

Object Oriented Design

In OOD, you have objects. You can create objects and handled them based on their attributes and behaviors.

Examples:

Objects	Attributes and states	Behaviors
aRectangle	x, y, width, height	create, translate
cerealBox	x,y,width, height	create, translate
aCar	mileage, model, gasTank	addGas, useGas
worldGreeter	name	sayHello
aBankAccount	balance	deposit, withdrawal
currentTemp	fahrenheit, celsius	converToCels, converToFahr
aPerson	height, weight, gender	growth, updateWeight

In java every object belongs to a class. Attributes and states of an object make up the instance fields of an object. The behaviors of an object are what is called its methods.

Classes are like factories for objects. You can also say that a Class is like the object's blue print.

In the examples above, the object class could be named as follows:

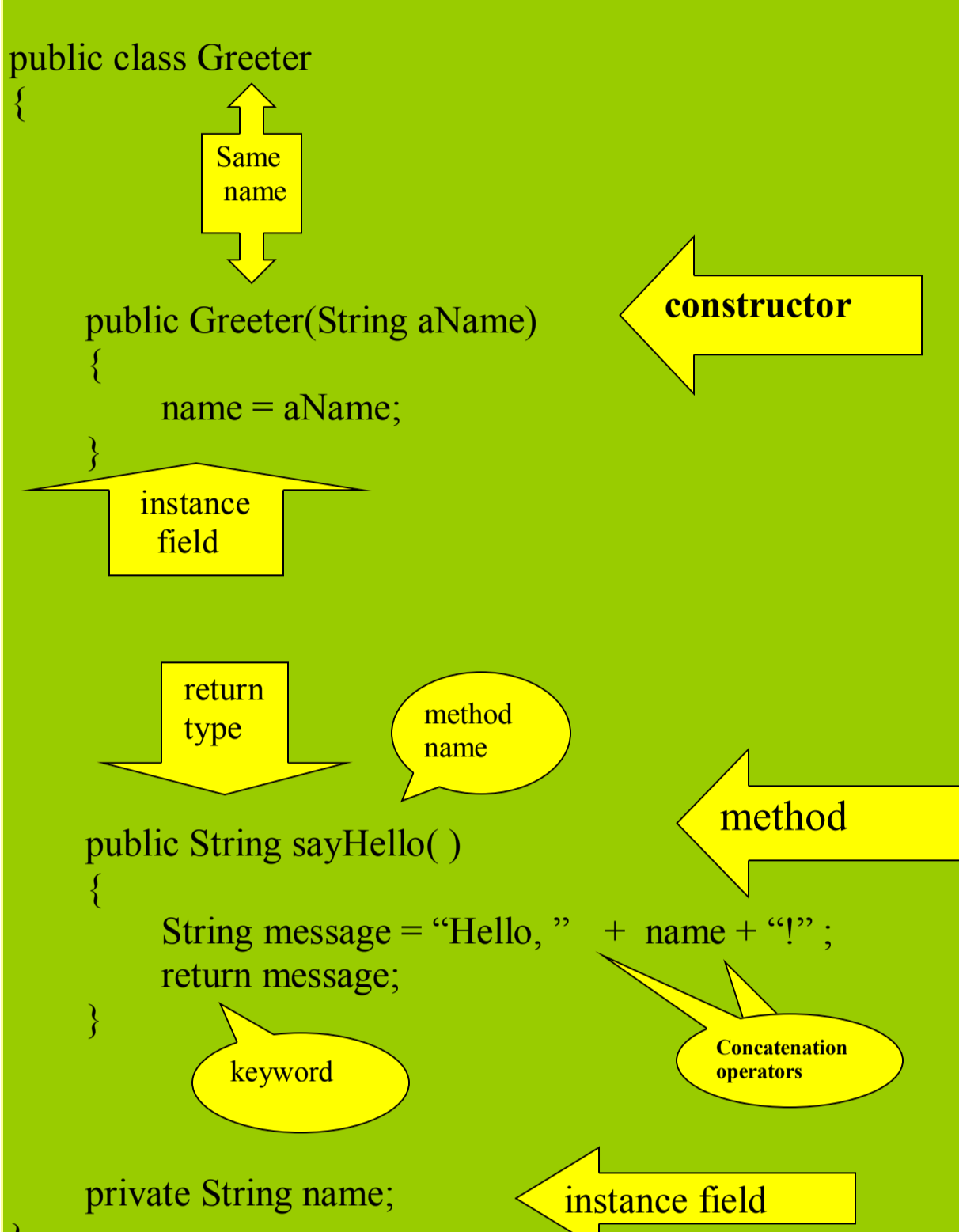
Object	Class
aRectangle	Rectangle
cerealBox	Rectangle
aCar	Car
worldGreeter	Greeter
aBankAccount	BankAccount
currentTemp	Temperature
aPerson	Person

