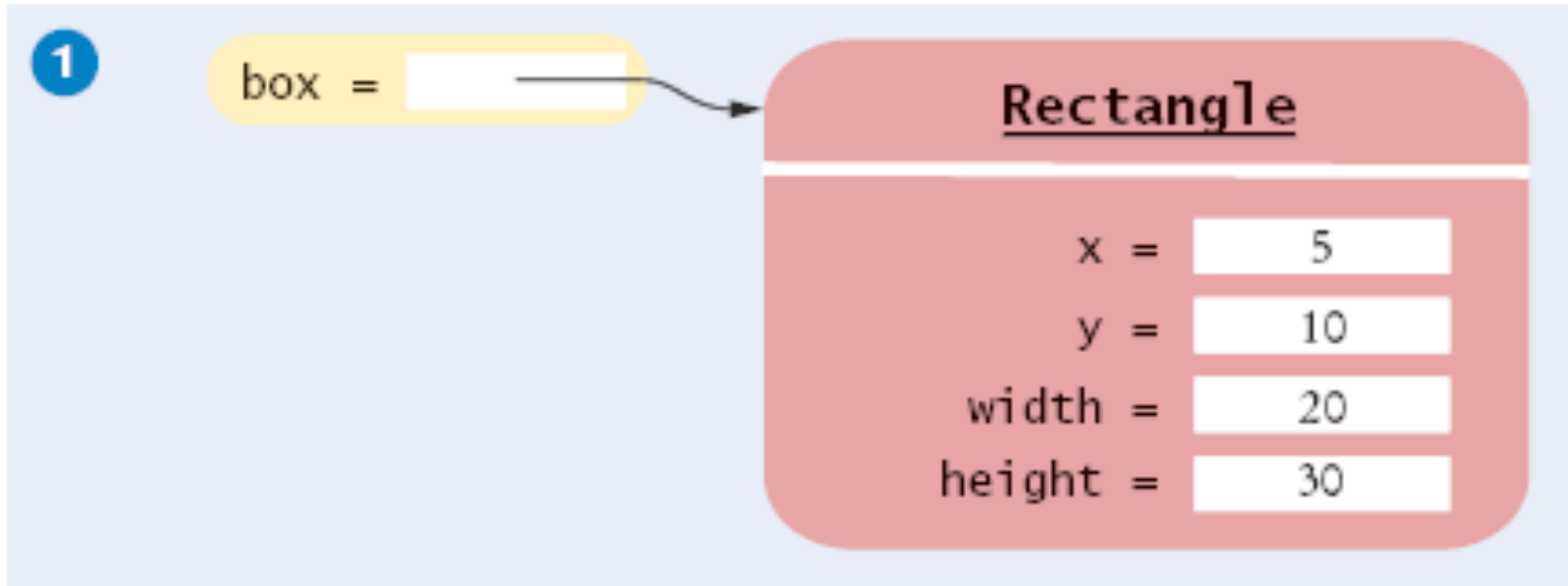
```
luckyNumber2 = 12; ③
```



**Figure 19**  
Copying Numbers

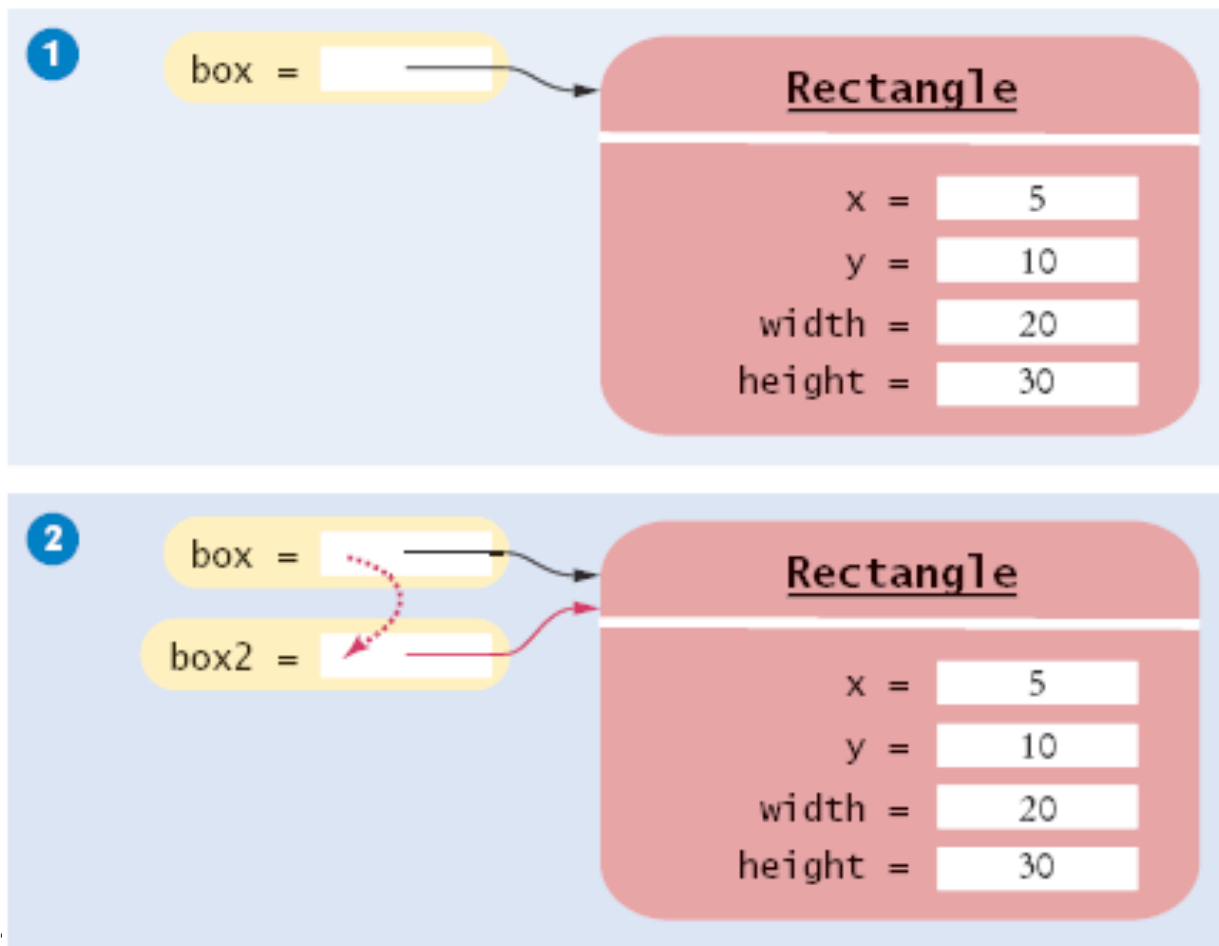
# Copying Object References

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



