JAVA PROGRAMMING 1

Summer 2018 - Christian Hur

Unit 1 Lecture - Introduction to Java

## What is Java?

Java is a powerful, versatile, general-purpose programming language for developing software running on mobile devices, desktop computers, and servers.

History and API’s:

* Developed by team led by James Gosling at Sun Microsystems in 1991 (bought by Oracle in 2010)
* Originally Java was called “Oak” in 1991 - use in embedded chips in consumer electronic appliances
* 1995 - renamed to Java because Oak was already taken.
* Java Applets - standalone applications that run on Web server (web page)
* Java Library - the API that contains predefined classes and interfaces for developing Java programs.
* Java Development Toolkit (JDK) - the software for developing and running Java programs
* Java Language Specification defines the Java language’s syntax and semantics (<http://docs.oracle.com/javase/specs/>)
* Java is modeled after C++.
* “**Write once, run anywhere**” (**WORA**) is the slogan developed by Sun Microsystems to describe the ability of one Java program version to work correctly on multiple platforms.

## 

## Editions and Versions of Java

Java comes in three editions:

1. **Java Standard Edition** (Java SE) - develop client-side applications. Standalone or applets
2. **Java Enterprise Edition** (Java EE) - develop server-side applications such as Java servlets, JavaServer Pages (JSP), and JavaServer Faces (JSF).
3. **Java Micro Edition** (Java ME) - develop applications for mobile devices such as cell phones.

Latest stable version: 8 (JRE 1.8.x, JDK 1.8.x)

## 

## Dimensions of Computer Languages

Computer languages can be classified in many dimensions. Below is a pictorial of where Java lies in relation to other computer languages.

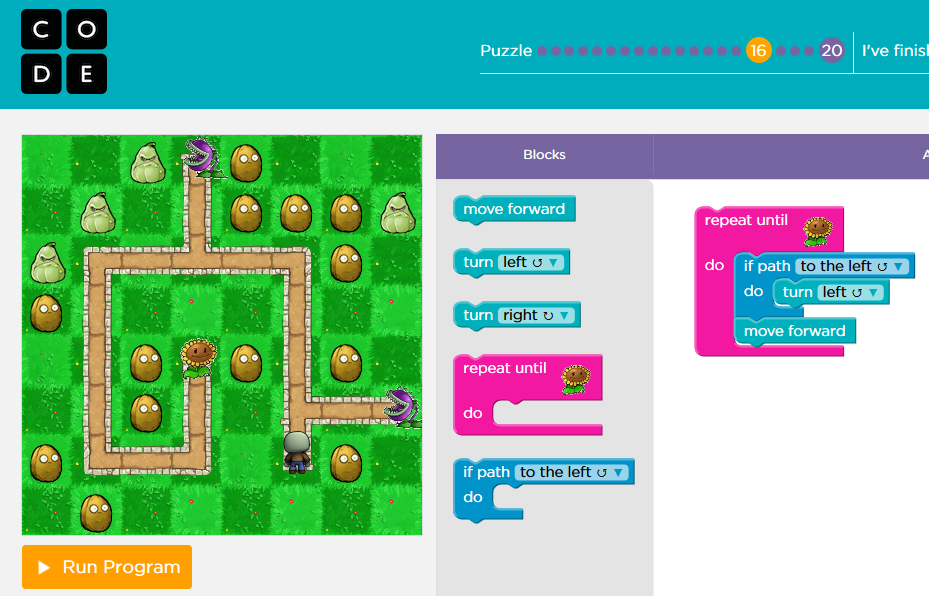


As you can see in the pictorial above, most of computer languages are High-Level. I can’t think of a specific purpose low-level language. I am sure there is at least one, but I can’t name one. That’s because if it did exist, no one is using it now, much like not everyone is really using Assembly anymore.

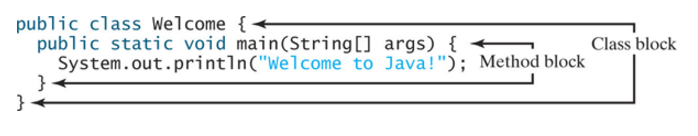
We can classify computer languages even further. Programming languages are also classified as a First Generation (1GL), Second Generation (2GL), and Third Generation (3GL). First generation languages are primarily referred to low level or machine level languages like Assembly. These languages are so close to the machine level that no compiler/interpretter is needed.

