Object-Oriented Programming I

Methods

Slides by Magdin Stoica
Updates by Georg Feil

Learning Outcomes

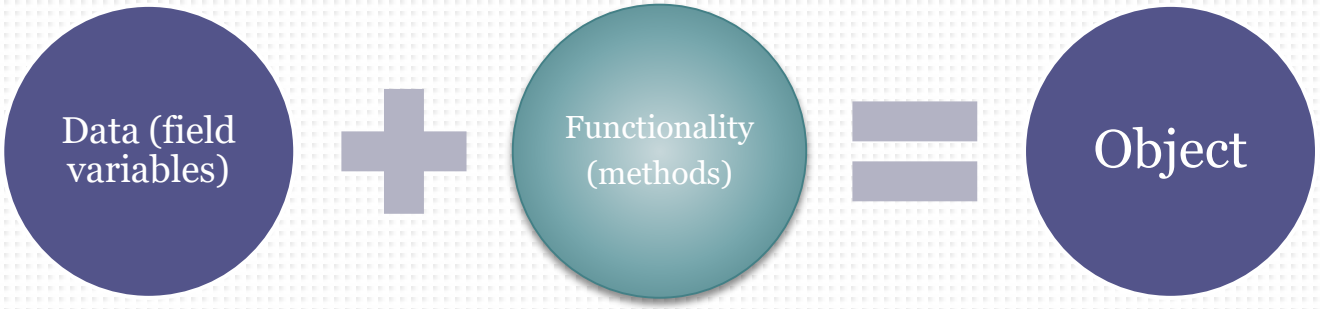
1. Characterize a method, its purpose and properties
2. Identify the role objects play in the process of calling a method to execute the functionality it defines
3. Analyze the syntax of method declarations and definitions as well as the relationship between the method declarations and definitions
4. Identify naming conventions used for naming methods
5. Analyze the syntax and process of calling a method
6. Use return statements to communicate the result of a method to the caller
7. Create object-oriented programs that use various methods with and without return value

Reading Assignments

- Introduction to Java Programming (required)
  - Chapter 5: Methods (except sections 5.8, 5.10.5 and 5.11)
    - Please ignore the “static” keyword. Static methods are used in this chapter to create programs without objects, procedural programs. We will not use static in this fashion.
- Head First Java (recommended) (See link in SLATE)
  - Chapter 4: Methods Use Instance Variables: How Objects Behave

What's in a class?

- Information or data (what the objects will be made of) and functionality (what the object will be able to do)
- The elements of a class that contain information (data) are called variables. The variables defined for the whole class are called fields.
- The elements of a class that execute commands to implement functionality are called methods.



What are **programs** made of?

Objects

What are **objects** made of?

Variables declared inside the object called fields, and methods

What can objects **do**?

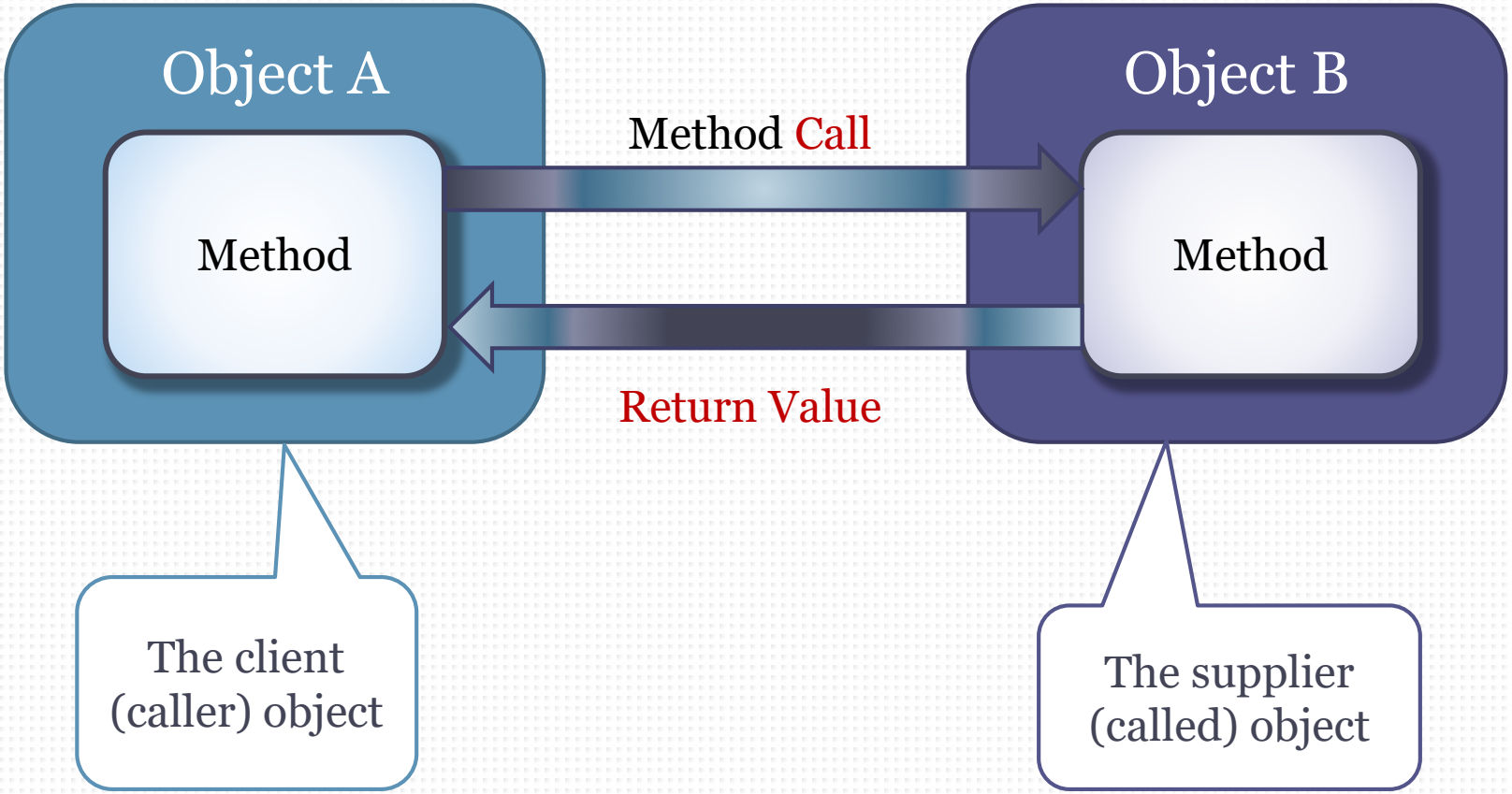
Methods contain statements that carry out the object's functionality