# Copying Object References

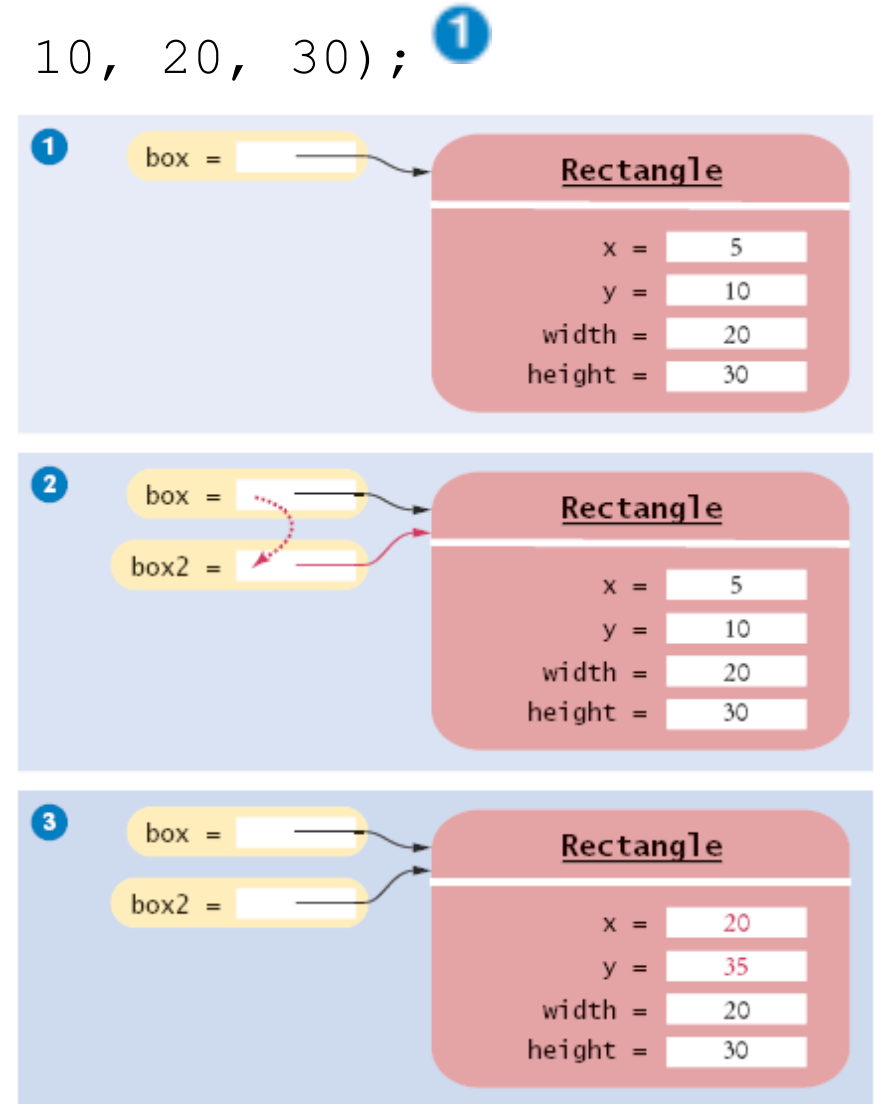
```
Rectangle box = new Rectangle(5, 10, 20, 30);  
Rectangle box2 = box;
```



ICOM .

# Copying Object References

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



## 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);
```

