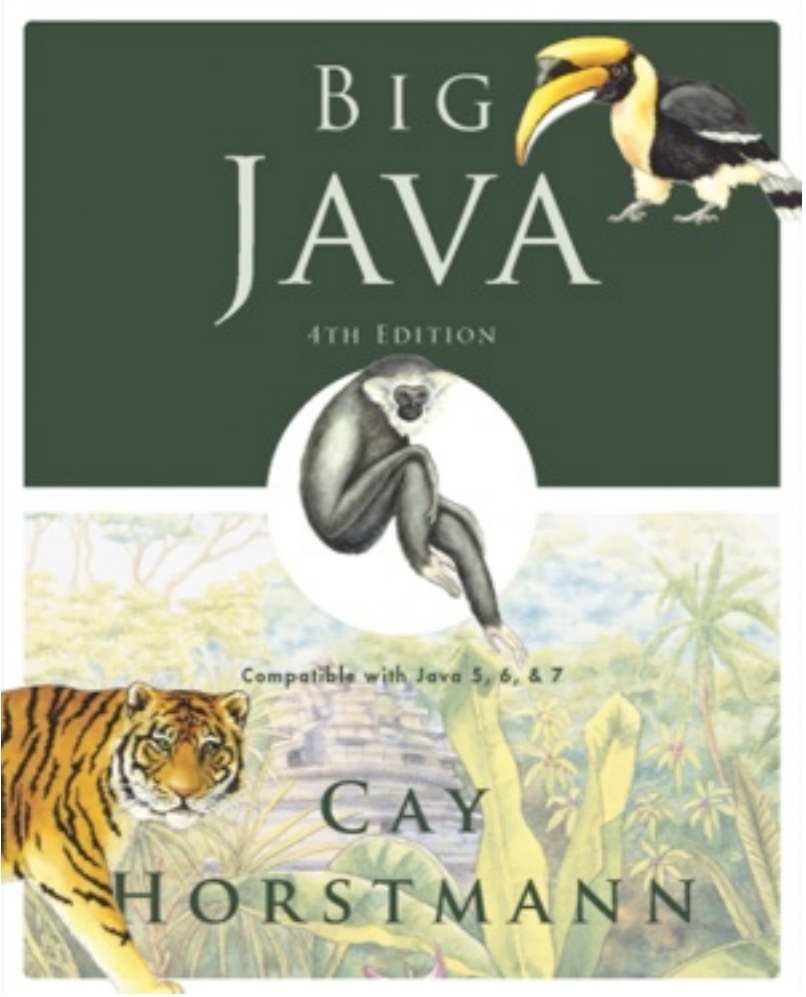


Chapter 2 – Using Objects

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 `*`

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

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

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

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

A variable declaration ends with a semicolon.

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

Use a descriptive variable name.

Variable Names

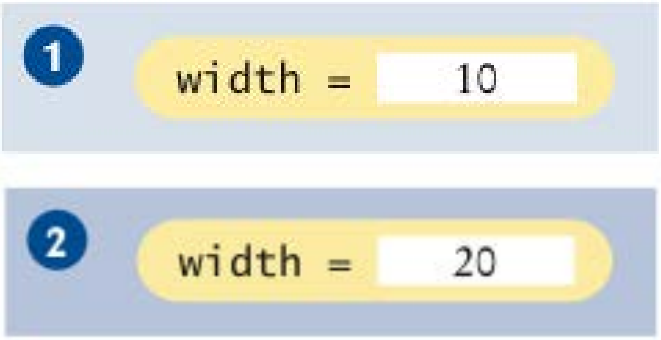
Table 3 Variable Names in Java

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.

The Assignment Operator

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

```
int width= 10; ❶
width = 20; ❷
```



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

```
double width = 30;
.
.
width = 30;
.
.
.
width = width + 10;
```

This is a variable declaration.

This is an assignment statement.

The value of this variable is changed.

The new value of the variable

The same name can occur on both sides. See Figure 3.

