# CS 225L Lab 02: Methods

# Learning Outcomes

* Develop simple methods within a single class.
* Utilize methods to modify object attribute values.
* Become familiar with the format of method signatures
* Become familiar with the use of public, private, and static keywords
* Become familiar with method return types

# Pre-Lab

There are no pre-lab deliverables for this lab.

This lab give you practice in writing methods. Methods are also known as functions

or procedures, but the term ‘methods’ is preferred in the object-oriented world.

## Why Methods? DRY!

One of the most significant principles in software engineering is known as the “Don’t Repeat Yourself” Principle, or DRY Principle. In general, when you are typing very similar code repeatedly, there is a high chance of making mistakes while you type. Copy/pasting code in blocks is also a bad idea: A bug in the original source block is propagated to all of the copies of the block. In addition, code changes must also be made to each copy of the block.

Methods allow us to avoid these problems. Instead of copying the raw code from place to place, a method allows you to define a block of code to do a particular task once, then refer to it by name wherever the task arises. Changes only have to be made in one place, but they propagate everywhere the name is used.

## Java Syntax for Methods

In general, a method definition in Java looks like this:

<visibility modifiers> <static (optional)> <return type> <methodName >( <parameter list> ) {

[[ Method body : the code that does the work ]]

}

The visibility modifiers determine which objects can access the method: public methods are accessible from any object, while private methods can only be used by the object containing them. The optional static keyword determines when the method is available. Static methods exist – and can be used - prior to the creation of the object, while non-static methods do not exist – and cannot be used - until an object of the class containing them is created. For example, the *main* method is static, making it available for use prior to the creation of any objects. The return type specifies the type of data returned by the method, if any. Methods that do not contain a return statement must have the type *void*. Methods with a non-void return type must have a return statement in the body of the method. The methodName can be any text that does not violate the rules for naming things in Java. Stick with letters, numbers, and underscores. By convention, you should start your method names with a lowercase letter and capitalize the first letter of every new word that appears in the name. For example: thisIsAReallyLongMethodName(). Method names should tell a human reader what the method does. Finally, the parameter list specifies the inputs required by the method. The parameter list is composed of data type – variable name pairs separated by commas: for example, int var1, int var2, double var3. The method body comprises the statements that are executed when the method is called.

Here are some random examples of method headers:

public static void main ( String [] args ) // Everyone 's favorite

public void start ( Stage stage ) // Your future favorite ( Stage is a GUI class )

public boolean equals ( Object obj ) // Object - oriented ==

public static int parseInt ( String s) // String to int conversion

public static int [] copyOf (int [] original , int newLength ) // Array copying

public void println () // System .out .<-

In each of these cases, it should be obvious what the purpose of the method is (from its name), what inputs you need (from the parameter list), and the type of output you can expect (the return type).

Note that non-void methods can stand in for variables that match the return type:

public class MethodsEverywhere {

public static int integerGen () {

return ( int) System . currentTimeMillis ();

}

public static void main ( String [] args ) {

// You can assign a var ...

int i = integerGen ();

// Or use it more directly -- anywhere an int can go!

int twice = integerGen () + integerGen ();

System . out . println ( integerGen ());

}

}

Methods may even stand in as parameters for other methods.

public class AllAtOnce {

public static void repeatAwesome (int times ) {

for(int i = 0; i < times ; i++) {

Awesome . myAwesomeMethod ();

}

}

public static void main ( String [] args ) {

repeatAwesome ( MethodsEverywhere . integerGen ());

}

}

## The Java API and JavaDocs

One advantage of Java over other languages extremely diverse and well-documented API that comes with the language. The developers of Java have already built over four-thousand classes, each with plenty of methods for your use. You can look at the root of the documentation at: <https://docs.oracle.com/javase/8/docs/api/>

You can browse through this website to explore the myriad of packages and classes. I recommend looking at the java.lang and java.util packages for some commonly-used goodies. If you're looking for a specific class, the API website is well-indexed by search engines, so searching for something like “java 8 Date class" will likely give you the API website as the top result. (I specify the 8 in particular since the APIs for older Java versions are still online.)

Once you drill down to an individual class, you'll get plenty of information, in this order:

1. Name and hierarchy information (what class does this extend, what

extends this)

2. Usage summary (a textual description)

3. Constructor Summary (making objects of this class|see next lab)

4. Method Summary (What can you actually do with this class?)

5. Detailed constructor/method descriptions

It's actually possible to generate a clone of the API website if you use a special form of comments called JavaDoc comments. They resemble regular block comments, except they start with two asterisks: /\*\* JavaDoc... \*/ Eclipse will highlight these comments in a different color than normal comments. Information you write in JavaDoc comments will show up in Eclipse's Documentation view, and will also pop up when you mouse-over documented methods. If you want to know more, you can read up here.

## Pre-Lab Exploration

Note that the blue text is clickable (points to a web URL).

1. Explore the documentation for the Math class. Be prepared to list five mathematical operations that you can perform using the Math class.

2. Explore the documentation for the Arrays class. Be prepared to list three things that you can do with arrays using the methods in this class.

# Lab 2: Methods

Instructions for the lab:

1. Import the java file into Eclipse
   1. Download this document and the MethodsLabExercise.java file.
   2. In Eclipse, select File ! Import... (or right-click in the Package Explorer ! Import...)
   3. Select General, Existing Projects into Workspace and click Next.
   4. Select “Select archive file:" and find the java file. Click Finish.
2. Examine MethodsLabExercise.java. Make sure you understand what changes are required by the lab.
3. The method required to allow a user to input an array of integer values is already written in the Java file. Create the other methods specified:
   1. calculateSum(): Simply sums the elements of the integer data.
   2. findMax(): Identifies the largest data element.
   3. findMaxMult(): find the largest value that can be obtained by multiplying two different data elements together.
   4. calculateAverage(): Calculate the average value of the data elements. This is simply the sum divided by the number of elements.
   5. calculateStdDev(): Look online, or use the following procedure ….
      1. Find the average value for the data, µ
      2. Sum the value of (di – µ)2 for each data element di
      3. Divide by the number of data elements, N
      4. Find the square root of the result.
      5. The equation for this is stdDev =
   6. printResults(): Prints out results as listed in the Java file.
4. Demonstrate your completed program to a lab instructor for grading.

# Post-Lab 2: Methods

There are no post-lab activities or deliverables required for this lab.