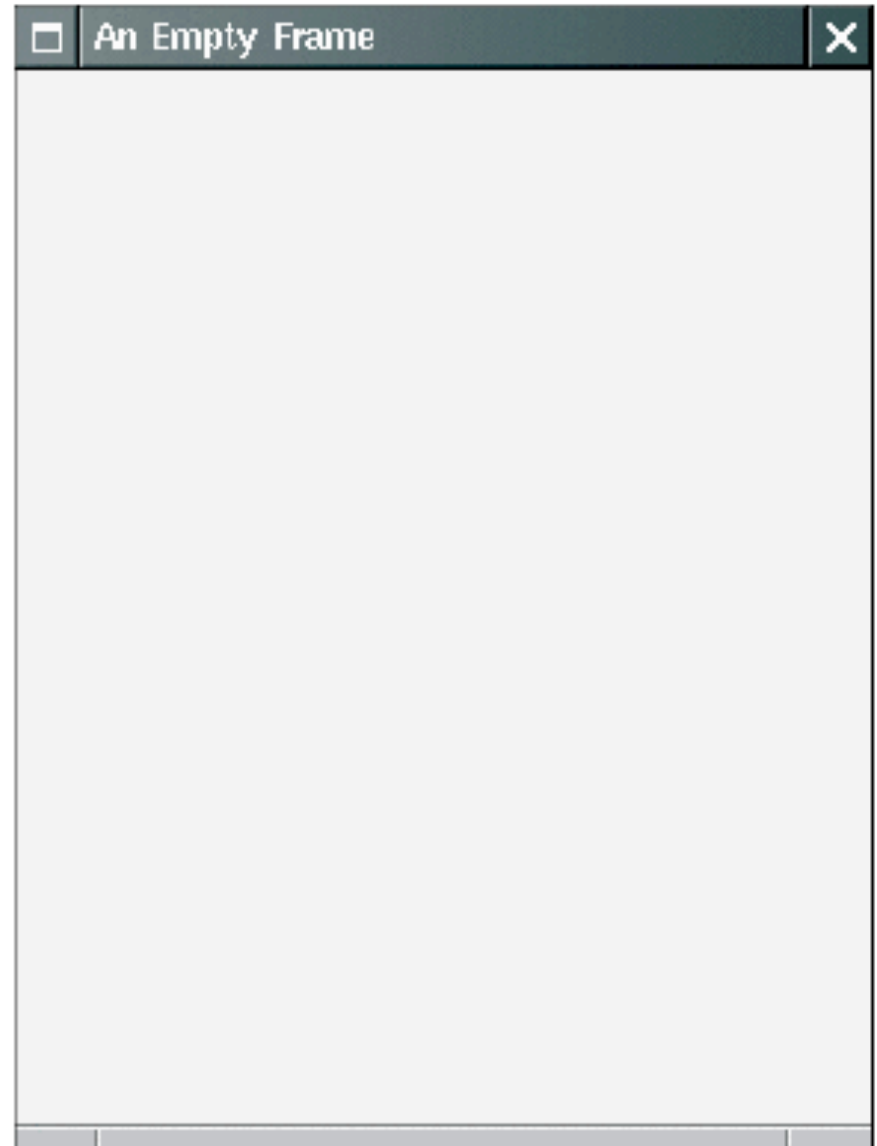
5. Make the frame visible:

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



# A Frame Window

---



IC **Figure 21** A Frame Window

## ch02/emptyframe/EmptyFrameViewer.java

---

