Classes and Objects

Advanced Programming

ICOM 4015

Lecture 2

Reading: Java Concepts Chapter 2

Lecture Goals

- To learn about variables
- To understand the concepts of classes and objects
- To be able to call methods
- To be able to browse the API documentation
- To realize the difference between objects and object references

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

Continued…

Identifiers

- By convention, variable names start with a lowercase letter
- By convention, class names start with an uppercase letter

Self Check

1. What is the type of the values `0`, `'0'` and `"0"`?
2. Which of the following are legal identifiers?

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

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

Answers

1. `int`, `char`, and `String`
2. Only the first two are legal identifiers
3.
```
String myName = "John Q. Public";
```

The Assignment Operator

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

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

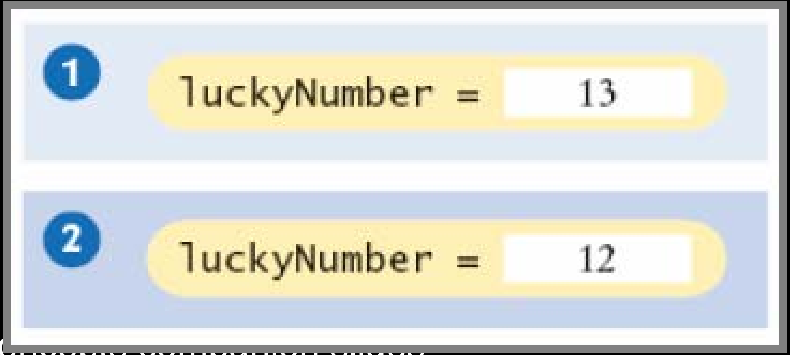


Figure 1:
Assigning a New Value to a Variable

Uninitialized Variables

- **Error:**

```
int luckyNumber;
System.out.println(luckyNumber);
   // ERROR - tryin to use and uninitialized variable
```

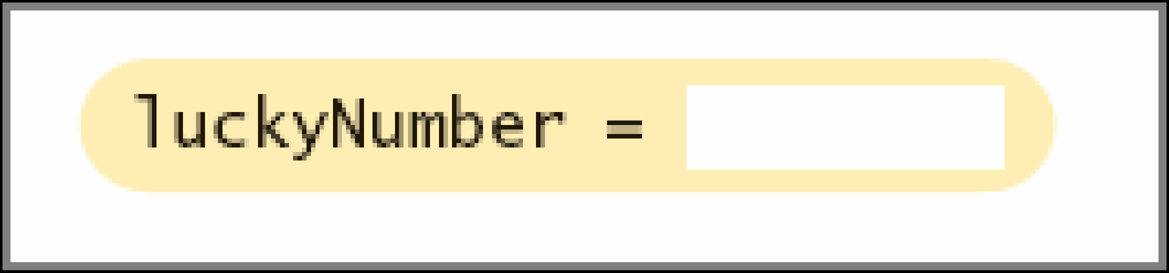


Figure 2:
An Uninitialized Object Variable

Syntax 2.2: Assignment

variableName = *value*;

Example:

```
luckyNumber = 12;
```

Purpose:

To assign a new value to a previously defined variable.

Self Check

1. Is `12 = 12` a valid expression in the Java language?
2. How do you change the value of the greeting variable to `"Hello, Nina!"`?

Answers

1. No, the left-hand side of the = operator must be a variable
2.
```
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`

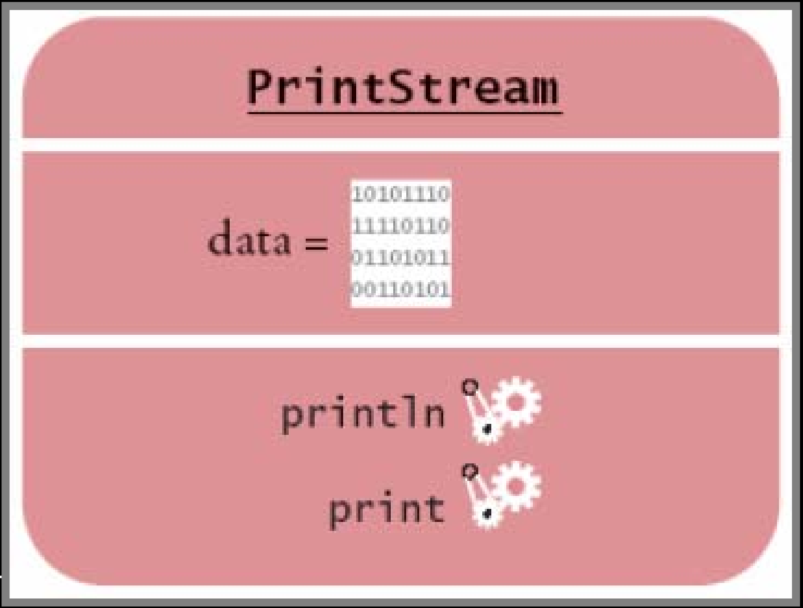


Figure 3:
Representation of the System.out object

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
```

Continued…

Methods

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

A Representation of Two String Objects

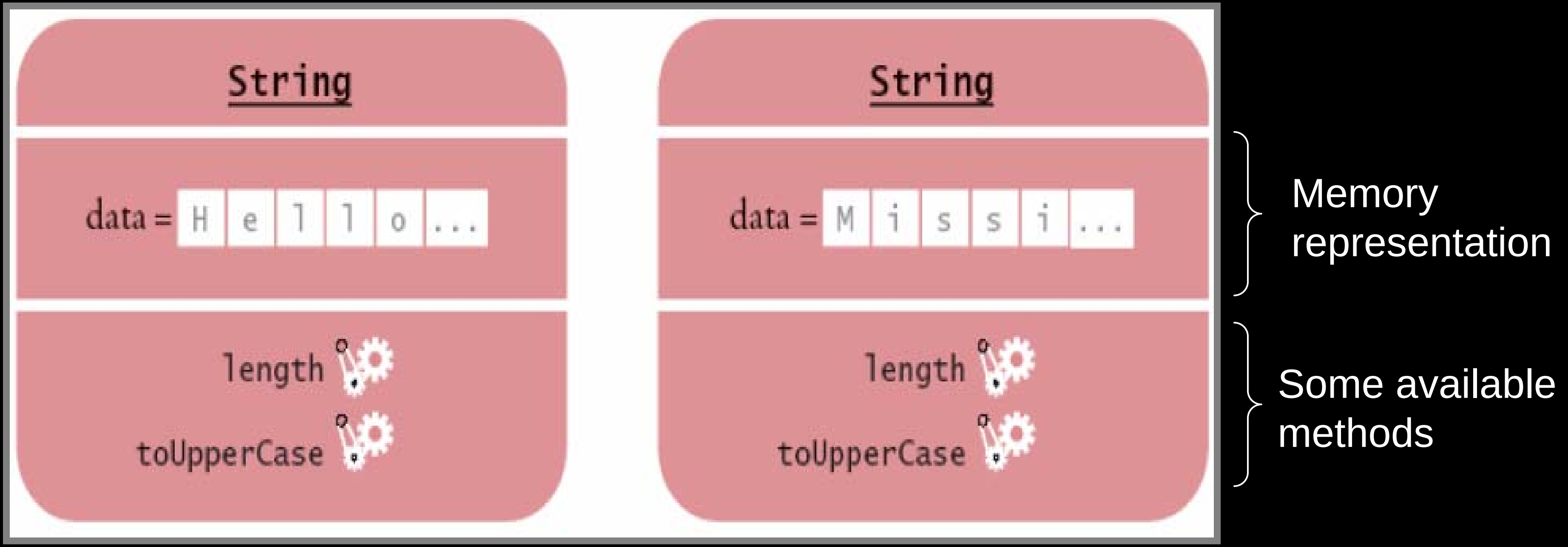


Figure 4:
A Representation of Two String Objects

String Methods

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

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

Continued…

String Methods

- **`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"
```

Continued…

String Methods

- **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
```

… since **`length()`** is not a valid or defined method that can be applied to this class of objects…
… in order to be so, it has to be explicitly defined somewhere as a valid method for such class, but it is not…

… in the other hand, it is a valid method for class **`String`**, because it has been explicitly included in this class…

Self Check

1. How can you compute the length of the string `"Mississippi"`?
2. How can you print out the uppercase version of `"Hello, World!"`?
3. Is it legal to call `river.println()`? Why or why not?

