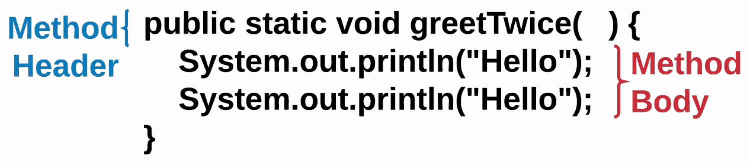
Learning Objectives: Method Basics

- **Demonstrate how to define a method**
- **Differentiate between the method header and the method body**
- **Identify the importance of the order of method definitions**
- **Identify the purpose of Java code documentation (Javadoc)**

Method Definition

Method Syntax

You have seen and used built-in methods like the length method (`myString.length()`). This unit deals with user-defined methods. Methods are composed of two parts, the header and the body.

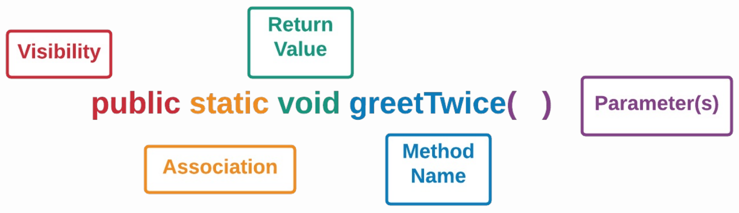


```
public static void greetTwice(   ) {
   System.out.println("Hello");
   System.out.println("Hello");
}
```

.guides/img/MethodFull

The method header contains several keywords to determine the method type. Next is the name of the method. Names of “[m]ethods should be verbs, in mixed case with the first letter lowercase, with the first letter of each internal word capitalized” (https://www.oracle.com/java/technologies/javase/codeconventions-namingconventions.html). Some examples include: `greetTwice`, `addThree`, `findArea`, etc. Parentheses are required but the parameter(s) within them are not. Any command(s) associated with the method should be indented and enclosed within curly braces `{}`. This command(s) comprises the body of the method.

Method Header



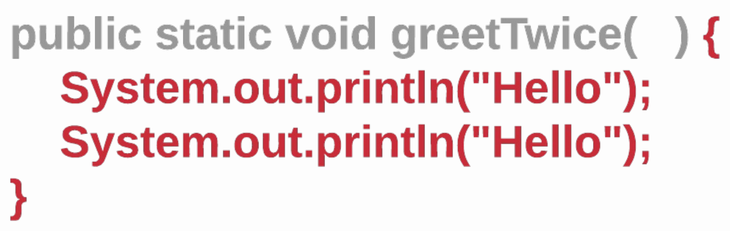
```
public static void greetTwice(   )
```

.guides/img/MethodHeader

The method header usually starts with a few keywords:
* `public` - Determines from where the method can be accessed. Other options include **private** and **protected**. For this module, we will be working with mostly **public** methods.
* `static` - Determines what can access the method. **static** methods can be accessed without needing a particular instance or object. If you want the

method to be exclusive to an object, do not include **static**. For this module, we will be working with mostly **static** methods.
* `void` - Determines whether there is a return value or not for the method. If the method has no return value, use **void**. If the method returns an integer, use **int**, etc.
* `greetTwice` - This is an example of a method name. See above for naming conventions.
* `()` - Parentheses are required for all methods. Any parameters that the method takes in will go into the parentheses but they are optional.

Method Body



```
public static void greetTwice(   ) {
    System.out.println("Hello");
    System.out.println("Hello");
}
```

.guides/img/MethodBody

The method body is the list of actions the method performs. All of the code for the method body should be enclosed within curly braces `{}` and indented to show association. This convention is similar to how conditionals and loops are written.

Method Calls

Calling a Method

Copy the entire code below into the text editor to your left. Then click the `TRY IT` button to see what happens.

```
public class MethodCall {

  public static void greetTwice() {
    System.out.println("Hello");
    System.out.println("Hello");
  }

}
```

You'll notice that an error is produced. This happens because when running Java programs, a `main` class is required. Let's go ahead and add `public static void main(String args[])` to the code like so.

```
public class MethodCall {

  public static void greetTwice() {
    System.out.println("Hello");
    System.out.println("Hello");
  }

  public static void main(String args[]) {

  }

}
```

Nothing is outputted when the program is executed. This happens because creating a method alone does not cause Java to run it. You have to explicitly call the method if you want it to run. Methods are usually called within the `main` class. To call a method, simply start with its name, provide any parameters if needed, and end with a semicolon.

```java
public class MethodCall {

  public static void greetTwice() {
    System.out.println("Hello");
    System.out.println("Hello");
  }

  public static void main(String args[]) {
    greetTwice(); //method call to greetTwice
  }

}
```

challenge

What happens if you:

- Call `greetTwice` again by adding another `greetTwice();` to the `main()` method?
- Modify `greetTwice` by adding the line `System.out.println("Goodbye");` underneath the second `System.out.println("Hello");`?

Order of Method Definitions

The order of method definitions is important in Java. If the code is changed to the following, what do you think will be outputted?

```java
public class MethodCall {

  public static void greetTwice() {
    System.out.println("Goodbye");
    System.out.println("Hello");
    System.out.println("Hello");
  }

  public static void main(String args[]) {
    greetTwice();
    greetTwice();
  }

}
```

Like how a regular Java program runs, the method is executed line by line from top to bottom. Thus, the order of statements within the method will determine what actions are performed first, second, etc.

Order of Method and Main

On the other hand, the order of the classes themselves, `main()` versus `greetTwice()`, does not matter. What happens if you swap the position of `main()` with `greetTwice()` and vice versa?

```
public class MethodCall {

  public static void main(String args[]) {
    greetTwice();
    greetTwice();
  }

  public static void greetTwice() {
    System.out.println("Goodbye");
    System.out.println("Hello");
    System.out.println("Hello");
  }

}
```

Javadoc

Java Code Documentation

Including Java code documentation prior to method definitions is standard. Doing so enables users to gain clarity on the purpose of the method, any parameters that are used, and what the method returns. Users can also see who wrote the method as well as what version the method is on. Here is an example, also present within the text editor to your left:

```
/**
  * This is an example Javadoc
  *
  * @author  FirstName LastName
  * @version 1.0
  */
public class Javadoc {

/**
  * This method greets the user twice
  *
  * @param   specify parameters if any
  * @return  specify return value if any
  */
  public static void greetTwice() {
    System.out.println("Hello");
    System.out.println("Hello");
  }

  public static void main(String args[]) {
    greetTwice();
  }

}
```

The Java code documentation does not affect the output of the code. However, it provides more clarity to how the method is used.

Javadoc Tool

There is a Javadoc tool that is available which can take in a Java file and generate an HTML page with all of the documentation. For more information, you may visit the Javadoc website at this link: Javadoc. However, for the purposes of this module, we will only focus on self-created documentation. In particular, we will be using mostly `@param` for parameters and `@return` for return values.