**Java Class Methods**

You learned from the [Java Methods](https://www.w3schools.com/java/java_methods.asp) chapter that methods are declared within a class, and that they are used to perform certain actions:

**Example**

Create a method named myMethod() in MyClass:

public class MyClass {  
  static void **myMethod()** {  
    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 parantheses **()** and a semicolon**;**

**Example**

Inside main, call the myMethod() method:

public class MyClass {  
  static void myMethod() {  
    System.out.println("Hello World!");  
  }  
  
  public static void main(String[] args) {  
    **myMethod();**  
  }  
}  
  
// Outputs "Hello World!"

**Static or Public**

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 creating objects:

**Example**

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

public class MyClass {  
  // Static method  
  static void myStaticMethod() {  
    System.out.println("Static methods can be called without creating objects");  
  }  
  
  // Public method  
  public void myPublicMethod() {  
    System.out.println("Public methods must be called by creating objects");  
  }  
  
  // Main method  
  public static void main(String[] args) {  
    myStaticMethod(); // Call the static method  
    // myPublicMethod(); This would compile an error  
  
    MyClass myObj = new MyClass(); // Create an object of MyClass  
    myObj.myPublicMethod(); // Call the public method on the object  
  }  
}

**Note:** You will learn more about these keywords (called modifiers) in the [Java Modifiers](https://www.w3schools.com/java/java_modifiers.asp) 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 Car class  
public class Car {  
   
  // 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) {  
    Car myCar = new Car();     // 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

**Example explained**

1) We created a custom Car class with the class keyword.

2) We created the fullThrottle() and speed() methods in the Car 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 Car class and its methods, we need to create an **object** of the Car 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 a Car 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.

**Remember that..**

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 parantheses **()**, followed by a semicolon (;).

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

**Using Multiple Classes**

Like we specified in the [Classes chapter](https://www.w3schools.com/java/java_classes.asp), 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:

* Car.java
* OtherClass.java

**Car.java**

public class Car {  
  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);  
  }  
}

**OtherClass.java**

class OtherClass {  
  public static void main(String[] args) {  
    Car myCar = new Car();     // Create a myCar object  
    myCar.fullThrottle();      // Call the fullThrottle() method  
    myCar.speed(200);          // Call the speed() method  
  }  
}