Answers

1.
```
river.length() or "Mississippi".length()
```
2.
```
System.out.println(greeting.toUpperCase());
```
3. 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
```

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

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

Continued…

Implicit and Explicit Parameters

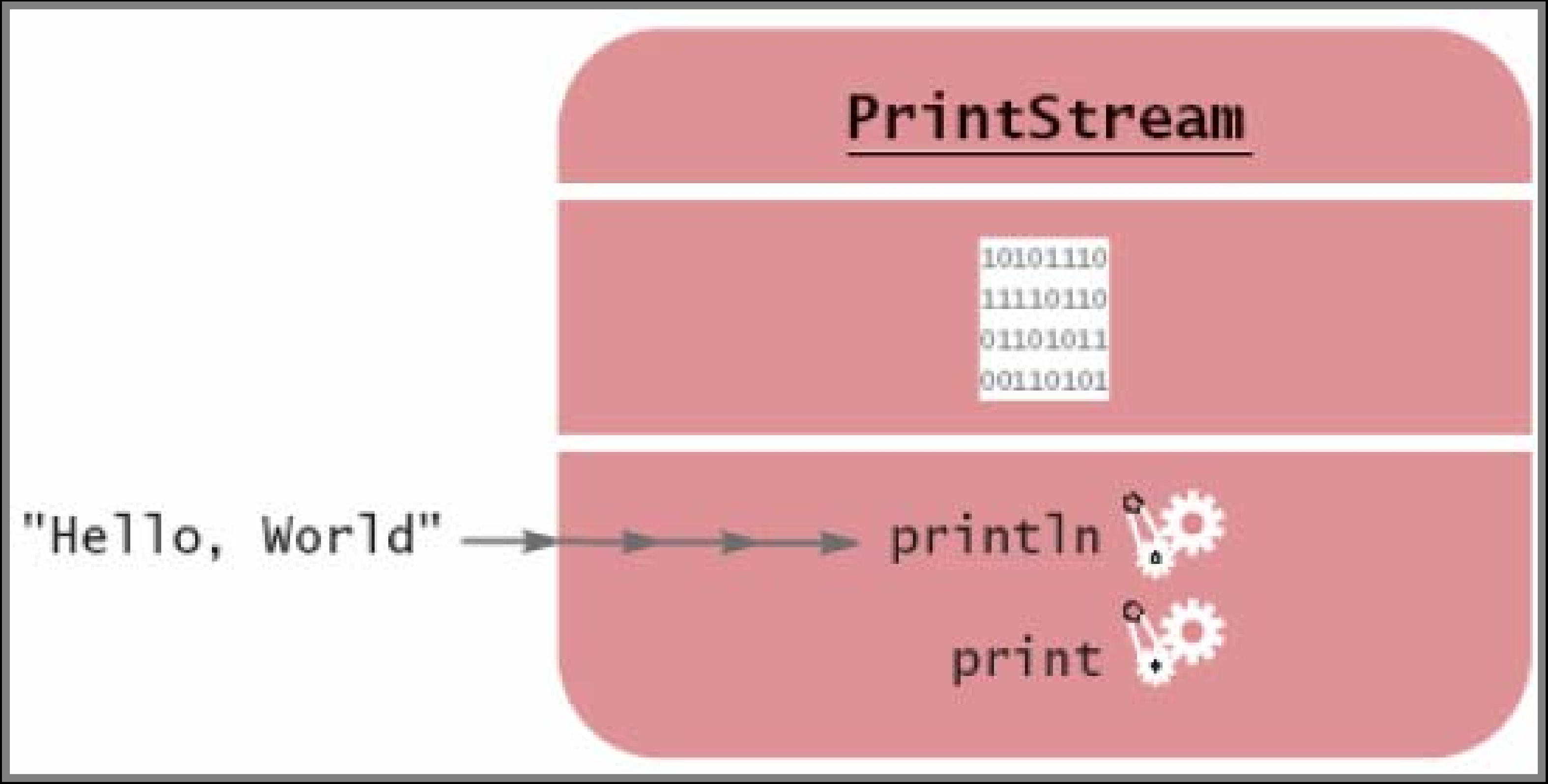


Figure 5:
Passing a parameter to the `println` method

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
```

Continued…

Return Values

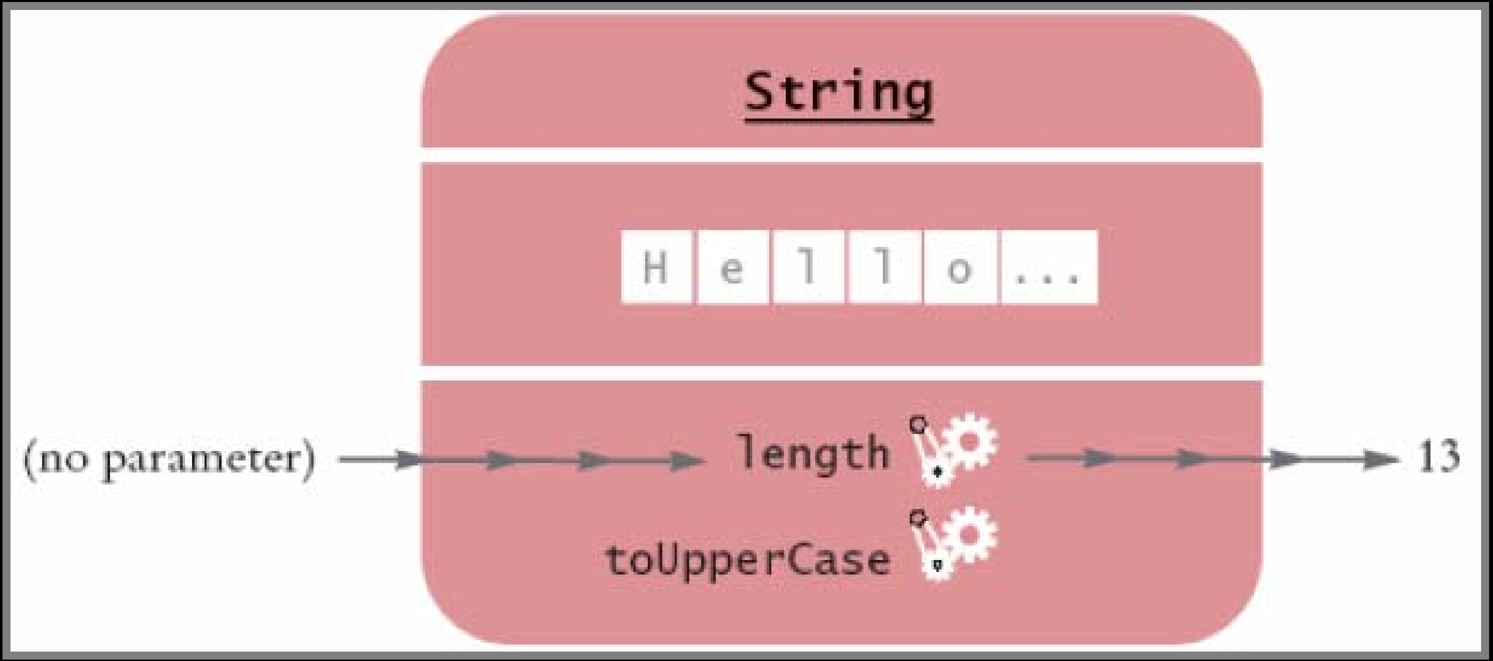


Figure 6:
Invoking the length Method on a String Object

Passing Return Values

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

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

- Not all methods return values. Example:

```
println
```

Continued…

Passing Return Values

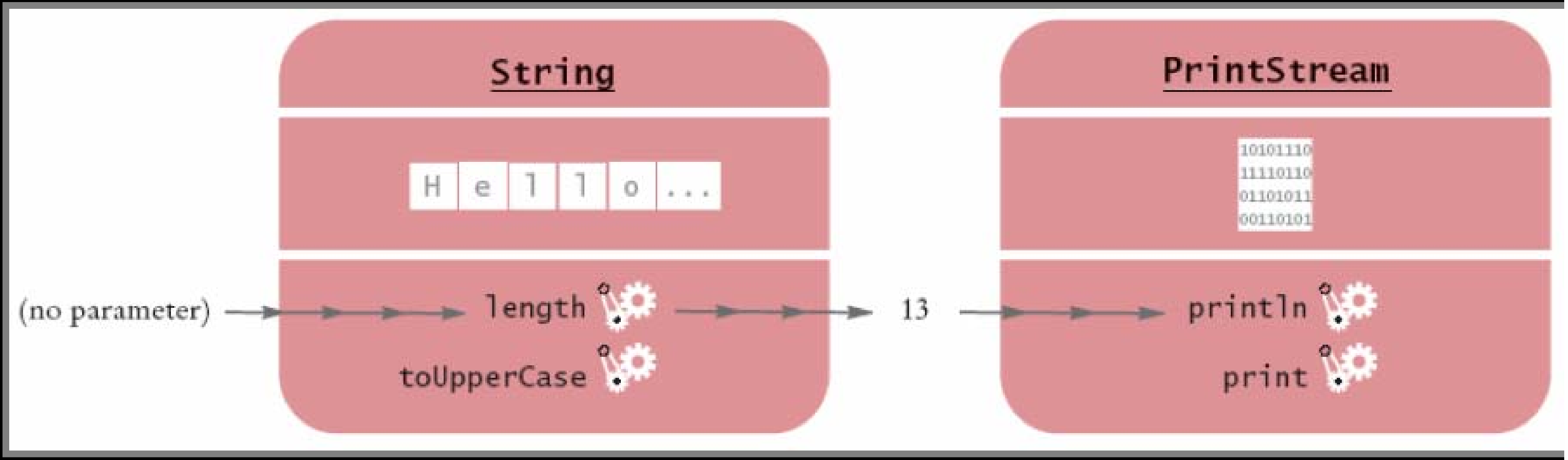


Figure 7:
Passing the Result of a Method Call to Another Method

A More Complex Call

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

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

- **As Figure 8 shows, this method call has**
  - one implicit parameter: the string **`"Mississippi"`**
  - two explicit parameters: the strings **`"issipp"`** and **`"our"`**
  - a return value: the string **`"Missouri"`**

