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:

class

Fruit

objects

Apple

Banana

Mango

Another example:

class

Car

objects

Volvo

Audi

Toyota

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 and Objects**

## 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;

}

Remember from the [Java Syntax chapter](https://www.w3schools.com/java/java_syntax.asp) that a class should always start with an uppercase first letter, and that the name of the java file should match the class name.

## 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);

}

}

# **Java Class Attributes**

## 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;

}

Another term for class attributes is **fields**.

## 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);

}

}

The final keyword is useful when you want a variable to always store the same value, like PI (3.14159...).

The final keyword is called a "modifier". You will learn more about these in the [Java Modifiers Chapter](https://www.w3schools.com/java/java_modifiers.asp).

## 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);

}

}

The next chapter will teach you how to create class methods and how to access them with objects.

## ****Inheritance****

Classes can share, obtain or “inherit” properties and methods that belong to existing classes. This lets you reuse existing code and reduces the time you spend coding. A class that inherits from another is called a derived class or a child class. A class that shares its properties and methods is called a base class or parent class. A child class can have additional properties and methods not present in the parent class that distinguishes it and provides additional functionality.

If you want a real world example, consider the Windows operating system. Windows 98 had certain properties and methods that were used in Windows XP. Windows XP was derived from Windows 98, but it was still different from it. Windows 7 was derived from Windows XP, but they were both different. They both followed a certain basic template and shared properties, however. This is true for base classes and derived classes too.

In Java, you can inherit a class using the extends keyword:

public class Flower {

String name;

String color;

void pollination() {

}

}

public class Rose extends Flower {

}

The inherited class Rose can now share the attributes and methods of the class Flower. For more info on declaring objects or classes in Java , you can [see this course on Java fundamental](https://www.udemy.com/java-fundamentals-i-and-ii/?tc=blog.oopsconceptswithexamples&couponCode=half-off-for-blog" \t "_blank)s.

## ****Polymorphism****

Polymorphism refers to the ability of a method to be used in different ways, that is, it can take different forms at different times (poly + morphos). There are two types of polymorphism: compile time polymorphism and run time polymorphism.

Compile time (static) polymorphism occurs when a method is overloaded; that is, when the argument used with the method is changed. This is done to obtain different results. An example of this would be suggesting different names for being the President of a country, which would get you different results each time – but they would still be called the President.

Run time (dynamic) polymorphism occurs when the methods itself are changed. When you don’t need a President but a Prime Minister, you would have to override the existing method.

We’ll explain the concept with a simple example. Suppose you wanted to multiply two numbers together:

public int mulNum (int x, int y) {

//

}

But what if you had other numbers to multiple, but they had “double” values instead of “int” values? You would just change the argument:

public int mulNum (double x, double y) {

//

}

All you’ve done here is change the signature of the method – the name of the method remains the same. JVM (and other OOP compilers) can differentiate between methods and provide the right result depending on the input you give to them

## ****Data Abstraction****

Data abstraction refers to the process of only displaying relevant properties and methods to handle an object, while hiding the rest. Data abstraction lets you reduce the complexity (apparently) of a program and is a large advantage offered by classes in OOP languages.

An example would help you understand this better. You can operate your phone as long as you have a keypad and a screen. You don’t to know about its micro circuitry, antenna, software or other technology to operate it. Its complexity is hidden.

You can declare a class as abstract in Java to use the data abstraction functionality. Once a class has been declared as abstract, however, it cannot be instantiated. The class doesn’t become useless- if it has any methods and fields, they can still be accessed. However, it’s impossible to declare an instance of the abstract class. They can be sub-classed.

You can create an abstract class using the abstract keyword. An abstract class may or may or may not contain abstract methods. A class containing abstract methods must be declared abstract – a normal class cannot contain abstract methods.

Here’s an example of how you can declare an abstract class in Java:

public abstract class ExampleAbs {

// declare your fields

// declare normal methods

}

Here’s an example of an abstract method declared within an abstract class in Java:

public abstract class ExampleAbs {

// declare your fields

abstract void delay();

}

Public class test extxends ExampleAbs {

Delay() {

System.out.println(“Tavi”);

The method delay() is the abstract method. Notice that its declaration is different from other method declarations – you don’t have to use curly braces after it. Also, you have to use a semicolon “;” after the declaration. An abstract method can exist side by side with non abstract methods.

## ****Encapsulation****

Encapsulation refers to keeping objects with their methods in one place. It also protects the integrity of the data – prevents it from being needlessly altered by restricting access to the data, preferably by hiding it from outside elements. Encapsulation is often confused with data abstraction, but they are different concepts entirely. Data hiding, or data abstraction, has more to do with access specifiers. A programmer must first encapsulate the data and then he can take steps to hide it.

A normal class in Java follows the encapsulation principle by default. Let’s create a basic class that deals with an employee and his tasks at the office:

public class Employeedetails {

private String employeeName;

private String employeeDept;

private int salary;

private int hoursperday;

public void work () {

}

public void train () {

}

}

As you can see, we are keeping the data that deals with the Employee together with the methods (work and train). This is encapsulation. We specified the name of the employee, his salary and hours he worked per day as private, which means only the members of the program can access this data. This helps protect the integrity of the data.