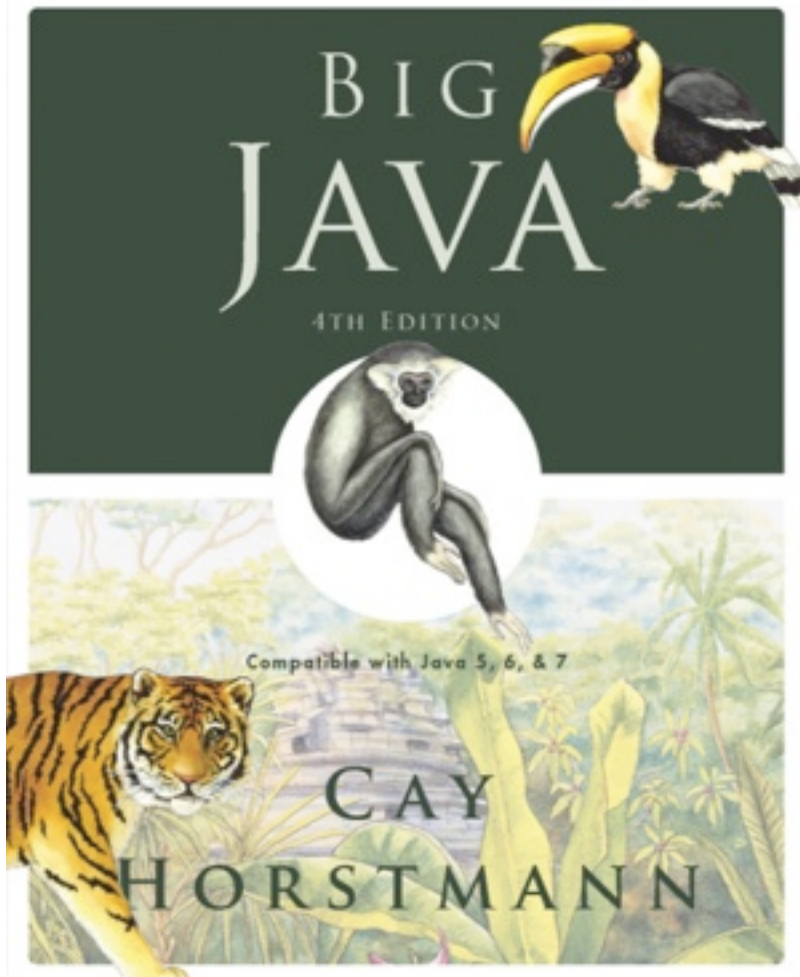


# ICOM 4015: Advanced Programming

## Lecture 2

**Reading: Chapter Two: Using Objects**



## 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

# Key Concepts

---

- Types are sets of objects with same behavior
- The type determines the operations that can be performed on its member objects
- Variables can hold values of the types that they are declared
- Variables are accessible within specific SCOPES
- Not all “objects” are Object’s

# Primitive Types vs. Class 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**
- Java has two types of types: **Primitives** and **Classes**



# Eight primitive data types in Java

---

- **byte**: The byte data type is an 8-bit signed two's complement integer. It has a minimum value of -128 and a maximum value of 127 (inclusive). The byte data type can be useful for saving memory in large [arrays](#), where the memory savings actually matters. They can also be used in place of int where their limits help to clarify your code; the fact that a variable's range is limited can serve as a form of documentation.
- **short**: The short data type is a 16-bit signed two's complement integer. It has a minimum value of -32,768 and a maximum value of 32,767 (inclusive). As with byte, the same guidelines apply: you can use a short to save memory in large arrays, in situations where the memory savings actually matters.
- **int**: The int data type is a 32-bit signed two's complement integer. It has a minimum value of -2,147,483,648 and a maximum value of 2,147,483,647 (inclusive). For integral values, this data type is generally the default choice unless there is a reason (like the above) to choose something else. This data type will most likely be large enough for the numbers your program will use, but if you need a wider range of values, use long instead.
- **long**: The long data type is a 64-bit signed two's complement integer. It has a minimum value of -9,223,372,036,854,775,808 and a maximum value of 9,223,372,036,854,775,807 (inclusive). Use this data type when you need a range of values wider than those provided by int.
- **float**: The float data type is a single-precision 32-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](#) section of the Java Language Specification. As with the recommendations for byte and short, use a float (instead of double) if you need to save memory in large arrays of floating point numbers. This data type should never be used for precise values, such as currency. For that, you will need to use the [java.math.BigDecimal](#) class instead. [Numbers and Strings](#) covers BigDecimal and other useful classes provided by the Java platform.
- **double**: The double data type is a double-precision 64-bit IEEE 754 floating point. Its range of values is beyond the scope of this discussion, but is specified in the [Floating-Point Types, Formats, and Values](#) section of the Java Language Specification. For decimal values, this data type is generally the default choice. As mentioned above, this data type should never be used for precise values, such as currency.
- **boolean**: The boolean data type has only two possible values: true and false. Use this data type for simple flags that track true/false conditions. This data type represents one bit of information, but its "size" isn't something that's precisely defined.
- **char**: The char data type is a single 16-bit Unicode character. It has a minimum value of '\u0000' (or 0) and a maximum value of '\uffff' (or 65,535 inclusive).

# 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: $1 \times 10^6$ 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		<b>Error:</b> Do not use a comma as a decimal separator.
 3 1/2		<b>Error:</b> 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 as System.out.println(20)
```

# Variables



---

- It is a compiler 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
<code>int width = 10;</code>	Declares an integer variable and initializes it with 10.
<code>int area = width * height;</code>	The initial value can depend on other variables. (Of course, width and height must have been previously declared.)
 <code>height = 5;</code>	<b>Error:</b> 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.
 <code>int height = "5";</code>	<b>Error:</b> You cannot initialize a number with a string.
<code>int width, height;</code>	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

The type specifies what can be done with values stored in this variable.



Use a descriptive variable name.

String greeting = "Hello, Dave!";





See the rules for and table of examples of valid names.

A variable declaration ends with a semicolon.

Supplying an initial value is optional, but it is usually a good idea.

# 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	<b>Caution:</b> Variable names are case-sensitive. This variable name is different from greeting.
 6pack	<b>Error:</b> Variable names cannot start with a number.
 farewell message	<b>Error:</b> Variable names cannot contain spaces.
 public	<b>Error:</b> 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
```

①

width = 10

②

width = 20

# 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
```

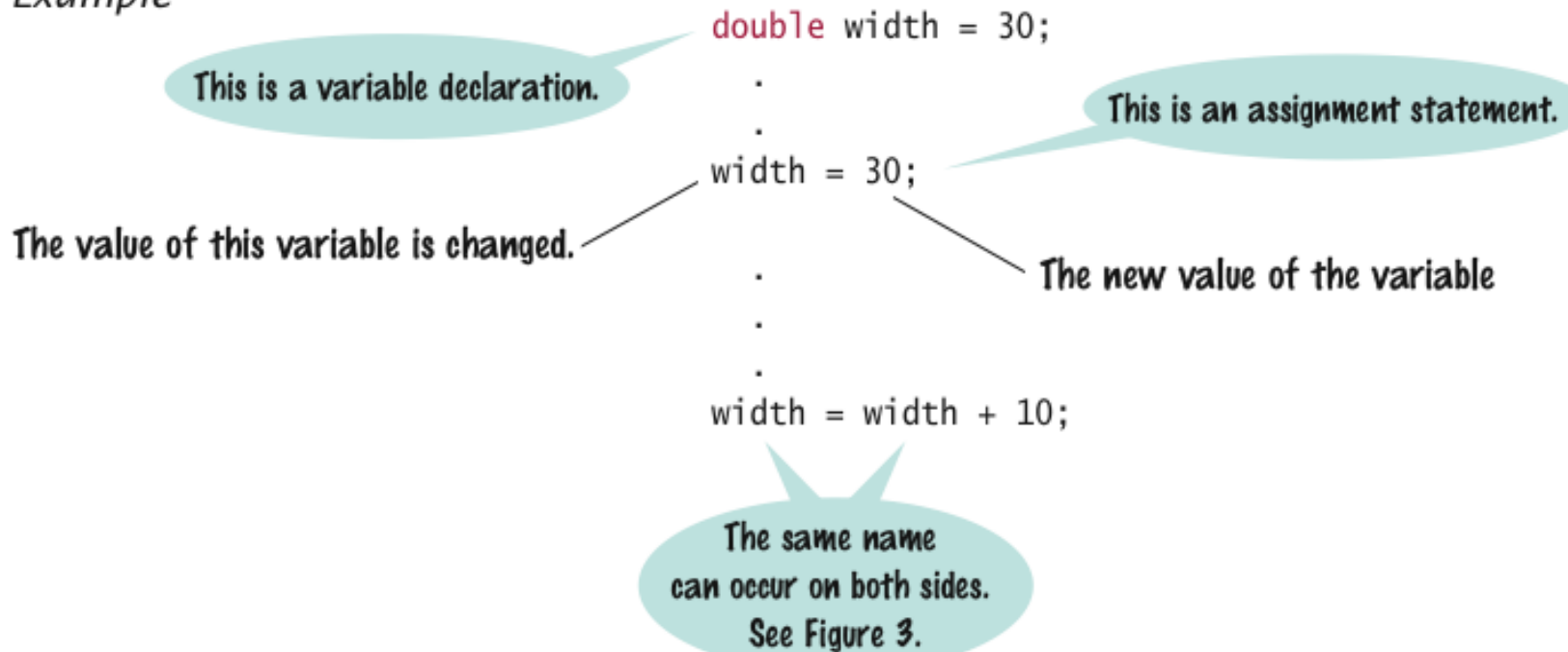
- Even better, initialize the variable when you declare it:

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

## Syntax 2.2 Assignment

*Syntax*    *variableName = value;*

*Example*





# Assignment

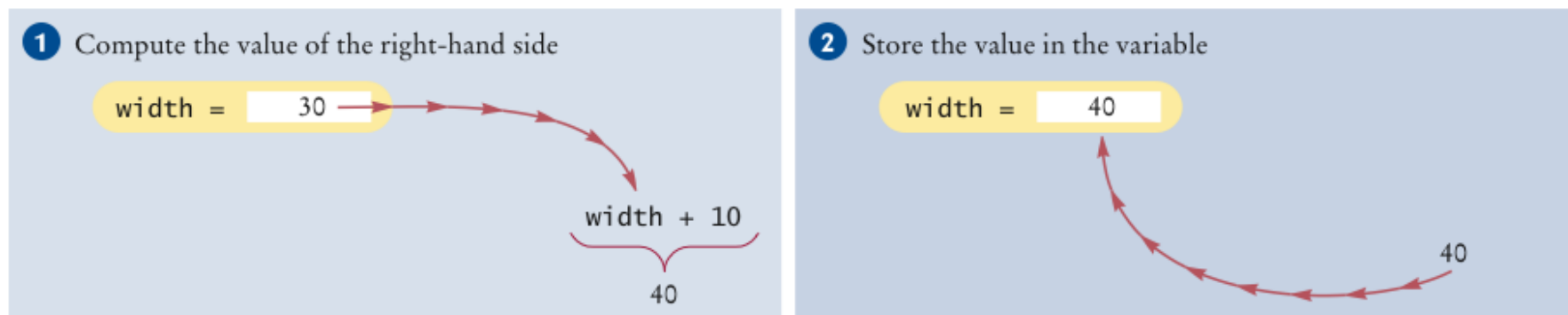
- The right-hand side of the = symbol can be a mathematical expression:

```
width = width + 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.

2-01 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

The screenshot displays the Java API documentation for the `PrintStream` class. The left sidebar shows the navigation pane with the `PrintStream` class selected. The main content area lists the following methods:

- `void println()`: A convenience method to write a formatted string to this output stream using the specified format string.
- `void println(boolean x)`: Terminates the current line by writing the line separator string. Prints a boolean and then terminate the line.
- `void println(char x)`: Prints a character and then terminate the line.
- `void println(char[] x)`: Prints an array of characters and then terminate the line.
- `void println(double x)`: Prints a double and then terminate the line.
- `void println(float x)`: Prints a float and then terminate the line.
- `void println(int x)`: Prints an integer and then terminate the line.
- `void println(long x)`: Prints a long and then terminate the line.
- `void println(Object x)`: Prints an Object and then terminate the line.
- `void println(String x)`: Prints a String and then terminate the line.
- `protected void setError()`: Sets the error state of the stream to `true`.
- `void write(byte[] buf, int off, int len)`: Writes `len` bytes from the specified byte array starting at offset `off` to this stream.
- `void write(int b)`: Writes the specified byte to this stream.

Methods inherited from class `java.io.FilterOutputStream`:

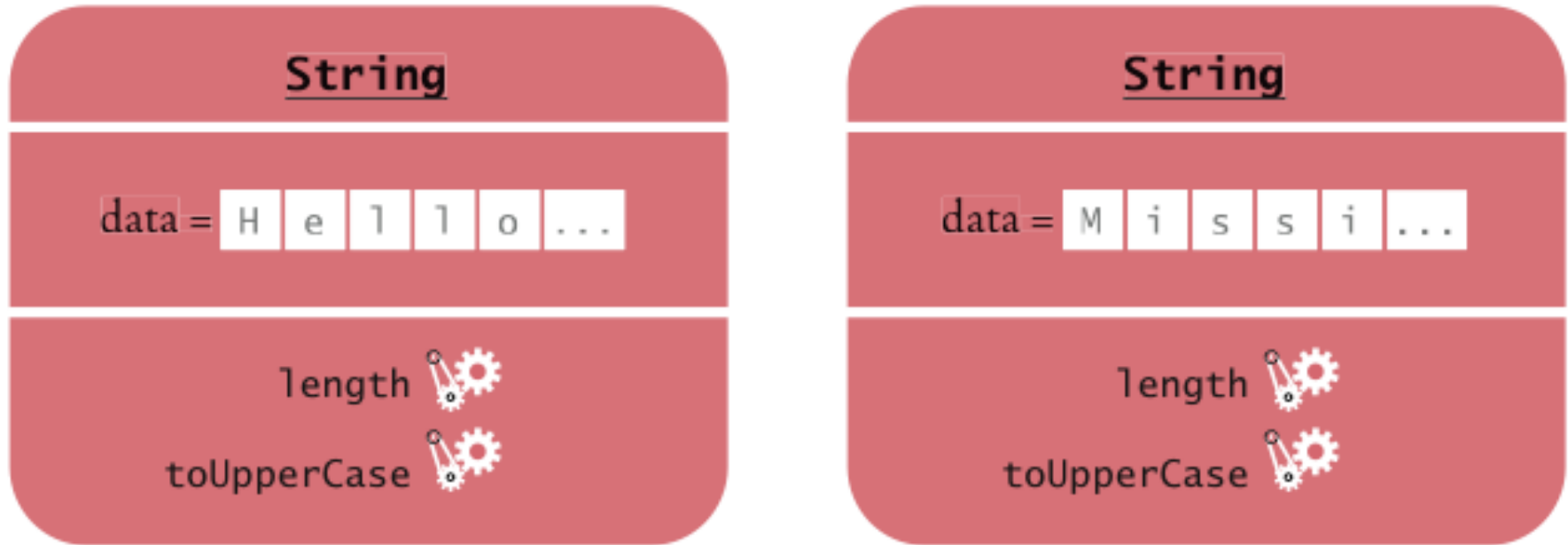
- `write`

Methods inherited from class `java.lang.Object`:

- `clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

# Objects of Same Class share methods, but not data

---



**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.

# Explicit vs. Implicit Method Parameters

---

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

```
String greeting = "Hello, World!";  
System.out.println(greeting);
```

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

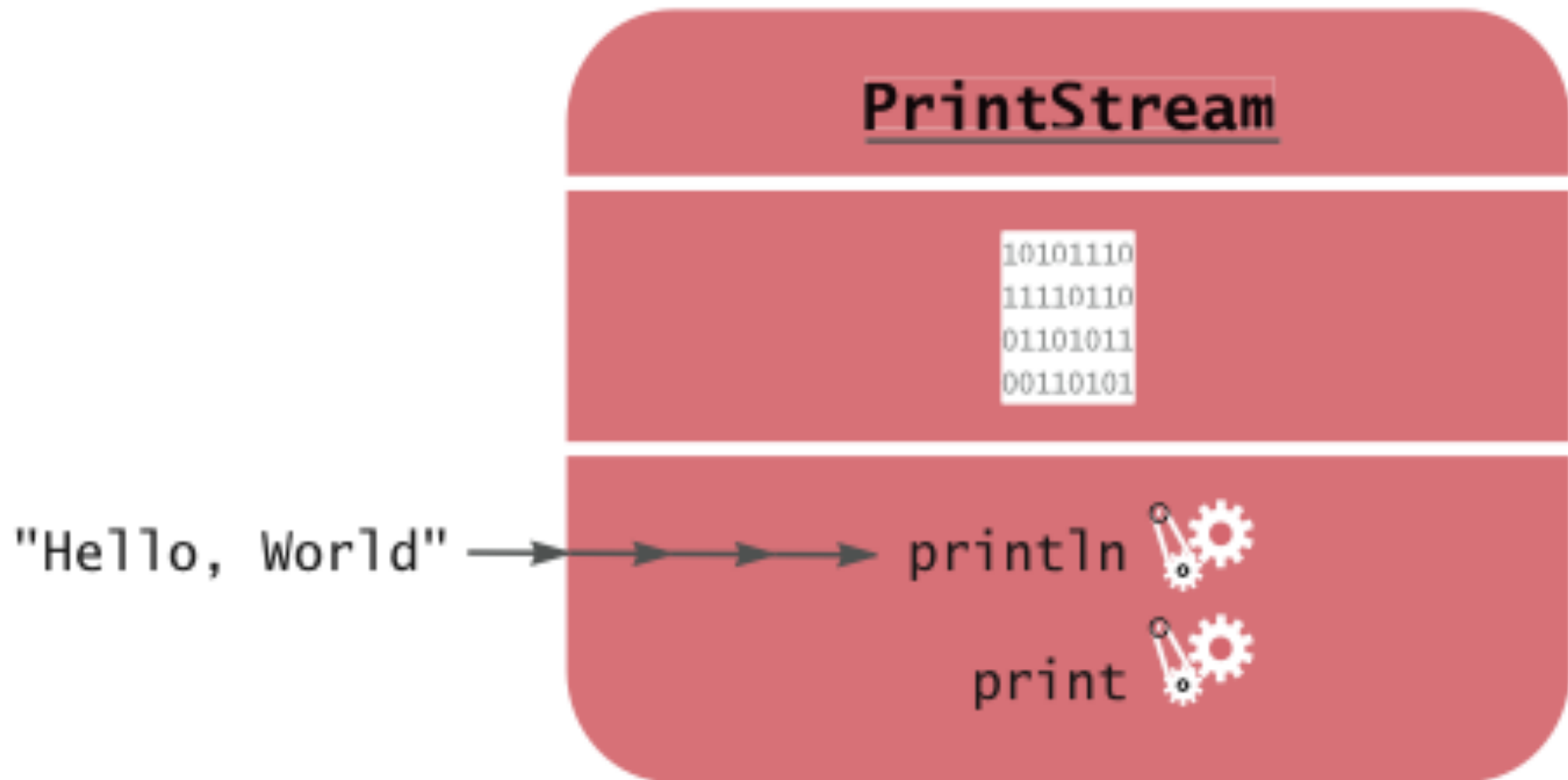
```
System.out.println(greeting)
```

- Not all methods have explicit parameters:

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

# Passing a Parameter

---

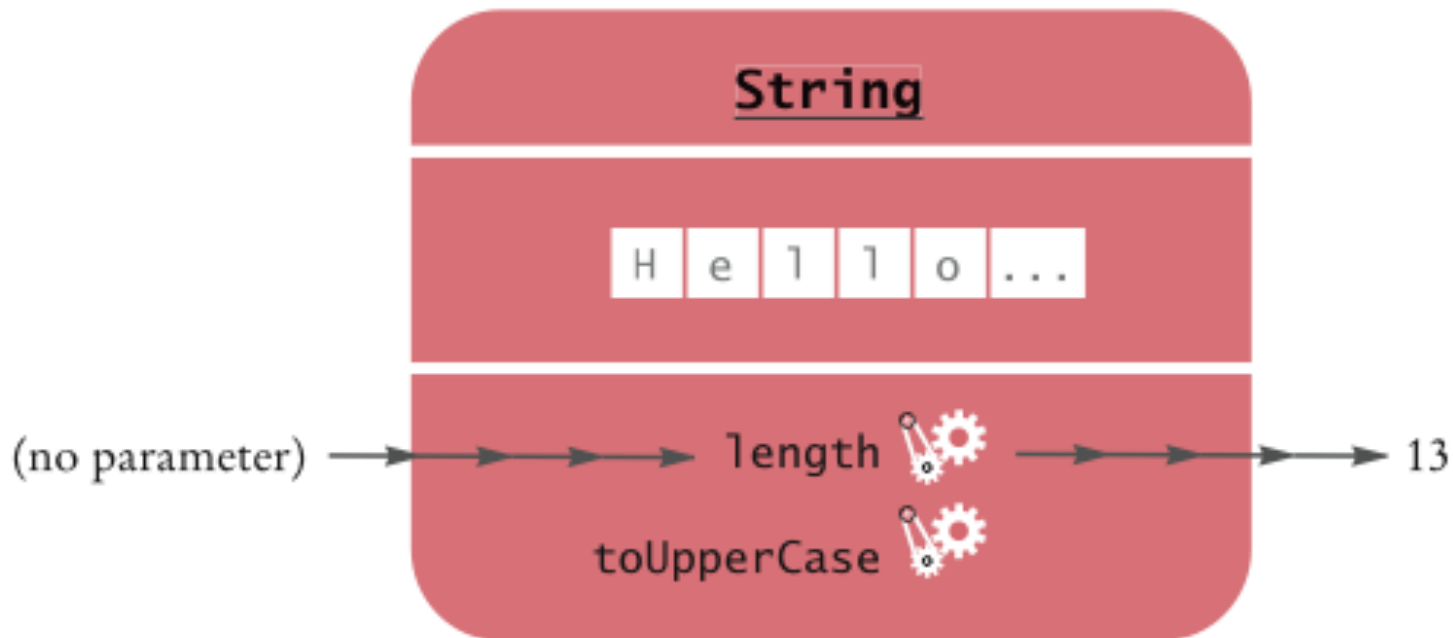


**Figure 6** Passing a Parameter to the `println` Method

# 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
```

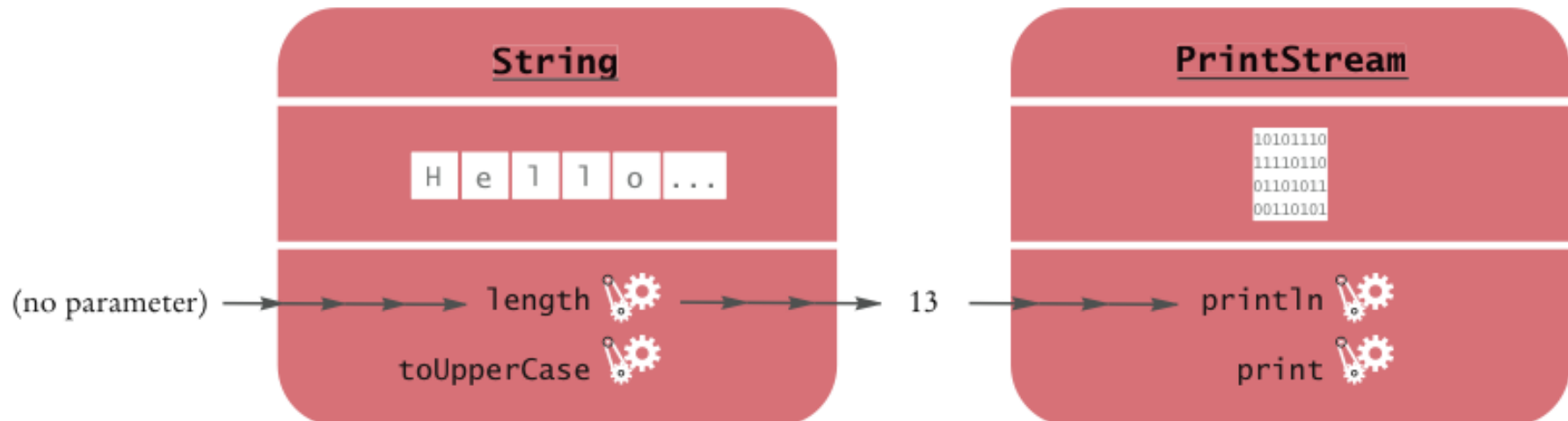


**Figure 7** Invoking the length Method on a String Object

# 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

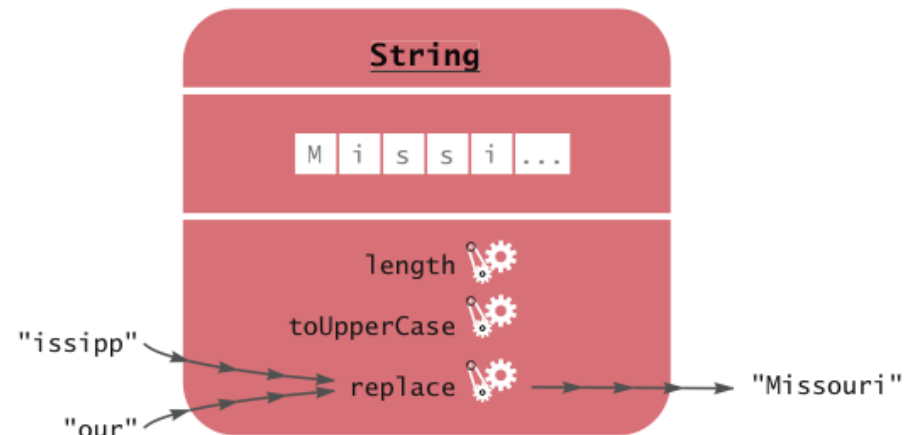
- Not all methods return values. Example: `println`



## 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

---

## 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:** `"Mississississippi"`.

## 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?

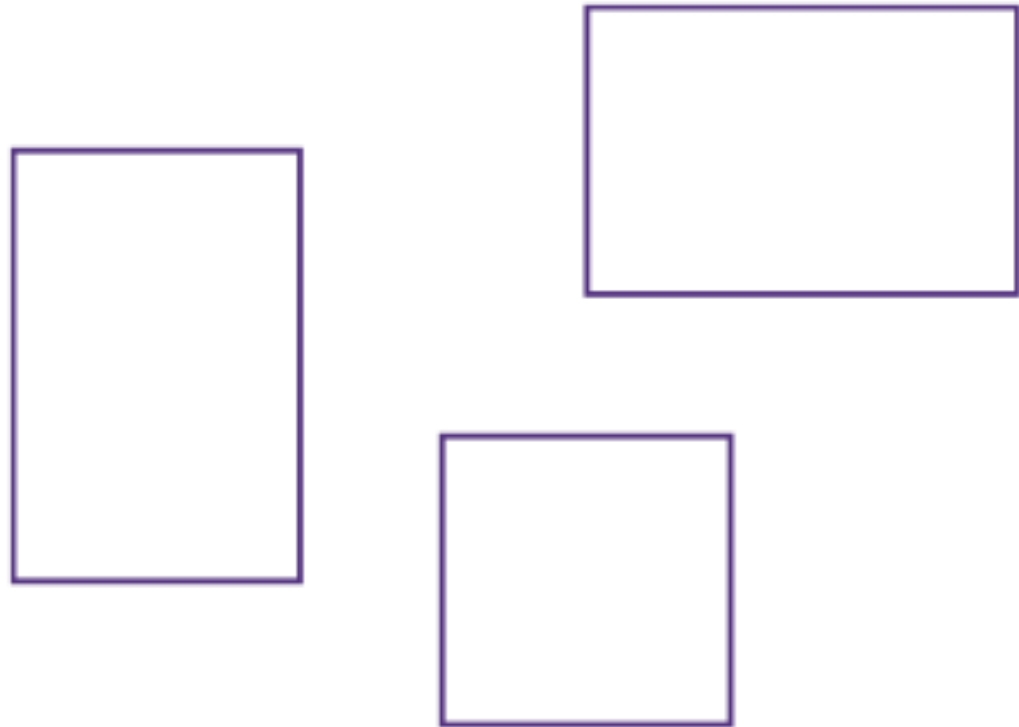
**Answer:** As `public String toUpperCase()`, with no explicit parameter and return type `String`.

# Rectangular Shapes and Rectangle Objects

---

- Objects of built-in class `Rectangle` *describe* rectangular shapes:

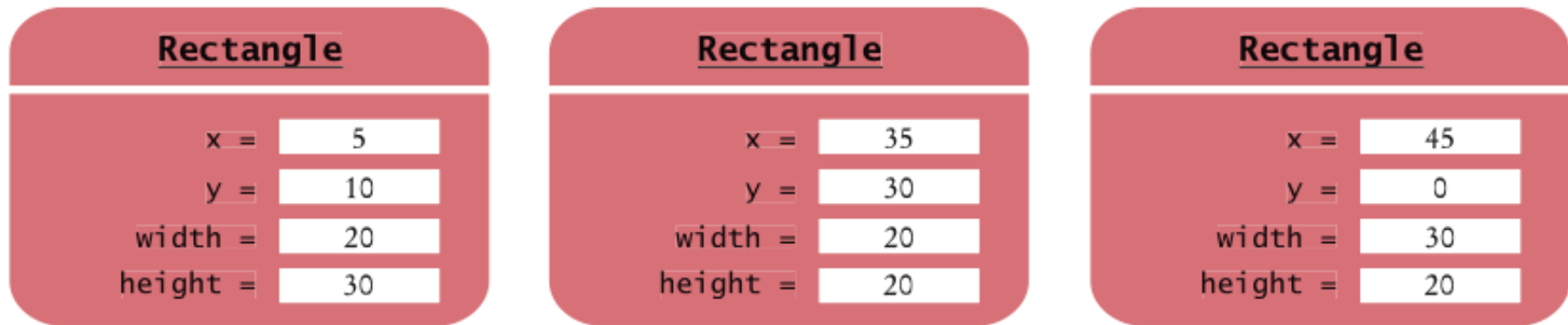
**Figure 10**  
Rectangular Shapes



# 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
```