Continued…

A More Complex Call

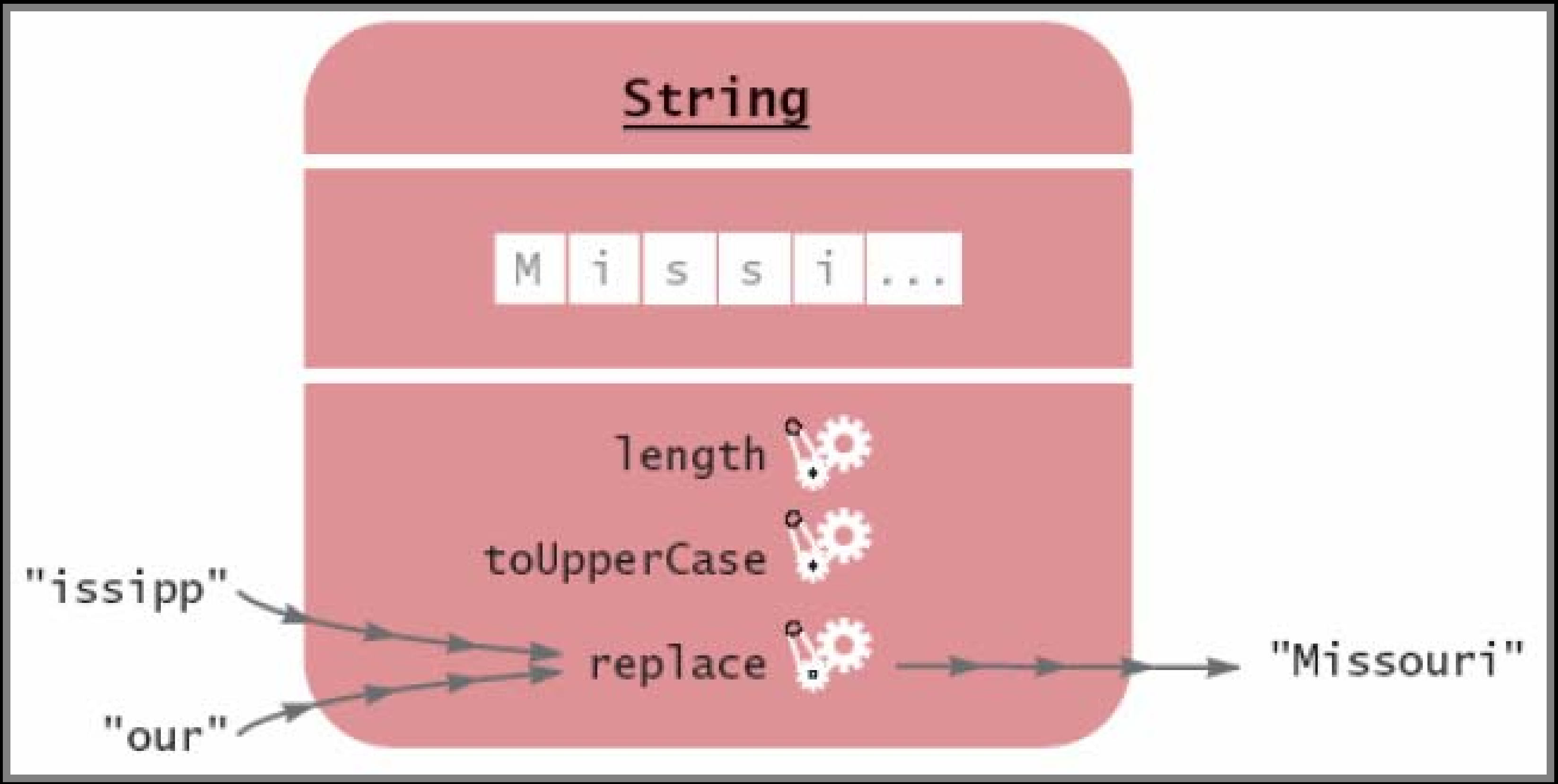


Figure 8:
Calling the `replace` Method

Method Definitions

- Method definition specifies types of explicit parameters and return value
- Type of implicit parameter = current class; not mentioned in method definition

Continued…

Method Definitions

- 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

1. What are the implicit parameters, explicit parameters, and return values in the method call `river.length()`?
2. What is the result of the call `river.replace("p", "s")`?
3. What is the result of the call `greeting.replace("World", "Dave").length()`?
4. How is the `toUpperCase` method defined in the `String` class?

Answers

1. The implicit parameter is `river`. There is no explicit parameter. The return value is `11`
2. `"Missississi"`
3. 12
4. 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`

Continued…

Number Types

- 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

1. Which number type would you use for storing the area of a circle?
2. Why is the expression `13.println()` an error?
3. Write an expression to compute the average of the values `x` and `y`.

Answers

1. `double`
2. An `int` is not an object, and you cannot call a method on it
3.
```
(x + y) * 0.5
```

