东方瑞通
终身学习
Founded in 1998

Java Fundamentals Part 1

IT管理方向

办公应用与协作

软件开发

数据库

运维方向

通用技能

云计算大数据

ERP工程师

云大物移智

关注微信公众号
享终身免费学习

1. OOP

1.1. 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:

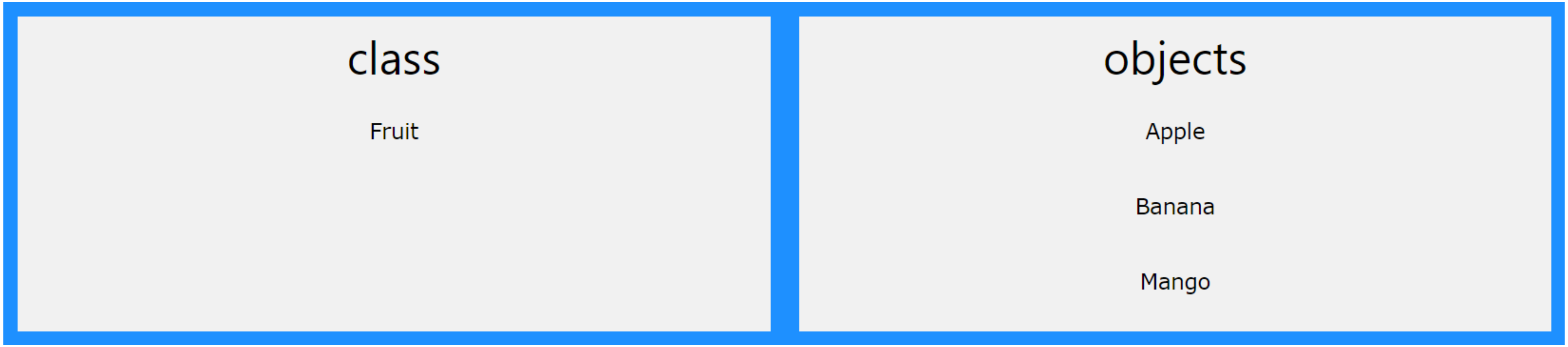
- OOP is faster and easier to execute
- OOP provides a clear structure for the programs
- OOP helps to keep the Java code DRY “Don’t Repeat Yourself”, and makes the code easier to maintain, modify and debug
- OOP makes it possible to create full reusable applications with less code and shorter development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

1.2. 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

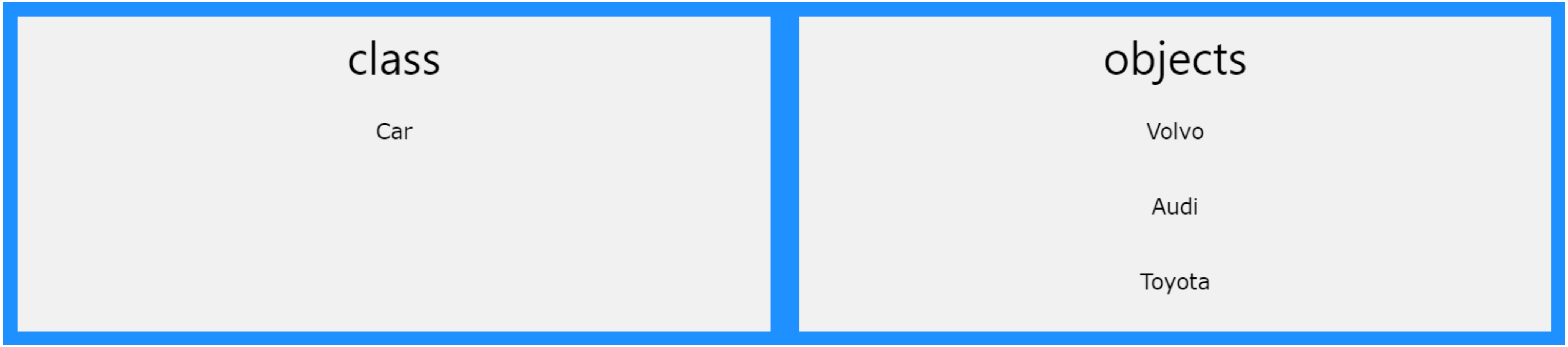
objects

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.