# Some *Rectangle* class methods

Rectangle (Java Platform SE 7)

docs.oracle.com/javase/7/docs/api/

Method Summary

Methods

Modifier and Type	Method and Description
void	<b>add(int newx, int newy)</b> Adds a point, specified by the integer arguments newx, newy to the bounds of this Rectangle.
void	<b>add(Point pt)</b> Adds the specified Point to the bounds of this Rectangle.
void	<b>add(Rectangle r)</b> Adds a Rectangle to this Rectangle.
boolean	<b>contains(int x, int y)</b> Checks whether or not this Rectangle contains the point at the specified location (x,y).
boolean	<b>contains(int X, int Y, int W, int H)</b> Checks whether this Rectangle entirely contains the Rectangle at the specified location (X,Y) with the specified dimensions (W,H).
boolean	<b>contains(Point p)</b> Checks whether or not this Rectangle contains the specified Point.
boolean	<b>contains(Rectangle r)</b> Checks whether or not this Rectangle entirely contains the specified Rectangle.
Rectangle2D	<b>createIntersection(Rectangle2D r)</b> Returns a new Rectangle2D object representing the intersection of this Rectangle2D with the specified Rectangle2D.
Rectangle2D	<b>createUnion(Rectangle2D r)</b> Returns a new Rectangle2D object representing the union of this Rectangle2D with the specified Rectangle2D.
boolean	<b>equals(Object obj)</b> Checks whether two rectangles are equal.
Rectangle	<b>getBounds()</b> Gets the bounding Rectangle of this Rectangle.
Rectangle2D	<b>getBounds2D()</b> Returns a high precision and more accurate bounding box of the Shape than the getBounds method.
double	<b>getHeight()</b> Returns the height of the bounding Rectangle in double precision.
Point	<b>getLocation()</b> Returns the location of this Rectangle.
Dimension	<b>getSize()</b> Gets the size of this Rectangle, represented by the returned Dimension.

## Syntax 2.3 Object Construction

**Syntax** `new ClassName(parameters)`

**Example**

The new expression yields an object.

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

Construction parameters

Usually, you save the constructed object in a variable.

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

You can also pass the constructed object to a method.

Supply the parentheses even when there are no parameters.

## 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:

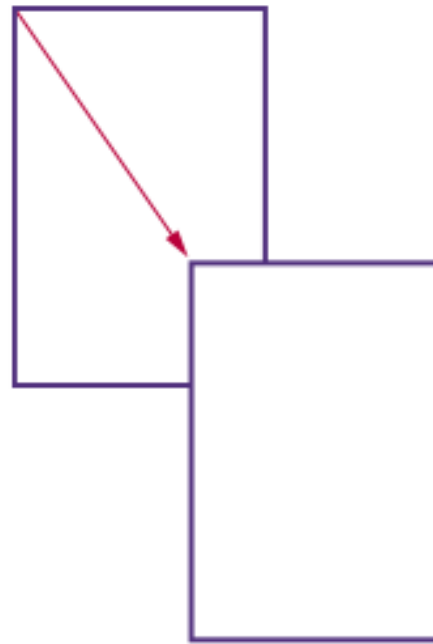
```
double width = box.getWidth();
```

- **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

The screenshot shows a web browser window displaying the 'Overview (Java Platform SE 7)' page. The browser's address bar shows 'docs.oracle.com/javase/7/docs/api/'. The page title is 'Java™ Platform, Standard Edition 7 API Specification'. Below the title, there is a navigation menu with 'Overview', 'Package', 'Class', 'Use', 'Tree', 'Deprecated', 'Index', and 'Help'. The main content area features a table of packages with columns for 'Package' and 'Description'. The table lists various packages such as 'java.applet', 'java.awt', 'java.awt.color', 'java.awt.datatransfer', 'java.awt.dnd', 'java.awt.event', 'java.awt.font', 'java.awt.geom', 'java.awt.im', 'java.awt.im.spi', 'java.awt.image', 'java.awt.image.renderable', 'java.awt.print', and 'java.beans'. A search bar at the bottom of the page shows the search term 'Rectan'.

Overview (Java Platform SE 7)

docs.oracle.com/javase/7/docs/api/

Overview Package Class Use Tree Deprecated Index Help

## Java™ Platform, Standard Edition 7 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

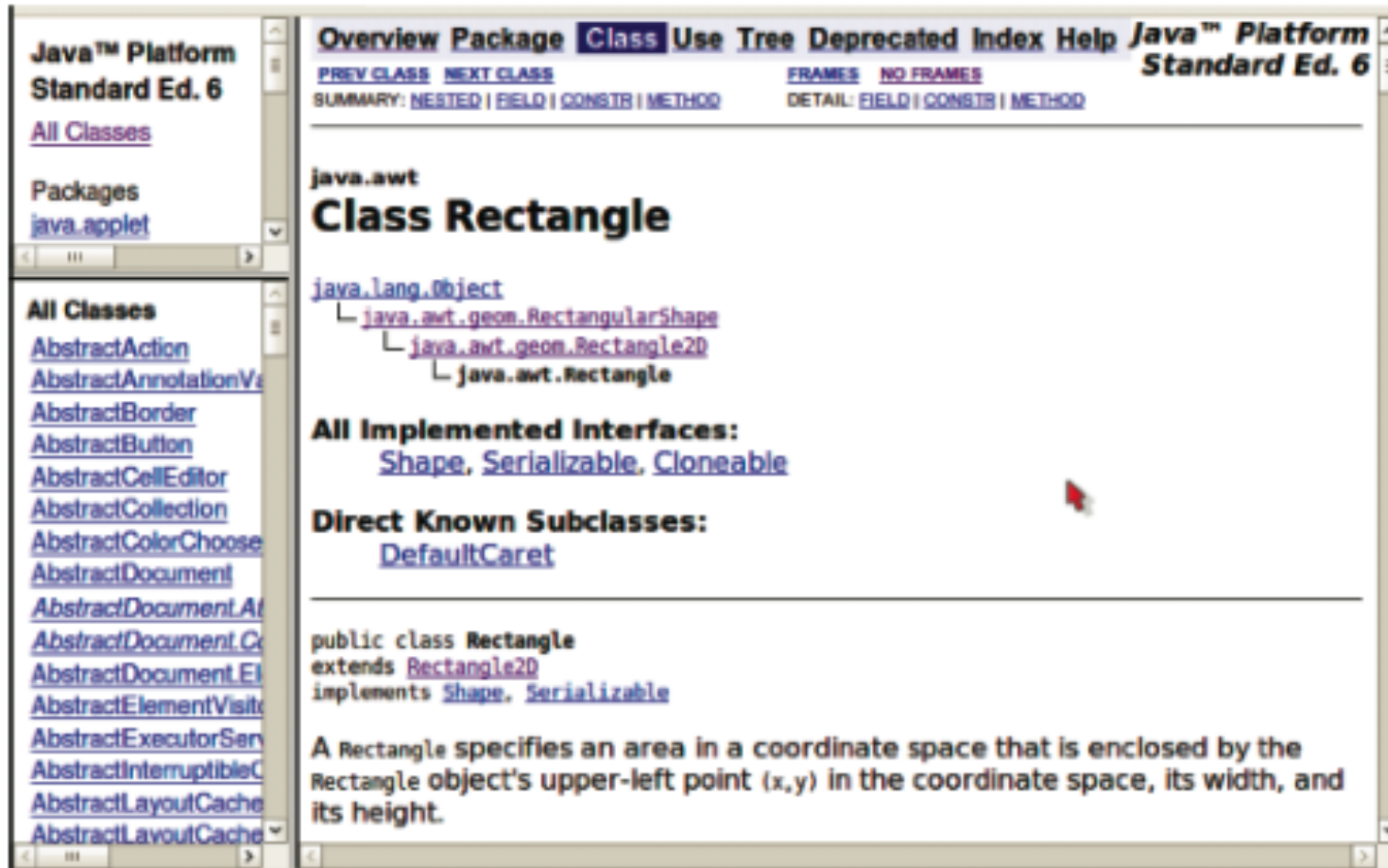
See: Description

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating user interfaces and for painting graphics and images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for transferring data between and within applications.
java.awt.dnd	Drag and Drop is a direct manipulation gesture found in many Graphical User Interface systems that provides a mechanism to transfer information between two entities logically associated with presentation elements in the GUI.
java.awt.event	Provides interfaces and classes for dealing with different types of events fired by AWT components.
java.awt.font	Provides classes and interface relating to fonts.
java.awt.geom	Provides the Java 2D classes for defining and performing operations on objects related to two-dimensional geometry.
java.awt.im	Provides classes and interfaces for the input method framework.
java.awt.im.spi	Provides interfaces that enable the development of input methods that can be used with any Java runtime environment.
java.awt.image	Provides classes for creating and modifying images.
java.awt.image.renderable	Provides classes and interfaces for producing rendering-independent images.
java.awt.print	Provides classes and interfaces for a general printing API.
java.beans	Contains classes related to developing <i>beans</i> – components based on the JavaBeans™ architecture.