Rectangular Shapes and Rectangle Objects

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

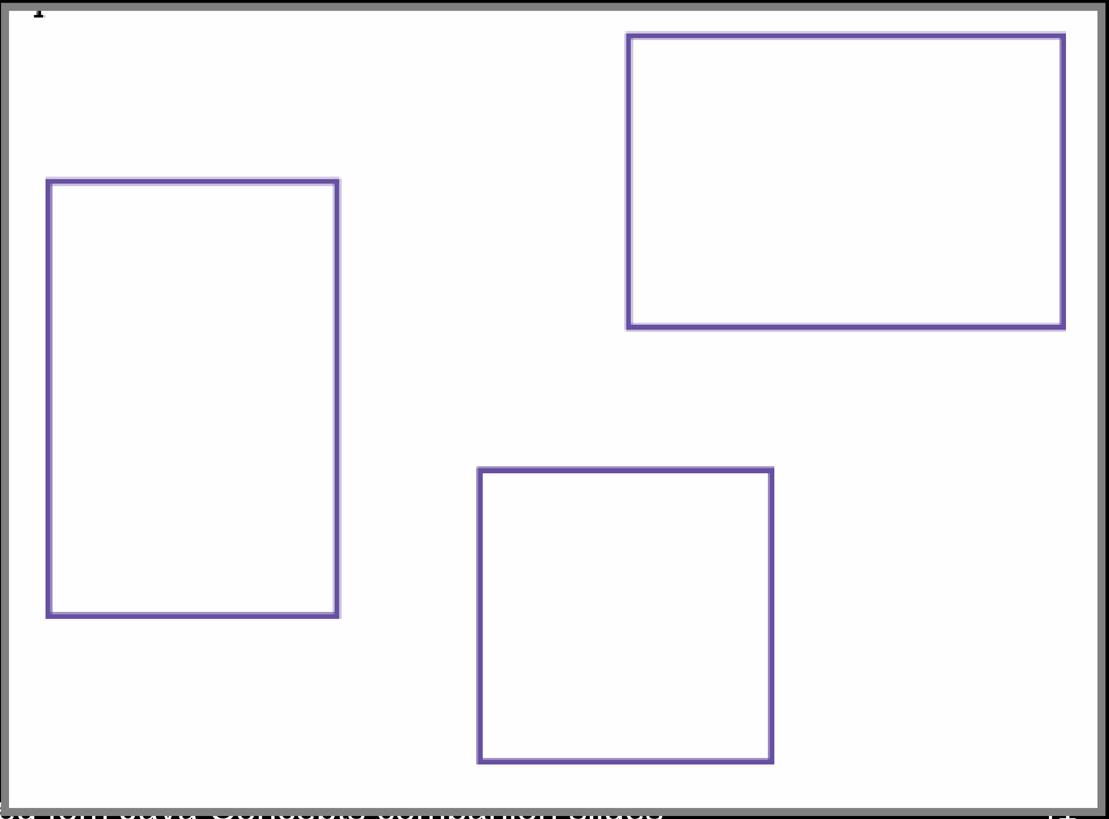


Figure 9: Rectangular Shapes

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

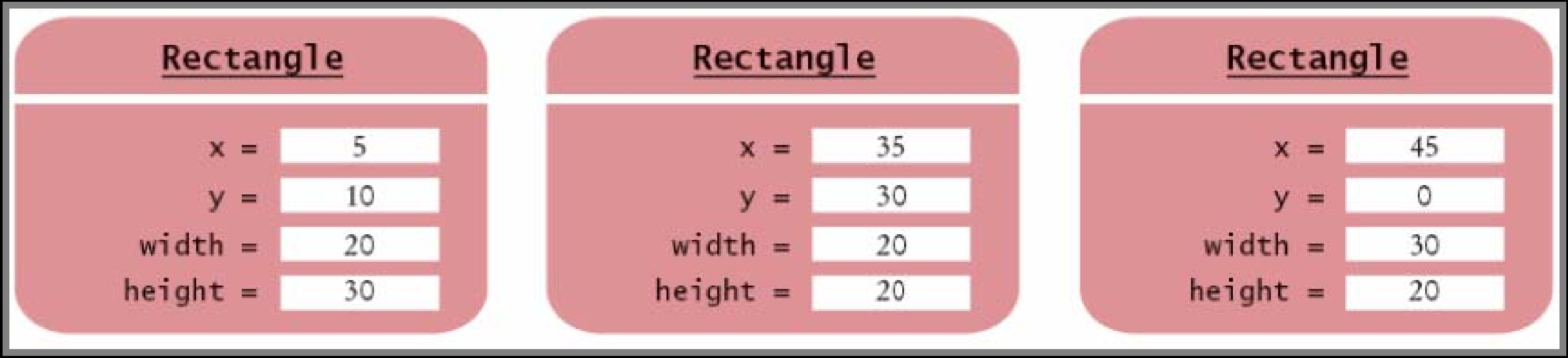


Figure 10: Rectangular Objects

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

1. How do you construct a square with center (100, 100) and side length 20?
2. What does the following statement print?

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

Answers

1. `new Rectangle(90, 90, 20, 20)`
2. 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);
```

Accessor and Mutator Methods

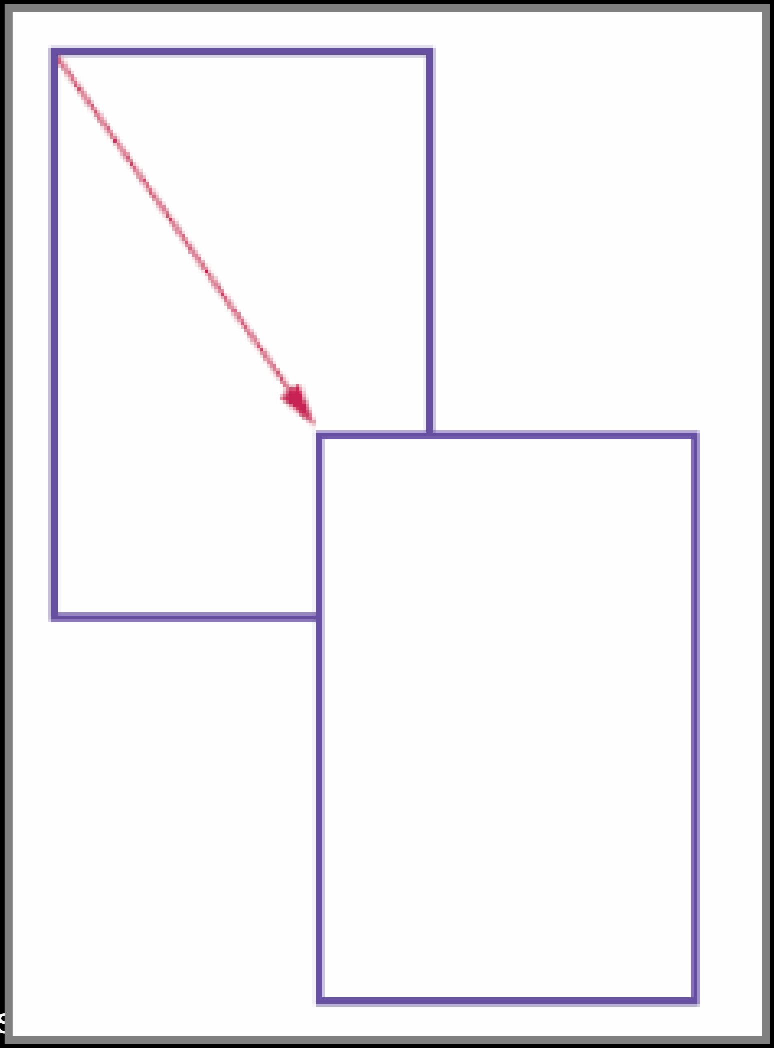


Figure 11:
Using the `translate` Method to Move a Rectangle

Self Check

1. Is the `toUpperCase` method of the `String` class an accessor or a mutator?
2. Which call to `translate` is needed to move the `box` rectangle so that its top-left corner is the origin (0, 0)?

Answers

1. An accessor–it doesn't modify the original string but returns a new string with uppercase letters
2. `box.translate(-5, -10),` provided the method is called immediately after storing the new rectangle into box

Implementing a Test Program

- Provide a new class
- Supply a `main` method
- Inside the `main` method, construct one or more objects
- Apply methods to the objects
- Display the results of the method calls

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.

File MoveTester.java