Before we look at a class, let's take a look at two variables and their "type":

Variables: String's and int's

```
String message = " Hello, World"
int aNumber = 14;
```

Now let's define the class Greeter:

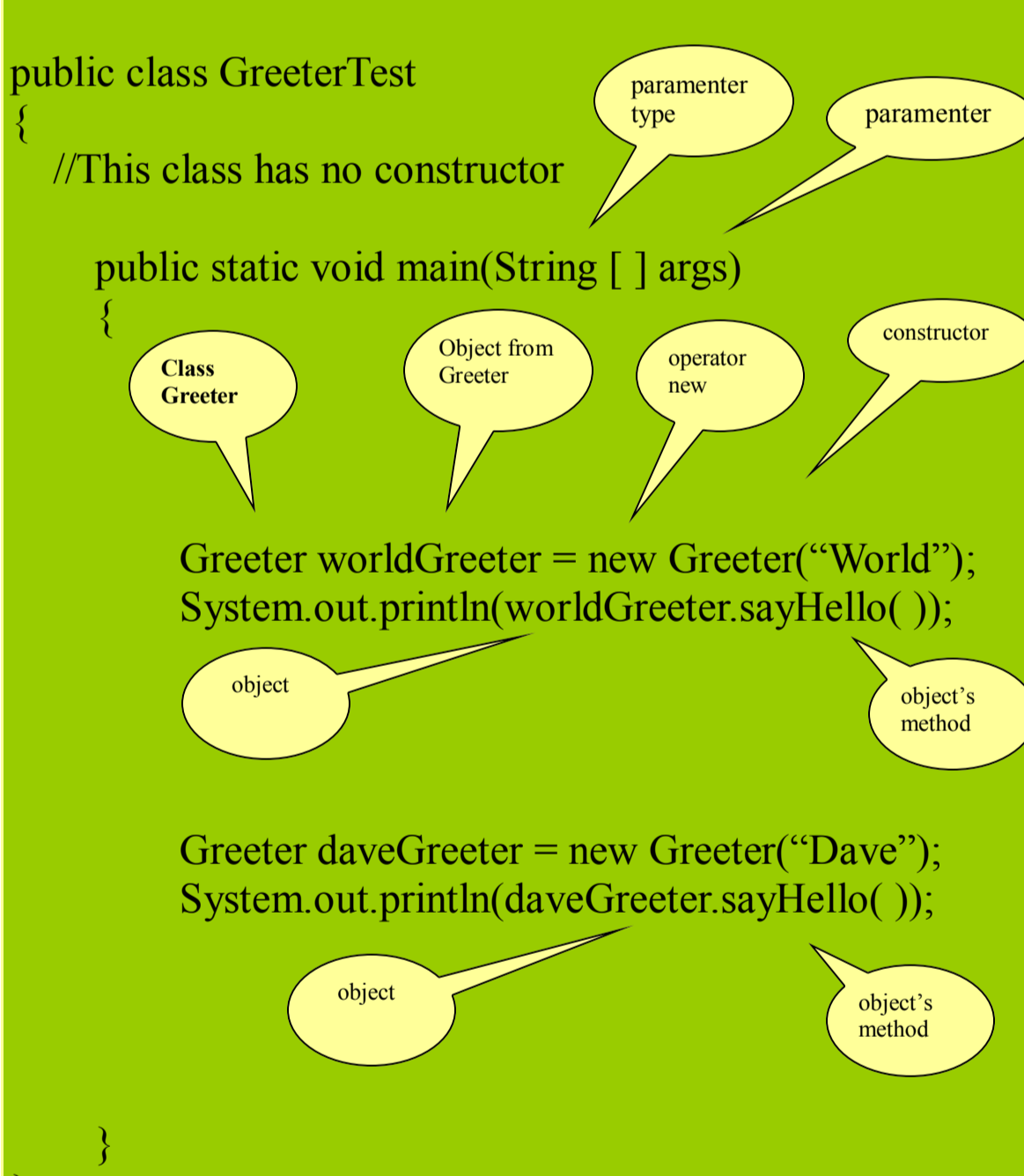


Testing a Class

You can compile the file *Greeter.java*. However, you can't execute the resulting *Greeter.class* file. It does not contain a *main* method. A test class is a class with a main method that contains statements to test another class. A test class typically carries out the following steps:

1. Construct one or more objects of the class that is being tested.
2. Invoke one or more methods.
3. Print out one or more results.

Here is a test class that you can use to confirm that the Greeter class works correctly.



The instructions or *statements* in the *body* of the *main* method—that is, the statements inside the curly brackets { }—are executed one by one .

Important notes

- A string is a sequence of characters enclosed in quotation marks.
- In this program, you need to specify the destination for the string is the system output—that is, a console window.
- The console window is represented in Java by an object called out.
- The **println** method prints a string or a number and then starts a new line.
- Use comments to help human readers understand your program.

Method Call Syntax