```
01: import javax.swing.JFrame;
02:
03: public class EmptyFrameViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:
09:         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;
        . . .
    }
}
```

***Continued***

## Classes Graphics and Graphics2D (cont.)

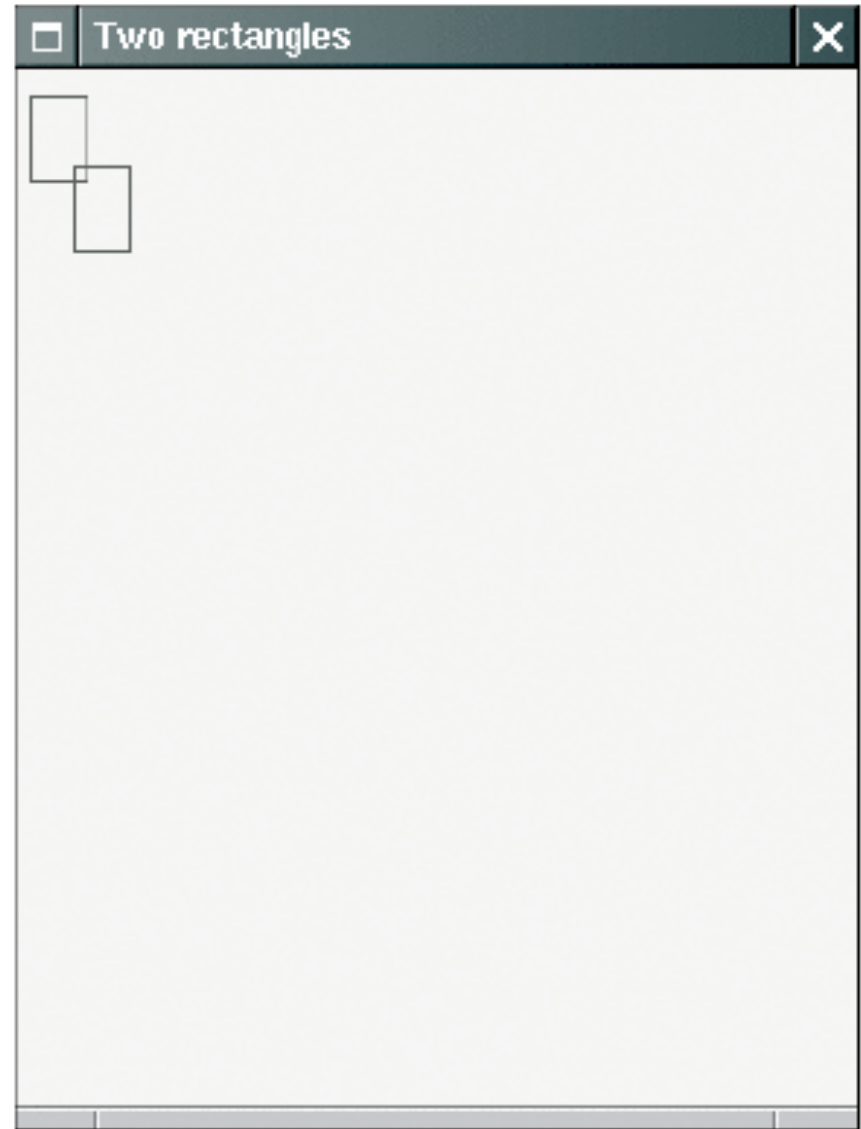
---

- 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 22**

Drawing Rectangles

ICOM 4015 Fall 20



## ch02/rectangles/RectangleComponent.java

---

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JComponent;
05:
06: /**
07:     A component that draws two rectangles.
08: */
09: 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

---