```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.println("After moving, the top-left
            corner is:");
      System.out.println(box.getX());
      System.out.println(box.getY());
   }
}
```

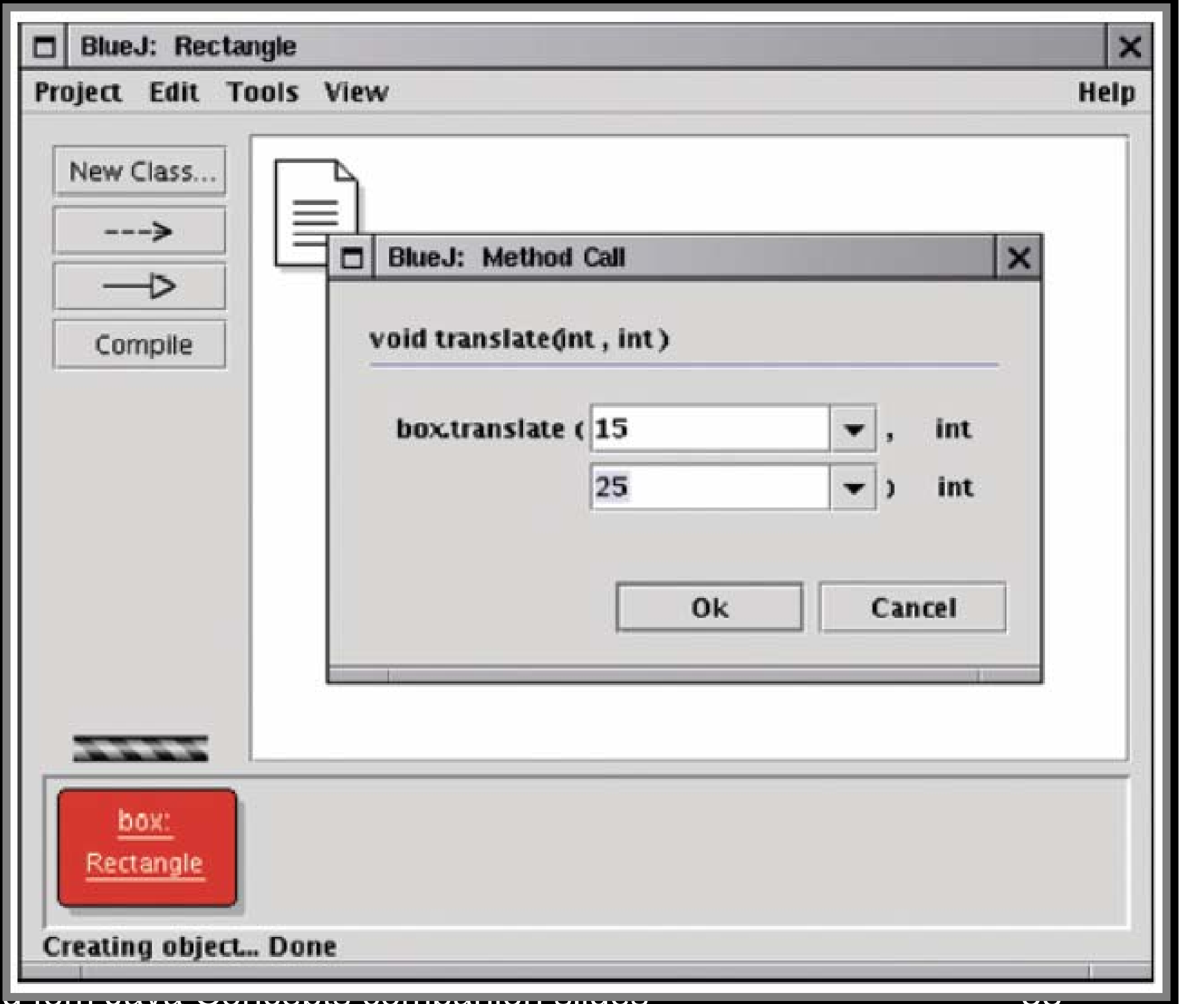
Self Check

1. 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?
2. Why doesn't the `MoveTester` program print the width and height of the rectangle?

Answers

1. Add the statement `import java.util.Random;` at the top of your program
2. Because the `translate` method doesn't modify the shape of the rectangle

Testing Classes in an Interactive Environment



void translate(int , int)

box.translate (15 , int
25) int

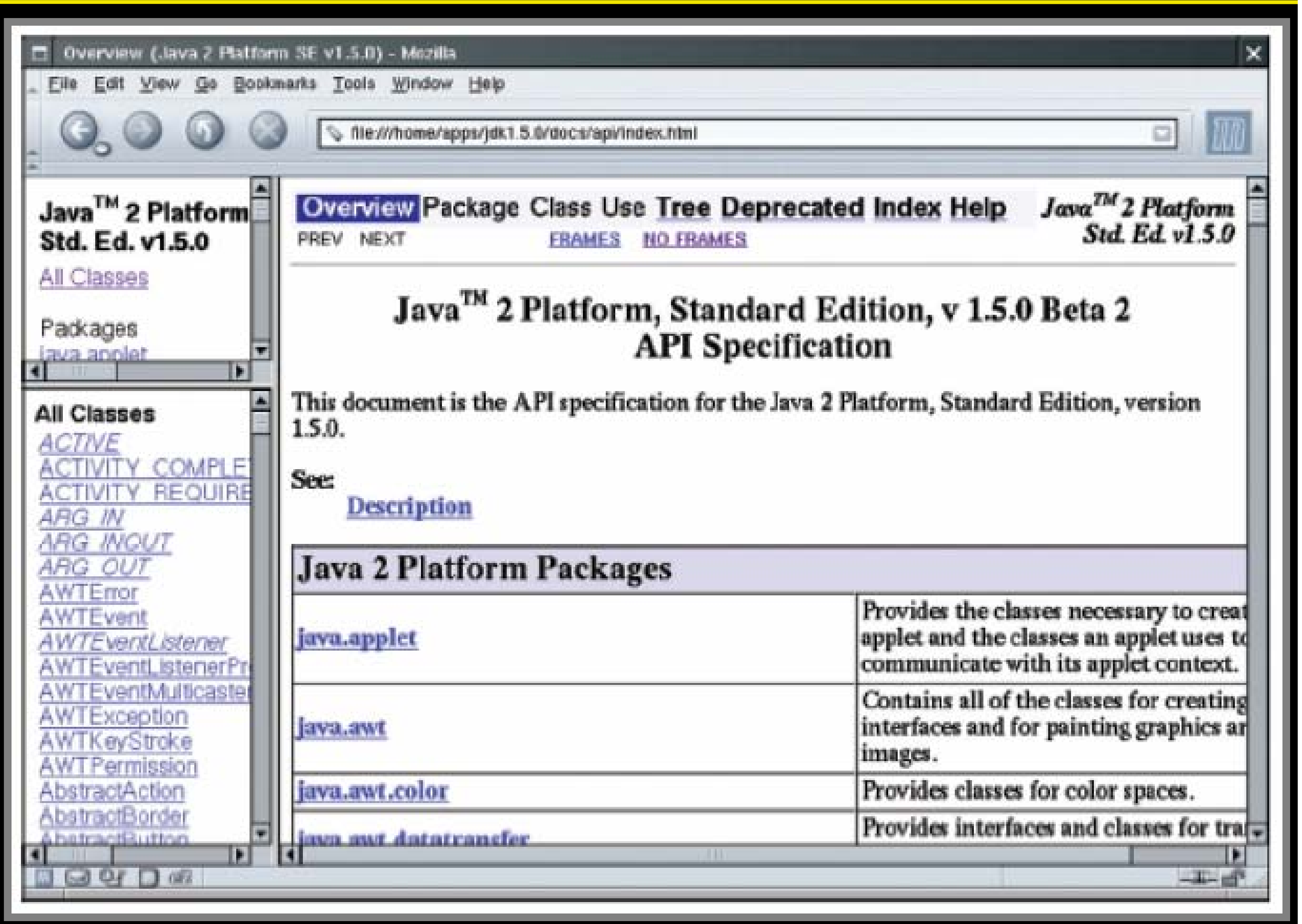
Creating object... Done

Figure 12:
Testing a Method Call in BlueJ

The API Documentation

- API: Application Programming Interface
- Lists classes and methods in the Java library
- http://java.sun.com/j2se/1.5/docs/api/index.html

The API Documentation of the Standard Java Library


Overview (Java 2 Platform SE v1.5.0) - Mozilla
File Edit View Go Bookmarks Tools Window Help
file:///home/apps/jdk1.5.0/docs/api/index.html
Overview Package Class Use Tree Deprecated Index Help
PREV NEXT
FRAMES NO FRAMES

Java™ 2 Platform Std. Ed. v1.5.0

All Classes

Packages

java.applet

All Classes

ACTIVE
ACTIVITY_COMPLE
ACTIVITY_REQUIRE
ARG_IN
ARG_INOUT
ARG_OUT
AWTError
AWTEvent
AWTEventListener
AWTEventListenerPr
AWTEventMulticaste
AWTException
AWTKeyStroke
AWTPermission
AbstractAction
AbstractBorder
AbstractButton

Java™ 2 Platform Std. Ed. v1.5.0

Java™ 2 Platform, Standard Edition, v 1.5.0 Beta 2
API Specification

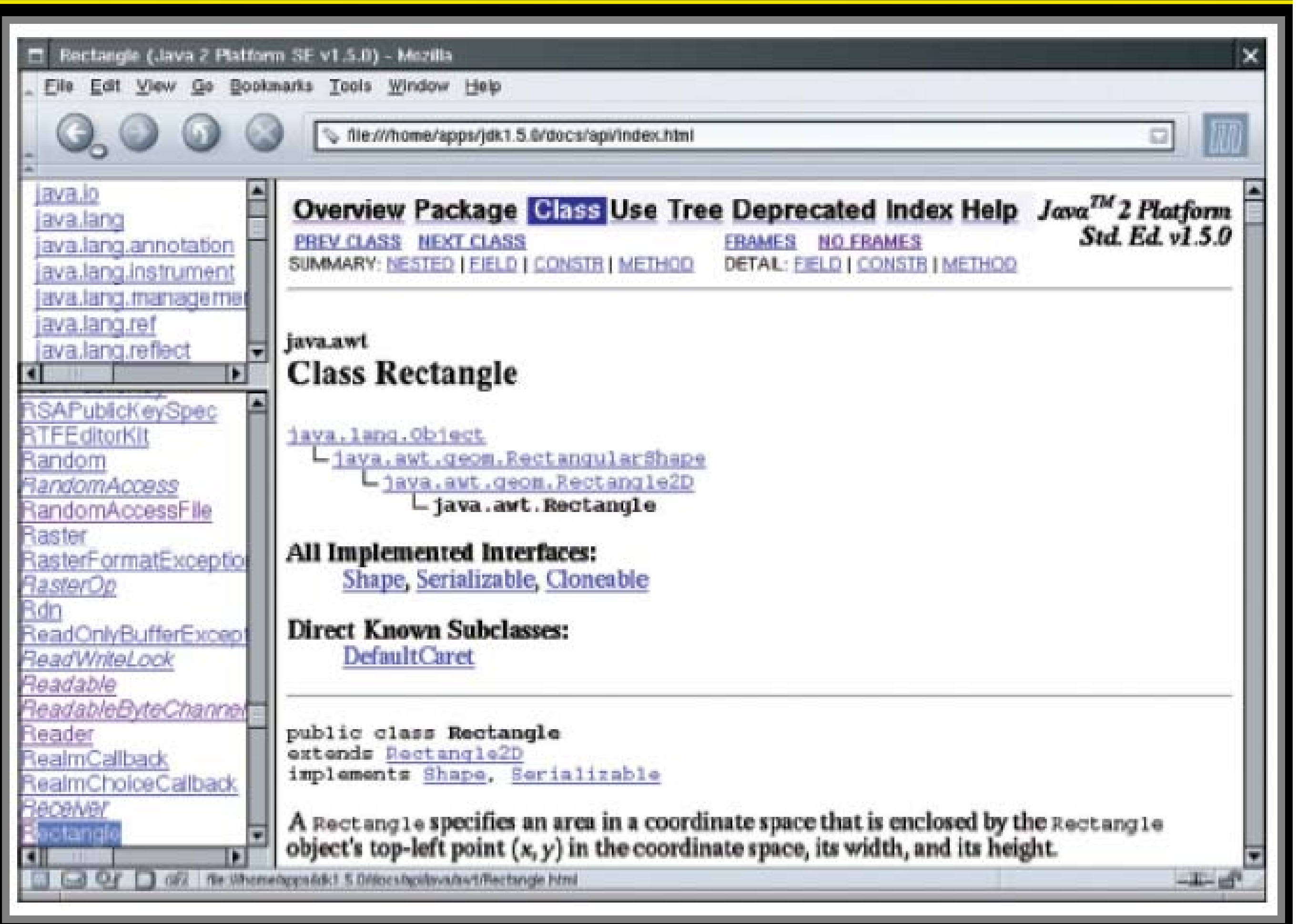
This document is the API specification for the Java 2 Platform, Standard Edition, version 1.5.0.

See:
Description

Java 2 Platform Packages	
java.applet	Provides the classes necessary to creat applet and the classes an applet uses to communicate with its applet context.
java.awt	Contains all of the classes for creating interfaces and for painting graphics ar images.
java.awt.color	Provides classes for color spaces.
java.awt.datatransfer	Provides interfaces and classes for tra

Figure 13:
The API Documentation of the Standard Java Library

The API Documentation for the Rectangle Class



java.awt
Class Rectangle