Find: Rectan

Downloads Canóvanas... Carolina 1... Carolina 1... Carolina 1... San Juan 0... San Juan 0... San Juan 0... San Juan 0... Cayey 0... installgoo... moodl... 12-13-1... Clear

# The API Documentation for the Rectangle Class



**Figure 14** The API Documentation for the Rectangle Class

# Method Summary

The screenshot displays the Java Platform Standard Ed. 6 documentation interface. On the left, there is a sidebar with the following content:

- Java™ Platform Standard Ed. 6
- [All Classes](#)
- Packages
- [java.applet](#)
- All Classes**
- [AbstractAction](#)
- [AbstractAnnotationV](#)
- [AbstractBorder](#)
- [AbstractButton](#)
- [AbstractCellEditor](#)
- [AbstractCollection](#)
- [AbstractColorChoose](#)
- [AbstractDocument](#)
- [AbstractDocumentAt](#)
- [AbstractDocument.C](#)
- [AbstractDocument.El](#)
- [AbstractElementVisi](#)
- [AbstractExecutorSer](#)

The main content area is titled "Method Summary" and contains a table of methods for the Rectangle class:

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

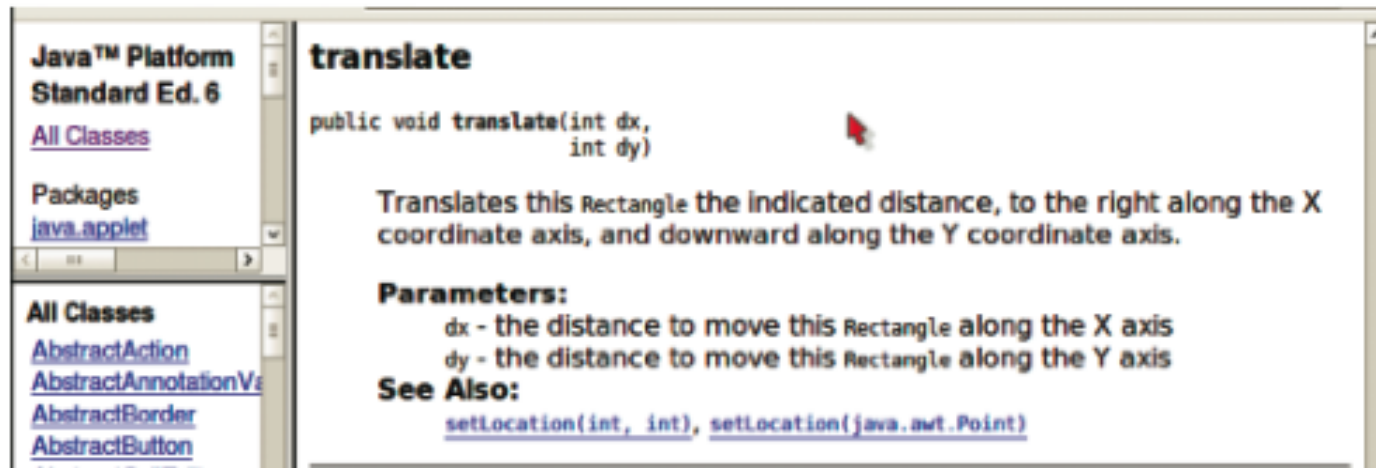
**Figure 15** The Method Summary for the Rectangle Class

# 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**

Import statements must be at the top of the source file.

Package name                      Class name

`import java.awt.Rectangle;`

You can look up the package name in the API documentation.



