



# D06

# PROGRAMMING with JAVA

## Ch2 – Using Objects

# Chapter 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

- In Java, every value has a **type** (e.g.: "Hello, World" has the type **String**). The type tells you what you can do with that value.
- To remember an object, you need to hold it in a **variable**, a storage location in the computer's memory that has a type, a name, and a content.
- 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 (`_`) 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**
  - Variable and method names should start with a lowercase letter
  - Class names should start with an uppercase letter

# Self Check

---

1. What is the type of the values `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` and `String`
2. Only the first two are legal identifiers

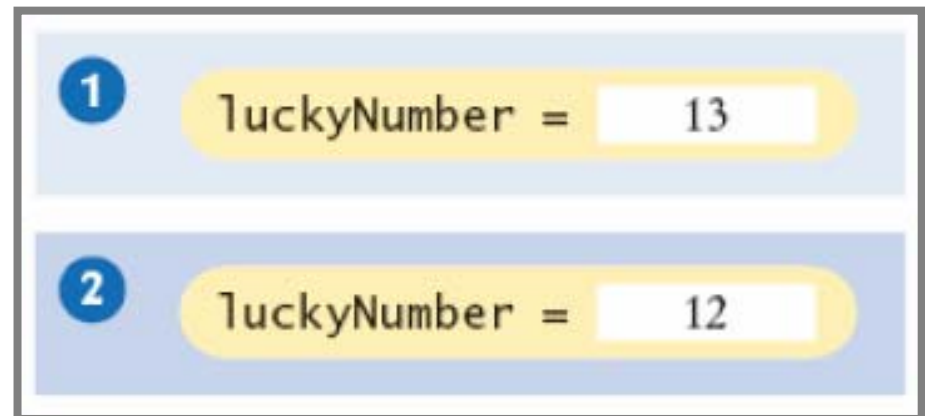
3.

```
String myName = "John Q. Public";
```

# The Assignment Operator

- You can change the value of an existing variable with the **assignment operator (=)**

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



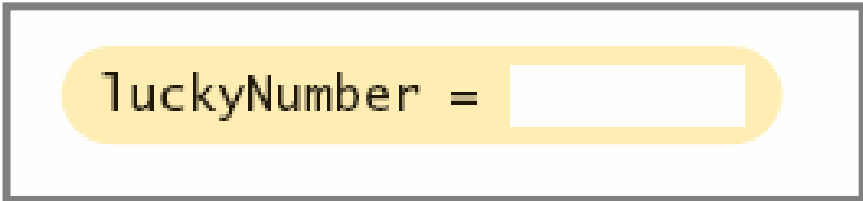
**Figure 1:**  
**Assigning a New Value to a Variable**

# Uninitialized Variables

- It is an error to use a variable that has never had a value assigned to it:

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

**Figure 2:**  
**An Uninitialized Object Variable**

A diagram illustrating an uninitialized variable. It consists of a rectangular box with a gray border. Inside the box, the text "luckyNumber =" is displayed in a light gray font. To the right of the equals sign is a white rectangular box with rounded corners, representing the uninitialized value of the variable.

luckyNumber =

 Assign a value to the variable before you use it or, even better, [initialize](#) the variable when you define it

# Syntax 2.2: Assignment

---

*variableName = value;*

**Example:**

```
luckyNumber = 12;
```

**Purpose:**

To assign a new value to a previously defined variable.

# Self Check

---

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

# Answers

---

4. No, the left-hand side of the = operator must be a variable

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

Note that

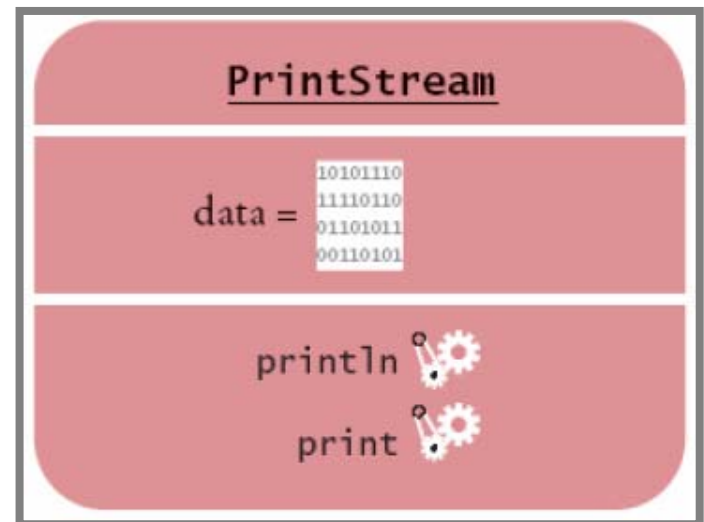
```
String greeting = "Hello, Nina!";
```