Second generation languages are most of the languages you see, hear, and know today. These include everything you see on the right side of the pictorial shown above. Second generation languages are primarily high level, which means the code can be easily read and write by a programmer and requires a compiler or interpretter to process the code to machine language.

Third generation languages are on the rise. These are languages that require no or very little coding but graphical in concept. To write code, a programmer would just drag and drop commands or blocks of commands (code blocks) to form a statement. A good example that simulates this is by visiting the Hour of Code (<http://hourofcode.com>) and run a few of their examples. Below is a screenshot of a JavaScript code written third generation.



Computer languages can also be classified as Procedural or Object-Oriented. Java is purely object-oriented starting with the moment you write your first line of code. Object-oriented always uses classes, and thus you’ve seen something similar to the following example throughout Java programs.



## Strongly-Typed vs. Weakly-Typed Language

A strongly-typed language is one in which a data type must be specified when declaring variables and functions.

For example, if you plan to use the following variables and functions (methods) in your program, you must first specify their data types.

int number = 12;

String name = “Christian Hur”;

double[] grades = {99.3, 89.5, 90.50};

boolean found = false;

public double sum(double a, double b) {

return (a + b);

}

public boolean isMatch(String a, Strong b){

return (a == b);  
}

A weakly-typed (or loosely-typed) language is one that does not require the programmer to specify a data type when declaring a variable or function. This is because the variable can hold any data type at any given time. Likewise, the function can return any type of data back. JavaScript, PHP, and Python are all weakly-typed languages. Here are some examples in PHP.

$name = “Christian Hur”; //initially, it’s a string data type

.

.

$name = 123; //later down the code, it’s now switched to integer

$name = true; //it’s now boolean

function result($a, $b){

If ($a > $b)

return 123; //return integer type

Else if ($a < $b)

return “Super!”; //return string type

Else

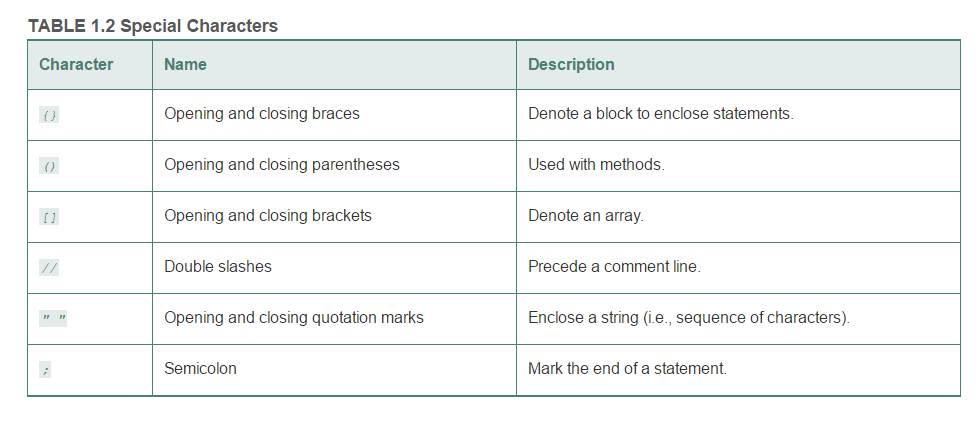
return true; //return boolean type

}

You can see the flexibility in a weakly-typed language, but also the danger in it because data can be changed at any given moment which is prone to produce logic errors. Thus, a weakly-typed language is not very suitable for object-oriented programming although it can be done with care.

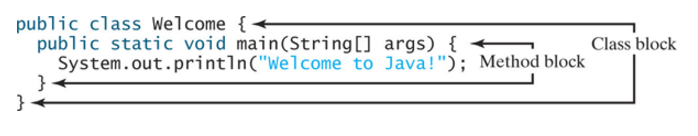
## Syntax

* A Java program is executed from the “**main**” method in the class.
* Java language is case sensitive (**area** , **Area** , and **AREA** are all different identifiers)
* Strongly-typed language
* Java source code is stored in a .java file
* Java source .java is compiled by the compiler (javac.exe) to .class file (bytecode)
* Bytecode is executed by Java Virtual Machine (JVM) on any system.