(We've seen an example already, the main method)

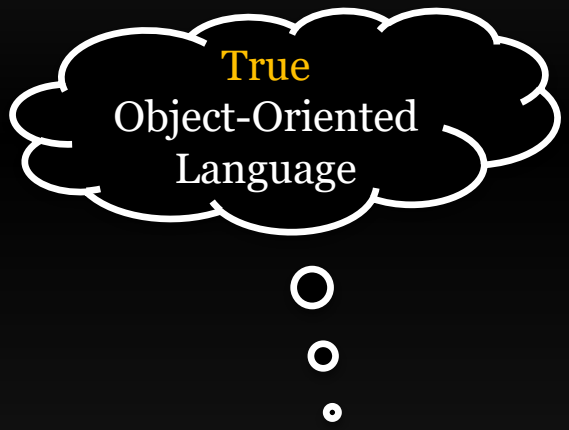
What is a method?

- A method is a block of statements defined in a class
  - The statements have a common purpose, solving a unique problem
  - The statements are related to the object, working with the variables defined in the object, its fields
- The statements defined inside the method are executed when the method is called or invoked
- A method has two parts
  - Declaration: introduces the method and instructs the compiler to consider the following block as a block of functionality. Also called the method's signature
  - Definition: defines the actual method block, the collection of statements that execute when the method is called or invoked

Example: Clients and Suppliers



In Java,
methods
can only be
defined
within a class



Method Declaration vs. Definition

(pseudocode)

Class declaration

```
public class <Name of Class>
{
```

Class definition

```
    …
```

Methods are ALWAYS defined in a class

Method declaration

```
{
    statement 1;
    …
}
```

Method declaration

Method Definition

No semicolon

```
}
```

Real Example of a Method

(Does not return any value, does not take any parameters)

Class declaration

```
public class Circle
{
```

Class definition

Method declaration

```
public void printArea()
```

Method Definition

```
{
    System.out.println(radius * radius * 3.14);
}
```

```
}
```

Real Example of a Method + Field Variable

Class declaration

```
public class Circle
{
```

Class definition

```
    private double radius;

    public void printArea()
    {
        System.out.println(radius * radius * 3.14);
    }
}
```

Method declaration

Method Definition

A method declaration
is always followed
by the method
definition block

A method's declaration is also called the method's signature

Method Properties

- A method has a list of “modifiers”, a return type, name and list of parameters
  - The method declaration defines all the method properties
- Modifiers
  - Visibility is a modifier that determines what parts of the program can access (call) the method. Works just like field variable visibility.
- Return Type
  - The type of the information returned by the method (e.g. int, double, Circle, etc.) used when the method calculates or generates a value
  - If the method does not return a value, its return type is defined as “void”
- Name identifies the method
- List of parameters declares the input variables needed by the method