```
01: import javax.swing.JFrame;
02:
03: public class RectangleViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:
09:         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 2.31

---

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

---

- Applets are programs that run 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

---

```
01: import java.awt.Graphics;
02: import java.awt.Graphics2D;
03: import java.awt.Rectangle;
04: import javax.swing.JApplet;
05:
06: /**
07:     An applet that draws two rectangles.
08: */
09: 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

---

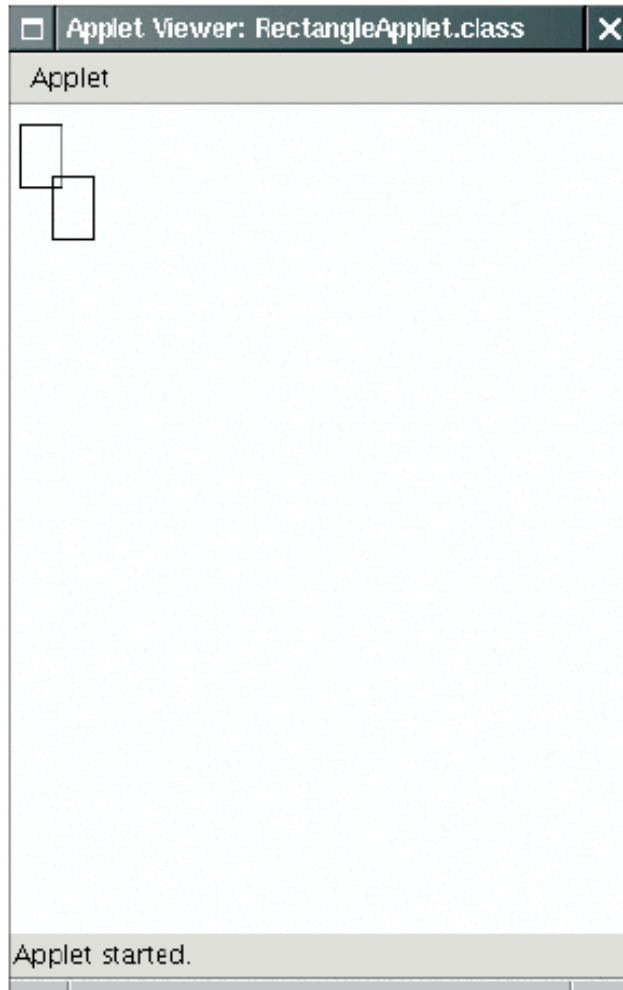
```
1: <applet code="RectangleApplet.class" width="300" height="400">  
2: </applet>
```

## ch02/applet/RectangleAppletExplained.html

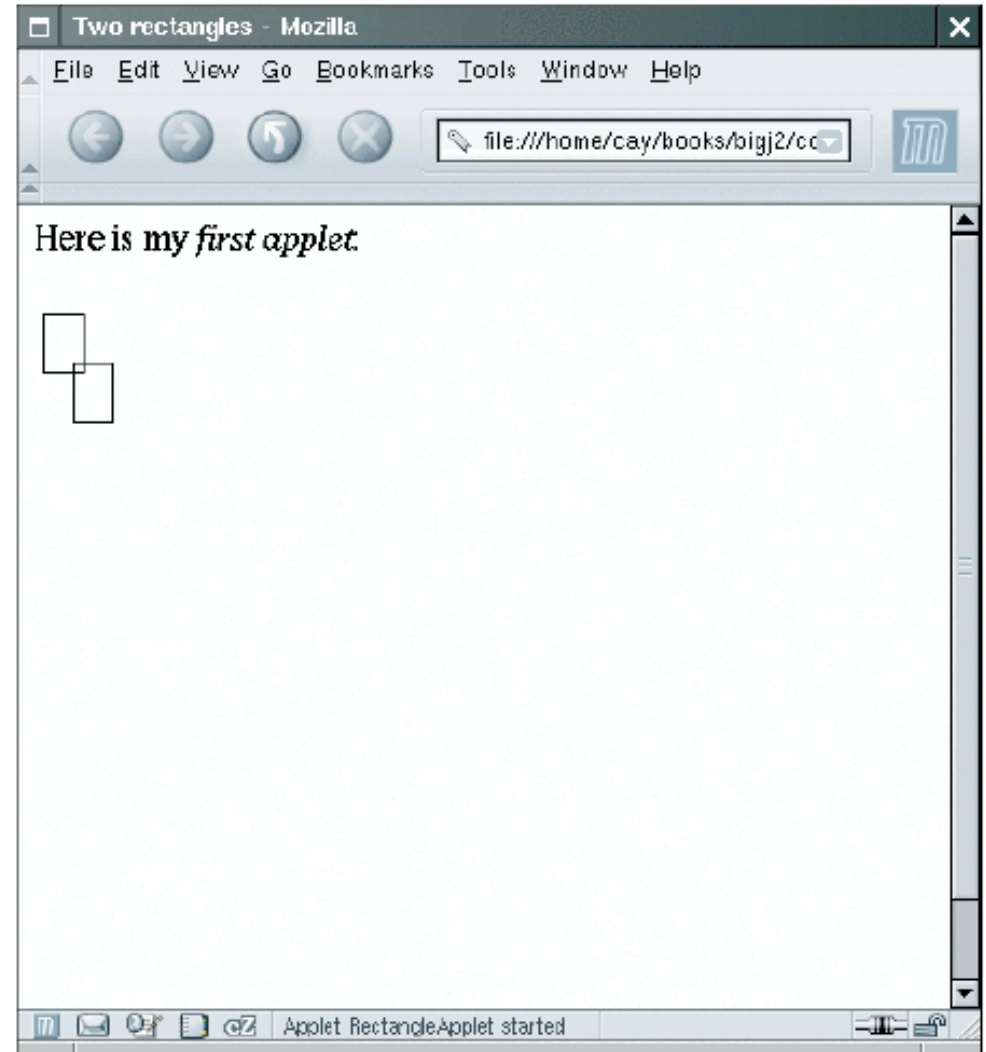
---