## 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

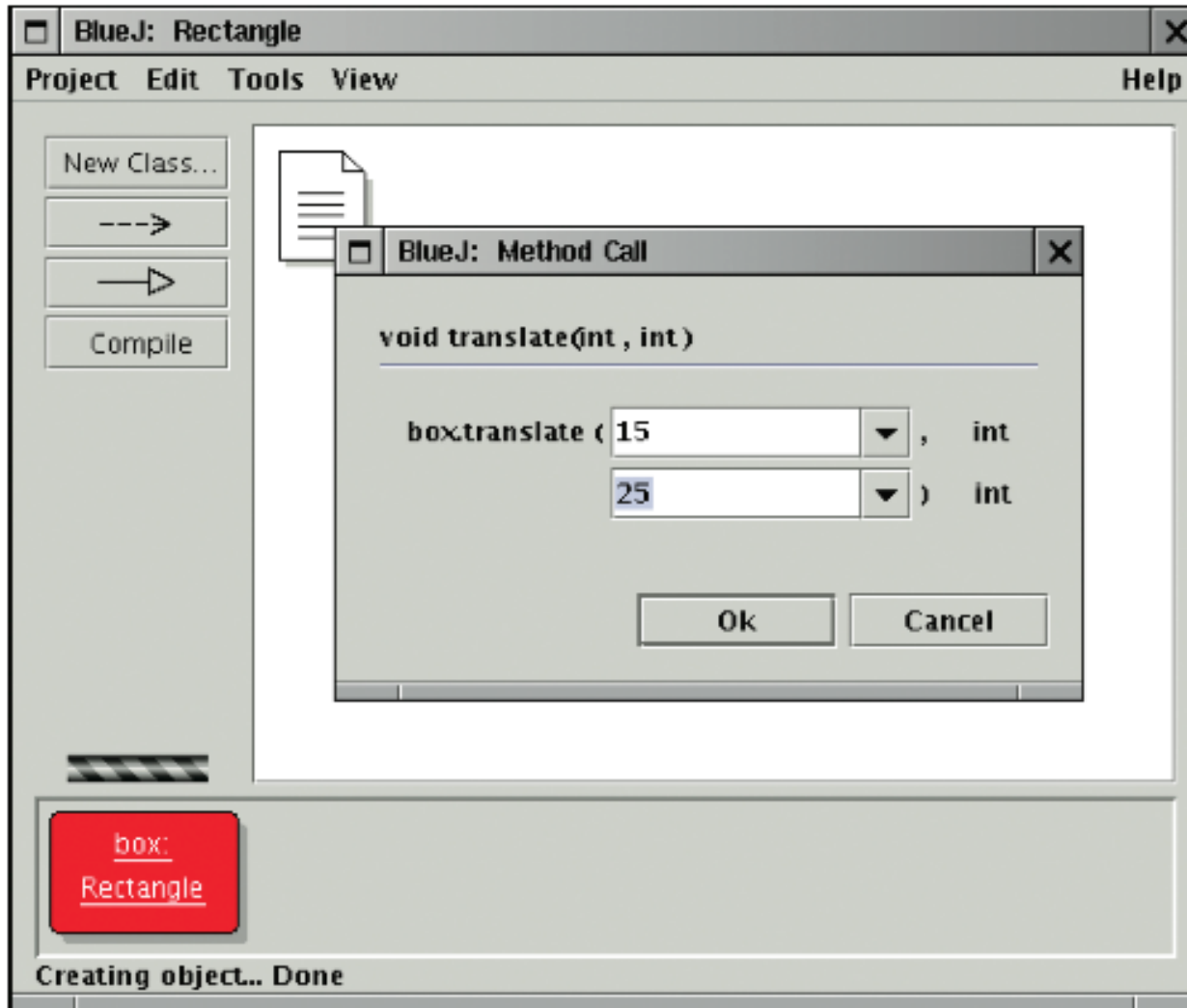
---

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();
```

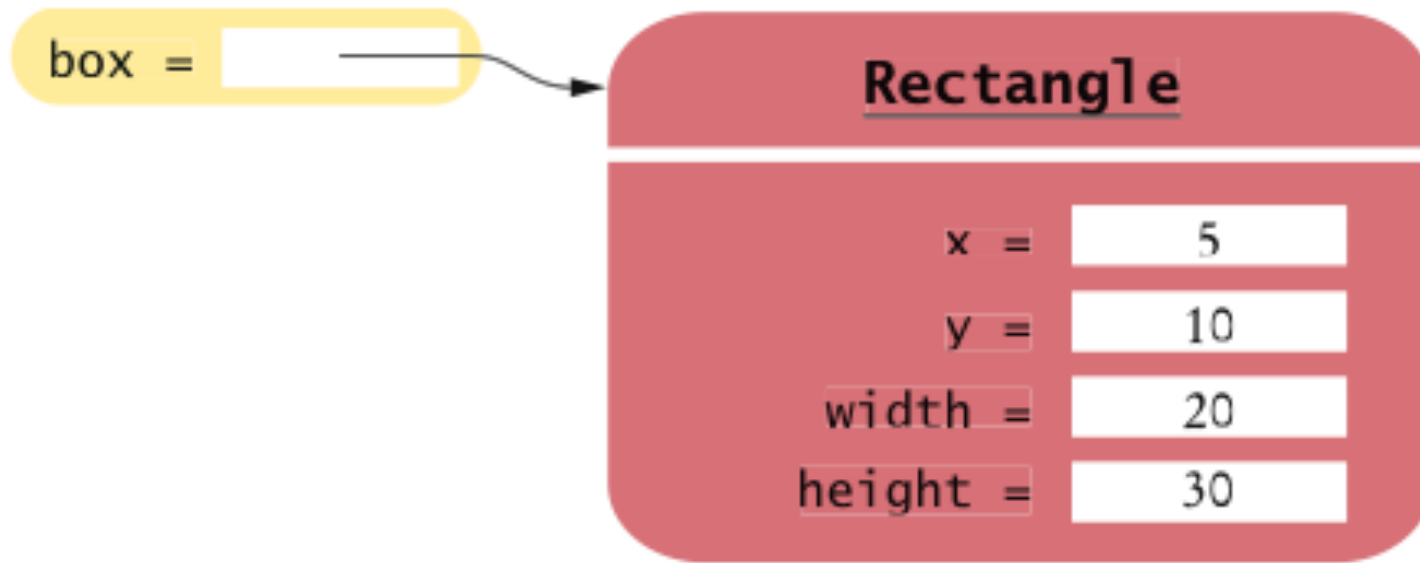
- Multiple object variables can refer to the same object:

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

- Primitive type variables  $\neq$  object variables

# Object Variables and Number Variables

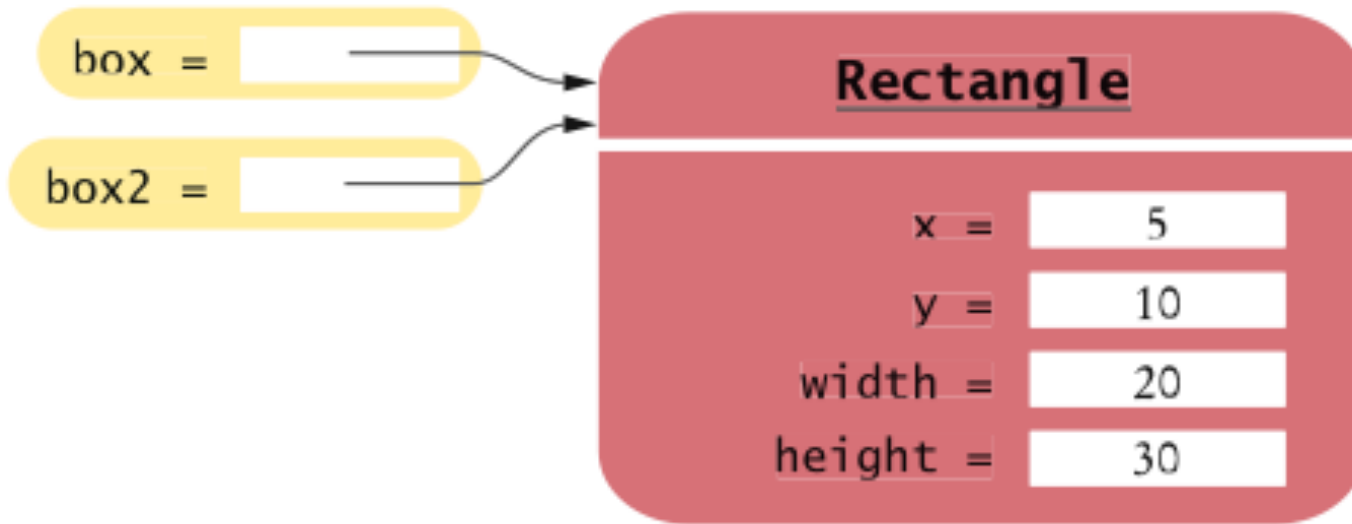
---



**Figure 17** An Object Variable Containing an Object Reference

# Object Variables and Number Variables

---



**Figure 18** Two Object Variables Referring to the Same Object

LuckyNumber = 13

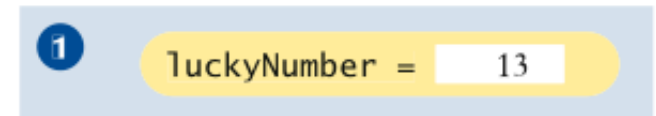
**Figure 19** A Number Variable Stores a Number

# Copying Numbers

---

```
int luckyNumber = 13; ①
```

**Figure 20**  
Copying Numbers



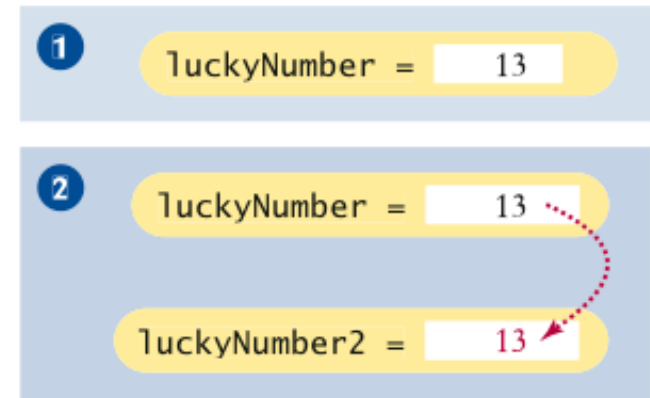
## Copying Numbers (cont.)

---

```
int luckyNumber = 13; ①
```

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

**Figure 20**  
Copying Numbers



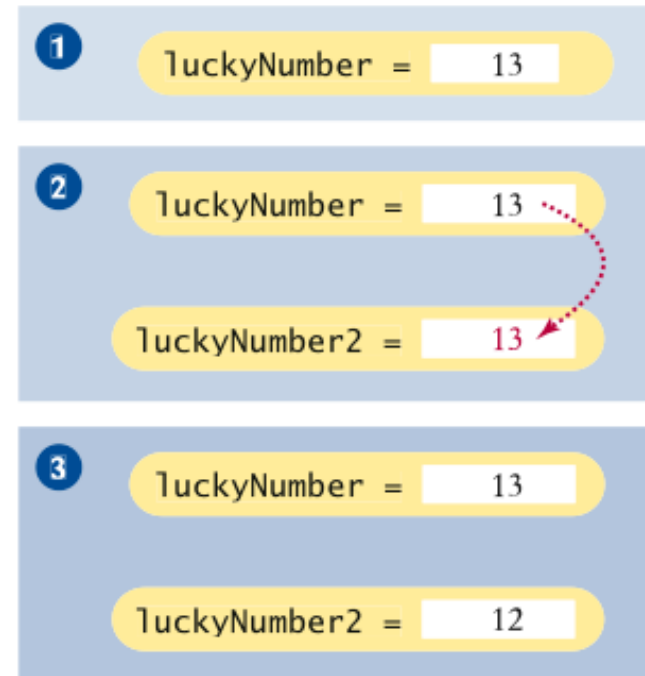
# Copying Numbers (cont.)

```
int luckyNumber = 13; ①
```

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

```
luckyNumber2 = 12; ③
```

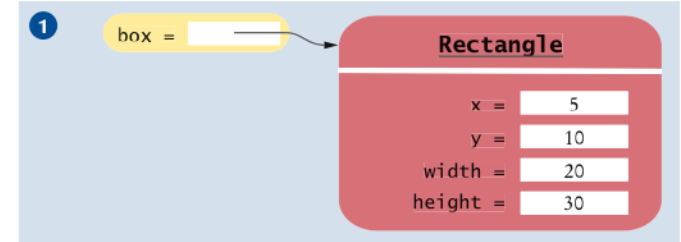
**Figure 20**  
Copying Numbers



# Copying Object References

---

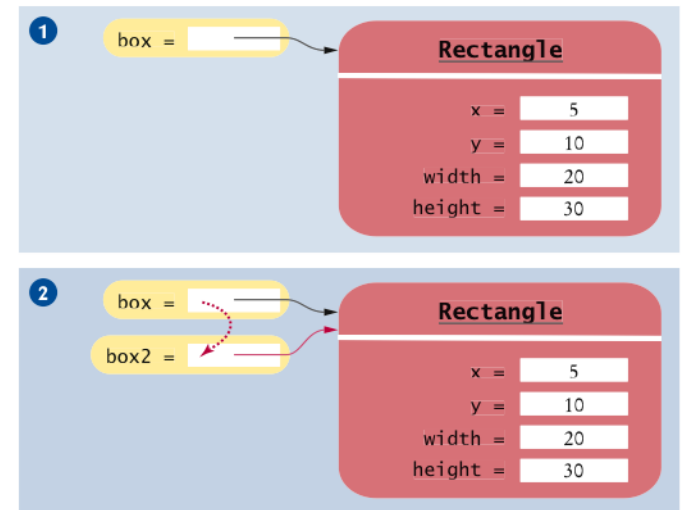
```
Rectangle box = new Rectangle(5, 10, 20, 30); 1
```





# 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
```