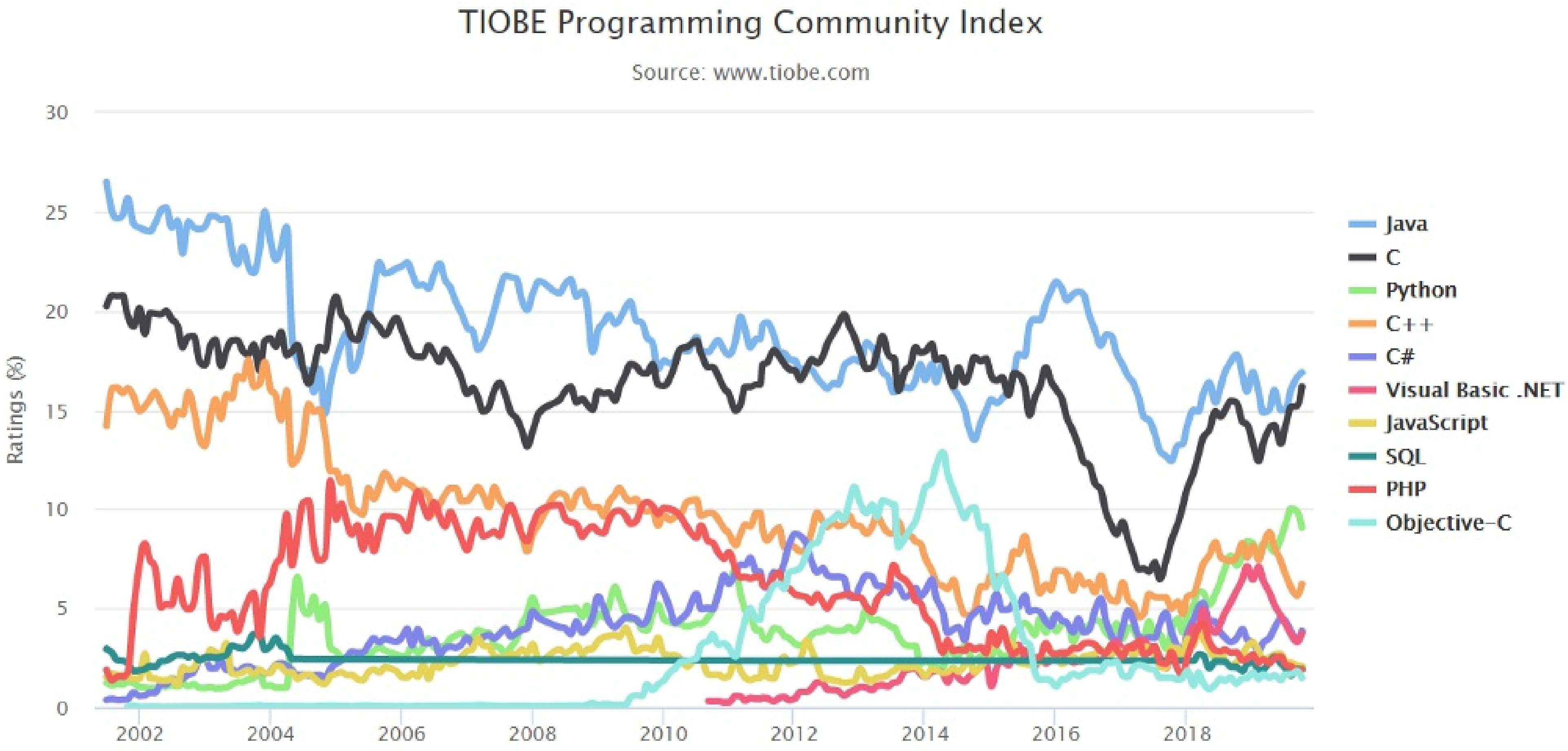
OOP

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

2. Java Introduction

Java is a powerful, general-purpose programming environment and one of the most widely used programming languages in the world. From it's inception, designers of Java focused on creating a language paradigm that is **simple**, **object-oriented**, and **platform independent**. These are just some aspects of the language which often make it a preferred choice for developing cutting-edge, enterprise-level applications.

Java is considered a higher-level programming language that enables application developers to **"write once, run anywhere"** meaning that code written and compiled in a certain environment can run on all platforms regardless of computer architecture. Over the past twenty years, Java has remained a go-to language for use in projects of all sizes… now running on over three billion devices!


TIOBE Programming Community Index
Source: www.tiobe.com
Ratings (%)
30
25
20
15
10
5
0
2002
2004
2006
2008
2010
2012
2014
2016
2018
Java
C
Python
C++
C#
Visual Basic .NET
JavaScript
SQL
PHP
Objective-C

2.1. What is Java?

Java is a popular programming language, created in 1995.

It is owned by Oracle, and more than **3 billion** devices run Java.

