**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 parentheses () and a semicolon**;

**Example**

Inside main, call **myMethod ();**

public class MyClass {

static void myMethod() {

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

}

public static void main(String[] args) {

myMethod();

}

}

// Outputs "Hello World!"

**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 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 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 parentheses (), 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

**Java Constructors**

A **constructor** in Java is a special method that is **used to initialize objects**. The **constructor** is *called when an object of a class is created*. It can be *used to set initial values for object attributes:*

**Example**

Create a constructor:

**// Create a MyClass class**

public class **MyClass** {

int x; **// Create a class attribute**

**// Create a class constructor for the MyClass class**

**public** MyClass() {

x = 5; **// Set the initial value for the class attribute x**

}

public static void main(String[] args) {

MyClass *myObj* = new MyClass(); **// Create an** *object* **of class MyClass (This will call the constructor)**

System.out.println(myObj.x); **// Print the value of x**

}

}

// Outputs 5

Note that the **constructor name** *must match* the **class name**, and it *cannot have a return type (like void).*

Also note that the **constructor** is called when the *object* is created.

**All classes have constructors by default:** if you do not create a class constructor yourself, Java creates one for you. However, then you are not able to set initial values for object attributes.

**Constructor Parameters**

Constructors can also take parameters, which is used to initialize attributes.

The following example adds an int y parameter to the constructor. Inside the constructor we set x to y (x=y). When we call the constructor, we pass a parameter to the constructor (5), which will set the value of x to 5:

**Example**

public class MyClass {

int x;

public MyClass(int y) {

x = y;

}

public static void main(String[] args) {

MyClass myObj = new MyClass(5);

System.out.println(myObj.x);

}

}

// Outputs 5

You can have as many **parameters** as you want:

**Example**

public class Car {

int modelYear;

String modelName;

public Car**(int year, String name**) {

modelYear = year;

modelName = name;

}

public static void main() {

Car myCar = new Car(1969, "Mustang");

System.out.println(myCar.modelYear + " " + myCar.modelName);

}

}

// Outputs 1969 Mustang