The anatomy of a method

(pseudocode)

The sign of a method

Method declaration

```
<visibility> <return type> <method name> (T1 p1, T2 p2, T3 p3)
```

List of parameters

Method Definition

```
{
        statement 1;
        statement 2;
        …
        statement n;
}
```

What the method does: the set of statements that execute when the method is called (invoked)

NO semicolon

NO semicolon

Method Names

- By convention method names start with a lower-case letter with the subsequent words capitalized (e.g. calculateArea)
  - Can contain numbers and _ but they rarely do
  - Same naming convention as a variables but should not start with _
- Method names should represent an action not a thing
  - They are the elements of an object that DO things
  - Methods are called, they execute, they may calculate a value
  - Names are usually made of two words: a verb (predicate) and noun (object). If only one word is used it should be the verb.
- Have to be descriptive to express what the method does
  - do(...) // what does this method do?
  - calculateArea() // what does this method do?

Method names vs. variable names

Method names seem to have the same convention as variable names.

1. How can we differentiate them?
2. Why would it be useful to differentiate them?

Exercise 1: Defining methods

- We'll extend the "Variables Example" program on the second-last slide of the "Inside Classes – Variables" slides
- Break down the main method into two new methods of the ProgramV class, as follows:
- Create a method called "helloOne"
  - Does not return any value (what is the return type?)
  - Does not take any parameters (what is the parameter list?)
  - Move the first part of the main method up to the 2nd println to this method
- Create a method called "helloTwo"
  - Does not return any value
  - Does not take any parameters
  - Move all the remaining lines from the main method into this method
- Compile and run the program
  - What do you notice?

Method parameters

- Method parameters are input variables whose values are provided by the calling code
  - Also called arguments
  - The caller supplies the values needed
  - The runtime places the values inside the parameter variables
  - The method "sees" the values through the parameter variables
- The list of method parameters is enclosed in (parenthesis)
- A parameter is declared like any other variable <type> <name> but *without a semicolon*
- When multiple parameters are needed they are separated using a comma
- It is common for methods to *not* need any parameters
  - Empty parentheses are still needed: ()

If I see ( ) it must be a method

Parameters or not, method names are ALWAYS followed by parentheses ( )

Example: Method with one parameter

One parameter of type double and name radius

Method declaration

```
public void printCircleArea (double radius)
```

Method Definition

```
{
    double area = radius * radius * 3.14;

    System.out.println(area);

}
```

What the method does: the set of statements that execute when the method is called (invoked)

Container class omitted...

Example: Method with two parameters

Two parameters. Both have the type double and are named length and width. Divided using a comma.

Method declaration

```
public void printRectArea (double length, double width)
```

Method Definition

```
{
    double area = length * width;

    System.out.println(area);
}
```

What the method does: the set of statements that execute when the method is called (invoked)

Container class omitted...

Calling (Invoking) a method

- A method is called using a method call statement which specifies
  - The name of the method to be called
  - The list of values to be passed as parameters. Values are expressions which are made of variables, literals or combinations of variables and literals using operators.
    - The list of values is enclosed in parentheses
    - If there is more than one parameter the values are separated by commas
- If the method returns a value, the method call can be used as a value in an expression
  - The value returned may be stored in a variable, used in a calculation, printed out, passed as a parameter etc.
  - If the caller is not interested in the value they are free to ignore it

The anatomy of a method call (no object)

```
<method name>(<value1>, <value2>, …) ;
```

The “sign” of a method

The name of the method being called

List of values separated by comma

The type is not needed, just a value, literal or expression

The anatomy of a method call

The name of the object whose method is called

The "sign" of a method

```
<object name>.<method name>(<value1>, <value2>, …);
```

. (dot) operator

The name of the method being called

List of values separated by comma

The type is not needed, just a value, literal or expression

Method Call Examples

```
setRadius(20);  // Method is part of the same class
calculator.printSum(3,5);
rect.drawRectangle(2,6,num, 16);
setFirstName("Barack");
```

Methods that perform an action but do not return a value

```
double area = circle.calculateArea();
double diameter = circle.getRadius() * 2;
String firstName = getFirstName();
System.out.println(person.getName());
```

Methods that return a value

Exercise 2: Variable Tester Fix

- Fix the variable tester such that it displays the output as before
  - Create a ProgramV object variable named 'self' inside the main method
- Use the new object variable to
  - Call the helloOne() method from the main method
  - Call the helloTwo() method from the main method
- Compile and run the program
  - Do you notice an improvement?
  - What happens if you try to call a method from main without using an object variable?

