Java Class Methods

‹ Previous Next ›

Java Class Methods

You learned from the Java Methods chapter that methods are declared within a class, and that they are used to perform certain actions:

Example

Create a method named `myMethod()` in Main:

```
public class Main {

    System.out.println("Hello World!");
  }
}
```

`myMethod()` prints a text (the action), when it is **called**. To call a method, write the method's name followed by two parentheses **()** and a semicolon**;**

Example

Inside `main`, call `myMethod()`:

```
  static void myMethod() {
    System.out.println("Hello World!");
  }

  public static void main(String[] args) {

  }
}

// Outputs "Hello World!"
```

Try it Yourself »

Static vs. Non-Static

You will often see Java programs that have either `static` or `public` attributes and methods.

In the example above, we created a `static` method, which means that it can be accessed without creating an object of the class, unlike `public`, which can only be accessed by objects:

Example

An example to demonstrate the differences between `static` and `public` **methods**:

```
public class Main {
  // Static method

    System.out.println("Static methods can be called without creating objects");
  }

  // Public method

    System.out.println("Public methods must be called by creating objects");
  }
```

```
  public static void main(String[] args) {
    myStaticMethod(); // Call the static method
    // myPublicMethod(); This would compile an error

    Main myObj = new Main(); // Create an object of Main
    myObj.myPublicMethod(); // Call the public method on the object
  }
}
```

Try it Yourself »

Note: You will learn more about these keywords (called modifiers) in the Java Modifiers chapter.

Access Methods With an Object

Example

Create a Car object named `myCar`. Call the `fullThrottle()` and `speed()` methods on the `myCar` object, and run the program:

```
// Create a Main class
public class Main {

  // Create a fullThrottle() method
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  // Create a speed() method and add a parameter
  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
```

```
  // Inside main, call the methods on the myCar object
  public static void main(String[] args) {
    Main myCar = new Main();   // Create a myCar object
    myCar.fullThrottle();      // Call the fullThrottle() method
    myCar.speed(200);          // Call the speed() method
  }
}

// The car is going as fast as it can!
// Max speed is: 200
```

Try it Yourself »

Example explained

1) We created a custom `Main` class with the `class` keyword.

2) We created the `fullThrottle()` and `speed()` methods in the `Main` class.

3) The `fullThrottle()` method and the `speed()` method will print out some text, when they are called.

4) The `speed()` method accepts an `int` parameter called `maxSpeed` - we will use this in **8)**.

5) In order to use the `Main` class and its methods, we need to create an **object** of the `Main` Class.

6) Then, go to the `main()` method, which you know by now is a built-in Java method that runs your program (any code inside main is executed).

7) By using the `new` keyword we created an object with the name `myCar`.

8) Then, we call the `fullThrottle()` and `speed()` methods on the `myCar` object, and run the program using the name of the object (`myCar`), followed by a dot (`.`), followed by the name of the method (`fullThrottle();` and `speed(200);`). Notice that we add an `int` parameter of **200** inside the `speed()` method.

The dot (`.`) is used to access the object's attributes and methods.

To call a method in Java, write the method name followed by a set of parentheses **()**, followed by a semicolon (`;`).

A class must have a matching filename (`Main` and **Main.java**).

Using Multiple Classes

Like we specified in the Classes chapter, it is a good practice to create an object of a class and access it in another class.

Remember that the name of the java file should match the class name. In this example, we have created two files in the same directory:

- Main.java
- Second.java

Main.java

```
public class Main {
  public void fullThrottle() {
    System.out.println("The car is going as fast as it can!");
  }

  public void speed(int maxSpeed) {
    System.out.println("Max speed is: " + maxSpeed);
  }
}
```

Second.java

```
class Second {
  public static void main(String[] args) {
```

```
    myCar.speed(200);          // Call the speed() method
  }
}
```

When both files have been compiled:

```
C:\Users\Your Name>javac Main.java
C:\Users\Your Name>javac Second.java
```

Run the Second.java file:

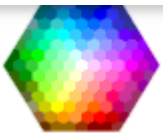
```
C:\Users\Your Name>java Second
```

And the output will be:

```
The car is going as fast as it can!
Max speed is: 200
```

Try it Yourself »

‹ Previous Next ›



LIKE US

CODE GAME

HTML
CSS
JavaScript
Front End
Python
SQL
And more