is not the right answer—that statement defines a new variable

# Objects and Classes

- ⚠ An object is an entity that you can manipulate in your programs. You don't usually know how the object is organized internally
- A class is a set of objects with the same behavior. Each object belongs to a class (an object is an **instance** of a class). For example, `System.out` belongs to the class `PrintStream`

Figure 3:  
Representation of the `System.out` object



# A Representation of Two String Objects

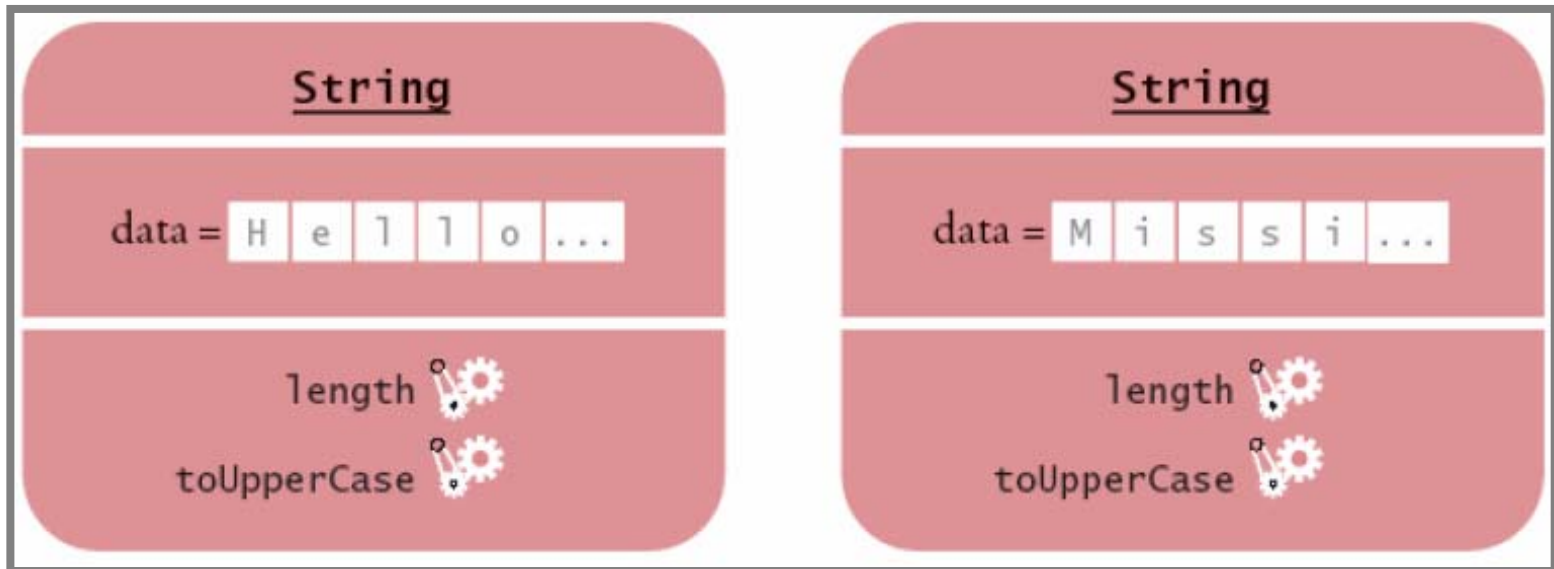


Figure 4:  
A Representation of Two String Objects

# Methods

---

- You manipulate an object by calling one or more of its methods. A **method** consists of a sequence of instructions that accesses the internal data
- A class specifies the methods that you can apply to its objects: e.g., the **length()** method counts the number of characters in a string; you can apply that method to any object of type **String**:

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

# 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
```

# Public interface of a class

---

- The methods form the **public interface** of the class, telling you what you can do with the objects of the class
- A class also defines a **private implementation**, describing the data inside its objects and the instructions for its methods. Those details are hidden from the programmers who use objects and call methods

# Self Check

---

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

# Answers

---

6. 