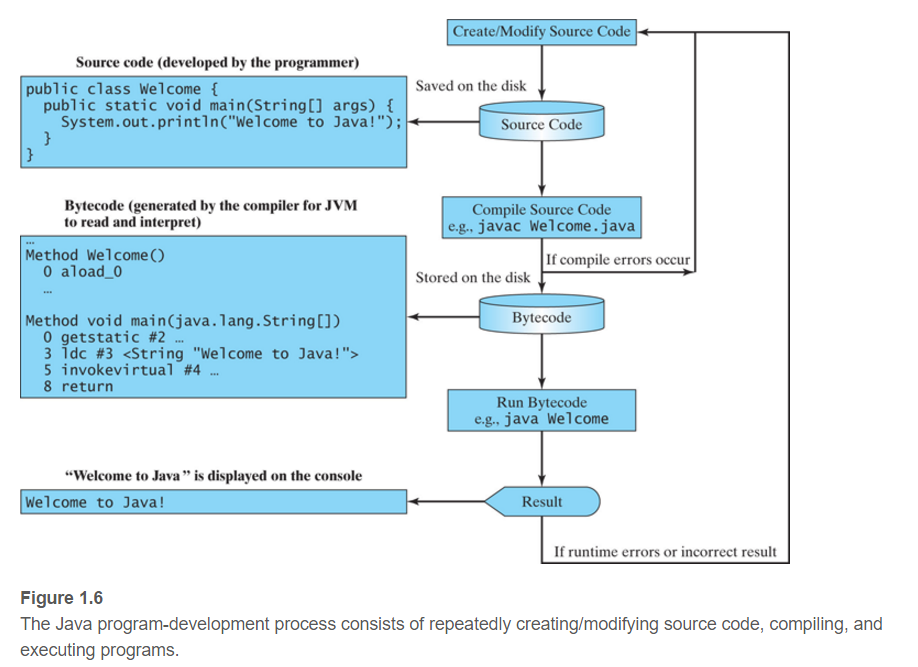


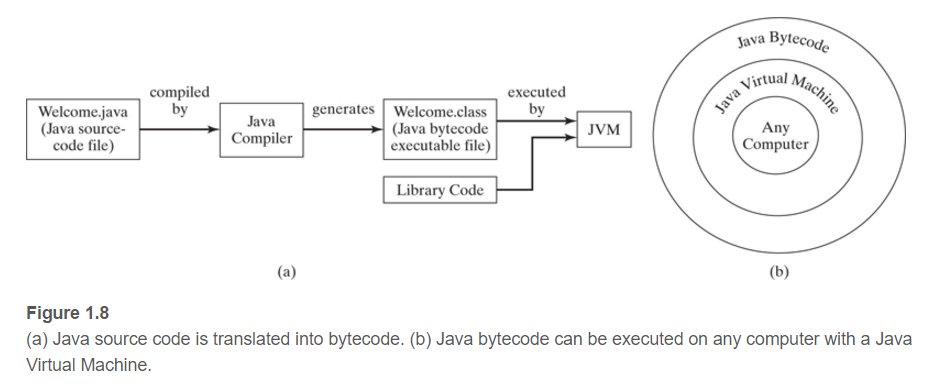
## Simple Java Program

A computer program is a set of instructions that tell a computer (machine) to perform a task. A program is composed of program statements. In order for a program to run (execute), it must be translated by a special program called a “compiler” or an “interpreter”. Consider the following simple Java program.



Below is a pictorial of the processes of how the simple program is compiled, interpretted, and executed.





# Compiler: javac.exe

The command to compile a Java program is “javac”. The interpreter to execute a compiled Java program is “java”.

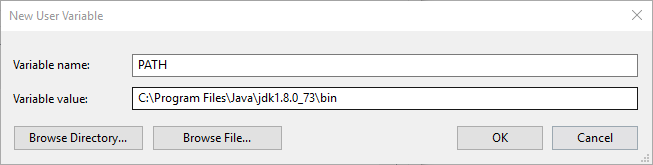
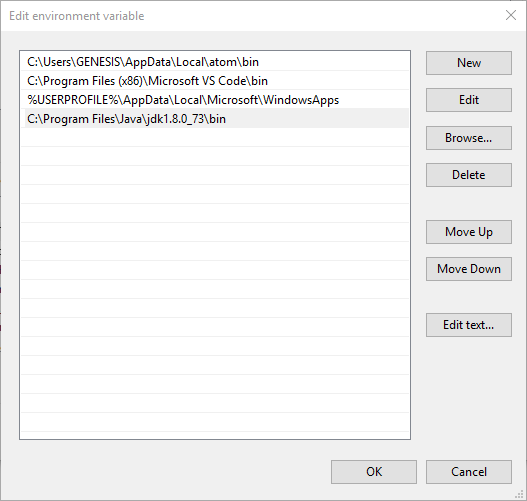
## Setup Environment Variables:

In order to compile and execute a Java program using the Command line, follow one of the following steps on your local machine to setup the environment.