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`

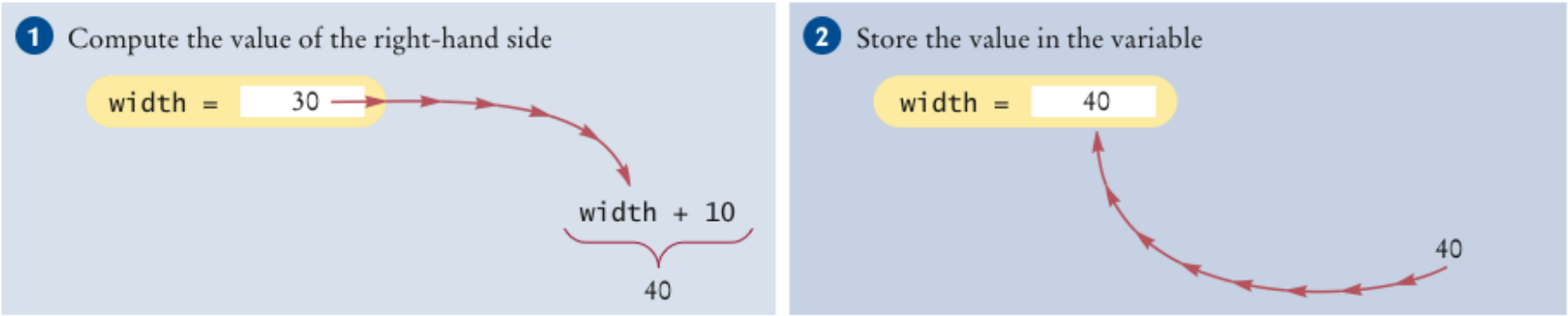


Figure 3 Executing the Statement `width = width + 10`

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`

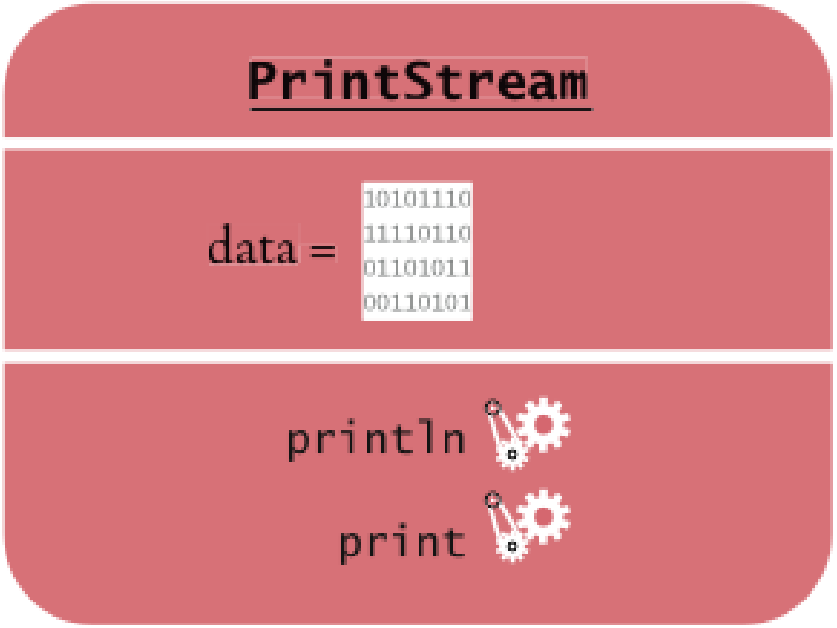


Figure 4 Representation of the `System.out` Object

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

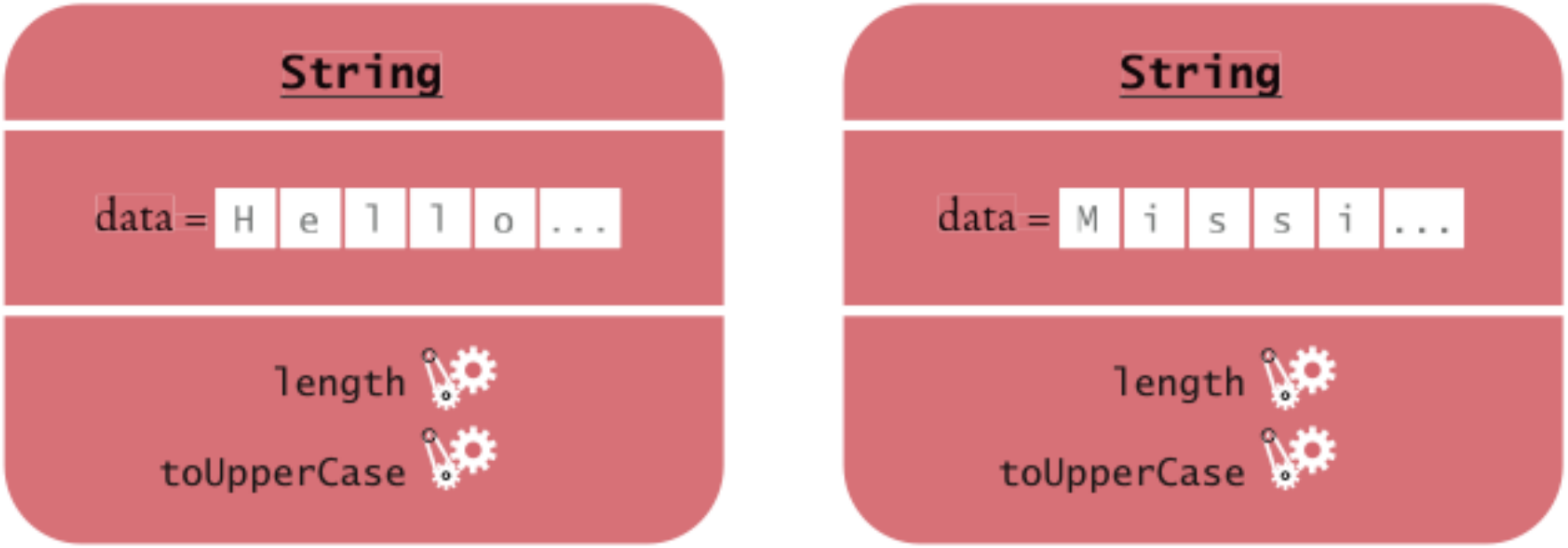


Figure 5 A Representation of Two `String` Objects

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
```

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

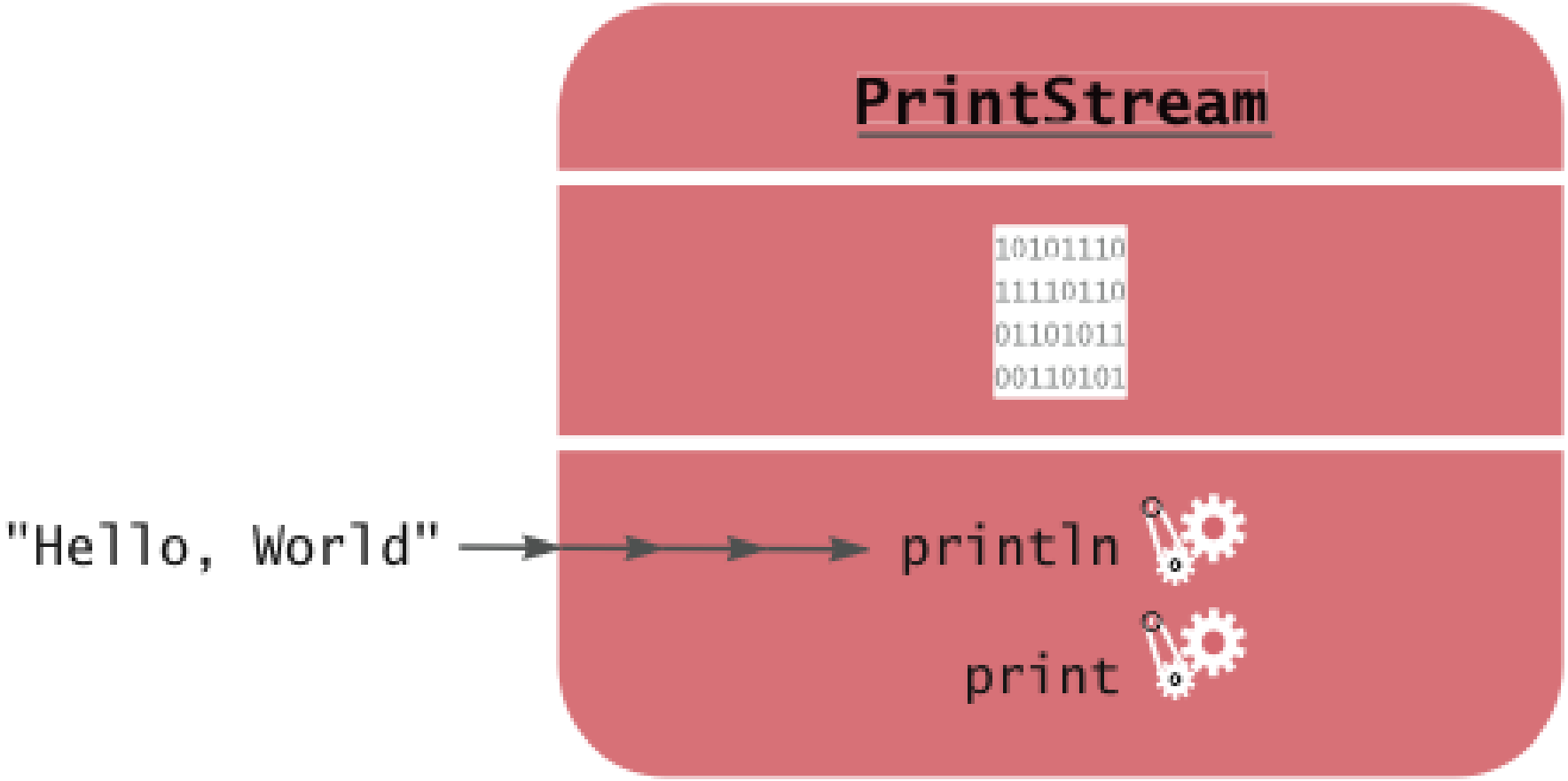


Figure 6 Passing a Parameter to the `println` Method

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
```