```
01: <html>
02:   <head>
03:     <title>Two rectangles</title>
04:   </head>
05:   <body>
06:     <p>Here is my <i>first applet</i>:</p>
07:     <applet code="RectangleApplet.class" width="300" height="400">
08:     </applet>
09:   </body>
10: </html>
```

# Applets



An Applet in the Applet Viewer



An Applet in a Web Browser

# Ellipses

---

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

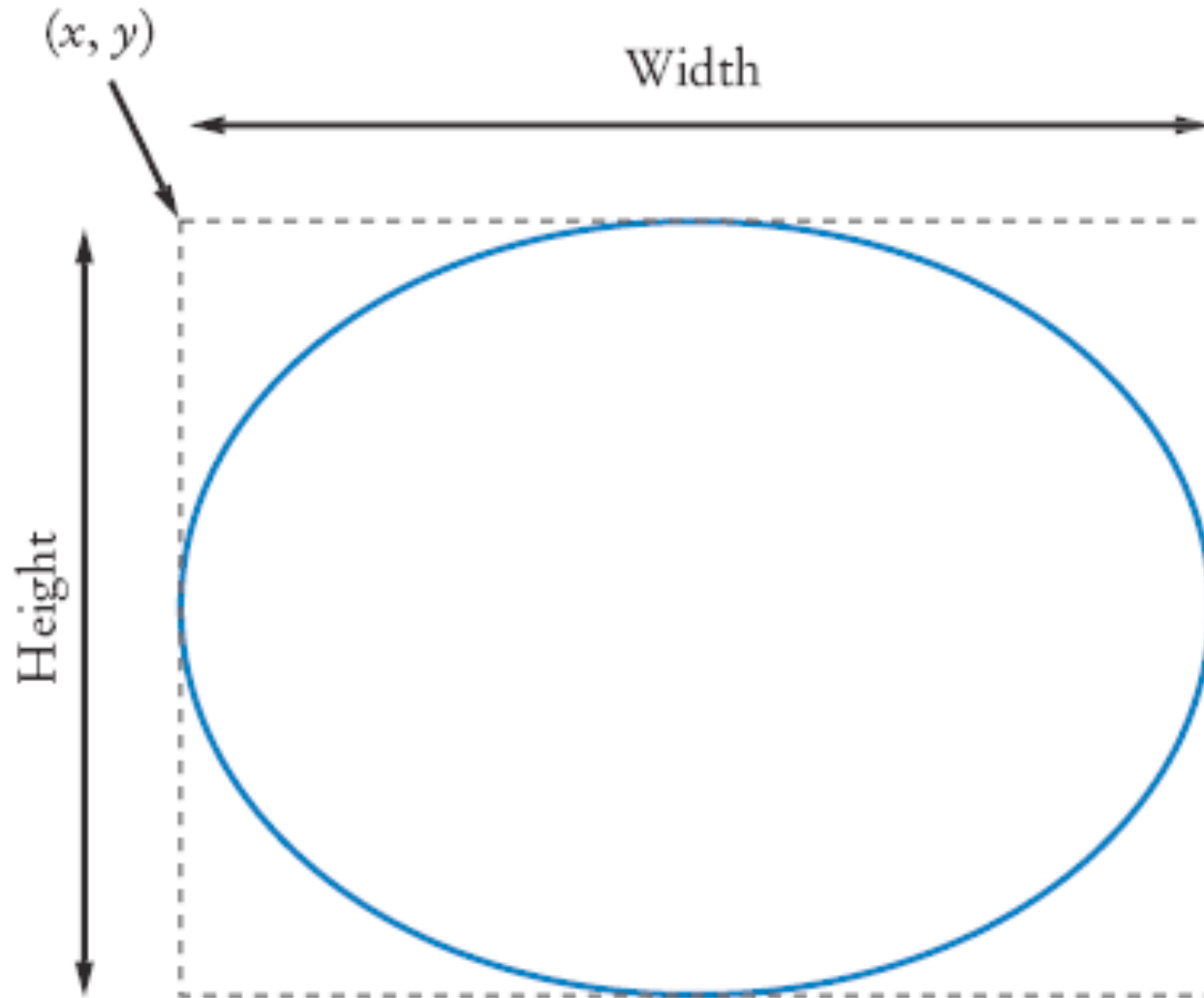
```
import java.awt.geom.Ellipse2D; // no .Double
```

- Must construct *and draw* the shape

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

