Java - What is OOP?

OOP stands for **Object-Oriented Programming**.

Procedural programming is about writing **procedures** or **methods** that perform operations on the data, while object-oriented programming is about creating **objects** that contain both **data** and **methods**.

Object-oriented programming has several advantages over procedural programming:

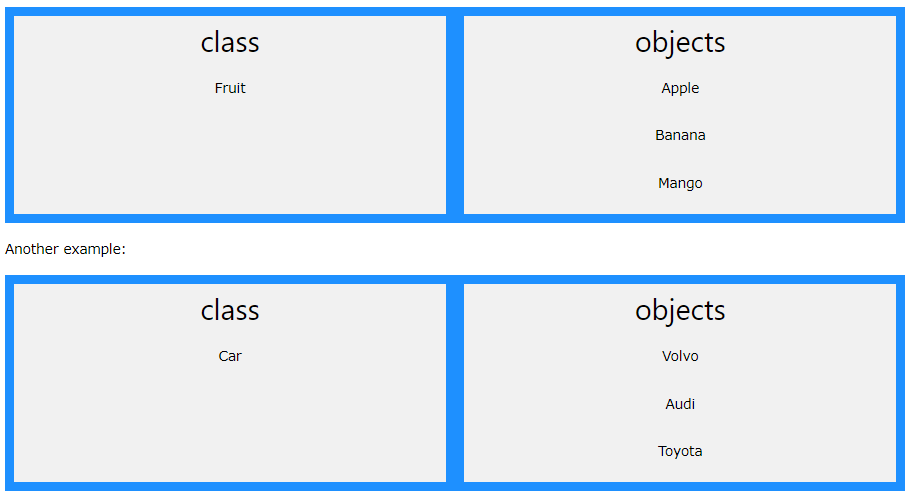
* OOP is faster and easier to execute
* OOP provides a clear structure for the programs
* OOP helps to keep the Java code DRY **"Don't Repeat Yourself"**, and makes the code easier to maintain, modify and debug
* OOP makes it possible to create full reusable applications with less code and shorter development time

**Tip:** The "Don't Repeat Yourself" (DRY) principle is about reducing the repetition of code. You should extract out the codes that are common for the application, and place them at a single place and reuse them instead of repeating it.

## Java - What are Classes and Objects?

Classes and objects are the two main aspects of object-oriented programming.

Look at the following illustration to see the difference between class and objects:



So, a class is a template for objects, and an object is an instance of a class.

When the individual objects are created, they inherit all the variables and methods from the class.

You will learn much more about [classes and objects](https://www.w3schools.com/java/java_classes.asp) in the next chapter.

## Java Classes/Objects

Java is an object-oriented programming language.

Everything in Java is associated with classes and objects, along with its attributes and methods. For example: in real life, a car is an object. The car has **attributes**, such as weight and color, and **methods**, such as drive and brake.

A Class is like an object constructor, or a "blueprint" for creating objects.

## Create a Class

To create a class, use the keyword class:

### MyClass.java

Create a class named "MyClass" with a variable x:

public class MyClass{

int x = 5; //Attribute

}

## Create an Object

In Java, an object is created from a class. We have already created the class named MyClass, so now we can use this to create objects.

To create an object of MyClass, specify the class name, followed by the object name, and use the keyword new:

### Example

Create an object called "myObj" and print the value of x:

Public class MyClass{

int x = 5;

public static void main(String[]args){

MyClass **myObj**=new MyClass();

System.out.println(myObj.x);

}

}

## Multiple Objects

You can create multiple objects of one class:

### Example

Create two objects of MyClass:

Public class MyClass{

int x =5;

public static void main(String[]args){

MyClass **myObj1**=new MyClass();// Object 1

MyClass **myObj2**=new MyClass();// Object 2

System.out.println(myObj1.x);

System.out.println(myObj2.x);

}

}

## Using Multiple Classes

You can also create an object of a class and access it in another class. This is often used for better organization of classes (one class has all the attributes and methods, while the other class holds the main() method (code to be executed)).

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/folder:

* MyClass.java
* OtherClass.java

#### MyClass.java

Public class MyClass{

int x =5;

}

#### OtherClass.java

Class OtherClass{

Public static void main(String[]args){

MyClass **myObj**=new MyClass();

System.out.println(myObj.x);

}

}

When both files have been compiled:

C:\Users\Your Name>javac MyClass.java  
C:\Users\Your Name>javac OtherClass.java

Run the OtherClass.java file:

C:\Users\Your Name>java OtherClass

And the output will be:

5

## Java Class Attributes

In the previous chapter, we used the term "variable" for x in the example (as shown below). It is actually an **attribute** of the class. Or you could say that class attributes are variables within a class:

### Example

Create a class called "MyClass" with two attributes: x and y:

Public class MyClass{

int x =5;

int y =3;

}

## Accessing Attributes

You can access attributes by creating an object of the class, and by using the dot syntax (.):

The following example will create an object of the MyClass class, with the name myObj. We use the x attribute on the object to print its value:

### Example

Create an object called "myObj" and print the value of x:

Public class MyClass{

int x =5;

public static void main(String[]args){

MyClass myObj=new MyClass();

System.out.println(myObj.x);

}

}

## Modify Attributes

You can also modify attribute values:

### Example

Set the value of x to 40:

Public class MyClass{

int x;

public static void main(String[]args){

MyClass myObj=new MyClass();

myObj.x=40;

System.out.println(myObj.x);

}

}

Or override existing values:

### Example

Change the value of x to 25:

Public class MyClass{

int x =10;

public static void main(String[]args){

MyClass myObj=new MyClass();

myObj.x=25;// x is now 25

System.out.println(myObj.x);

}

}

If you don't want the ability to override existing values, declare the attribute as **final**:

### Example

Public class MyClass{

**Final** int x =10;

Public static void main(String[]args){

MyClass myObj=new MyClass();

myObj.x=25;// will generate an error: cannot assign a value to a **final** variable

System.out.println(myObj.x);

}

}

## Multiple Objects

If you create multiple objects of one class, you can change the attribute values in one object, without affecting the attribute values in the other:

### Example

Change the value of x to 25 in myObj2, and leave x in myObj1 unchanged:

Public class MyClass{

int x =5;

public static void main(String[]args){

MyClass myObj1 =new MyClass();// Object 1

MyClass myObj2 =new MyClass();// Object 2

myObj2.x =25;

System.out.println(myObj1.x);// Outputs 5

System.out.println(myObj2.x);// Outputs 25

}

}

## Multiple Attributes

You can specify as many attributes as you want:

### Example

Public class Person{

String fname="John";

String lname="Doe";

int age =24;

public static void main(String[]args){

Person myObj=new Person();

System.out.println("Name: "+myObj.fname+" "+myObj.lname);

System.out.println("Age: "+myObj.age);

}

}

## 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 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**:

publicclassMyClass{

// Static method

staticvoidmyStaticMethod(){

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

}

}

## 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.

## 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

publicclassCar{

publicvoidfullThrottle(){

System.out.println("The car is going as fast as it can!");

}

publicvoidspeed(intmaxSpeed){

System.out.println("Max speed is: "+maxSpeed);

}

}

#### OtherClass.java

classOtherClass{

publicstaticvoidmain(String[]args){

CarmyCar=newCar();// 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