It is used for:

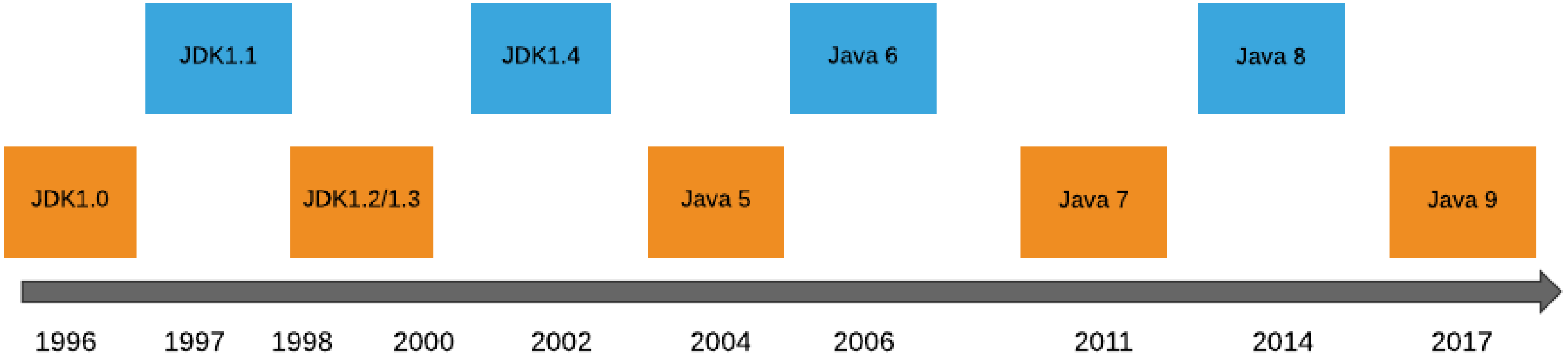
- Mobile applications (specially Android apps)
- Desktop applications
- Web applications
- Web servers and application servers
- Games
- Database connection
- And much, much more!

2.2. Why Use Java?

- Java works on different platforms (Windows, Mac, Linux, Raspberry Pi, etc.)
- It is one of the most popular programming language in the world
- It is easy to learn and simple to use
- It is open-source and free
- It is secure, fast and powerful
- It has a huge community support (tens of millions of developers)
- Java is an object oriented language which gives a clear structure to programs and allows code to be reused, lowering development costs
- As Java is close to C++ and C#, it makes it easy for programmers to switch to Java or vice versa

2.3. Language Timeline

The Java programming language was originally developed by Sun Microsystems, first released in 1995 as a core component of the company's Java platform (Java 1.0 [J2SE]). James Gosling from Sun Microsystems, best known as the founder of Java, along with his team, began designing the first version primarily aimed at programming home appliances which are controlled by a wide variety of computer processors. Since its release, it has seen tremendous advancements in capabilities and as a result (to-date) — all Android apps are based on Java and 90 percent of Fortune 500 companies use Java as a server-side language for backend development. The language supports many configurations and is suitable for a variety of platforms including J2EE for Enterprise Applications, and J2ME for Mobile Applications.



2.4. Key features

- **Object-Oriented** − In Java, everything is an Object. Java is robust and can be easily extended since it is based on the Object model.
- **Platform Independent** − Unlike many other programming languages including C and C++, when Java is compiled, it is not compiled into platform specific machine code, rather into platform independent bytecode. This bytecode can be distributed over the web and interpreted by the Virtual Machine (JVM) on whichever platform it is being run on.
- **Multi-threaded** − This key feature allows developers to write programs that can perform many tasks simultaneously. This design feature allows for the construction of interactive applications that can run concurrently while maximizing the utilization of computing resources.
- **Interpreted** − Java bytecode is translated on the fly to native machine instructions and is not stored anywhere. The development and deployment process can be more efficient and calculated since the linking is an incremental and light-weight process.
- **High Performance** − With the use of Just-In-Time compilers, Java enables high performance. The JIT compiler is the component that converts Java’s intermediate language into native machine code as needed.
- **Memory Management** — Java comes with a built-in garbage collector that automatically manages the memory used by programs.

3. Java Getting Started

3.1. Java Install

Some PCs might have Java already installed.

To check if you have Java installed on a Windows PC, search in the start bar for Java or type the following in Command Prompt (cmd.exe):

```
java -version
```

If Java is installed, you will see something like this (depending on version):

```
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)
```

If you do not have Java installed on your computer, you can download it for free at **oracle.com**.