Exercise 2b: Method Call w/o Object

- In Exercise 2, your main method calls two other methods using an object variable
  - The methods called are named helloOne and helloTwo
  - The object variable is needed because the calls are inside the main method, which is static (this means it can't "access" regular methods or fields)
- Now change the program so that the main method just calls helloOne.
- Call the helloTwo() method from the helloOne() method
  - You do not need an object variable
  - You do not need to use the dot operator
  - Just call the method as follows: `helloTwo();`
  - Run the program and single-step using the debugger

We'll only make objects like 'self ' in the main method, so we can access (non-static) fields & methods

What's the benefit of using methods?

- You may be wondering why you should use methods
  - Why not just put all your code in the main method?
- Dividing code into different methods makes programs modular (composed of separate well-defined pieces)
  - You could think of each method or class as one module
  - Modular programs are easier to understand
- Solving a big programming problem is easier with modules ("divide-and-conquer")
- Using methods properly is more efficient and reliable
  - Allows you to reuse code and reduce code copying
  - Instead of copying code and maybe introducing a typo or a bug, you can call a method which has been tested and is known to work

Exercise 2c: Proper use of ‘static’

- In our program for Exercise 2b we still have a field variable that’s declared with the word ‘static’
  - We want to avoid using ‘static’ except for the main method declaration
- It’s safe now to remove the word static from the declaration of the field variable ‘pi’ since we’re no longer using it inside the main method
- … remove it and test your program to be sure it still works

The return statement (keyword)

- How does a method return a value it calculated inside to the caller?
  - Using the "return" statement
- The return statement consists of the **`return`** keyword followed by an expression (literal, variable, or calculation)

```
public int addNumbers(int num1, int num2)
{
        return num1 + num2;
}
```

keyword

expression

return statement

Don't forget the semicolon. It is a statement

Example: Method that returns a value

Class declaration

```
public class Circle
{
```

Class definition

```
    private double radius;

    public double calculateArea()
    {
        double area = radius * radius * 3.14;
        return area;
    }

}
```

Method declaration

Method Definition

No parameters for this method

A method can use any field variable defined in the same class

Example: Method with two parameters that returns a value

Two parameters. Both have the type double and are named length and width. Divided using a comma.

```
public double calculateRectArea (double length, double width)
{
    double area = length * width;
    return area;
}
```

Container class omitted…

Barking Dogs: An Example Program for Variables & Methods

- The next two slides show an example program that puts together the ideas of variables and methods
- It's based on the example from page 73 of Head First Java, but I've simplified it
- We'll be revisiting this example later and expanding it

Example Prog: “The size affects the bark” (pg 1)

```
package sheridan;
public class Dog {
    public String name;

    public void bigBark() {
        System.out.println(name + " says Woof! Woof!");
    }

    public void smallBark() {
        System.out.println(name + " says Yip! Yip!");
    }

…
```

Example Prog: “The size affects the bark” (pg 2)

```
    public static void main(String[] args) {
        Dog one = new Dog();
        one.name = "Thor";
        Dog two = new Dog();
        two.name = "Tiny";
        one.bigBark();
        two.smallBark();
    }

} // End of class Dog
```

Exercise 3: “The size affects the bark”

- Copy / paste the example program into Dr. Java
  - Copy both pages, one after the other
  - The class must be saved in a file called Dog.java
  - Fix formatting, indenting etc.
  - Have only this file open
- Compile by pressing the Compile button and fix any errors
- Run by pressing the Run button
- Try tracing through the program using Debug mode
  - Set a breakpoint at the start of main, then use “step into”

Now let’s jump to a different set of slides to learn about user input…

Exercise 4: Using private field variables

- The next two slides contain a modified version of the barking dogs example program with the field variable made *private* instead of *public*
  - Something else has been added
- Can you see what has changed?
- Enter this as a new file called Dog2.java and try it out

Example Prog: With private field variable (pg1)

```
package sheridan;
public class Dog2 {
    private String name;

    public void setName(String newName) {
        name = newName;
    }
    public void bigBark() {
        System.out.println(name + " says Woof! Woof!");
    }

    public void smallBark() {
        System.out.println(name + " says Yip! Yip!");
    }
…
```

Example Prog: With private field (pg 2)

```
    public static void main (String[] args) {
        Dog2 one = new Dog2();
        one.setName("Thor");
        Dog2 two = new Dog2();
        two.setName("Tiny");
        one.bigBark();
        two.smallBark();
    }

} // End of class Dog2
```

Exercise 4 Conclusion

- When a field variable is private we can use a “setter” method to set its value from outside the class
  - This example is all within one class, so a “setter” is not strictly required
  - We will soon see larger programs with more than one class
  - Note: We’ll learn all about the idea of “getter” and “setter” methods in a few weeks

Exercise 5: Barking dogs with string input

- Starting with the program from Exercise 4, modify the program so that the user is prompted to enter the name of each dog.
- Read the user input before setting the dogs' names