```
object.methodName(parameters)
```

Example: System.out.println("Hello, Dave!");

Purpose:
To invoke a method on an object and supply any additional parameters.

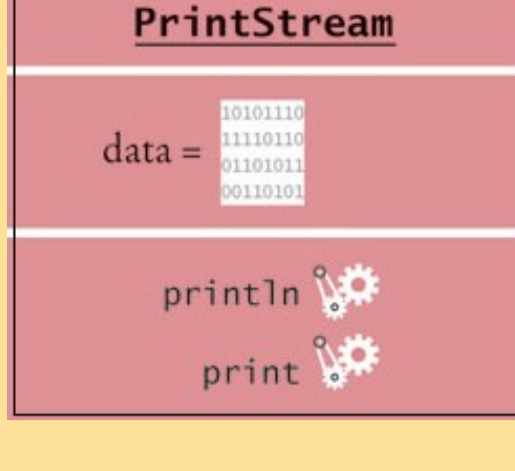
Objects, Classes, and Methods

- You can manipulate an object by calling one or more of its methods.
- A method consist of a sequence of instructions that accesses the internal data of an object.
- When you call a method, you do not know exactly what those instructions are, but you know the purpose of the method.
- A class defines the methods that you can apply to its objects.
- The methods do 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. This is called **encapsulation**.
- A **parameter** is an input to a method.
- The **implicit** parameter of a method call is the object on which the method is invoked.
- **Example:** in the class Greeter, sayHello() is being called daveGreeter. DaveGreeter is the implicit parameter.

The PrintStream class:

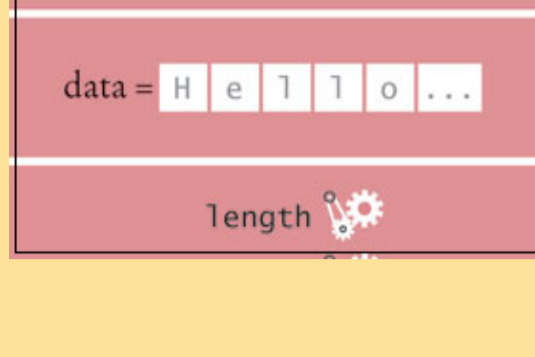
The System.out object belongs to the class PrintStream and this object has methods that have been defined in its class, print and println.

Class



The String Class:

The “Hello, World” object belongs to the class String. The String class provides methods that you can apply to any String object. Some methods to mention: length() and toUpperCase().