### A. Command Prompt

1. Locate the path to javac.exe in JDK/bin.
2. C:> set path=”C:\Program Files\Java\jdk1.8.0\_73\bin”
3. Navigate to the location of your Java program (i.e. HelloWorld.java)
4. javac HelloWorld.java //Compiles it to HelloWorld.class
5. java HelloWorld //Executes - notice it doesn’t include the “class” extension

### B. GUI (Advanced System Settings)

1. Locate the path to javac.exe in JDK/bin.
2. Right click on “Computer” => Properties
3. Select “Advanced system settings”
4. Under the “Advanced” tab, click “Environment Variables…”
5. Click “New” if no PATH variable exists  
   
6. Click “Edit” the PATH variable if exists.
7. Click OK to exit out of all windows.
8. Restart CMD
9. Javac HelloWorld.java //Compile
10. Java HelloWorld //Execute

## 

## Identifiers

* An identifier is a sequence of characters that consists of letters, digits, underscores ( \_ ), and dollar signs ( $ ).
* An identifier must start with a letter, an underscore ( \_ ), or a dollar sign ( $ ). It cannot start with a digit.
* An identifier cannot be a reserved word
* An identifier cannot be true , false , or null .
* An identifier can be of any length.

## Named Constants

* A named constant is an identifier that represents a permanent value.
* A constant must be declared and initialized in the same statement.
* The keyword “final” is used for declaring a constant

final double PI = 3.14;

## Naming Conventions

* Use lowercase for variables and methods. If a name consists of several words, concatenate them into one, making the first word lowercase and capitalizing the first letter of each subsequent word—for example, the variables radius and area and the method print .
* Capitalize the first letter of each word in a class name—for example, the class names ComputeArea and System .
* Capitalize every letter in a constant, and use underscores between words—for example, the constants PI and MAX\_VALUE .

## I/O - Reading Input from Console

Java uses System.in to read input from console (keyboard), and System.out to write to console (monitor).

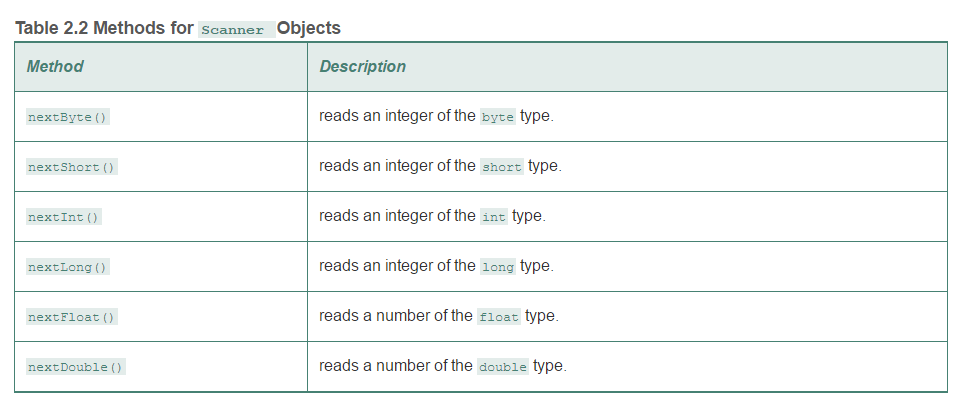
Output: System.out.print(“Hello world!”);

Input: Use the Scanner class to create an object to read input from the System.in.

Scanner input = new Scanner(System.in);

double radius = input.nextDouble();

int age = input.nextInt();