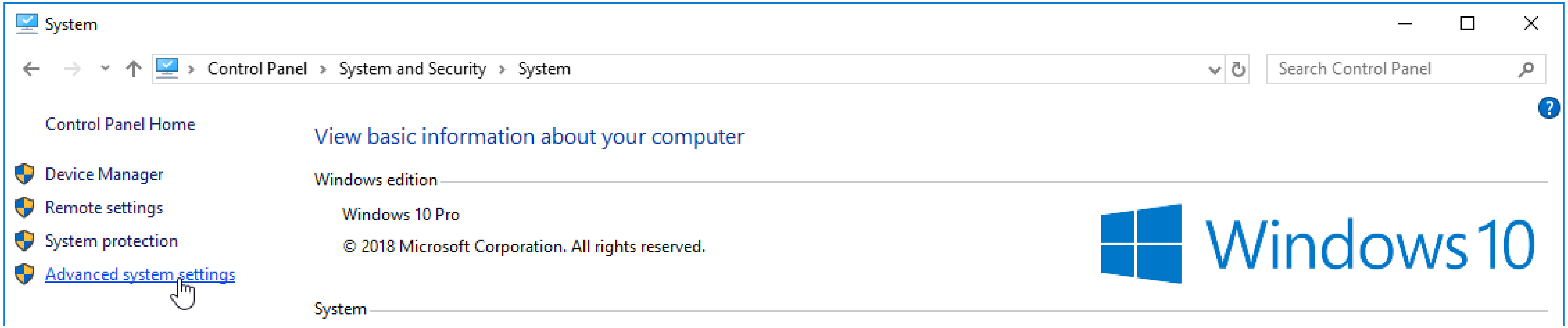
> **Note:** In this tutorial, we will write Java code in a text editor. However, it is possible to write Java in an Integrated Development Environment, such as IntelliJ IDEA, Netbeans or Eclipse, which are particularly useful when managing larger collections of Java files.

3.2. Setup for Windows

To install Java on Windows:

1. Go to “System Properties” (Can be found on Control Panel > System and Security > System > Advanced System Settings)
2. Click on the “Environment variables” button under the “Advanced” tab
3. Then, select the “Path” variable in System variables and click on the “Edit” button
4. Click on the “New” button and add the path where Java is installed, followed by ***. By default, Java is installed in C:Files.1 (If nothing else was specified when you installed it). In that case, You will have to add a new path with: **C:Files.1* Then, click “OK”, and save the settings
5. At last, open Command Prompt (cmd.exe) and type **java -version** to see if Java is running on your machine

3.2.1. Step 1



View basic information about your computer

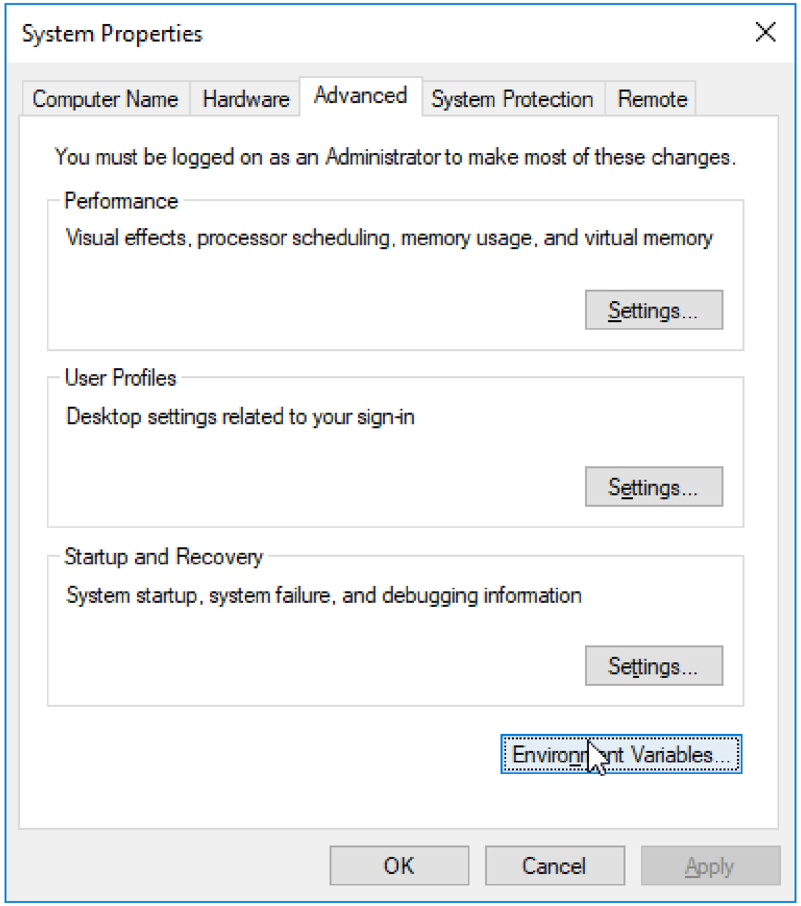
Windows edition

Windows 10 Pro

© 2018 Microsoft Corporation. All rights reserved.

System

3.2.2. Step 2



You must be logged on as an Administrator to make most of these changes.