```
java.lang.Object
  └java.awt.geom.RectangularShape
      └java.awt.geom.Rectangle2D
          └java.awt.Rectangle
```

All Implemented Interfaces:
Shape, Serializable, Cloneable

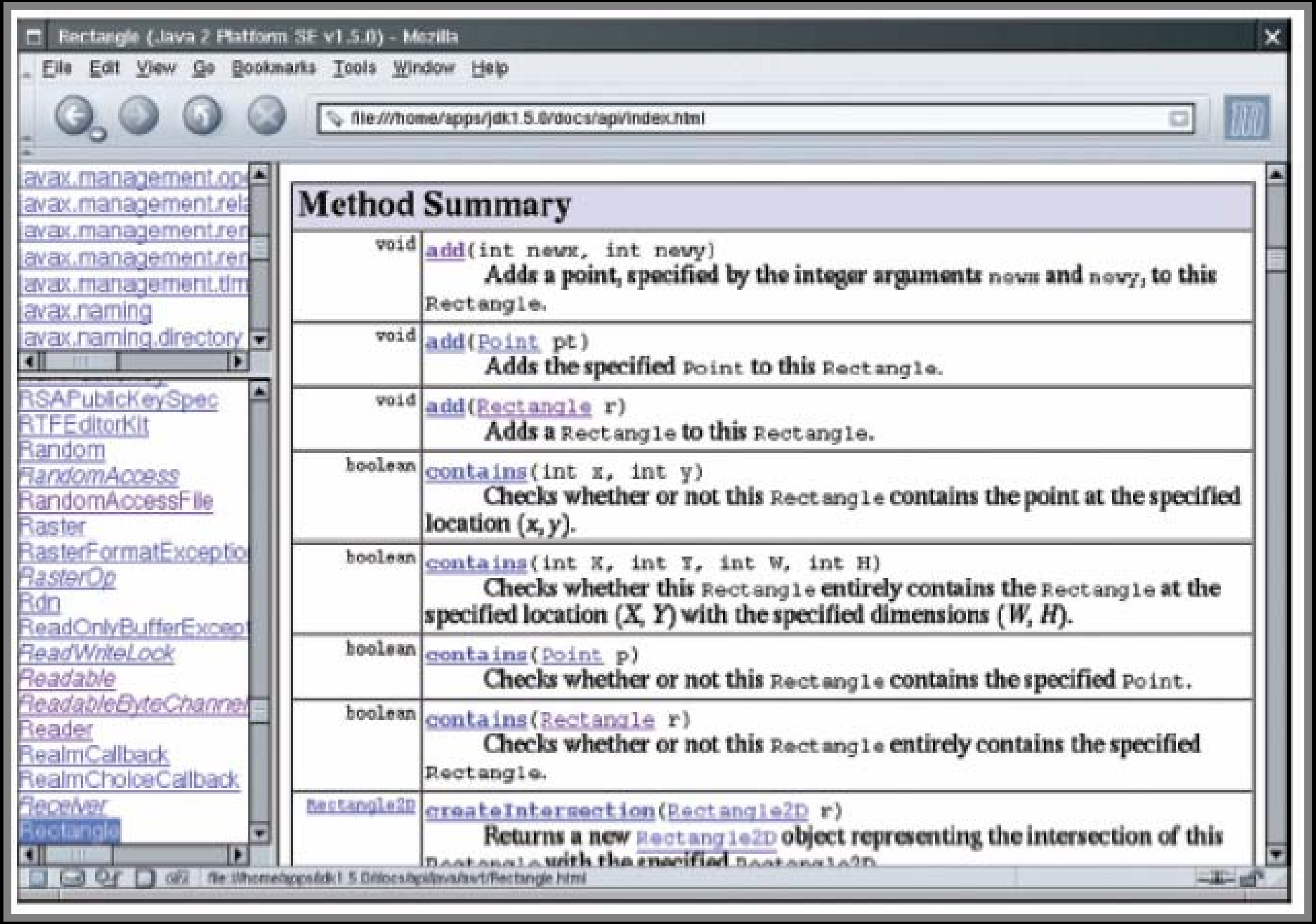
Direct Known Subclasses:
DefaultCaret

```
public class Rectangle
extends Rectangle2D
implements Shape, Serializable
```

A `Rectangle` specifies an area in a coordinate space that is enclosed by the `Rectangle` object's top-left point (*x*, *y*) in the coordinate space, its width, and its height.

Figure 14:
The API Documentation of the Rectangle Class

Javadoc Method Summary



Rectangle (Java 2 Platform SE v1.5.0) - Mozilla
File Edit View Go Bookmarks Tools Window Help
file:///home/apps/jdk1.5.0/docs/api/index.html

avax.management.op
avax.management.rel
avax.management.ren
avax.management.ren
avax.management.tim
avax.naming
avax.naming.directory

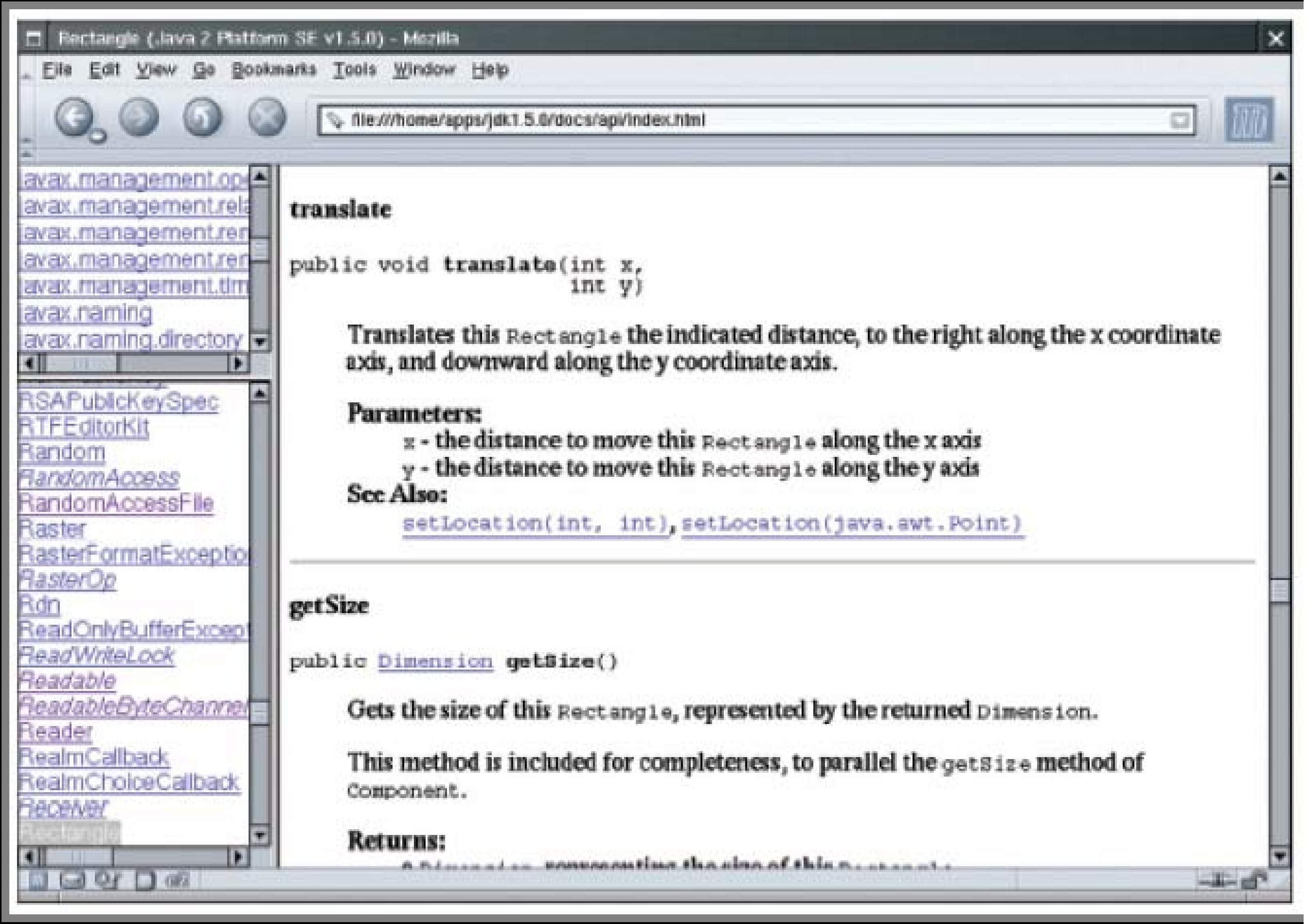
RSAPublicKeySpec
RTFEditorKit
Random
RandomAccess
RandomAccessFile
Raster
RasterFormatExceptio
RasterOp
Rdn
ReadOnlyBufferExcept
ReadWriteLock
Readable
ReadableByteChannel
Reader
RealmCallback
RealmChoiceCallback
Receiver
Rectangle

Method Summary

void	`add(int newx, int newy)` Adds a point, specified by the integer arguments `newx` and `newy`, to this `Rectangle`.
void	`add(Point pt)` Adds the specified `Point` to 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`.
Rectangle2D	`createIntersection(Rectangle2D r)` Returns a new `Rectangle2D` object representing the intersection of this `Rectangle` with the specified `Rectangle2D`.

Figure 15:
The Method Summary for the Rectangle Class

`translate` Method Documentation



translate