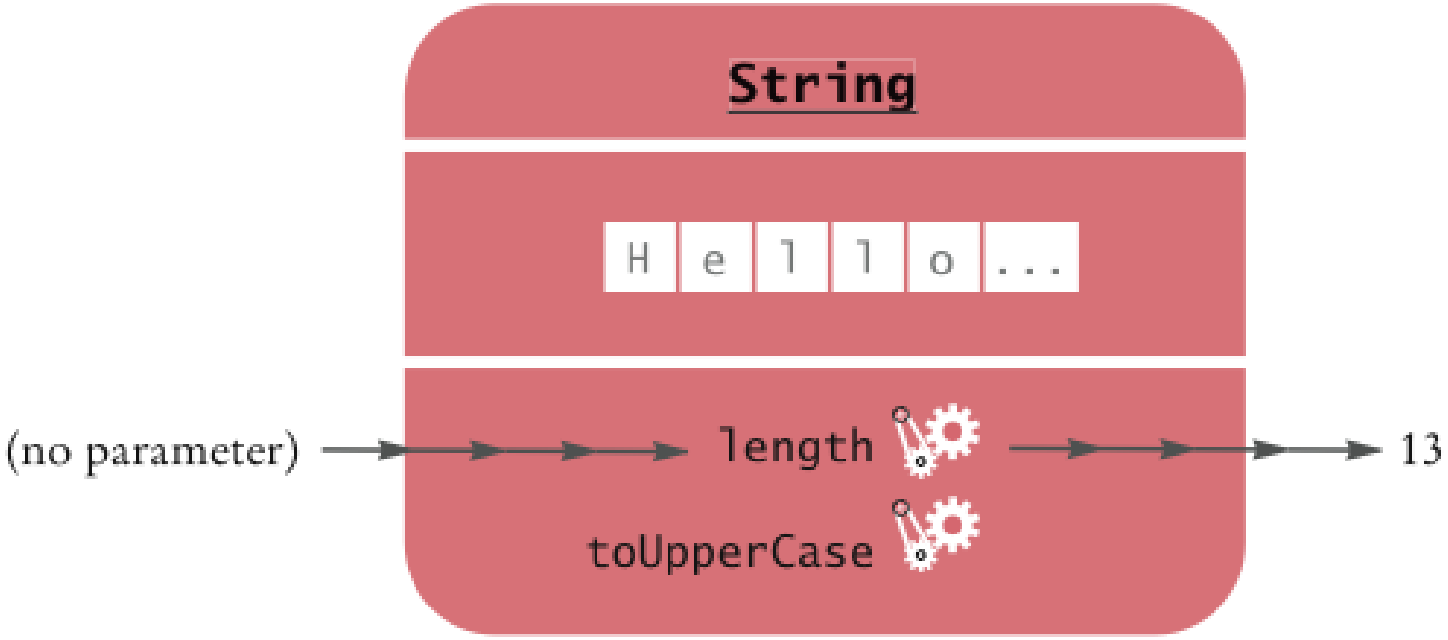


Figure 7 Invoking the `length` Method on a `String` Object

Passing Return Values

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

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

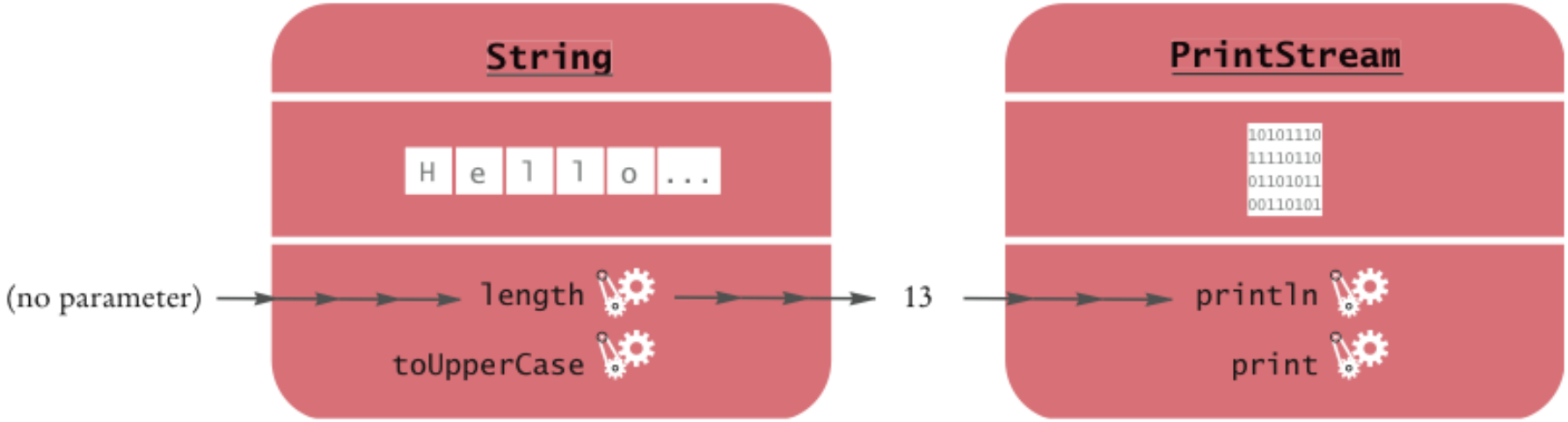


Figure 8 Passing the Result of a Method Call to Another Method

- 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")
```

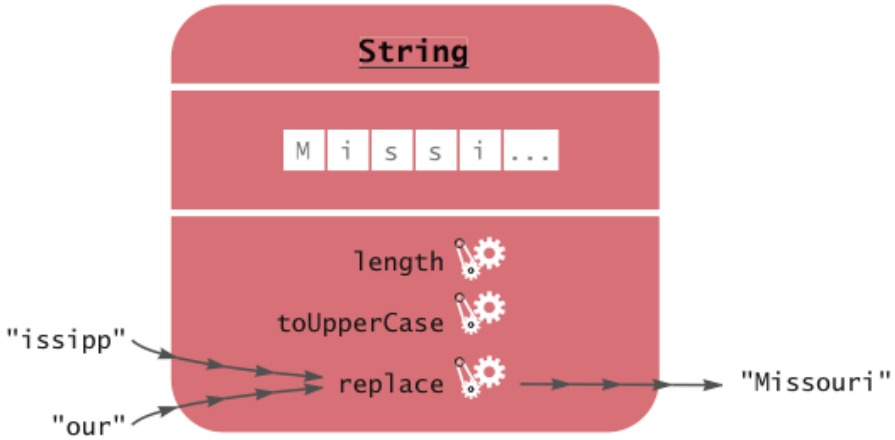


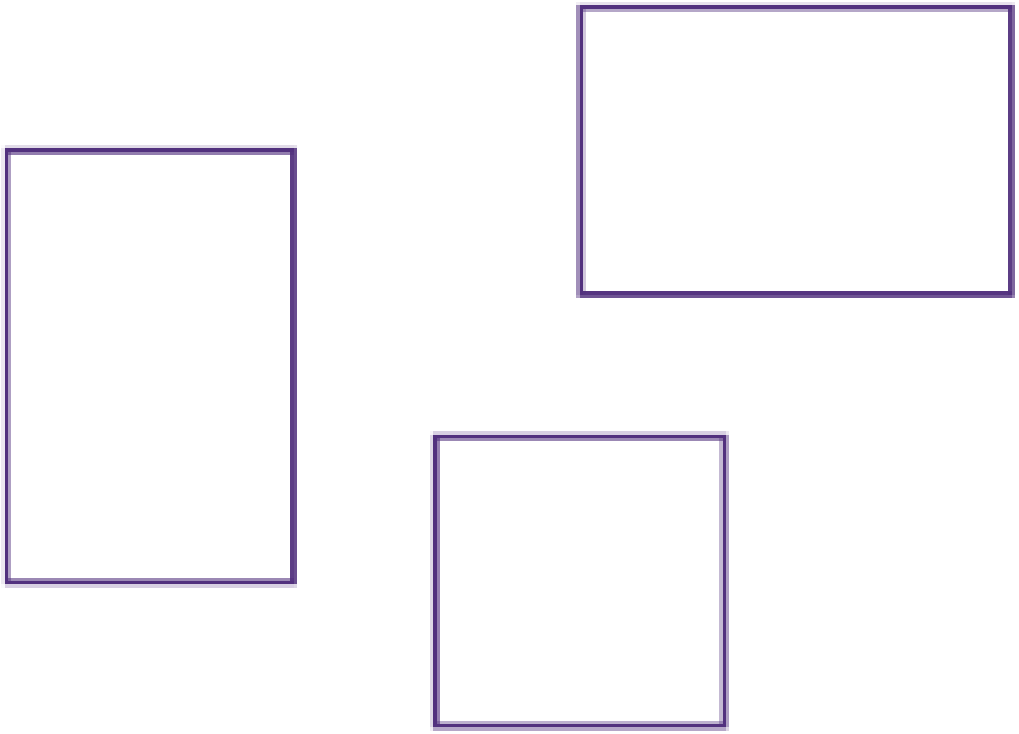
Figure 9 Calling the `replace` Method

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

Rectangular Shapes and Rectangle Objects

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

Figure 10
Rectangular Shapes



Rectangular Shapes and Rectangle Objects

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

