In this section, we will cover using methods. Encapsulated code that can be re-used through-out our programs.

Methods

CSCI 1250 Study Guide

Schneider, Michael Joseph

Method Signature

The starting point for creating a method is the method's signature. A method signature helps the compiler to know what type of data the method needs to take in (*parameters*) and what data the method will give back (*returnType*).

Access Specifier

The access specifier of a method tells the compiler "who" can access the method. The possible access specifiers for Java are:

`public`

-- Open Access –

The public access specifier states that anyone may call this method.

`private`

-- Restricted Access –

The private access specifier states that only "this" class may use the method. I.e. only the class that created the method may access the method.

`protected`

-- Semi-Restricted Access—

The protected access specifier is not used in our class, it plays a role in how access to methods is restricted to sub-classes (inheritance) and classes in the same Java package. You will get more into inheritance in future CS courses.

Static or Not-Static

`static` tells the compiler "when" the method should be placed into memory. If a method is static, it is placed into memory when our program starts and will exist for the life of the program. This qualifier is optional. If a method is not static, we simply leave static out.

A non-static method must be accessed from an instantiated object. This means that instead of placing the method into memory at the start, we place the code into memory only after creating an object for the method's class. This saves memory! We only have to put our method into memory when we need it!

Return Type

The return type defines the type of data being returned by the method. This can be a data primitive (int, char), a Class object (String, Circle), or even `void` if the method does not return data.

Parameters

The parameters list the data that needs to be passed to the method when it's called. I.e. the data the method needs in order to run. The parameters are, in fact, variables that have been declared for the scope of the method. These variables are then initialized with the data being passed into the method.

Remember, parameters are optional. If the method doesn't require inputted data, then it can leave the parameters as blank ( )

Method Signature Examples

```
public static void addNumbers(int a, int b)
```

This method signature states that *addNumbers* :

- can be called from any class (public)
- can be called at any time since it will be placed into memory when the program starts (static)
- will return nothing (void)
- requires two numbers in order to run (int a, int b)

```
private int findSum(int a, int b)
```

This method signature states that *findSum*:

- can only be used by the class that created it (private)
- can only be called from an initialized class object (non-static)
- will return a number (int)
- requires two numbers in order to run (int a, int b)

```
private String getName()
```

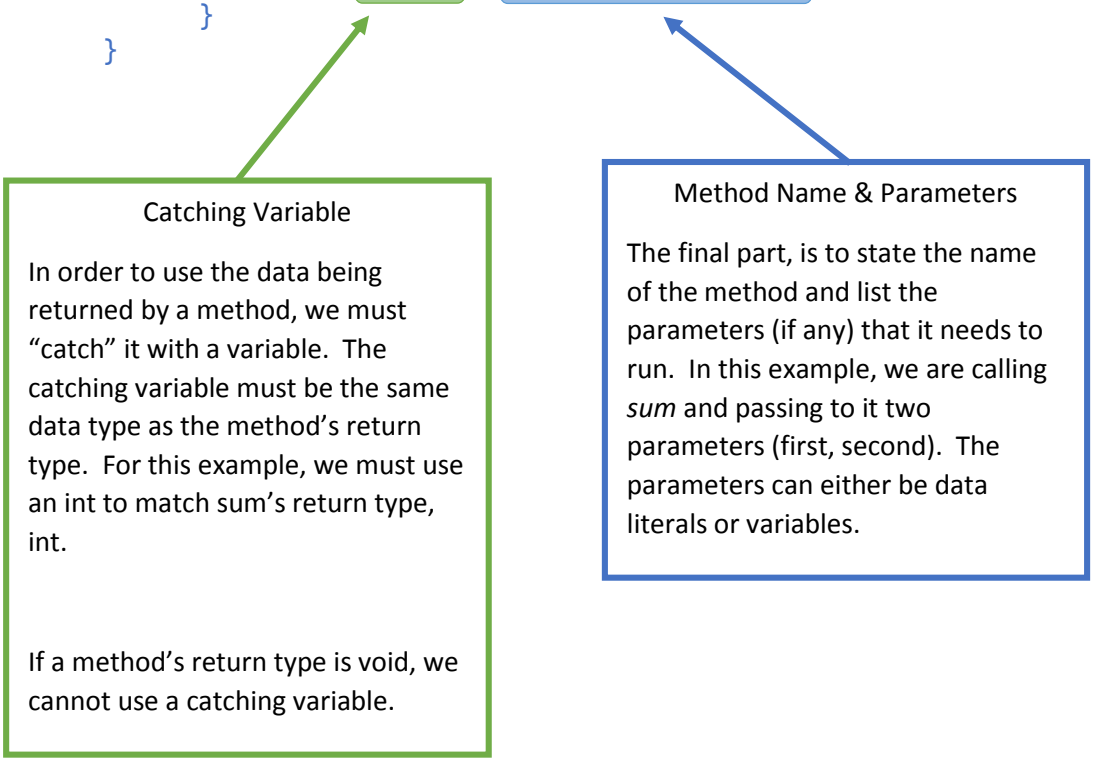
This method signature states that *getName*:

- can only be used by the class that created it (private)
- can only be called from an initialized class object (non-static)
- will return text data (String)
- requires no data in order to run ( )

Method Calls – From the Same Class

```
public class Math
{
	//Add two numbers and return the sum
	public int sum(int first, int second)
	{
		int total = first + second;
		return total;
	}


	public static void main(String[] args)
	{
		int first = 4;
		int second = 7;
		int number = sum(first, second);
	}
}
```



Catching Variable

In order to use the data being returned by a method, we must "catch" it with a variable. The catching variable must be the same data type as the method's return type. For this example, we must use an int to match sum's return type, int.

If a method's return type is void, we cannot use a catching variable.

Method Name & Parameters

The final part, is to state the name of the method and list the parameters (if any) that it needs to run. In this example, we are calling *sum* and passing to it two parameters (first, second). The parameters can either be data literals or variables.

Method Calls – From Another Class

```
public class DoMath
{
	public static int addNumbers(int first, int second)
	{
		int sum = first + second;
		return sum;
	}
}
```

To call a method in java we must use the following syntax:

```
int catchingVariable = DoMath.addNumbers(5, 13);
```

Class Name

The Java Compiler needs to know where to find the method being called. We need to list the name of the Class that "owns" the method. After the Class name, use "." or dot notation to access the Class's method. In this example, we are telling the compiler to look at the Math class to find the addNumbers method.

Remember

Place your Java files in the same location/directory so that the compiler knows where to find the other class(es) (in this case the DoMath class).