# CS 225L Lab 03: Methods-Racer

# 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

## 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.

## Pre-Lab Exploration

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

1. Explore the documentation for the [Math class](https://docs.oracle.com/javase/8/docs/api/java/lang/Math.html).

# Lab 3: Methods-Racer

Racer Game Requirements:

1. The program shall have the race be 100 units long.
2. The program shall have 4 racers in the race.
3. The Racers are name the following Racer1, Racer2, Racer3, Racer4.
4. Racer1 shall move randomly either 4 or 8 units per turn.
5. Racer2 shall move randomly 2 to 10 units per turn.
6. Racer3 shall move randomly either
   1. A range from 0 to 10
   2. 7 \* cos(7)
7. Racer4 shall move spaces per turn but must move at least 1 unit.
8. The program shall end the race once 1 racer passes the ending point.
9. The program shall print out who won the race.

Lab 3 Requirements:

1. The program shall use method called Racer1Move to move Racer1.
2. The program shall use method called Racer2Move to move Racer2.
3. The program shall use method called Racer3Move to move Racer3.
4. The program shall use method called Racer4Move to move Racer4.

Given these requirements, build the racing game using the given Racer.java.