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

The length method—unlike the println method—requires no input inside the parentheses. However, the length method yields an output, namely the character count.

Let us look at another method of the String class. When you apply the toUpperCase method to a String object, the method creates another String object that contains the characters of the original string, with lowercase letters converted to uppercase. For example, the sequence of statements

```
String river = "Mississippi";  
String bigRiver = river.toUpperCase();
```

sets bigRiver to the String object “MISSISSIPPI”.



When you apply a method to an object, you must make sure that the method is defined in the appropriate class. For example, it is an error to call

```
System.out.length(); // This method call is an error
```

The public interface of a class specifies what you can do with its objects. The hidden implementation describes how these actions are carried out.

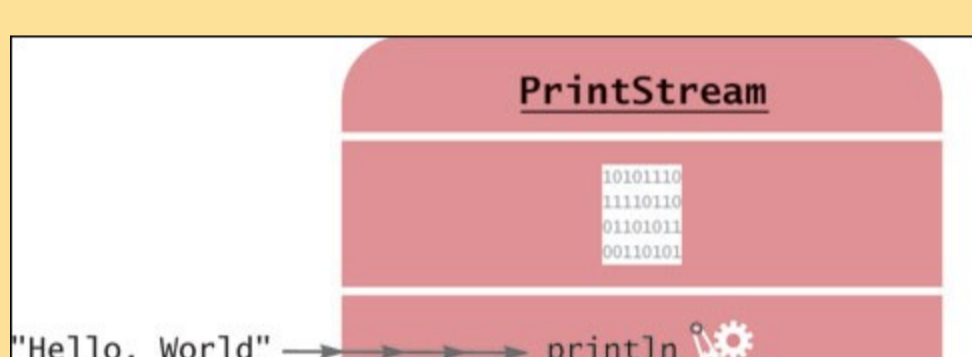
1. How can you compute the length of the string “Mississippi”?
2. How can you print out the uppercase version of “Hello, World!”?
3. Is it legal to call river.println()? Why or why not?

The length method differs from the println method in another way: it has an output. We say that the method *returns a value*, namely the number of characters in the string. You can store the return value in a variable:

```
int n = greeting.length();
```

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

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



Technically speaking, the greeting parameter is an explicit parameter of the println method. The object on which you invoke the method is also considered a parameter of the method call, called the implicit parameter. For example, System.out is the implicit parameter of the method call

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

Some methods require multiple explicit parameters, others don't require any explicit parameters at all. An example of the latter is the length method of the String class. All the information that the length method requires to do its job—namely, the character sequence of the string—is stored in the implicit parameter object.

