Introduction to OOP with Java

Instructor: AbuKhleif, Mohammad Noor
Sep 2017
www.abukhlief.com

Lecture 05:

Methods

Instructor: AbuKhleif, Mohammad Noor
Sep 2017
www.abukhlief.com
Instructor

- AbuKhleif, ‘Mohammad Noor’
- Computer Engineer (JU 2012-2017)
- Software Automation Engineer @ Atypon – John Wiley and Sons Company - Jordan Branch

- Reach me at:
  - www.abukhleif.com
  - moh.noor94@gmail.com
  - facebook.com/moh.noor94
  - twitter.com/moh_noor94

Course

- Java SE Basics
- Object Oriented Programming
- Course Page:
- Or, go to: www.abukhleif.com → Courses → Java 101 Course – Sep 2017
- Course Facebook Group:
  - www.facebook.com/groups/AKF2017Java
Let’s Start!

Opening Problem

• Find the sum of integers from 1 to 10, from 20 to 30, and from 35 to 45, respectively.
Opening Problem – *(Solution?)*

```java
int sum = 0;
for (int i = 1; i <= 10; i++)
    sum += i;
System.out.println("Sum from 1 to 10 is " + sum);

sum = 0;
for (int i = 20; i <= 30; i++)
    sum += i;
System.out.println("Sum from 20 to 30 is " + sum);

sum = 0;
for (int i = 35; i <= 45; i++)
    sum += i;
System.out.println("Sum from 35 to 45 is " + sum);
```

Motivations

Methods can be used to define **reusable** code and **organize** and simplify code

- It would be nice to write the common code once and reuse it.
- This is achieved by:
  - Defining a method that contains the common code.
  - Reuse it by invoking it with different values.
Methods

• A method is a collection of statements that are grouped together to perform an operation.

```java
public static int max(int num1, int num2) {
    int result;
    if (num1 > num2)
        result = num1;
    else
        result = num2;
    return result;
}
```
Methods

- *Method signature* is the combination of the method name and the parameter list.

```java
public static int max(int num1, int num2)
```

- The variables defined in the method header are known as *formal parameters*.

```java
max(int num1, int num2)
```

- When a method is invoked, you pass a value to the parameter. This value is referred to as *actual parameter or argument*.

```java
int z = max(x, y);
```

Return Value Type

- A method may return a value. The *returnValueType* is the data type of the value the method returns.

- If the method does not return a value, the *returnValueType* is the keyword *void*.
  - For example, the *returnValueType* in the `main` method is *void*.
Example

```java
public class TestMax {
    /** Main method */
    public static void main(String[] args) {
        int i = 5;
        int j = 2;
        int k = max(i, j);
        System.out.println("The maximum between " + i + 
        " and " + j + " is " + k);
    }
    /** Return the max between two numbers */
    public static int max(int num1, int num2) {
        int result;
        if (num1 > num2) {
            result = num1;
        } else {
            result = num2;
        }
        return result;
    }
}
```

Example, cont.

- Calling a method executes the code in the method.
- The main method is just like any other method except that it is invoked by the JVM to start the program.
Example, cont.

- When a program calls a method, program control is transferred to the called method.
- A called method returns control to the caller when:
  - Either its return statement is executed, or
  - Its method-ending closing brace is reached.

Caution

- A return statement is required for a value-returning method.
- The method shown below in (a) is logically correct, but it has a compilation error because the Java compiler thinks it possible that this method does not return any value.

```java
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else if (n < 0)
        return -1;
}
```

Should be:

```java
public static int sign(int n) {
    if (n > 0)
        return 1;
    else if (n == 0)
        return 0;
    else
        return -1;
}
```
Let’s Code

- Write a program that have a method called ‘printMultiplications’.
- ‘printMultiplications’ methods receives 2 integers: ‘x’ and ‘y’ and prints first ‘y’ multiplications of ‘x’ each on a line.
- Use the ‘printMultiplications’ with, at least, 3 examples in the main method.

What Happens when a Method is Invoked?
What happens when a method is invoked?

- Each time a method is invoked, the system creates an activation record.
- Activation record stores parameters and variables for the method.
- Activation record is placed in an area of memory known as the call stack, or simply the stack.
- When a method invokes another method, the caller’s activation record is kept intact, and a new activation record is created.
- When a method finishes its work and returns to its caller, its activation record is removed from the stack.
- A call stack stores methods in last-in, first-out fashion.