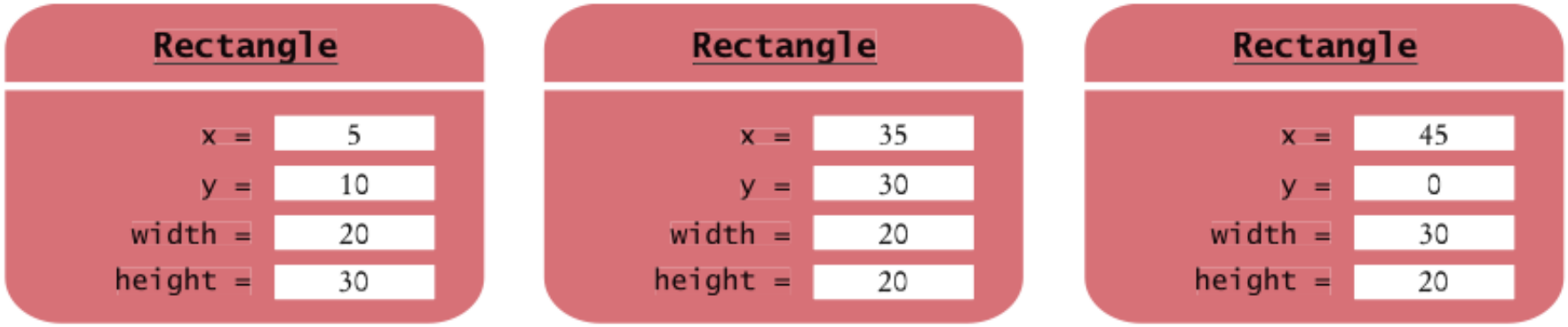


Figure 11 Rectangle Objects

Constructing Objects

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

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

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

Constructing Objects

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

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

Syntax 2.3 Object Construction

Syntax new *ClassName(parameters)*

Example

The new expression yields an object.

Construction parameters

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

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.

Accessor and Mutator Methods

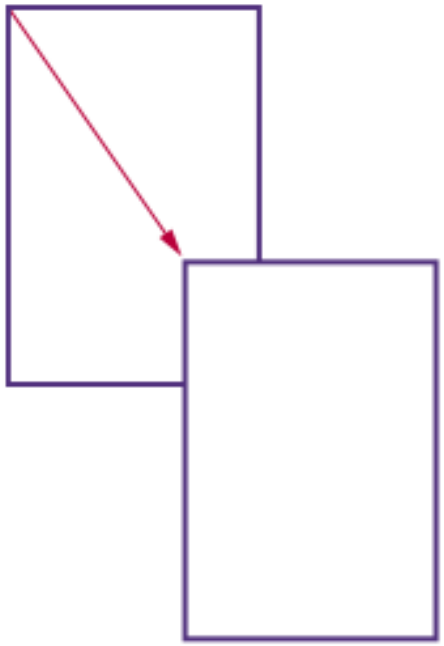
- **Accessor method**: does not change the state of its implicit parameter:

```
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



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

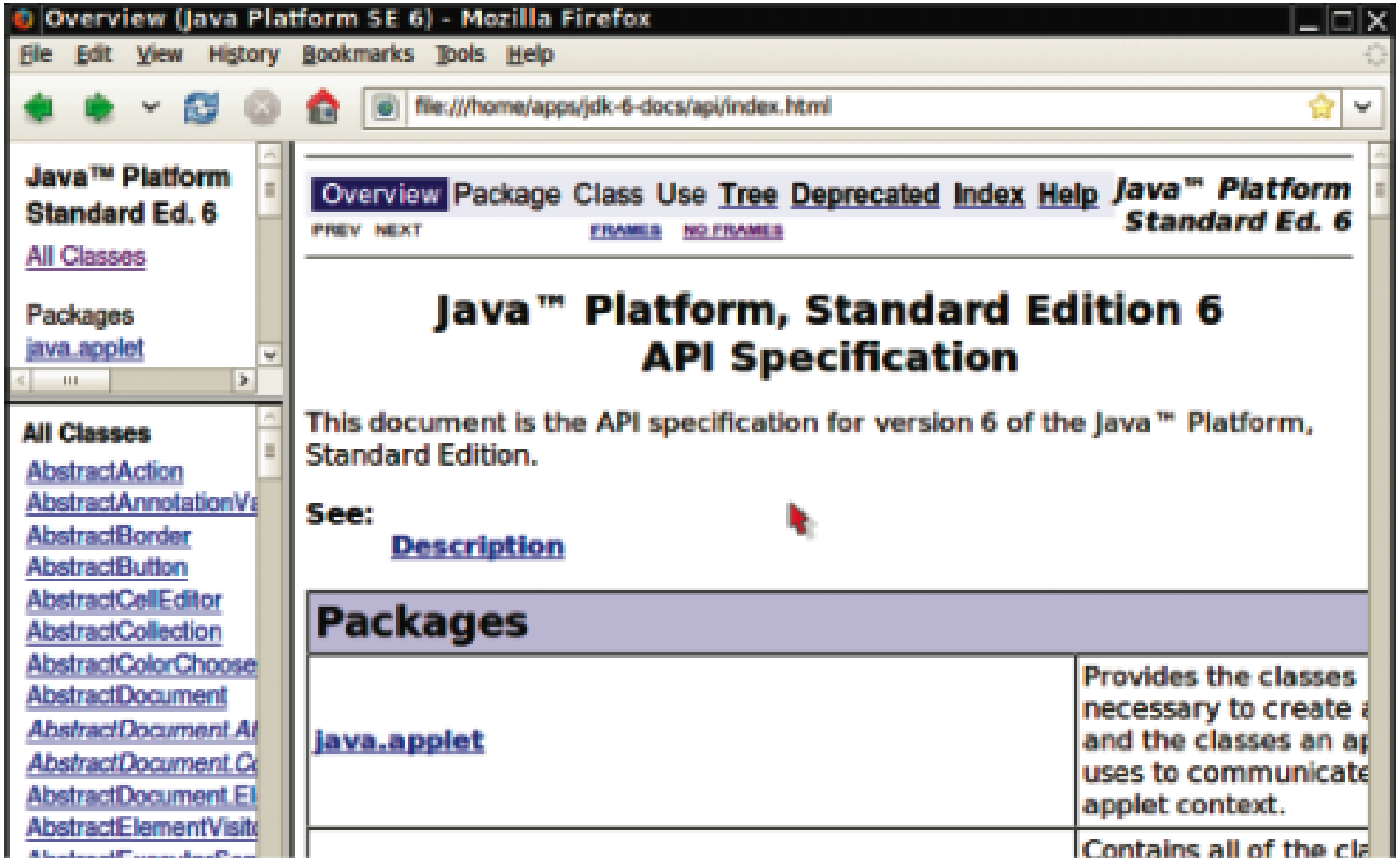


Figure 13 The API Documentation of the Standard Java Library

The API Documentation for the Rectangle Class

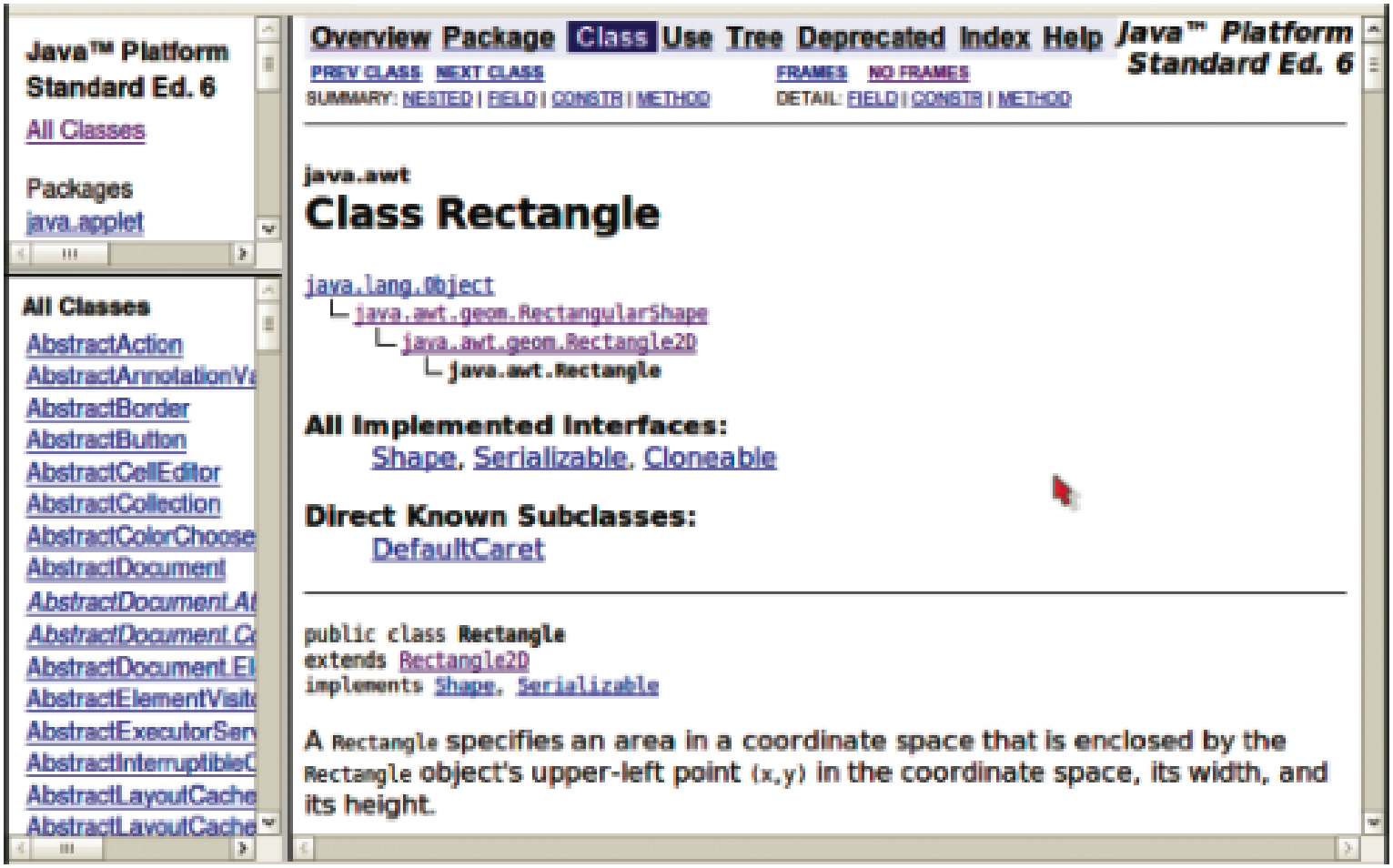


Figure 14 The API Documentation for the Rectangle Class

Method Summary

Java™ Platform Standard Ed. 6

All Classes

Packages

java.applet

All Classes

AbstractAction
AbstractAnnotationVa
AbstractBorder
AbstractButton
AbstractCellEditor
AbstractCollection
AbstractColorChoose
AbstractDocument
AbstractDocument.At
AbstractDocument.Co
AbstractDocument.El
AbstractElementVisito
AbstractExecutorSer

Method Summary

void	add(int newx, int newy) Adds a point, specified by the integer arguments newx,newy to the bounds of this Rectangle.
void	add(Point pt) Adds the specified Point to the bounds of this Rectangle.
void	add(Rectangle r) Adds a Rectangle to this Rectangle.
boolean	contains(int x, int y) Checks whether or not this Rectangle contains the point at the specified location (x,y).
boolean	contains(int X, int Y, int W, int H) Checks whether this Rectangle entirely contains the Rectangle at the specified location (X,Y) with the specified dimensions (W,H).