```
river.length() or "Mississippi".length()
```
7. 

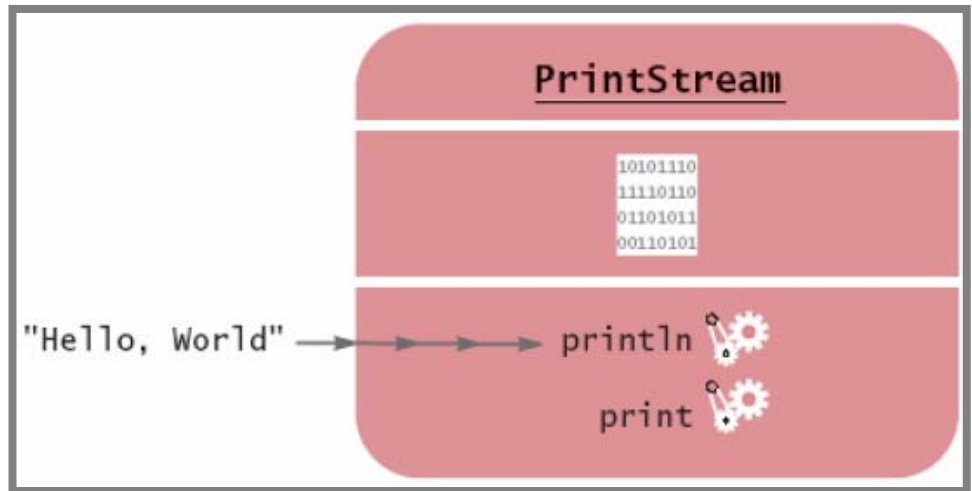
```
System.out.println(greeting.toUpperCase());
```
8. It is not legal. The variable `river` has type `String`. The `println` method is not a method of the `String` class.


# Implicit and Explicit Parameters

- Some methods (not all) require one or more **explicit parameters** (inputs) that give details about the work that they need to do

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

Figure 5:  
Passing a parameter to the  
println method



-  The object on which you invoke the method is also considered a parameter of the method call, called the **implicit parameter**

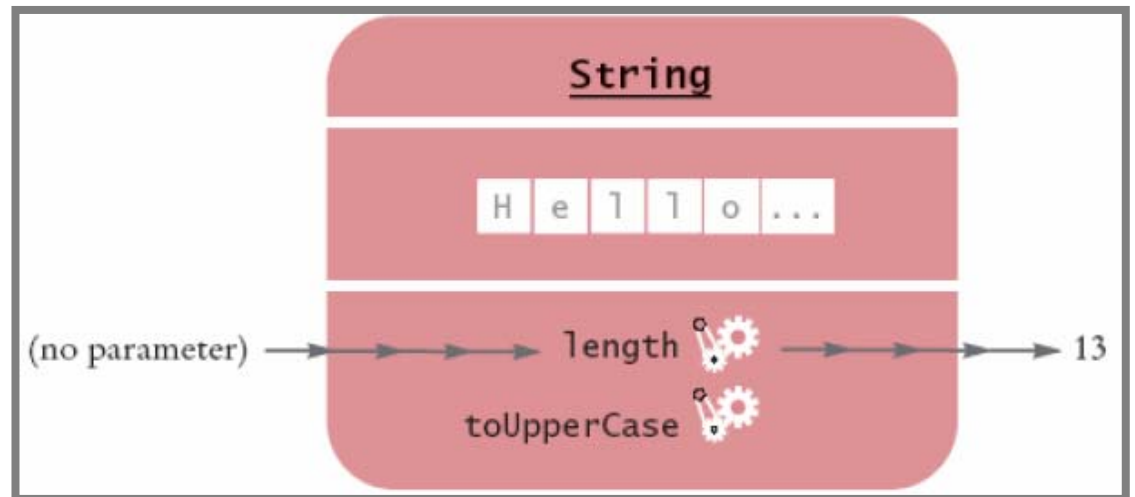
```
System.out.println(greeting)
```

# Return Values

- Some methods (not all) return a value; you can store the return value in a variable