Performance

Visual effects, processor scheduling, memory usage, and virtual memory

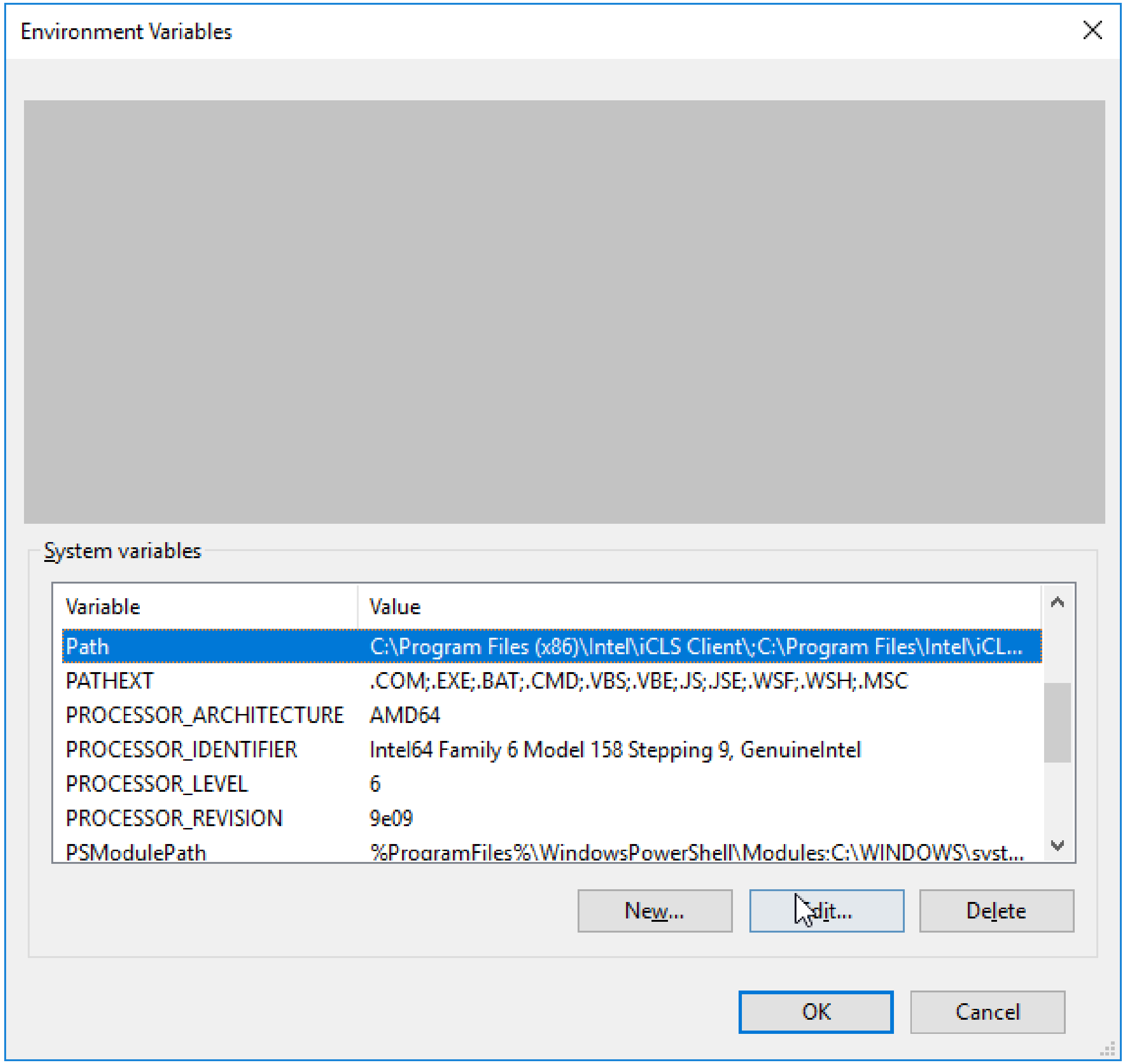
User Profiles

Desktop settings related to your sign-in

Startup and Recovery

System startup, system failure, and debugging information

3.2.3. Step 3

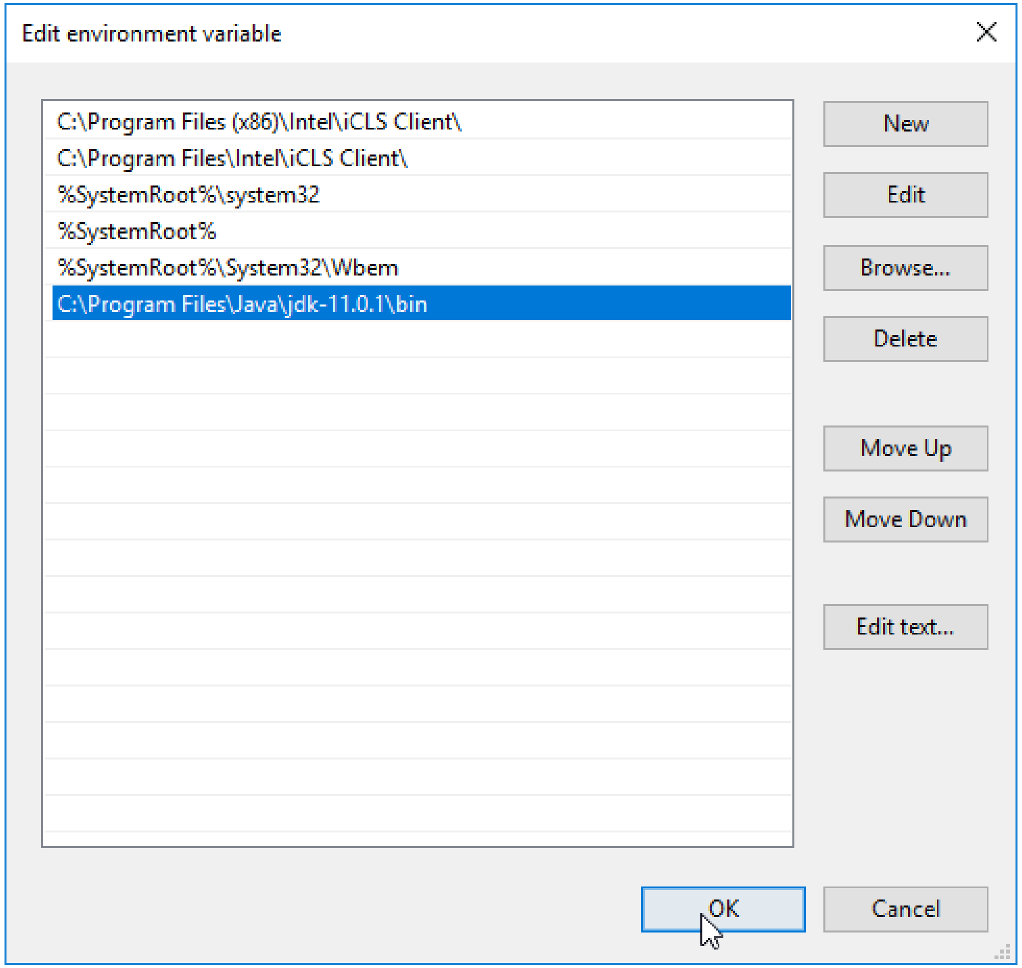


Environment Variables

System variables

Variable	Value
Path	C:\Program Files (x86)\Intel\iCLS Client\;C:\Program Files\Intel\iCL...
PATHEXT	.COM;.EXE;.BAT;.CMD;.VBS;.VBE;.JS;.JSE;.WSF;.WSH;.MSC
PROCESSOR_ARCHITECTURE	AMD64
PROCESSOR_IDENTIFIER	Intel64 Family 6 Model 158 Stepping 9, GenuineIntel
PROCESSOR_LEVEL	6
PROCESSOR_REVISION	9e09
PSModulePath	%ProgramFiles%\WindowsPowerShell\Modules;C:\WINDOWS\syst...

3.2.4. Step 4



Edit environment variable

C:\Program Files (x86)\Intel\iCLS Client\
C:\Program Files\Intel\iCLS Client\
%SystemRoot%\system32
%SystemRoot%
%SystemRoot%\System32\Wbem
C:\Program Files\Java\jdk-11.0.1\bin

3.2.5. Step 5

Write the following in the command line (cmd.exe):

```
java -version
```

If Java was successfully installed, you will see something like this (depending on version):

```
java version "1.8.0_261"
Java(TM) SE Runtime Environment (build 1.8.0_261-b12)
Java HotSpot(TM) 64-Bit Server VM (build 25.261-b12, mixed mode)
```

3.3. Java Quickstart

In Java, every application begins with a class name, and that class must match the filename.

Let’s create our first Java file, called MyClass.java, which can be done in any text editor (like Notepad).

The file should contain a “Hello World” message, which is written with the following code:

3.3.1. write code

MyClass.java

```java
public class MyClass {
 public static void main(String[] args) {
  System.out.println("Hello World");
 }
}
```

3.3.2. compile your code

Save the code in Notepad as “MyClass.java”. Open Command Prompt (cmd.exe), navigate to the directory where you saved your file, and type “javac MyClass.java”:

```
javac MyClass.java
```

3.3.3. run the file

This will compile your code. If there are no errors in the code, the command prompt will take you to the next line. Now, type “java MyClass” to run the file:

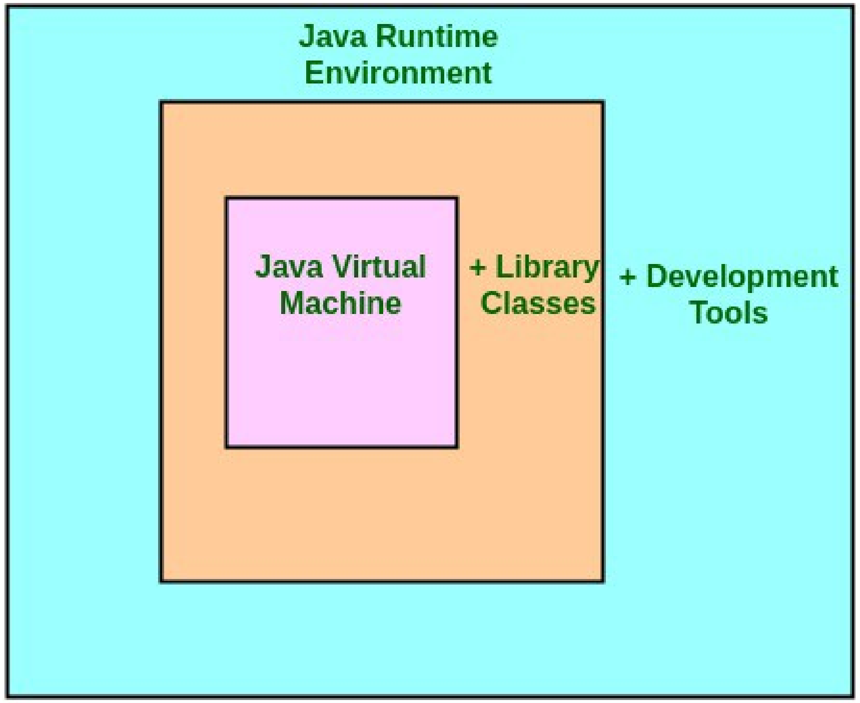
```
java MyClass
```

The output should read:

```
Hello World
```

4. Java Platform

The platform required to develop and execute Java programs consists of three key components that are important to understand — the Java Development Kit (JDK), the Java Runtime Environment (JRE), and the Java Virtual Machine (JVM). The JDK consists of the JRE (and other development tools) which includes the JVM — the graphic below depicts the relationship between the three components.



JDK = JRE + Development Tool
JRE = JVM + Library Classes

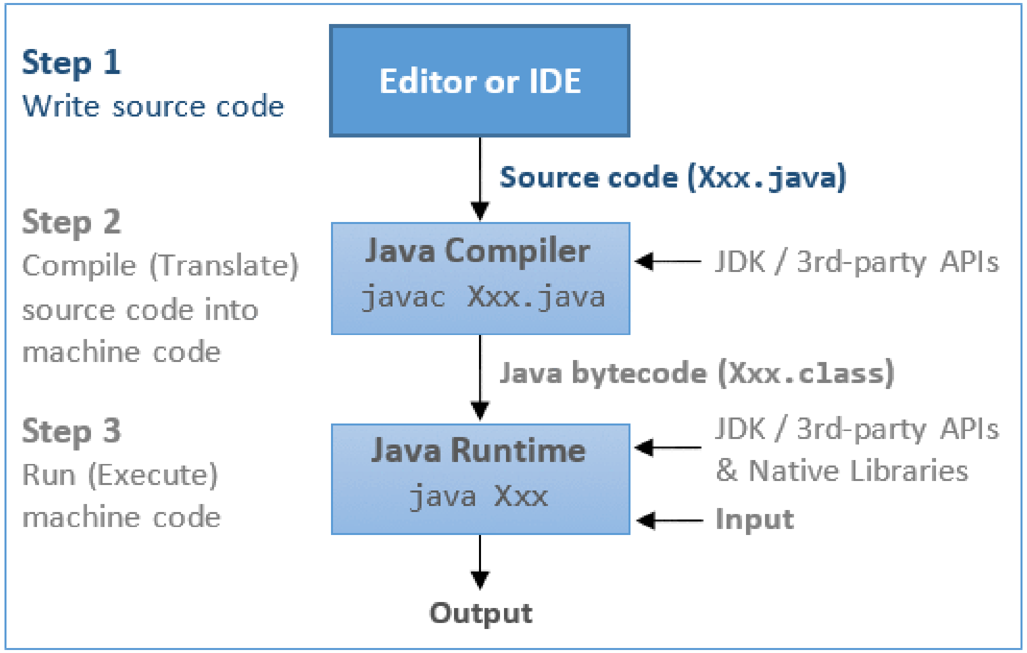
Breaking down components that make-up the platform —

Java Development Kit (JDK) — Composed of the tools required to compile, document, and package Java programs. This includes the JRE, an interpreter/class loader, a compiler, an archiver, and a documentation generator. Put simply, the JDK provides a complete toolkit for standard Java

development and program **execution**.

Java Runtime Environment (JRE) — Although it is included in the JDK, it is also available for use as a standalone component. The JRE provides the minimum requirements for **executing** a Java application but doesn't include all features required for development; it consists of the JVM and the some core libraries which enable end-users to run Java applications.

Java Virtual Machine (JVM) — The component responsible for executing the Java program line by line hence it is also known as a run-time **interpreter**. It interprets compiled Java bytecode for a computer's hardware platform so that it can perform a Java program's instructions.