# An Ellipse

---



ICOM 4015 Fall 200 **Figure 23** An Ellipse and Its Bounding Box

*Big Java* by Cay Horstmann

Copyright © 2008 by John Wiley & Sons. All rights reserved.

## 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 24** 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`

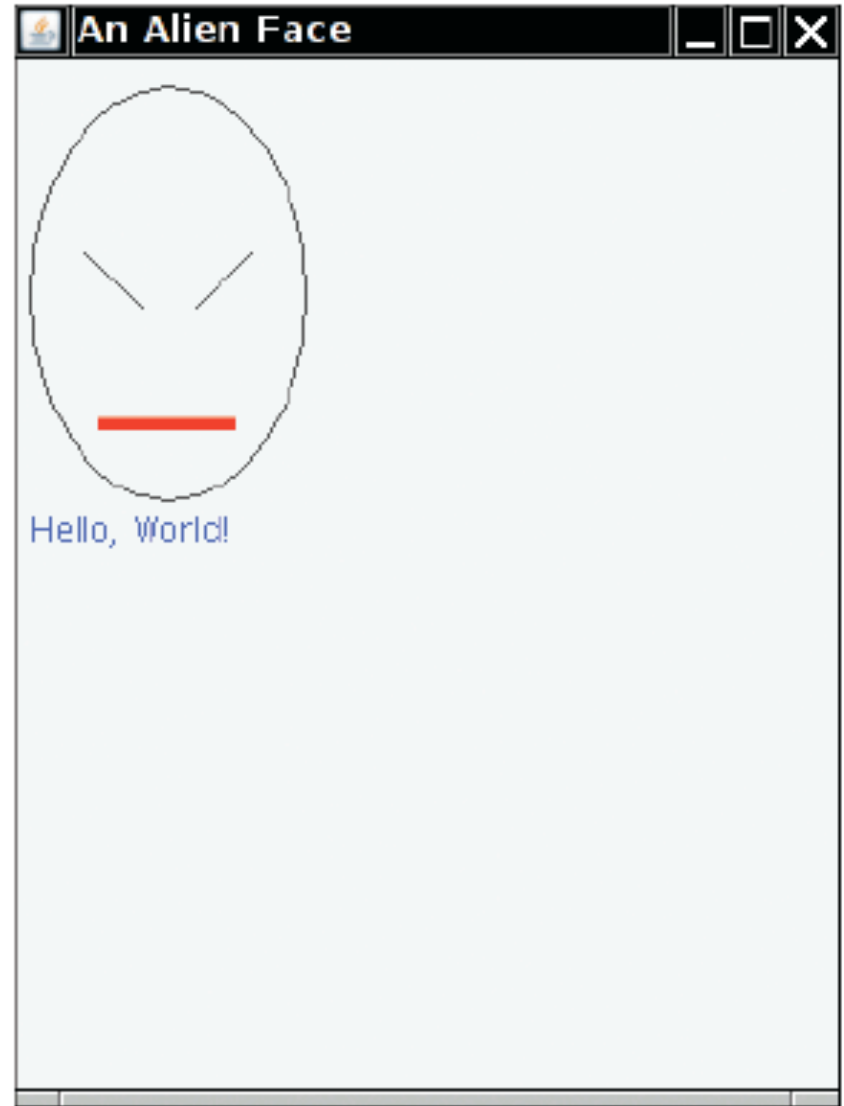
# Predefined Colors and Their RGB Values

---

<b>Color</b>	<b>RGB Value</b>
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 25** An Alien Face

ICOM 4015 Fall 2008

## ch02/faceviewer/FaceComponent.java

---