```
int n = greeting.length(); // return value stored in n
```

Figure 6:  
Invoking the length  
Method on a String  
Object



*Continued...*

# Passing Return Values

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

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

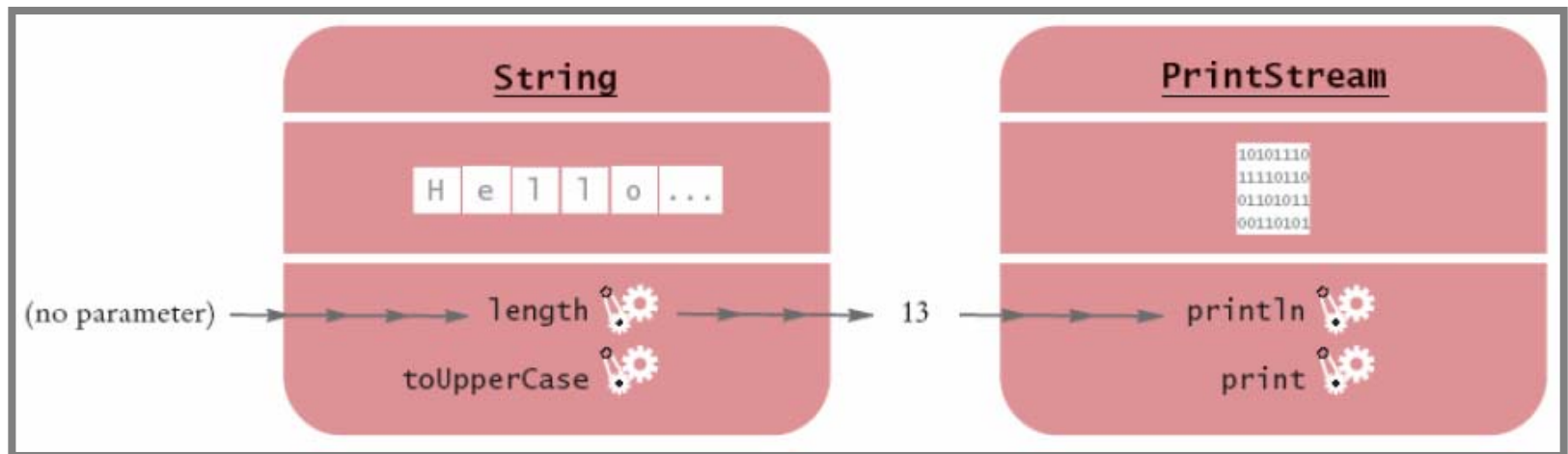


Figure 7: Passing the Result of a Method Call to Another Method

- Not all methods return values. Example:

```
println()
```

# A More Complex Call

- The `replace()` method carries out a search-and-replace operation

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

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

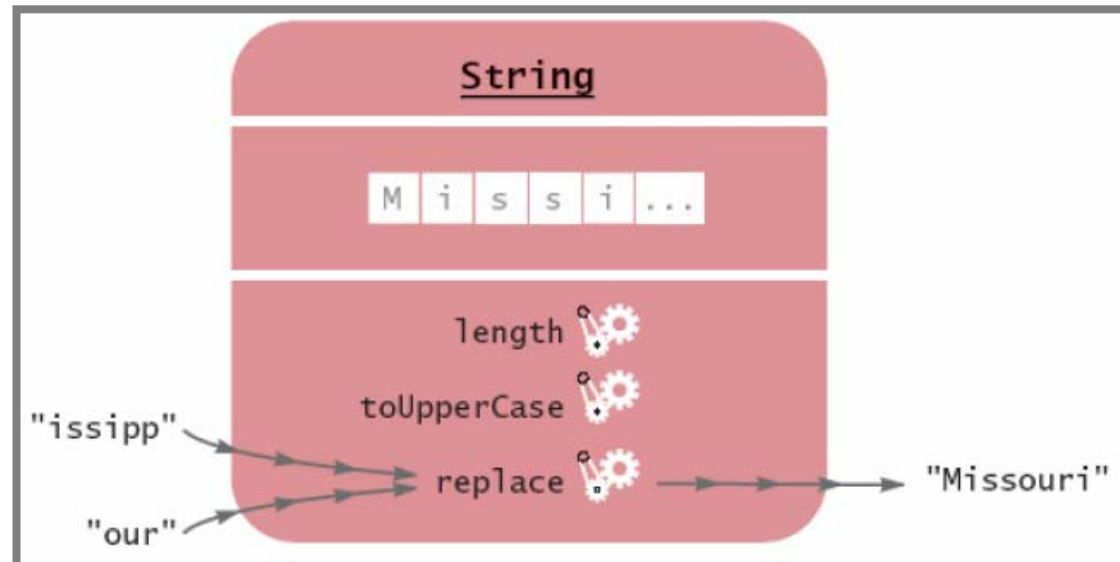


Figure 8:  
Calling the `replace` Method

# Method Definitions

- ⚠ When a method is defined in a class, the **definition** specifies the types of the explicit parameters (if any) and the return value
- The **type** of the implicit parameter is the class that defines the method
- **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
```