Step 1
Write source code

Editor or IDE

Source code (Xxx.java)

Step 2
Compile (Translate) source code into machine code

Java Compiler
javac Xxx.java

JDK / 3rd-party APIs

Java bytecode (Xxx.class)

Step 3
Run (Execute) machine code

Java Runtime
java Xxx

JDK / 3rd-party APIs & Native Libraries

Input

Output

4.1. Understanding Java Garbage Collection

Java is object-oriented! It has a concept of object instances not raw bytes (in contrast to C), initializes object memory, has no pointer arithmetic and uses automatic memory management, called “garbage collection”. To optimize application execution, the JVM divides memory into **stack** and **heap** memory, put simply this is a built-in mechanism for both **static** and **dynamic** memory allocation. When Java programs run on the JVM, objects are created on the heap, and **it’s the garbage collectors job to free up memory when these objects become unused.**

Defining the Java Memory Models —

Stack Memory — Stores primitive values that are specific to a method and references to objects that are in a heap, referred from the method. When the method is finished execution, its’ corresponding stack frame is retired.

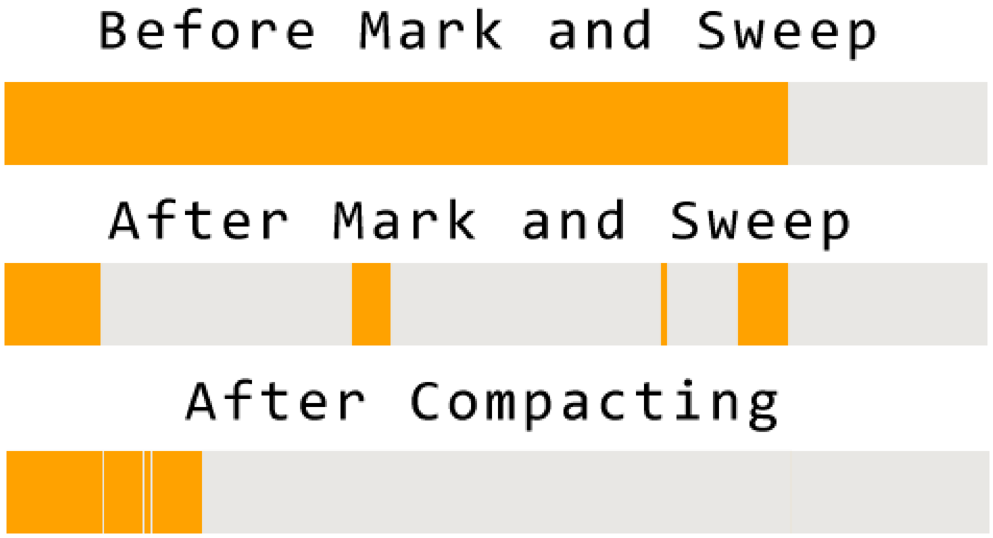
Heap Memory — New objects are created in heap space and the references to these objects are stored in stack memory. Objects have global access and can be accessed from anywhere in the application.

4.2. describe how Java’s garbage collector works

The garbage collector’s job is to automate memory management as objects are created, referenced, and deemed no longer needed by a program.

The main component of the garbage collection process is a “mark and sweep algorithm”. So how does it work at a high-level?

- The garbage collector **scans** Java’s dynamic memory areas for objects that are **referenced**.
- **Unreferenced** objects are identified and **marked** as **ready** for garbage collection. Two common ways objects can become unreferenced is by nulling the reference to an instance, or by assigning a reference to another object.
- **Marked** objects are deleted, and memory is **compacted** by moving referenced objects with allocated memory together forming a contiguous block on the **heap**. This makes it easier to allocate memory to new objects as program execution continues.



5. Getting Started with Java in Eclipse

Getting Started with Eclipse | The Eclipse Foundation

6. Java OOP

6.1. 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.

6.1.1. Create a Class

To create a class, use the keyword `class`:

MyClass.java

Create a class named “`MyClass`” with a variable x:

```java
public class MyClass {
  int x = 5;
}
```

6.1.2. 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);
  }
}
```

6.1.3. 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:

```
javac MyClass.java
javac OtherClass.java
```

Run the OtherClass.java file:

```
java OtherClass
```

And the output will be:

```
5
```

6.2. Java Class Attributes

we used the term “variable” for `x` in the example. 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;
}
```

6.2.1. 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);
  }
}
```

6.2.2. 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...).

6.2.3. 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:

```java
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
  }
}
```

6.2.4. Multiple Attributes

You can specify as many attributes as you want:

Example

```java
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);
  }
}
```

6.3. Java Class Methods

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!"
```

6.3.1. 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
  }
}
```

6.3.2. 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.

Thank you

Thank you

全国统一咨询热线：400-690-6115

北京|上海|广州|深圳|天津|成都|重庆|武汉|济南|青岛|杭州|西安