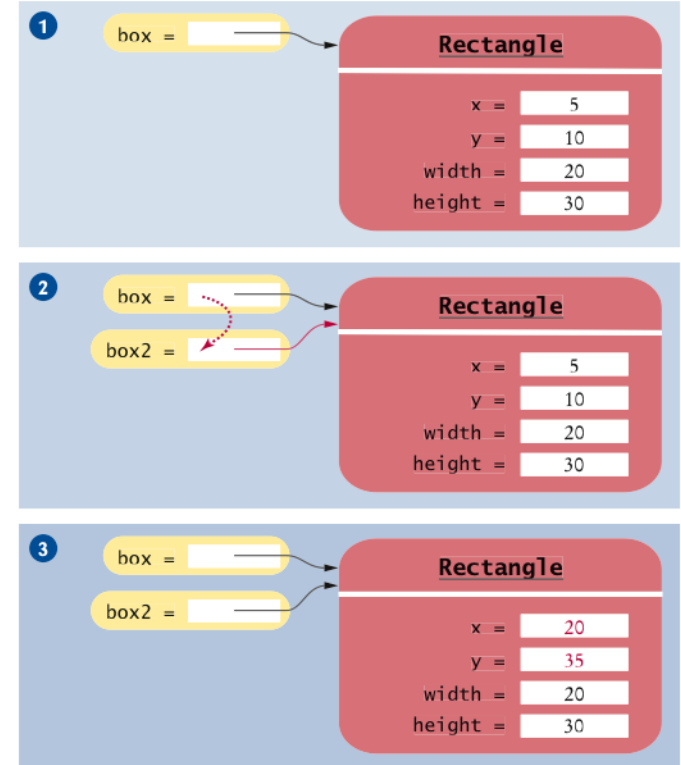


Figure 21 Copying Object References

## 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”:

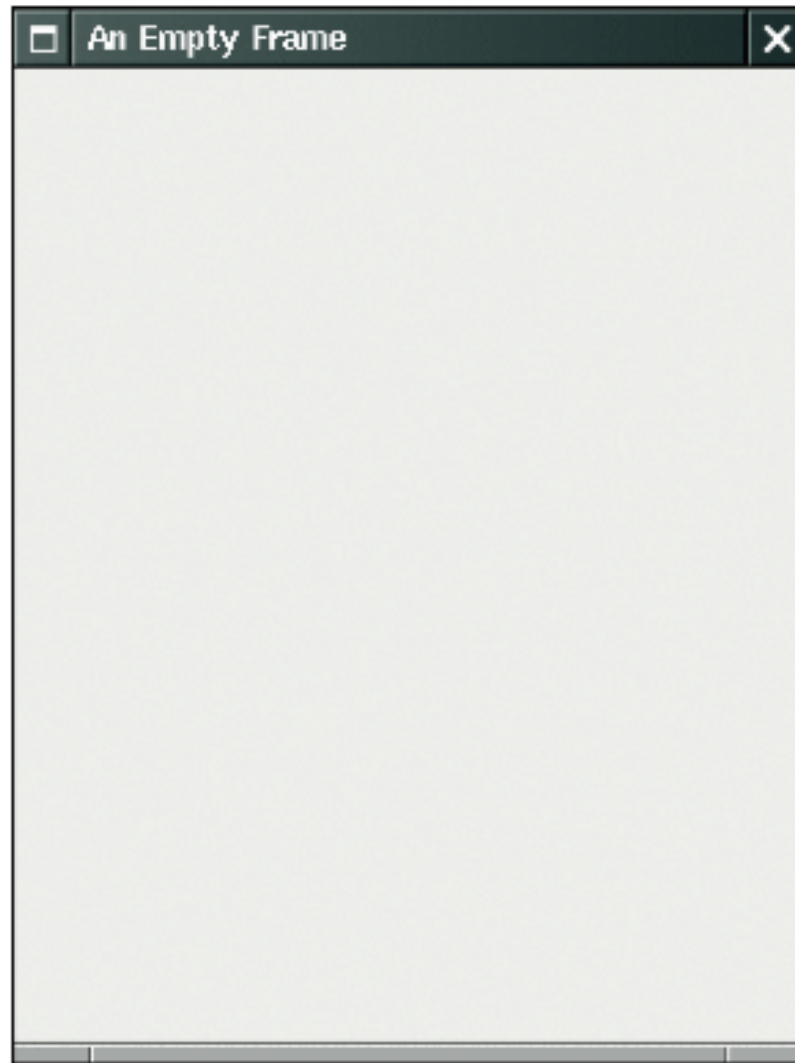
```
frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
```

5. Make the frame visible:

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

# A Frame Window

---



**Figure 22**  
A Frame Window

## ch02/emptyframe/EmptyFrameViewer.java

---

```
import javax.swing.JFrame;

public class EmptyFrameViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        frame.setSize(300, 400);
        frame.setTitle("An Empty Frame");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

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



## 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)
    {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;
        Rectangle box = new Rectangle(5, 10, 20, 30);
        g2.draw(box);
        . . .
    }
}
```

# Import Required Classes

---

```
import java.awt.Graphics;  
import java.awt.Graphics2D;  
import java.awt.Rectangle;  
import javax.swing.JComponent;
```

***Continued***

# Creating a Component Object

---

```
/**
    A component that draws two rectangles.
 */
public class RectangleComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;

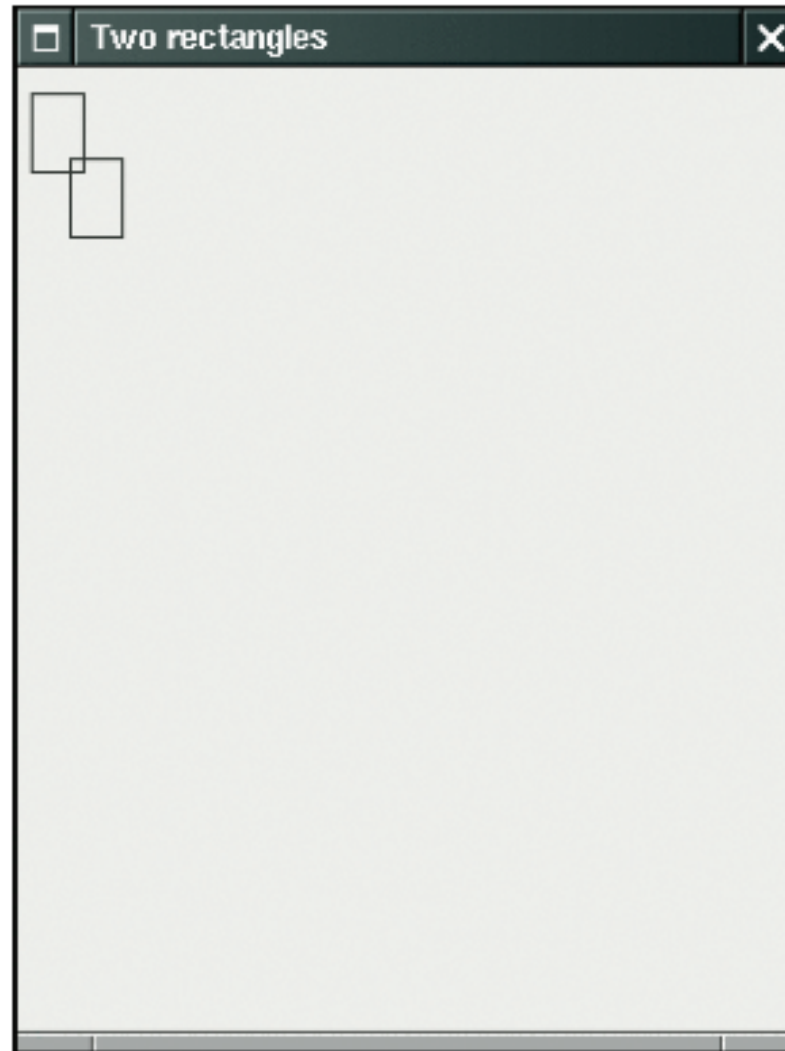
        // Construct a rectangle and draw it
        Rectangle box = new Rectangle(5, 10, 20, 30);
        g2.draw(box);

        // Move rectangle 15 units to the right and 25 units down
        box.translate(15, 25);

        // Draw moved rectangle
        g2.draw(box);
    }
}
```

# Drawing **Two** Rectangles in a Frame

---



**Figure 23**  
Drawing Rectangles



# Using a Component Within a Frame

---

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.

# Main Method to Display Frame with Components

---

```
import javax.swing.JFrame;

public class RectangleViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();

        frame.setSize(300, 400);
        frame.setTitle("Two rectangles");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        RectangleComponent component = new RectangleComponent();
        frame.add(component);

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

## 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>
```

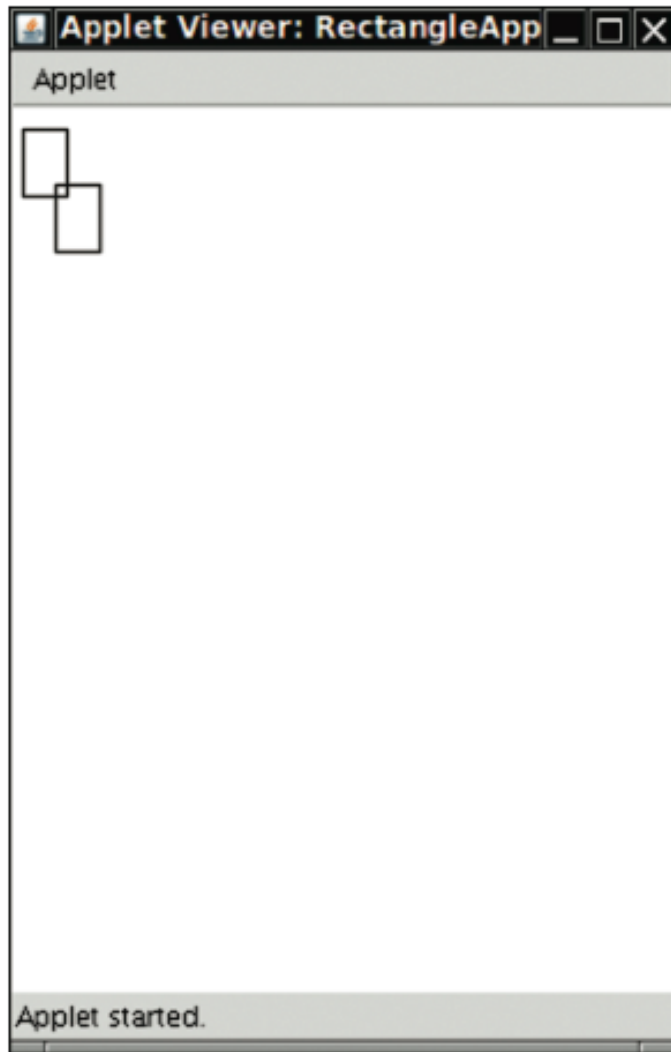
# ch02/applet/RectangleAppletExplained.html