(Example)

(a) The main method is invoked.

(b) The max method is invoked.

(c) The max method is being executed.

(d) The max method is finished and the return value is sent to k.

(e) The main method is finished.

Stack is empty
void Methods

public class TestVoidMethod {
    public static void main(String[] args) {
        System.out.print("The grade is ");
        printGrade(78.5);
        System.out.print("The grade is ");
        printGrade(59.5);
    }
    public static void printGrade(double score) {
        if (score >= 90.0) {
            System.out.println('A');
        } else if (score >= 80.0) {
            System.out.println('B');
        } else if (score >= 70.0) {
            System.out.println('C');
        } else if (score >= 60.0) {
            System.out.println('D');
        } else {
            System.out.println('F');
        }
    }
}

void Methods Example
Passing Parameters

public static void nPrintln(String message, int n) {
    for (int i = 0; i < n; i++)
        System.out.println(message);
}

• Suppose you invoke the method using
  nPrintln(“Welcome to Java”, 5);
  What is the output?

• Suppose you invoke the method using
  nPrintln(“Computer Science”, 15);
  What is the output?

• Can you invoke the method using
  nPrintln(15, “Computer Science”);
Passing Parameters

• When calling a method, you need to provide arguments, which must match the parameters defined in the method signature in:
  • Order
  • Number
  • Compatible type

• When you invoke a method with an argument, the value of the argument is passed to the parameter.
  • This is referred to as pass-by-value.
  • If a value of a variable is passed as an argument to a parameter, the variable is not affected, regardless of the changes made to the parameter inside the method.

Example

```java
public class Increment {
    public static void main(String[] args) {
        int x = 1;
        System.out.println("Before the call, x is " + x);
        increment(x);
        System.out.println("after the call, x is " + x);
    }

    public static void increment(int n) {
        n++;
        System.out.println("n inside the method is " + n);
    }
}
```
Modularizing Code

• Modularizing makes the code:
  • Clear and easy to read.
    • Isolates parts used to perform specific computations from the rest of the code.
  • Easy to maintain and debug.
    • Narrows the scope of debugging.
  • Reusable.
    • Code can be reused by other programs (or other classes).
Overloading Methods

- Two methods that have the same name, but different parameter lists within one class.
- The Java compiler determines which method to use based on the method signature.
  - It finds the most specific method for a method invocation.
```java
public class TestMethodOverloading {
    /** Main method */
    public static void main(String[] args) {
        System.out.println("The maximum of 3 and 4 is " + max(3, 4));
        System.out.println("The maximum of 3.0 and 5.4 is " + max(3.0, 5.4));
        System.out.println("The maximum of 3.0, 5.4, and 10.14 is " + max(3.0, 5.4, 10.14));
    }

    /** Return the max of two int values */
    public static int max(int num1, int num2) {
        if (num1 > num2) return num1;
        else return num2;
    }

    /** Find the max of two double values */
    public static double max(double num1, double num2) {
        if (num1 > num2) return num1;
        else return num2;
    }

    /** Return the max of three double values */
    public static double max(double num1, double num2, double num3) {
        return max(max(num1, num2), num3);
    }
}
```

**Example**

**Ambiguous Invocation**

- Sometimes there may be two or more possible matches for an invocation of a method, but the compiler cannot determine the most specific match.
- This is referred to as **ambiguous invocation**.
- Ambiguous invocation is a compilation error.
Ambiguous Invocation - Example

```java
public class AmbiguousOverloading {
    public static void main(String[] args) {
        System.out.println(max(1, 2));
    }

    public static double max(int num1, double num2) {
        if (num1 > num2) {
            return num1;
        } else {
            return num2;
        }
    }

    public static double max(double num1, int num2) {
        if (num1 > num2) {
            return num1;
        } else {
            return num2;
        }
    }
}
```

Scope of Variables
Scope of Variables

• The scope of a variable is the part of the program where the variable can be referenced.
• A variable defined inside a method is referred to as a local variable.
• The scope of a local variable starts from its declaration and continues to the end of the block that contains the variable.
  • A local variable must be declared before it can be used.