*Continued...*

# Method Definitions

- When a method returns no value, the return type is declared with the reserved word **void**

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

- ⚠ Occasionally, a class defines two or more methods with the same name and different explicit parameter types. In this cases, we say that the method name is **overloaded** (since it refers to more than one method)

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

# Self Check

---

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

# Answers

---

9. The implicit parameter is `river`. There is no explicit parameter. The return value is `11`
10. `"Mississippi"`
11. `12`
12. `As public String toUpperCase()`, with no explicit parameter and return type `String`.

# Number Types

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


13

- Floating point numbers: **float**, **double**

1.3, 0.00013

- When a floating-point number is multiplied or divided by 10, only the position of the decimal point changes; it "floats". This representation is related to the "scientific" notation  $1.3 \times 10^{-4}$

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

-  Numbers are not objects; numbers types are **primitive types**, not classes

# 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

---

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

# Answers

---

13. `double`

14. An `int` is not an object, and you cannot call a method on it

15.  $(x + y) * 0.5$

# Rectangular Shapes and Rectangle Objects

- Objects of type `Rectangle` describe rectangular shapes
- A `Rectangle` object isn't a rectangular shape—it is an object that contains a set of numbers that describe the rectangle

Figure 9:  
Rectangular Shapes

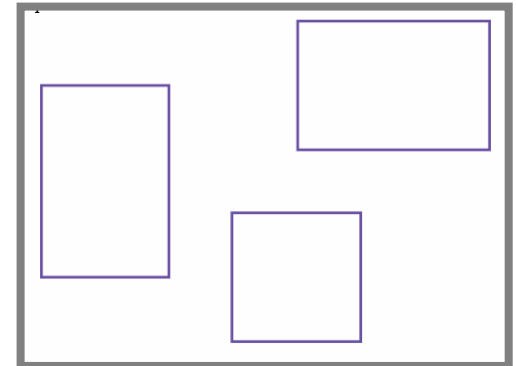


Figure 10: Rectangular Objects

<u>Rectangle</u>	<u>Rectangle</u>	<u>Rectangle</u>
x = 5	x = 35	x = 45
y = 10	y = 30	y = 0
width = 20	width = 20	width = 30
height = 30	height = 20	height = 20

# Constructing Objects

- ⚠ Invoke the **new** operator, specifying the name of the class and the required parameters:

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

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

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

*Continued...*

# 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**
- You use the value of a **new** expression just like a method return value: assign it to a variable or pass it to another method
- Some classes let you construct objects in multiple ways (e.g., you can also obtain a **Rectangle** object by supplying no construction parameters at all, but you must still supply the parentheses):

```
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

---

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

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

# Answers

---

16. `new Rectangle(90, 90, 20, 20)`

17. 0

# Accessor and Mutator Methods

- **Accessor method:** a method that accesses an object and returns some information about it, without changing the object

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

- **Mutator method:** changes the state of its implicit parameter (the object on which the method is invoked)

```
box.translate(15, 25);
```

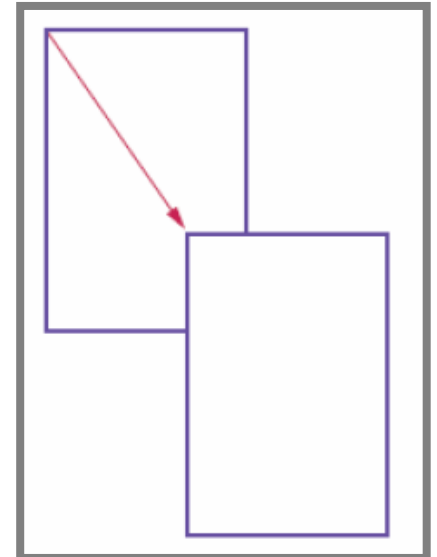


Figure 11:  
Using the `translate` Method to  
Move a Rectangle

# Self Check

---

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

# Answers

---

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

# Implementing a Test Program



To construct a **test program** –a program that allows to test the behavior of an object, do the following:

1. Provide a new 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
- Don't forget to include (import) appropriate packages:
    - Java classes are grouped into **library packages**
    - **import** library classes by specifying the package and class name:

```
import java.awt.Rectangle;
```
    - Classes from the **java.lang** package -such as `String` and `System`, are automatically imported

# 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.

The `java.awt` package contains many classes for drawing windows and graphical shapes

# File MoveTester.java

1. Provide a new class

```
01: import java.awt.Rectangle;
```

```
02:
```

```
03: public class MoveTester
```

```
04: {
```

```
05:     public static void main(String[] args)
```

```
06:     {
```

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

```
08:
```

```
09:         // Move the rectangle
```

```
10:         box.translate(15, 25);
```

```
11:
```

```
12:         // Print information about the moved rectangle
```

```
13:         System.out.println("After moving, the top-left  
corner is:");
```

```
14:         System.out.println(box.getX());
```

```
15:         System.out.println(box.getY());
```

```
16:     }
```

```
17: }
```

2. Supply a main method

3. Construct objects

4. Apply methods to the objects

5. Display the results of the calls

# Self Check

---

20. 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?
21. Why doesn't the `MoveTester` program print the width and height of the rectangle?

# Answers

---

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

# The API Documentation

---



The classes and methods of the Java library are documented in the **Application Programming Interface (API) documentation**

- A programmer who uses the Java classes to put together a computer program (or **application**) is an **application programmer**; in contrast, the programmers who designed and implemented the library classes are **system programmers**
- The API documentation for each class starts out with a section that describes the purpose of the class. Then come summary tables for the constructors and methods
- You can find the API documentation at:  
<http://java.sun.com/j2se/1.5/docs/api/index.html>

# Self Check

---

22. 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!"`?
23. 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

---

22. `toLowerCase`

23. "Hello, Space !" –only the leading and trailing spaces are trimmed

# Object References



In Java, a variable whose type is a class does not actually hold an object. It merely holds a **reference** to the object (i.e., the memory location of the object)

- The **new** operator returns a reference to a new object, and that reference is stored in the `box` variable:

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

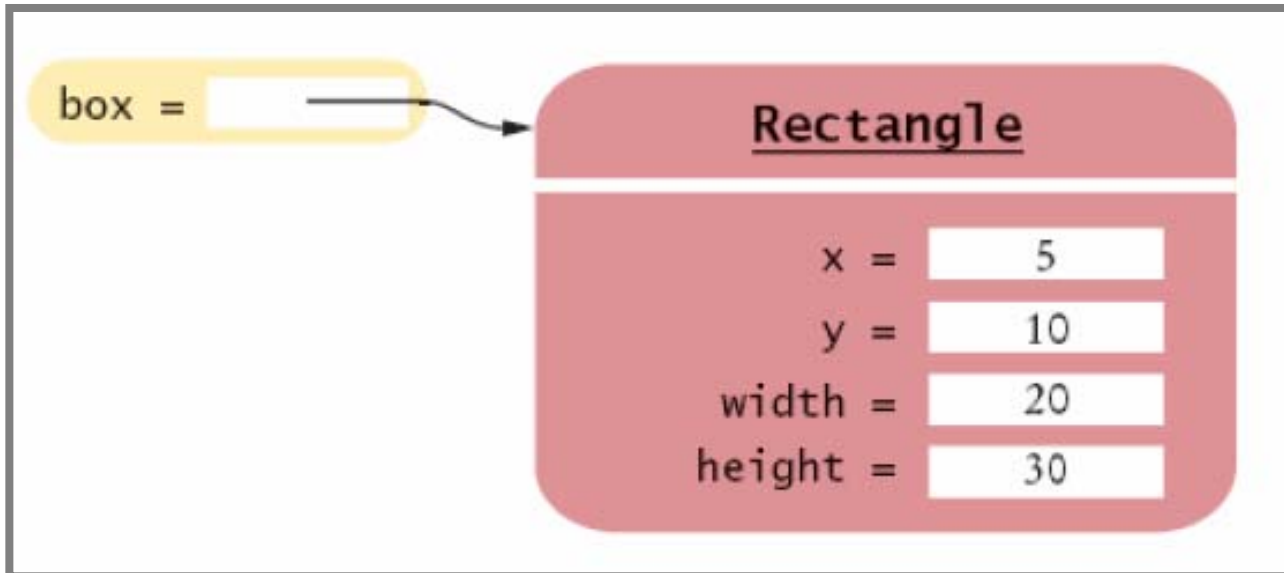


Figure 17:  
An Object Variable  
Does Not Contain the  
Object. It Refers to  
the Object

*Continued...*

# Object References

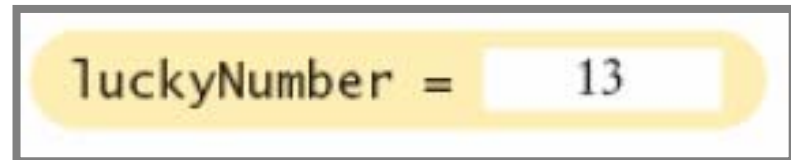
- You can have two or more object variables referring to the same object:

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

- However, number variables actually store numbers:

```
int luckyNumber = 13;
```

Figure 19: A Number Variable Stores a Number



- ⚠ Primitive type variables  $\neq$  object variables: each number requires a small amount of memory, but objects can be very large (it is far more efficient to only manipulate the memory location)

# Copying Numbers

- When you copy a primitive type value, the original and the copy of the number are independent values:

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

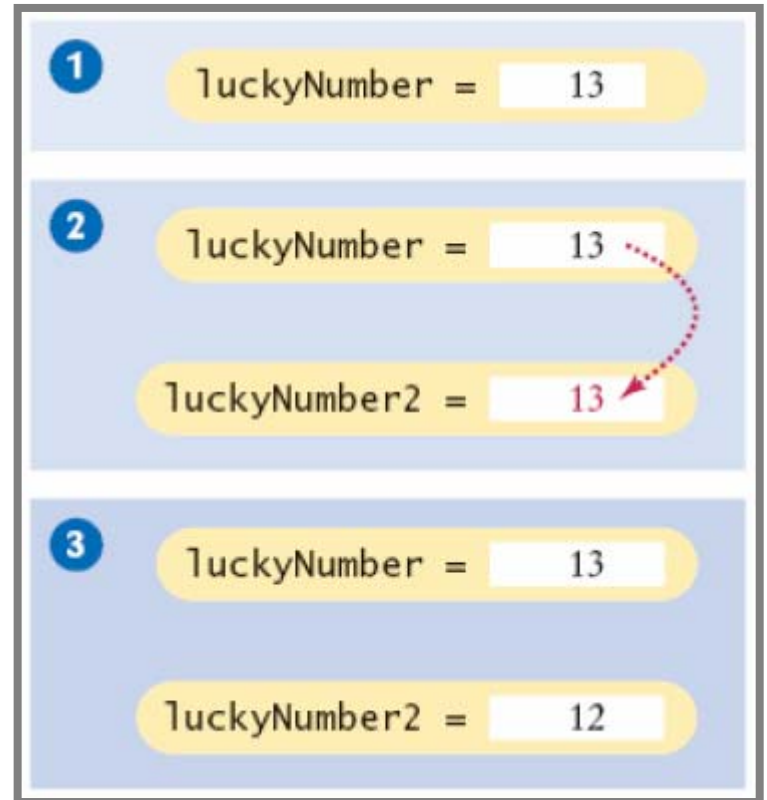


Figure 20:  
Copying Numbers

# Copying Object References

- But, when you copy an object reference, both the original and the copy are references to the same object:

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

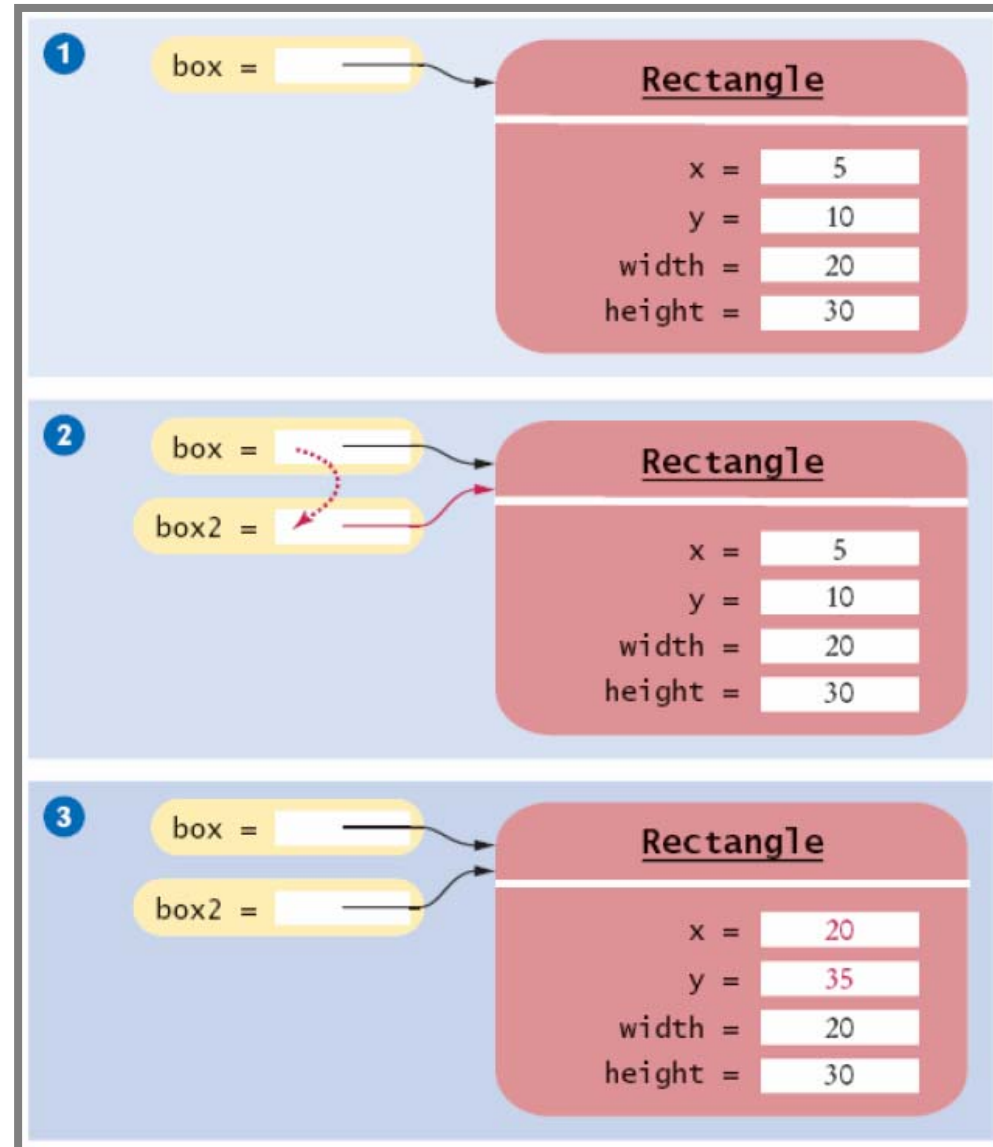


Figure 21:  
Copying Object References

# Self Check

---

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

# Answers

---

- 24. Now `greeting` and `greeting2` both refer to the same `String` object.
- 25. 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.

# Chapter Summary

---

- In Java, every value has a type
- **Identifiers** for variables, methods, and classes are composed of letters, digits, and the characters “\_” and “\$”
- Variable and method names should start with a lowercase letter
- All variables should be **initialized** before you access them
- **Objects** are entities that you manipulate by calling methods
- A **method** is a sequence of instructions that accesses the data of an object
- A **class** defines the methods that you can apply to its objects
- The **public interface** of a class specifies what you can do with its objects. The **hidden implementation** describes how these actions are carried out

*Continued...*

# Chapter Summary

---

- An **explicit parameter** is an input to a method
- The **implicit parameter** of a method call is the object on which the method is invoked
- The **return value** of a method is a result that the method has computed for use by the code that called it
- A method name is **overloaded** if a class has more than one method with the same name (but different parameter types)
- Numbers are not objects and number types are not classes
- Use the **new** operator, followed by a class name and parameters, to construct new objects
- An **accessor method** does not change the state of its implicit parameter. A **mutator method** changes the state

*Continued...*

# Chapter Summary

---

- Java classes are grouped into **packages**. Use the **import** statement to use classes that are defined in other packages
- The **API** (Application Programming Interface) **documentation** lists the classes and methods of the Java library
- An **object reference** describes the location of an object
- Multiple object variables can contain references to the same object
- Number variables store numbers. Object variables store references