```
public void translate(int x,
                      int y)
```

Translates this `Rectangle` the indicated distance, to the right along the x coordinate axis, and downward along the y coordinate axis.

Parameters:
`x` - the distance to move this `Rectangle` along the x axis
`y` - the distance to move this `Rectangle` along the y axis
See Also:
`setLocation(int, int)`, `setLocation(java.awt.Point)`

getSize

```
public Dimension getSize()
```

Gets the size of this `Rectangle`, represented by the returned `Dimension`.

This method is included for completeness, to parallel the `getSize` method of `Component`.

Returns:

Figure 16:
The API Documentation of the translate Method

Self Check

1. 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!"`?
2. 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.)

Answers

1. `toLowerCase`
2. `"Hello, Space !"`–only the leading and trailing spaces are trimmed

Object References

- Describe the location of objects
- 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);
```

Continued…

Object References

- Primitive type variables ≠ object variables

Object Variables and Number Variables

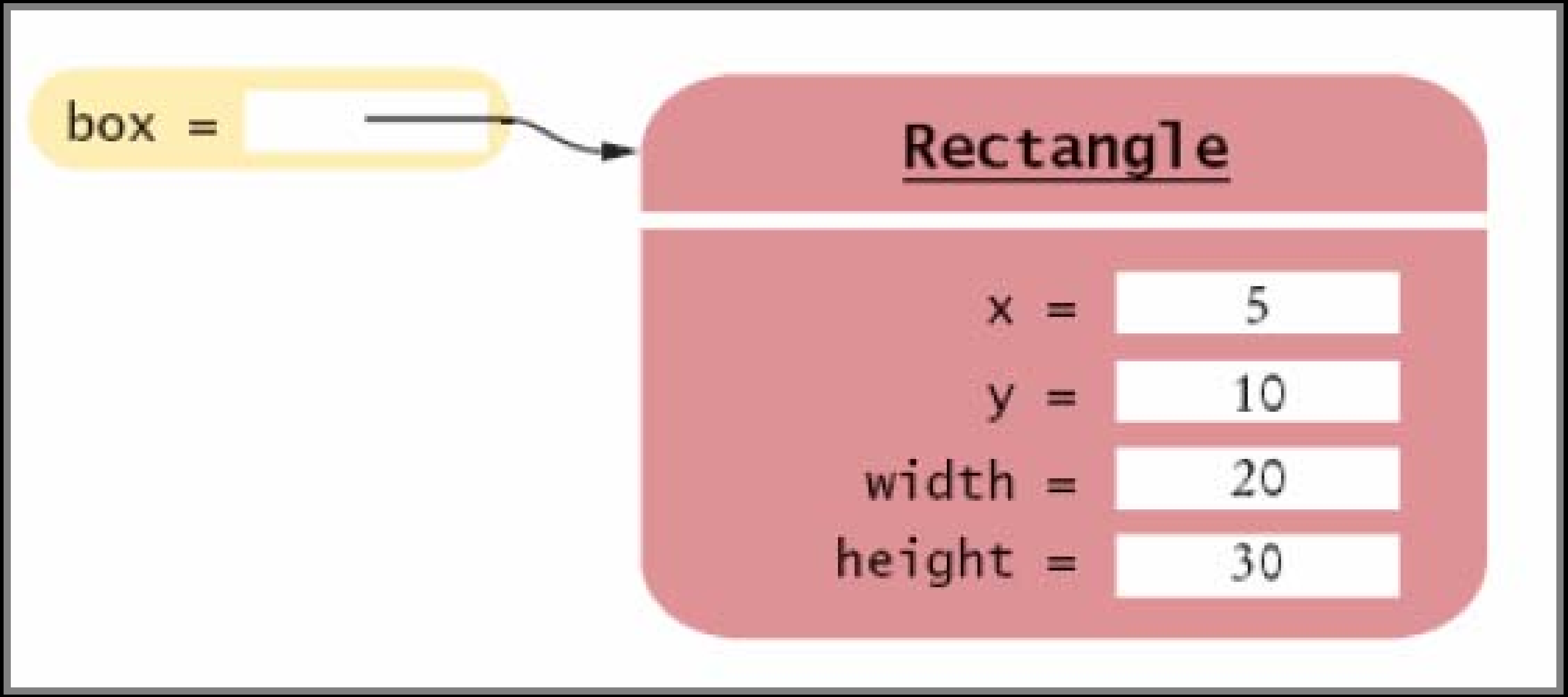


Figure 17:
An Object Variable containing an Object Reference

Object Variables and Number Variables

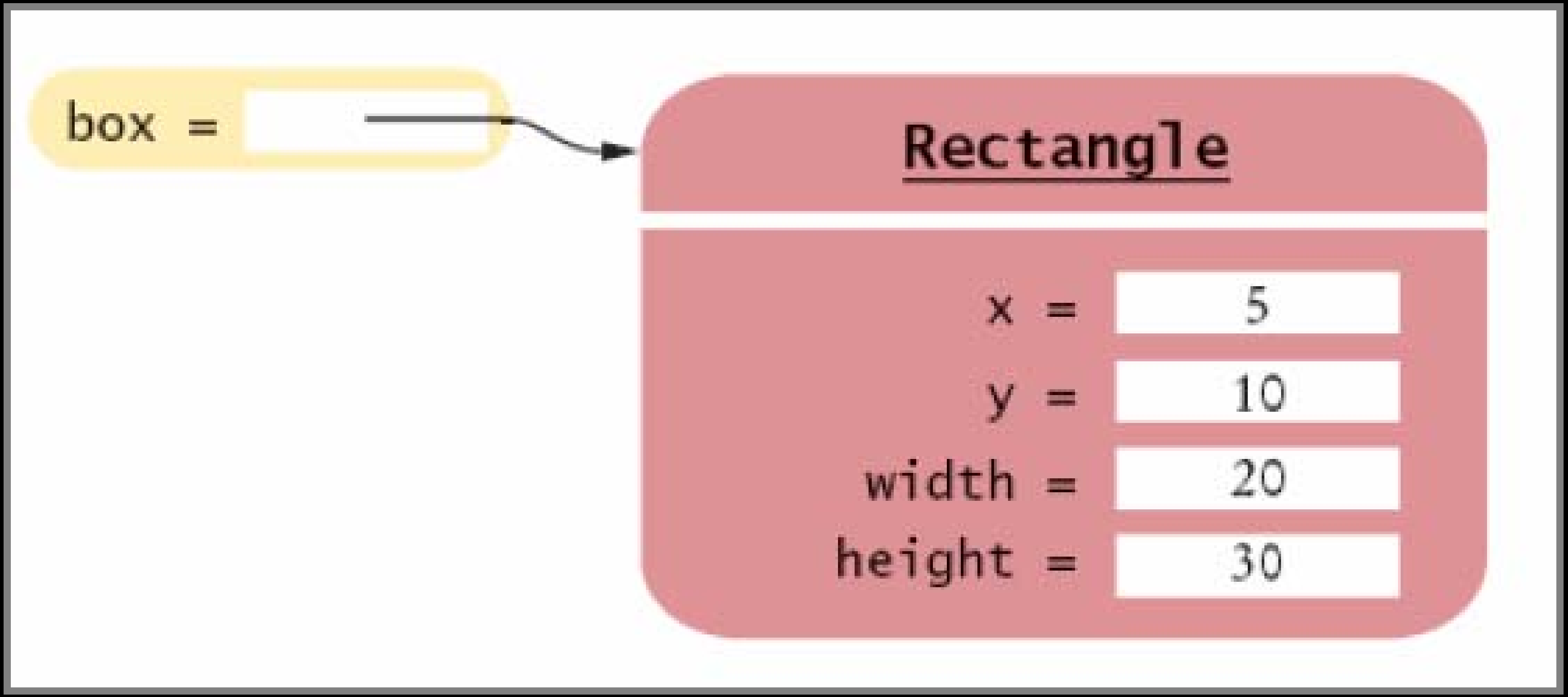


Figure 17:
An Object Variable containing an Object Reference

Object Variables and Number Variables

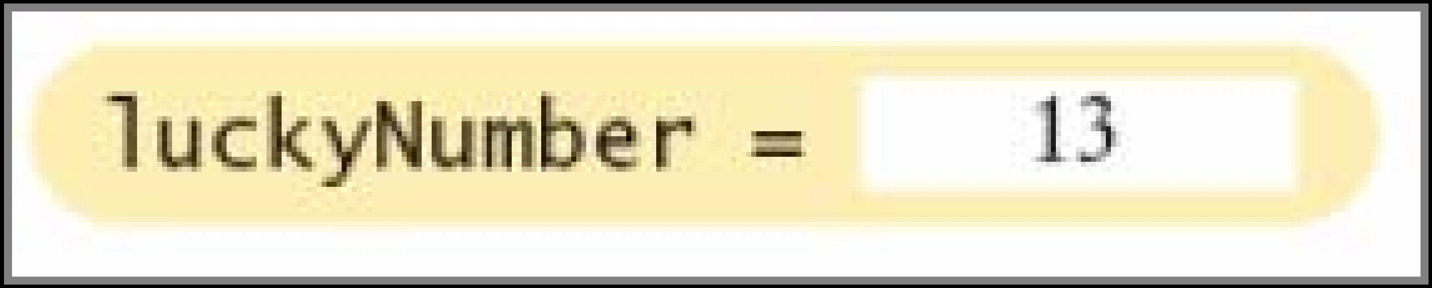


Figure 19:
A Number Variable Stores a Number

Copying Numbers

-
  ```
  int luckyNumber = 13;
  int luckyNumber2 = luckyNumber;
  luckyNumber2 = 12;
  ```

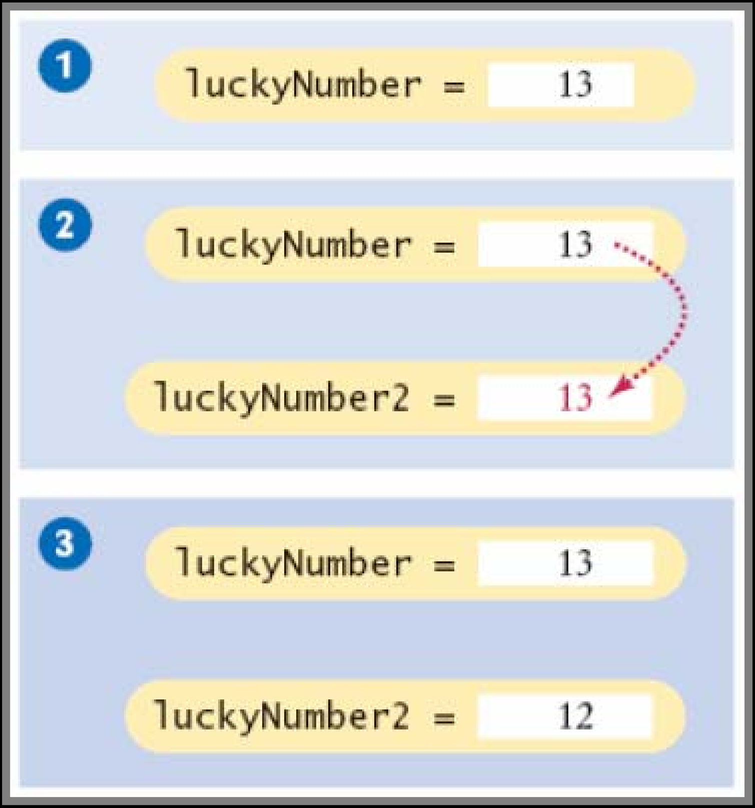


**Figure 20:
Copying Numbers**

Copying Object References

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

Continued…

Copying Object References

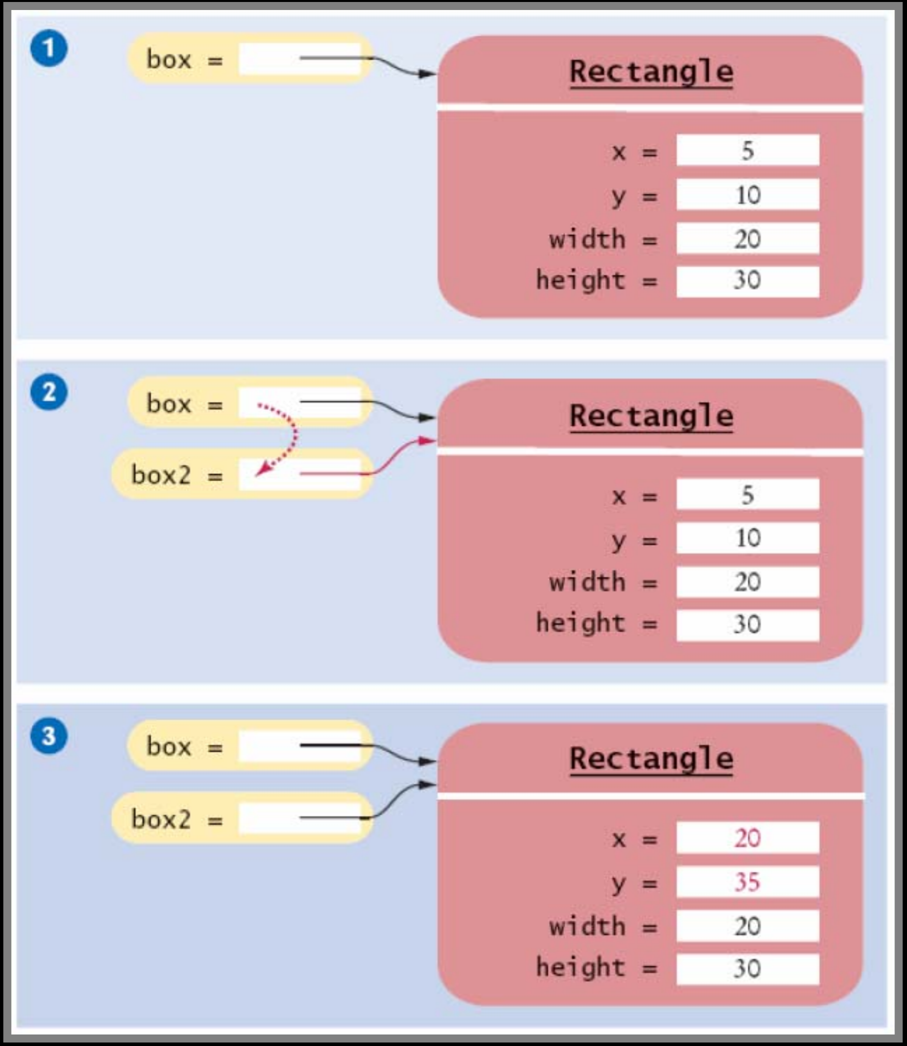


Figure 21:
Copying Object References

Self Check

1. What is the effect of the assignment `greeting2 = greeting`?
2. After calling `greeting2.toUpperCase()`, what are the contents of `greeting` and `greeting2`?

Answers

1. Now `greeting and greeting2` both refer to the same `String` object.
2. 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

Figure 22:
A Mainframe Computer