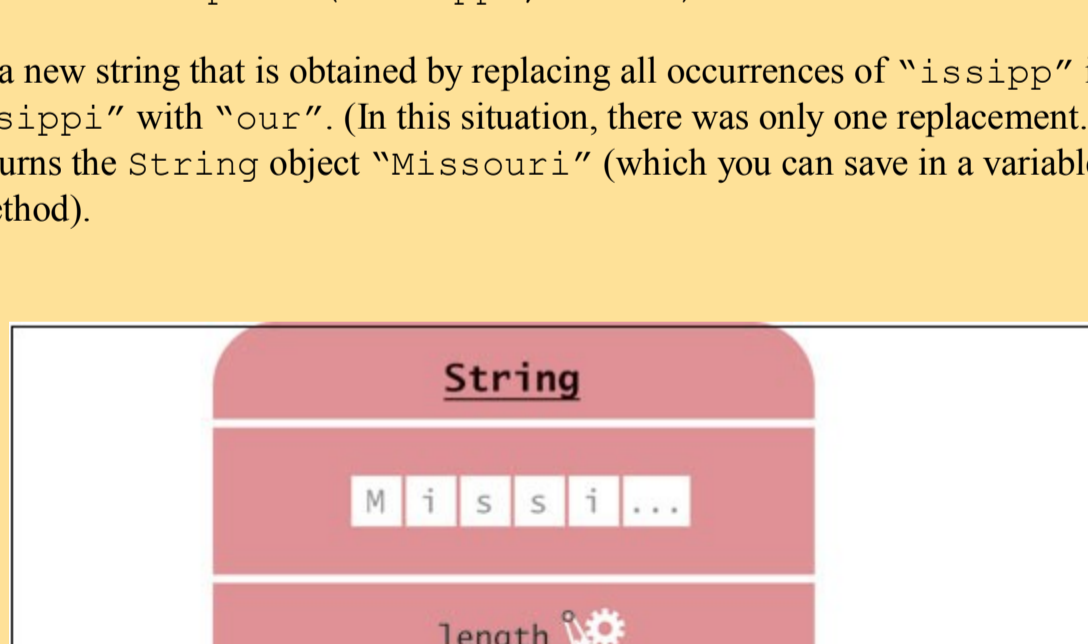
```
greeting.length()
```

Not all methods return values. One example is the println method. The println method interacts with the operating system, causing characters to appear in a window. But it does not return a value to the code that calls it.

Let us analyze a more complex method call. Here, we will call the replace method of the String class. The replace method carries out a search-and-replace operation, similar to that of a word processor. For example, the call

```
river.replace("issipp", "our")
```

constructs a new string that is obtained by replacing all occurrences of “issipp” in “Mississippi” with “our”. (In this situation, there was only one replacement.) The method returns the String object “Missouri” (which you can save in a variable or pass to another method).



- * one implicit parameter: the string “Mississippi”
- * two explicit parameters: the strings “issipp” and “our”
- * a return value: the string “Missouri”

When a method is defined in a class, the definition specifies the types of the explicit parameters and the return value.

For example, the String class defines the length method as

```
public int length()
```

That is, there are no explicit parameters, and the return value has the type int.

The type of the implicit parameter is the class that defines the method.

In this case, String. It is not mentioned in the method definition—hence the term “implicit”.

The replace method is defined as

```
public String replace(String target, String replacement)
```

To call the replace method, you supply two explicit parameters, target and replacement, which both have type String. The returned value is another string.

When a method returns no value, the return type is declared with the reserved word void. For example, the PrintStream class defines the println method as

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

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

4. What are the implicit parameters, explicit parameters, and return values in the method call river.length()?

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

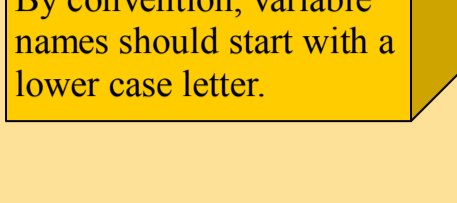
6. What is the result of the call greeting.replace("World", "Dave").length()?

7. How is the toUpperCase method defined in the String class?

Identifiers for variables, methods, and classes are composed of letters, digits, and underscore characters.

An *identifier* is the name of a variable, method, or class. Java imposes the following rules for identifiers:

- Identifiers can be made up of letters, digits, and the underscore (`_`) and dollar sign (`$`) characters. They cannot start with a digit, though. For example, `greeting1` is legal but `1greeting` is not.
- You cannot use other symbols such as `?` or `%`. For example, `hello!` is not a legal identifier.
- Spaces are not permitted inside identifiers. Therefore, `lucky number` is not legal.
- Furthermore, you cannot use reserved words, such as `public`, as names; these words are reserved exclusively for their special Java meanings.
- Identifiers are also case sensitive ; that is, `greeting` and `Greeting` are *different*.



Other conventions to follow so that other programmers will find your programs easy to read:

- Variable and method names should start with a lowercase letter. It is OK to use an occasional uppercase letter, such as `LuckyNumber`. This mixture of lowercase and uppercase letters is sometimes called “camel case” because the uppercase letters stick out like the humps of a camel.
- Class names should start with an uppercase letter. For example, `Greeting` would be an appropriate name for a class, but not for a variable.

4. What is the Type of the Values 0 and “0”?

5. Which of the following are legal identifiers?

Greeting1

g

void

101dalmatians

Hello, World

<greeting>

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

Number Types

Constructing objects revisited

Accessor and Mutator Methods

Implementing a test program revisited

The API documentation

Object References

GUI's