boolean	contains(Point p) Checks whether or not this Rectangle contains the specified Point.
boolean	contains(Rectangle r) Checks whether or not this Rectangle entirely contains the specified Rectangle.

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)

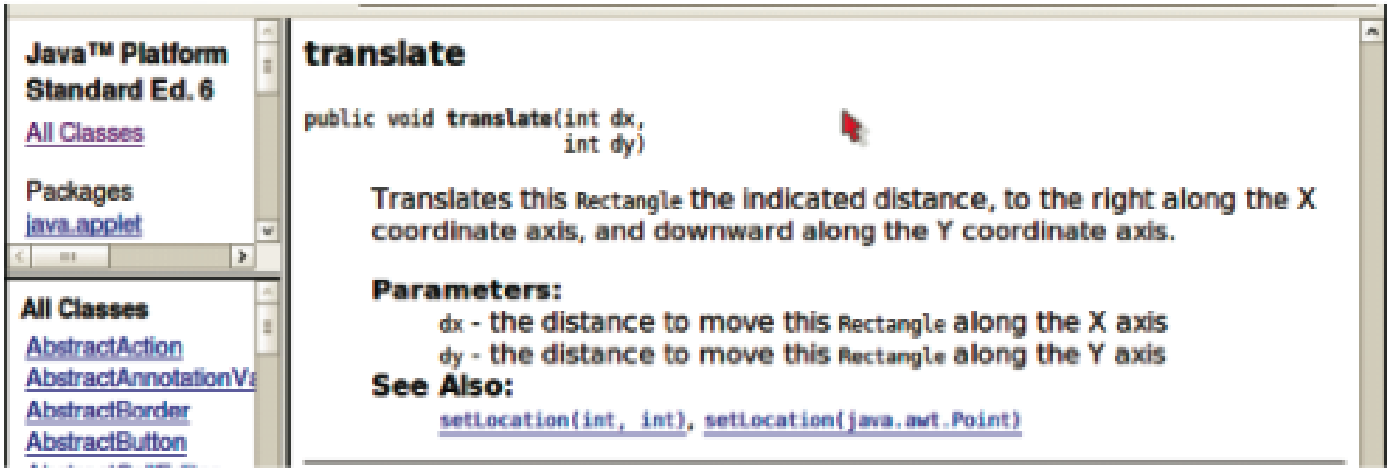


Figure 16 The API Documentation of the `translate` Method

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

Package name

Class name

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

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

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

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

```
import java.awt.Rectangle;

public class MoveTester
{
   public static void main(String[] args)
   {
      Rectangle box = new Rectangle(5, 10, 20, 30);

      // Move the rectangle
      box.translate(15, 25);

      // Print information about the moved rectangle
      System.out.print("x: ");
      System.out.println(box.getX());
      System.out.println("Expected: 20");

      System.out.print("y: ");
      System.out.println(box.getY());
      System.out.println("Expected: 35");
   }
}
```

ch02/rectangle/MoveTester.java (cont.)