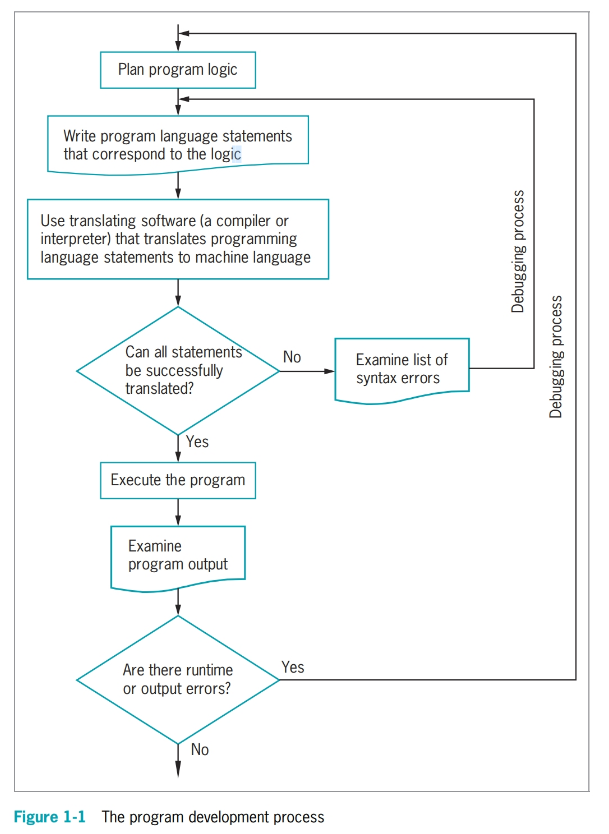
# 

# Code Indentation Styles

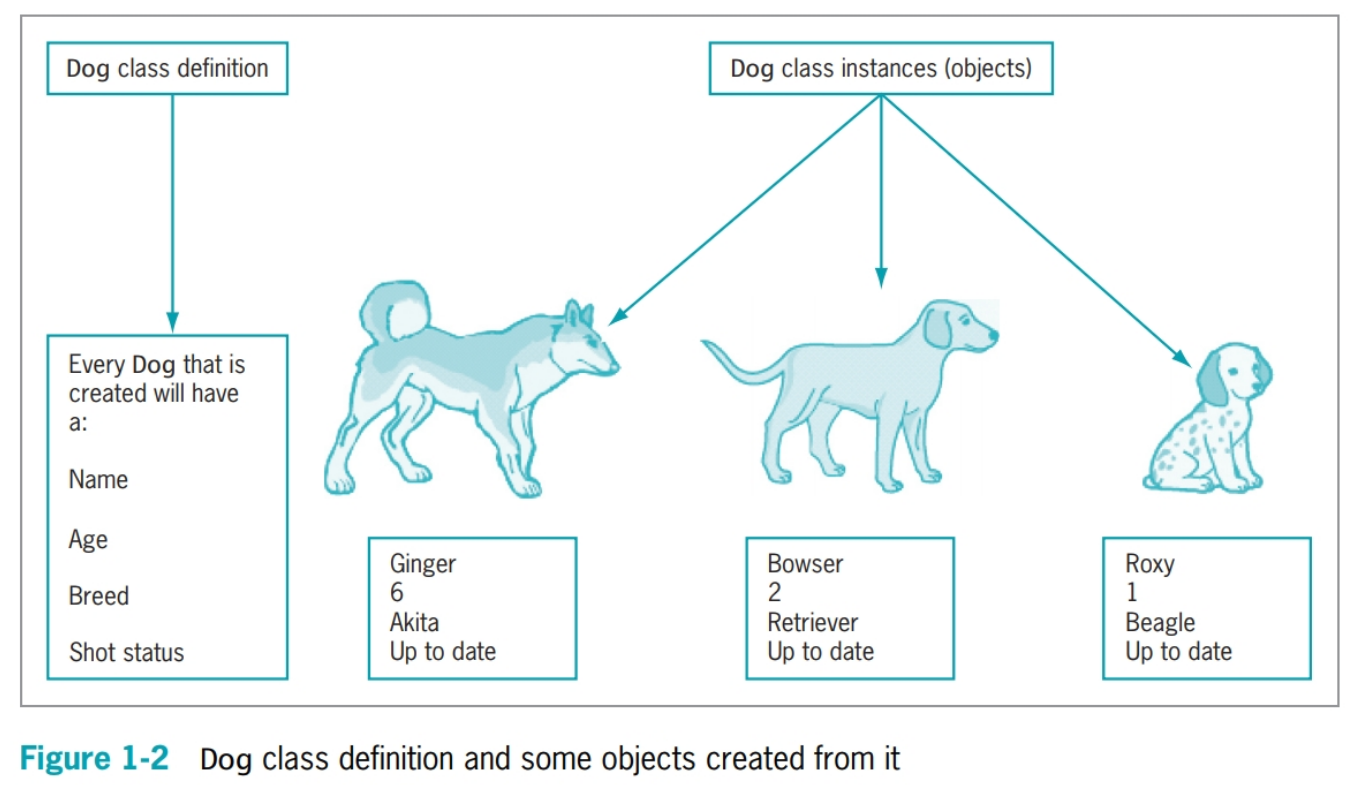
The curly braces, **{}**, are used in pairs to define a class and a method body. Note that in a properly formatted Java program, the curly brace that opens a method should be placed either at the end of the method header line or on the next line. Whitespace in the code is ignored by the compiler. The **K & R** style and **Allman** style of brace placement styles are good to use, and you should pick one that you understand.

