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 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!"
```

Run example »

Method Parameters

Methods can also have parameters, which is used to pass information:

Example

Create a method that accepts an `int` parameter called x. In the `main()` method, we call `myMethod()` and set an `int` parameter of 10:

```
public class MyClass {
  static void myMethod(int x) {
    System.out.println(x);
  }

  public static void main(String[] args) {
    myMethod(10);
  }
}

// Outputs 10
```

Run example »

Static or Public

In the examples above we used the `static` keyword for the method instead of `public`. A static method means that it belongs to the class, and cannot be accessed from outside of the

class, unlike `public`. To access a method with an object, you must must declare the method as `public`:

Example

```
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");
  }

}
```

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

Access Methods With Objects

Now that you know how to create classes and methods, we can start creating objects that we will use to access class methods.

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
```

Run example »

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, 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
  }
}
```

When both files have been compiled:

```
C:\Users\Your Name>javac Car.java
C:\Users\Your Name>javac OtherClass.java
```

Run the OtherClass.java file:

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

And the output will be:

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

Run example »

‹ Previous

Next ›