Program Run:

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

Object References

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

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

- Multiple object variables can refer to the same object:

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

- Primitive type variables ≠ object variables

Object Variables and Number Variables

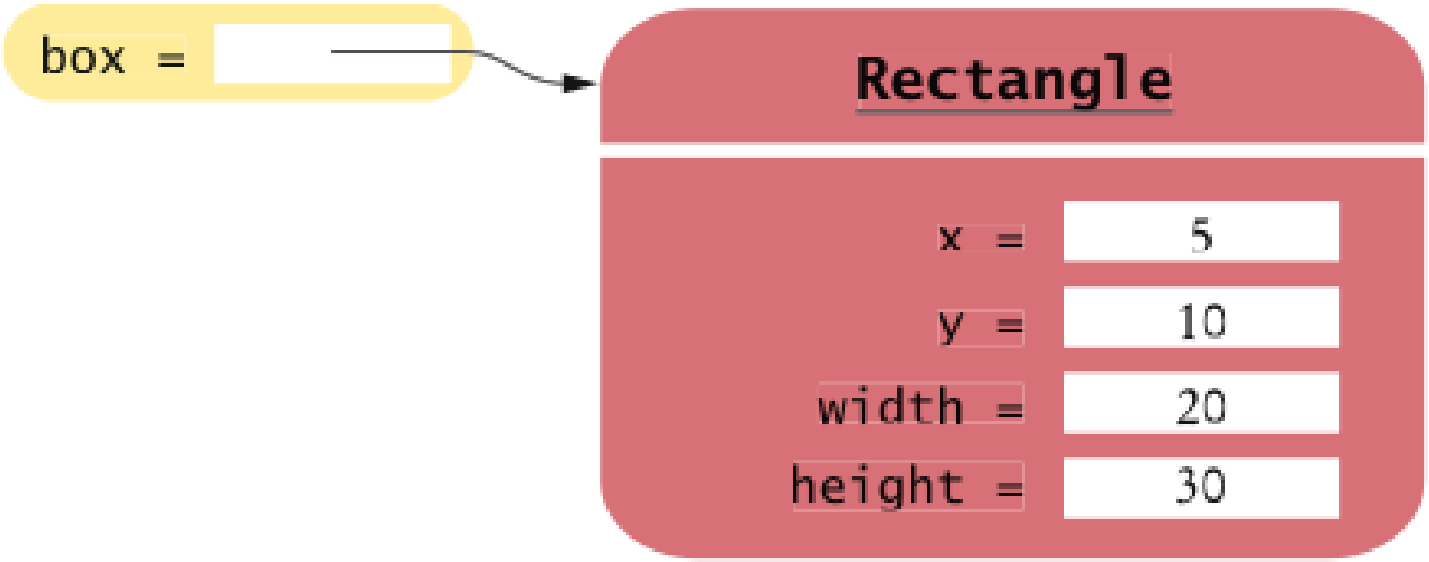


Figure 17 An Object Variable Containing an Object Reference

Object Variables and Number Variables

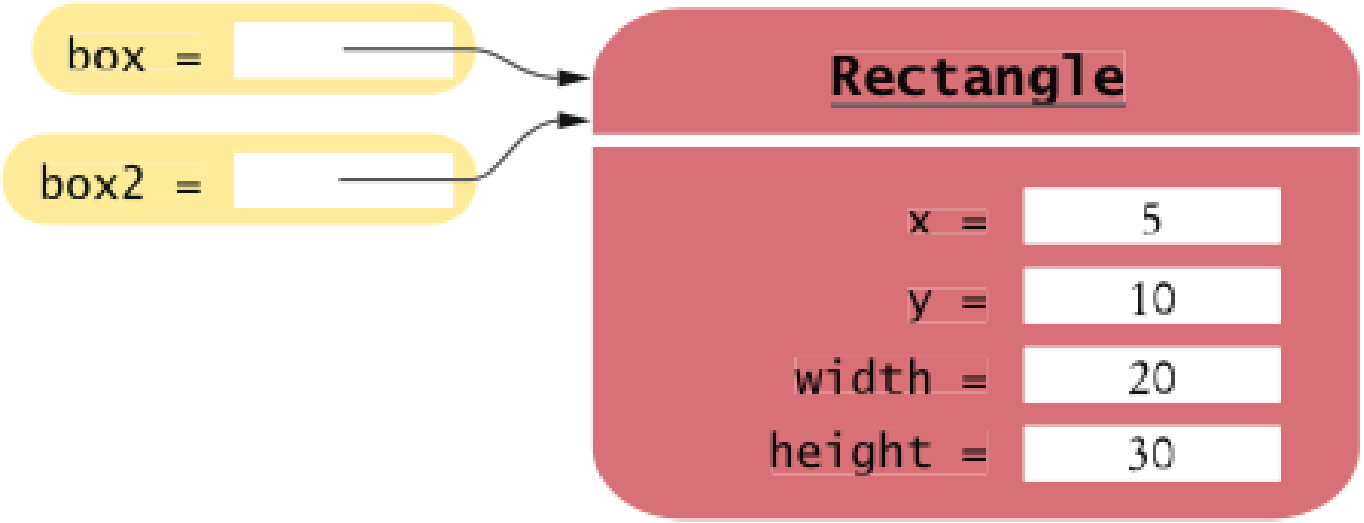


Figure 18 Two Object Variables Referring to the Same Object

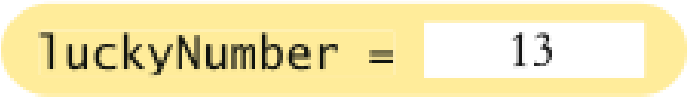
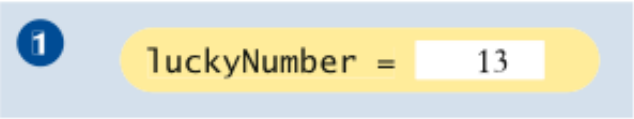