|  |  |
| --- | --- |
| **Brace placement** | **Styles** |
| **while** (x == y) {  something();  somethingelse(); } | [K&R](https://en.wikipedia.org/wiki/Indent_style#K.26R) and variants:  [1TBS](https://en.wikipedia.org/wiki/Indent_style#1TBS), [Stroustrup](https://en.wikipedia.org/wiki/Indent_style#Variant:_Stroustrup), [Linux kernel](https://en.wikipedia.org/wiki/Indent_style#Variant:_Linux_kernel), [BSD KNF](https://en.wikipedia.org/wiki/Indent_style#Variant:_BSD_KNF) |
| **while** (x == y) {  something();  somethingelse(); } | [Allman](https://en.wikipedia.org/wiki/Indent_style#Allman_style) |
| **while** (x == y)  {  something();  somethingelse();  } | [GNU](https://en.wikipedia.org/wiki/Indent_style#GNU_style) |
| **while** (x == y)  {  something();  somethingelse();  } | [Whitesmiths](https://en.wikipedia.org/wiki/Indent_style#Whitesmiths_style) |
| **while** (x == y) { something();  somethingelse(); } | [Horstmann](https://en.wikipedia.org/wiki/Indent_style#Horstmann_style) |
| **while** (x == y) { something();  somethingelse(); } | [Pico](https://en.wikipedia.org/wiki/Indent_style#Pico_style) |
| **while** (x == y) {  something();  somethingelse();  } | [Ratliff](https://en.wikipedia.org/wiki/Indent_style#Ratliff_style) |
| **while** (x == y) {  something();  somethingelse(); } | [Lisp](https://en.wikipedia.org/wiki/Indent_style#Lisp_style) |

## The Program Development Process



## Understanding the relationship between Class and Objects



## The PIE in OOP (cover in Chapter 10)

The three pillars of a true object-oriented programming.

**Polymorphism**

* Principle in biology in which an organism/species can have many different forms/stages
* Allows the same word to be interpreted correctly in different situations based on context
* Feature of *method override*

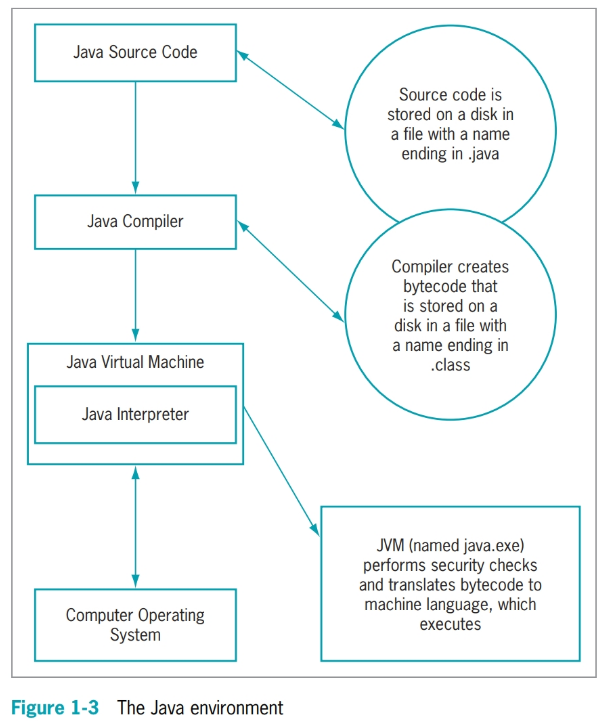
**Inheritance**

* Classes share attributes and methods of existing classes but with more specific features
* Helps you understand real-world objects
* The “extends” keyword (e.g. public class Dog extends Pet { … } )

**Encapsulation**

* Conceals internal values and methods from outside sources
* Provides security (e.g. private int age; private int calculate(){ … })
* Keeps data and methods safe from inadvertent changes