---

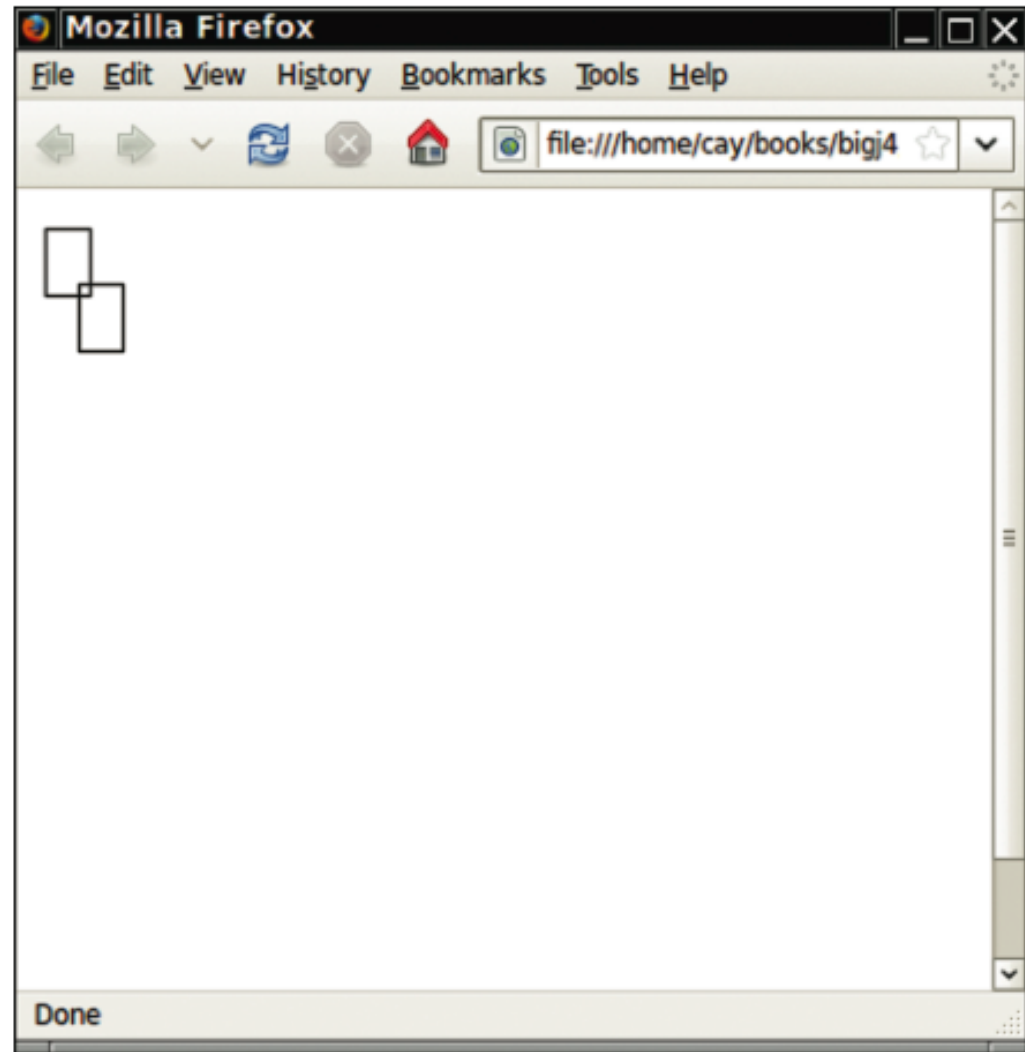
```
1 <html>
2   <head>
3     <title>Two rectangles</title>
4   </head>
5   <body>
6     <p>Here is my <i>first applet</i>:</p>
7     <applet code="RectangleApplet.class" width="300" height="400">
8     </applet>
9   </body>
10 </html>
```

# 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:

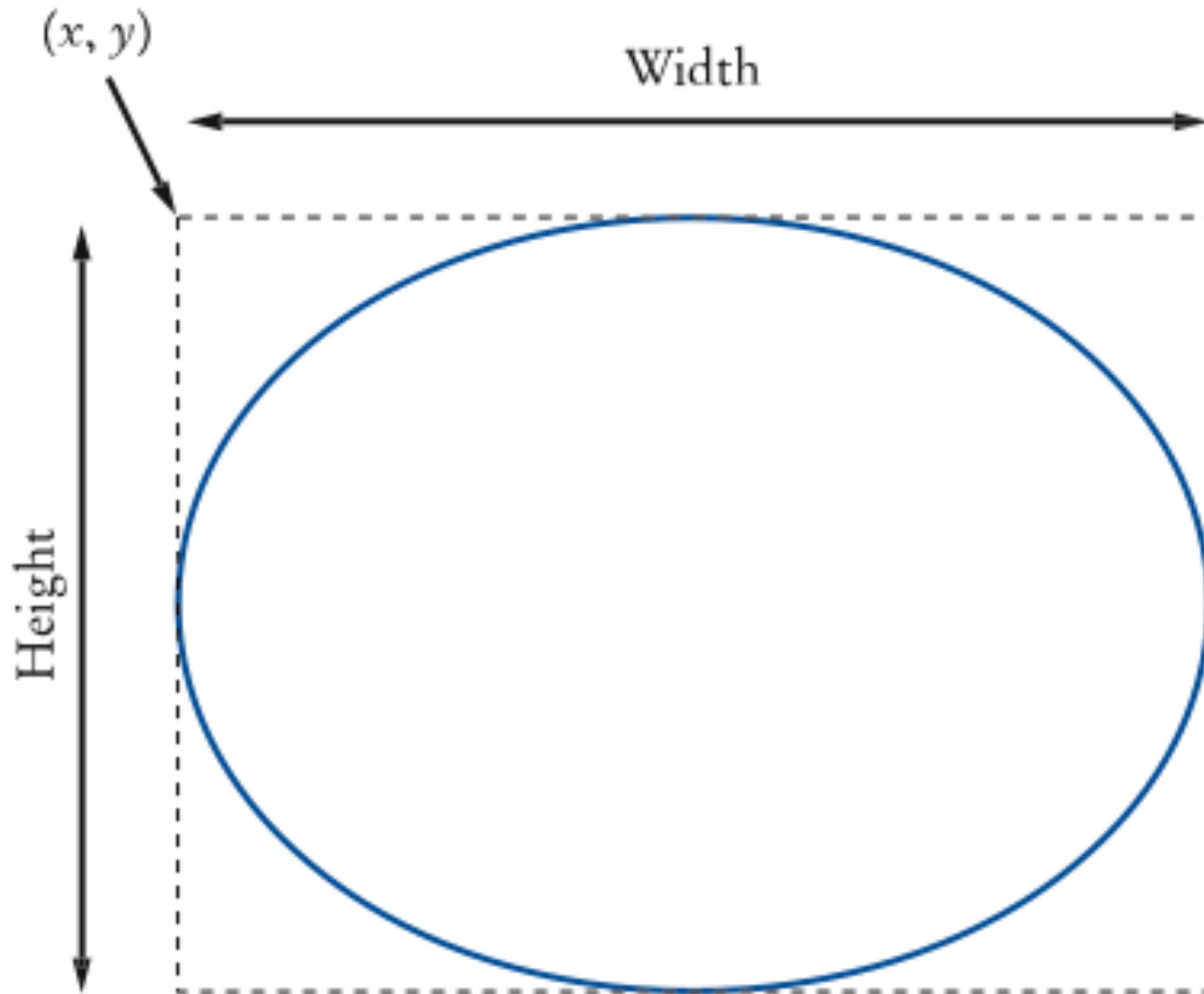
```
import java.awt.geom.Ellipse2D; // no .Double
```

- 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:

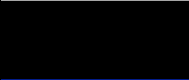


```
g2.setColor(magenta);
```

- 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

---

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import java.awt.geom.Ellipse2D;
import java.awt.geom.Line2D;
import javax.swing.JComponent;

/**
 * A component that draws an alien face
 */
public class FaceComponent extends JComponent
{
    public void paintComponent(Graphics g)
    {
        // Recover Graphics2D
        Graphics2D g2 = (Graphics2D) g;
```

***Continued***

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

---

```
// Draw the head
Ellipse2D.Double head = new Ellipse2D.Double(5, 10, 100, 150);
g2.draw(head);

// Draw the eyes
g2.setColor(Color.GREEN);
Rectangle eye = new Rectangle(25, 70, 15, 15);
g2.fill(eye);
eye.translate(50, 0);
g2.fill(eye);

// Draw the mouth
Line2D.Double mouth = new Line2D.Double(30, 110, 80, 110);
g2.setColor(Color.RED);
g2.draw(mouth);

// Draw the greeting
g2.setColor(Color.BLUE);
g2.drawString("Hello, World!", 5, 175);
}
}
```

## ch02/face/FaceViewer.java

---

```
import javax.swing.JFrame;

public class FaceViewer
{
    public static void main(String[] args)
    {
        JFrame frame = new JFrame();
        frame.setSize(150, 250);
        frame.setTitle("An Alien Face");
        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

        FaceComponent component = new FaceComponent();
        frame.add(component);

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

## 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));
```