Figure 19 A Number Variable Stores a Number

Copying Numbers

```
int luckyNumber = 13;
```

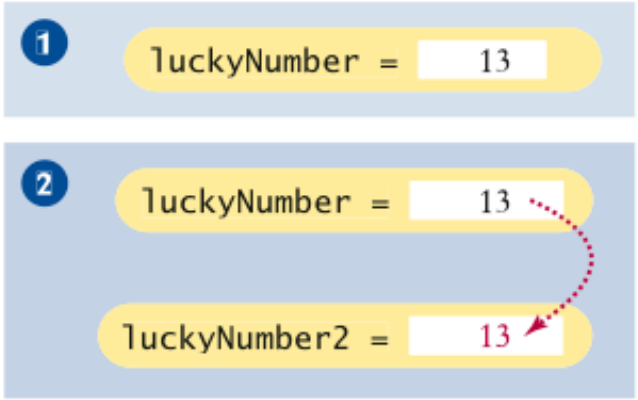
Figure 20
Copying Numbers



Copying Numbers (cont.)

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

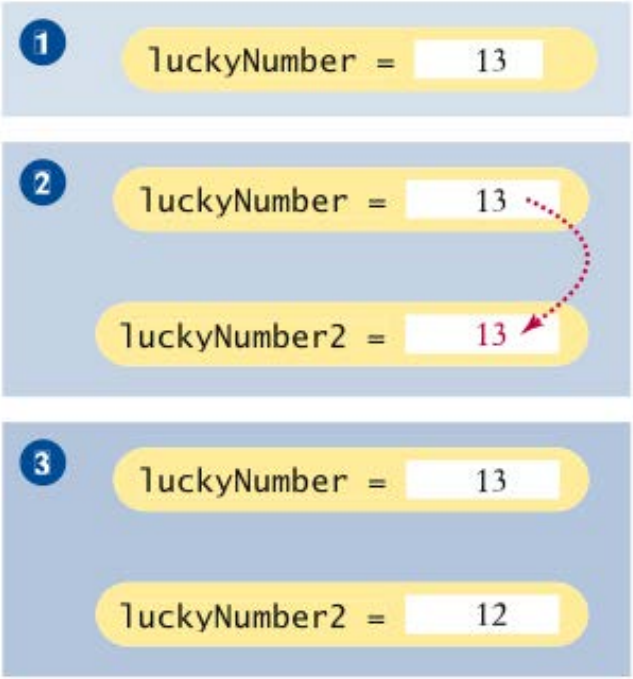
Figure 20
Copying Numbers



Copying Numbers (cont.)

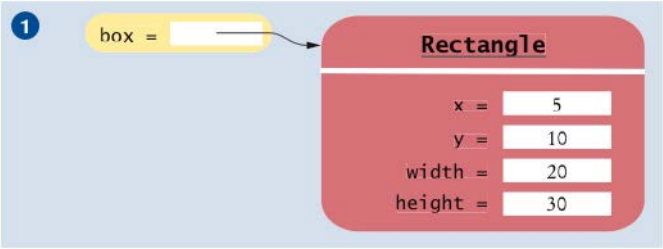
```
int luckyNumber = 13;  ❶
int luckyNumber2 = luckyNumber;  ❷
luckyNumber2 = 12;  ❸
```

Figure 20
Copying Numbers



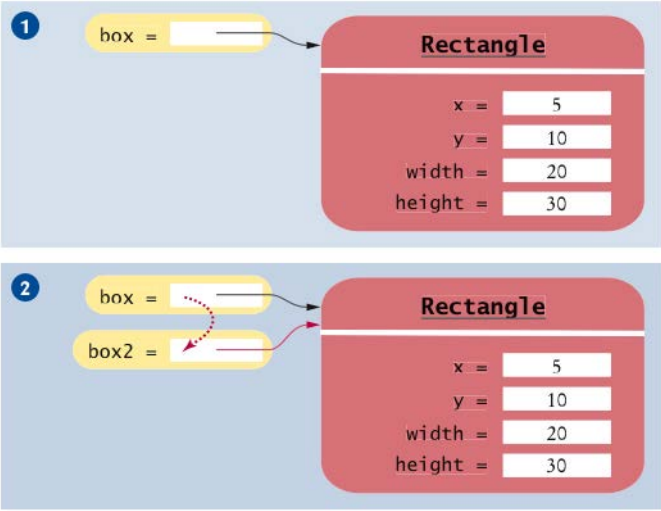
Copying Object References

```
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);  ❶
Rectangle box2 = box;  ❷
Box2.translate(15, 25);  ❸
```

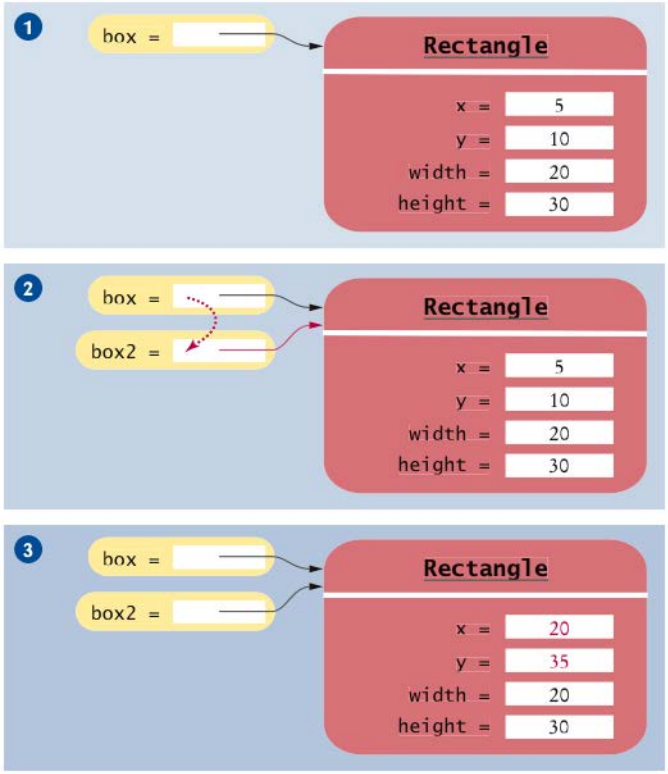


Figure 21 Copying Object References