## Steps of how a Java program is compiled and executed

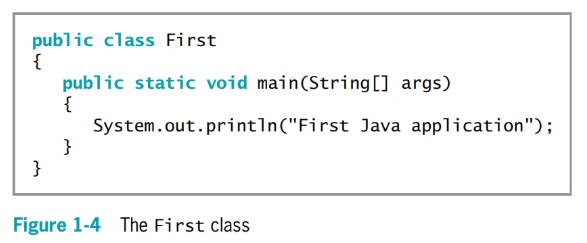


## Shell code for a typical Java class with a main() method

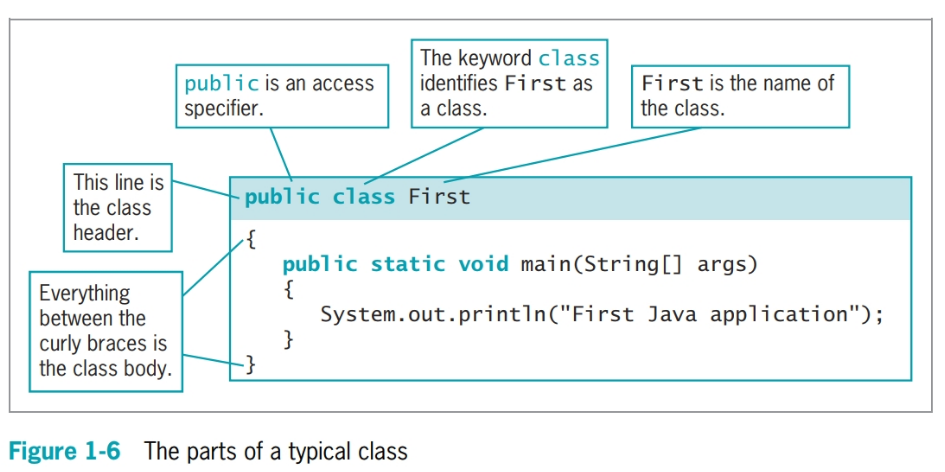
## 

***Note: Not all classes must have a main method - only one method will run per program.***

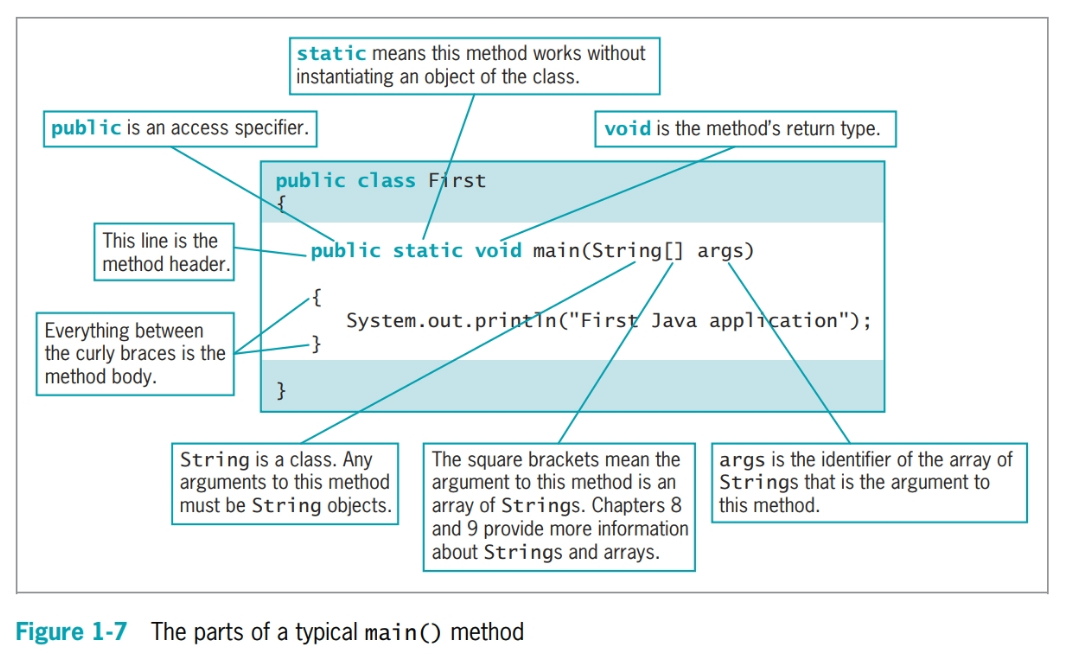
## A simple Java program



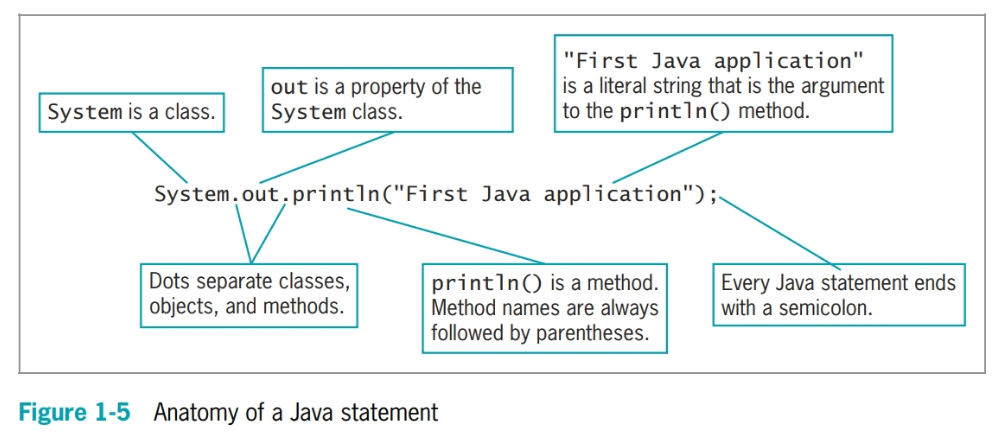
## Anatomy of a simple Java class



## Anatomy of the **main()** method

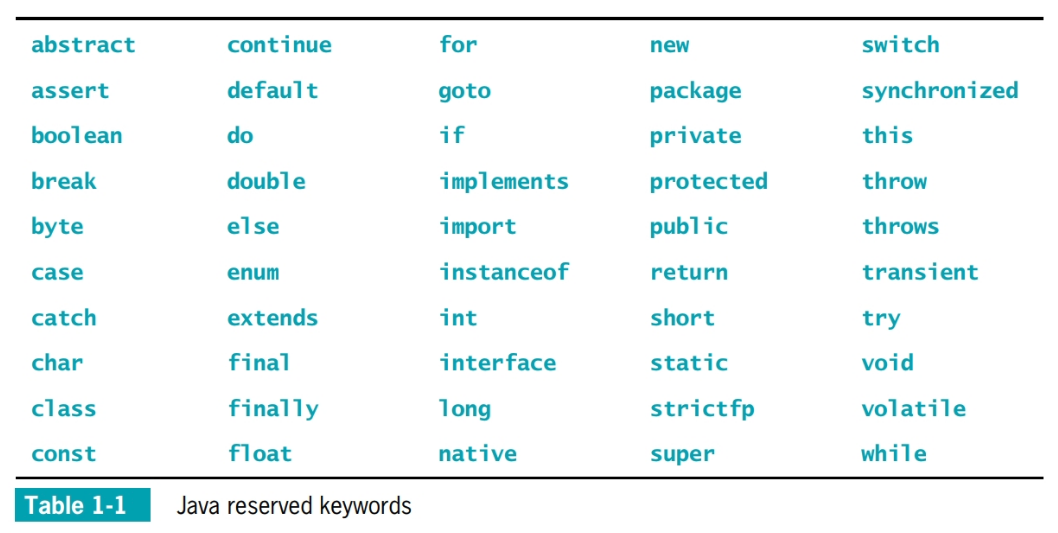


## Anatomy of a Java Statement.



## 

## All the keywords in Java - only 50 of them (all lowercase)



You cannot use any of the above keywords as standalone identifiers in your programs.

The following are not allowed:

boolean **final** = true; // illegal identifier

String **this** = “Java is fun!”; // illegal identifier

public void **try**() { … } // illegal method name

public class **goto** { … } // illegal class name

Keywords used as a part of an identifier or multiple words are legal:

boolean **finals** = true; // final is a keyword but not finals

//***try*** and ***this*** are keywors but are joined so it’s no longer a keyword.

String **trythis** = “Java is fun!”;

public void **Try**() { … } //Try is capitalized and Java is case sensitive

public class **Goto** { … } //Goto is capitalized

Details here: <https://en.wikipedia.org/wiki/List_of_Java_keywords>

## Three Types of Computer Errors.

1. **Syntax error**: incorrect use of keywords or code that violate the rule of the language. Easiest to fix.
2. **Run-time error**: an error not detected until the program asks the computer to do something wrong, or even illegal, while executing. It could be caused by syntax error or logic error during execution (e.g. the erronous code only runs when a condition is met - such as in an IF block).
3. **Logic error**: a bug that allows a program to run, but that causes it to operate incorrectly. This is the most difficult to debug (fix).