```
01: import java.awt.Color;
02: import java.awt.Graphics;
03: import java.awt.Graphics2D;
04: import java.awt.Rectangle;
05: import java.awt.geom.Ellipse2D;
06: import java.awt.geom.Line2D;
07: import javax.swing.JPanel;
08: import javax.swing.JComponent;
09:
10: /**
11:     A component that draws an alien face
12: */
13: public class FaceComponent extends JComponent
14: {
15:     public void paintComponent(Graphics g)
16:     {
17:         // Recover Graphics2D
18:         Graphics2D g2 = (Graphics2D) g;
19:
```

***Continued***

## ch02/faceviewer/FaceComponent.java (cont.)

---

```
20:         // Draw the head
21:         Ellipse2D.Double head = new Ellipse2D.Double(5, 10, 100, 150);
22:         g2.draw(head);
23:
24:         // Draw the eyes
25:         Line2D.Double eye1 = new Line2D.Double(25, 70, 45, 90);
26:         g2.draw(eye1);
27:
28:         Line2D.Double eye2 = new Line2D.Double(85, 70, 65, 90);
29:         g2.draw(eye2);
30:
31:         // Draw the mouth
32:         Rectangle mouth = new Rectangle(30, 130, 50, 5);
33:         g2.setColor(Color.RED);
34:         g2.fill(mouth);
35:
36:         // Draw the greeting
37:         g2.setColor(Color.BLUE);
38:         g2.drawString("Hello, World!", 5, 175);
39:     }
40: }
```

## ch02/faceviewer/FaceViewer.java

---

```
01: import javax.swing.JFrame;
02:
03: public class FaceViewer
04: {
05:     public static void main(String[] args)
06:     {
07:         JFrame frame = new JFrame();
08:         frame.setSize(300, 400);
09:         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));
```