Scope of Variables, cont.

• A parameter is actually a local variable.
  • The scope of a method parameter covers the entire method.
• You can declare a local variable with the same name multiple times in different non-nesting blocks in a method.
  • But you cannot declare a local variable twice in nested blocks.
Scope of Variables, cont.

• A variable declared in the initial action part of a `for` loop header has its scope in the entire loop.
• But a variable declared inside a `for` loop body has its scope limited in the loop body from its declaration and to the end of the block that contains the variable.

```java
public static void method1() {
    for (int i = 1; i < 10; i++) {
        int j;
    }
}
```

It is fine to declare `i` in two non-nesting blocks:

```java
public static void method1() {
    int x = 1;
    int y = 1;
    for (int i = 1; i < 10; i++) {
        x += i;
    }
    for (int i = 1; i < 10; i++) {
        y += i;
    }
}
```

It is wrong to declare `i` in two nesting blocks:

```java
public static void method2() {
    int i = 1;
    int sum = 0;
    for (int i = 1; i < 10; i++) {
        sum += i;
    }
}
```
Methods Abstraction and Benefits of Methods

- You can think of the method body as a black box that contains the detailed implementation for the method.

```
Method Header

<table>
<thead>
<tr>
<th>Optional arguments for Input</th>
</tr>
</thead>
<tbody>
<tr>
<td>Optional return value</td>
</tr>
</tbody>
</table>
```

Black Box
Benefits of Methods

• Write a method once and reuse it anywhere
• Information hiding
  • Hide the implementation from the user.
• Reduce complexity

Program Design
Top-Down Design

• Top-down approach is to implement one method in the structure chart at a time from the top to the bottom.
• Stubs can be used for the methods waiting to be implemented.
  • A stub is a simple but incomplete version of a method.
  • The use of stubs enables you to test invoking the method from a caller.
  • Implement the main method first and then use a stub for the each method to test.

Bottom-Up Design

• Bottom-up approach is to implement one method in the structure chart at a time from the bottom to the top.
• For each method implemented, write a test program to test it.
Which Design Approach to Choose?

• Both top-down and bottom-up methods are fine.
• Both approaches implement the methods incrementally and help to isolate programming errors and makes debugging easy.
• Sometimes, they can be used together.
• You, choose!
• I will choose, let’s say always, the ‘top-down’ approach.

Self Reading

• A common error for the exercises in this chapter is that students don’t implement the methods to meet the requirements even though the output from the main program is correct.
• For an example of this type of error see www.cs.armstrong.edu/liang/CommonMethodErrorJava.pdf.
Tasks

All tasks should be well-documented, well-designed, and well-styled.

Task 01

(Check vowels)

• Write a method `isVowel`. That returns the value true if a given character is a vowel and otherwise returns false.
Task 02

(Display an integer reversed) *

• Write a method with the following header to display an integer in reverse order:

public static void reverse(int number)

• For example, reverse(3456) displays 6543.
• Write a test program that prompts the user to enter an integer and displays its reversal.
• Hint: Use %10 to get the last digit of a number and /10 to get rid of it.

Task 03

(Check password) **

• Some websites impose certain rules for passwords.
• Write a method that checks whether a string is a valid password.
• Suppose the password rules are as follows:
  • A password must have at least eight characters.
  • A password consists of only letters and digits.
  • A password must contain at least two digits.
• Write a program that prompts the user to enter a password and displays Valid Password if the rules are followed or Invalid Password otherwise.
Tasks Submission

• Submit a zipped file contains all the 3 .java files, 1 file for each task.
• Name your zipped file as follow [Lect5_YourName.zip].
• Upload your zipped file to the Facebook group.
• Submission due: Wednesday, Sep 20 - **10:00 PM**
• Late submission will not be reviewed by the instructor.
• Public solutions upload goal is to share knowledge, you can see other’s solutions, but, please, don’t cheat yourself!
• Don’t forget, all tasks should be well-documented, well-designed, and well-styled.

Test Yourself

• Answer all question:

References:

- Liang, Introduction to Java Programming 10/e
- Eng. Asma Abdel Karim Computer Engineering Department, JU Slides.

Instructor: AbuKhleif, Mohammad Noor
Sep 2017
www.abukhleif.com

End of Lecture =D