Chapter 1  
Introduction to Computers, Programs, and Java

Contents

[Introduction 2](#_Toc41813477)

[What is in a computer? 3](#_Toc41813478)

[Bus 3](#_Toc41813479)

[CPU 3](#_Toc41813480)

[Hertz 3](#_Toc41813481)

[Core 4](#_Toc41813482)

[Bits and Bytes 4](#_Toc41813483)

[Memory 4](#_Toc41813484)

[Programming Languages 4](#_Toc41813485)

[Machine Language 4](#_Toc41813486)

[Assembly Language = Low Level language 4](#_Toc41813487)

[High Level languages 5](#_Toc41813488)

[Interpreter 5](#_Toc41813489)

[Compiler 5](#_Toc41813490)

[Operating Systems – OS 5](#_Toc41813491)

[Java 5](#_Toc41813492)

[API – Application Program Interface 5](#_Toc41813493)

[JDK – Java Developing Toolkit 5](#_Toc41813494)

[IDE – Integrated developing environment 5](#_Toc41813495)

[Writing Java Programs 7](#_Toc41813496)

[One Line at a Time Approach 7](#_Toc41813497)

[Class name 8](#_Toc41813498)

[Main method 8](#_Toc41813499)

[Statement terminator 8](#_Toc41813500)

[Reserved words / Keywords 8](#_Toc41813501)

[String 9](#_Toc41813502)

[Comments 9](#_Toc41813503)

[Block 9](#_Toc41813504)

[Special Characters 9](#_Toc41813505)

[Case sensitive 9](#_Toc41813506)

[Syntax Rules 10](#_Toc41813507)

[Programming Style and Documentation 10](#_Toc41813508)

[Programming style 10](#_Toc41813509)

[Documentation 10](#_Toc41813510)

[Comments 10](#_Toc41813511)

[Indentation 11](#_Toc41813512)

[Spacing 11](#_Toc41813513)

[Block Styles 11](#_Toc41813514)

[Programming Errors 11](#_Toc41813515)

[Syntax Errors 11](#_Toc41813516)

[Runtime Errors 11](#_Toc41813517)

[Logic Errors 12](#_Toc41813518)

# Introduction

The central theme of this text is to learn how to solve problems utilizing a computer by writing a program.

Programming: create (or develop) software, which is called a program.

Software: a program (a set of detailed instructions) that instructs the computer what to do. Programs are written in a special language, a program language, which the computer will decode to machine language.

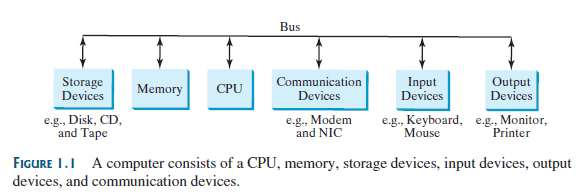
# What is in a computer?

Major hardware components:

* CPU: central processing unit
* Memory (main memory)
* Storage devices
* Input devices: mouse, keyboard, microphone
* Output devices: monitor, printers, speakers
* Communication devices: modem, network cards

# Bus

A computer’s component are interconnected by a subsystem called a bus – wires that connect all the parts together.



# CPU

The CPU, central processing unit, retrieves instructions from memory and executes those instructions. Usually consists of two units: a control unit (controls and coordinates the actions of the components) and an arithmetic/logic unit (performs numeric operations, like addition, subtraction, multiplication and division, and logical operations/comparisons).

# Hertz

The electronic pulses in a computer that control and synchronize the pace of operation in a computer are performed at a constant rate, that is measured in hertz (Hz), where 1 Hz is 1 pulse speed per second.

# Core

The core is part of the processor that performs the reading and executing of instructions. Most computers today are multicore, with somewhere between 2 to 36 cores.

# Bits and Bytes

A computer is nothing more than a series of switches (a device that can store electricity).

On = storing electricity = 1

Off = not storing electricy = 0

This 1, or 0, is called a bit, which stands for binary digit.

We can combine this values together, 8 bits = a byte, the bytes can represent values, like the number 3; but to represent this value we need an encoding scheme (a set of rules that govern how a computer translates characters, numbers, and symbols into data the computer can actually work with). Today our computers use the Unicode encoding scheme, which is based on the ASCII encoding scheme.

# Memory

A computer’s memory consists of an ordered sequence of bytes for storing programs as well as the data that the program is working with.

Every byte in the memory has a unique address, which allows storing a retrieving of data and instructions. When the CPU is using a program or data, the program and data is stored in the RAM (Random Access Memory), it is called RAM because the instructions/data can be accessed in any order.

# Programming Languages

## Machine Language

Is the CPU’s native language (a 1 or a 0, which is referred to as binary code) and is a set of instructions that is built into it.

## Assembly Language = Low Level language

A language that uses short descriptive words, known as mnemonic, like add, or sub, for addition and subtraction. An assembler is used to translate the mnemonic words into binary code. Assembly languages are machine dependent.

## High Level languages

Are English-like languages, uses statements, and are easy to learn and use. Each high-level language has pros and cons. A software developer should know several as these languages are the tools of their trade.

### Interpreter

An interpreter reads one statement and executes that one statement, then moves to the next statement and continues in this fashion until the program terminates.

### Compiler

A compiler reads the entire source code, then translates that code into machine level code all at once (after reading the entire source code). The machine level code is then executed.

# Operating Systems – OS

The operating system is a program that manages and controls a computer’s activities.

The major tasks of an operating system:

* Controlling and monitoring system activities
* Allocating and assigning system resources
* Scheduling operations

# Java

Java is a high-level language; it is an objected-oriented programming language.

## API – Application Program Interface

Aka the library. It contains the predefined classes and interfaces for developing Java programs.

## JDK – Java Developing Toolkit

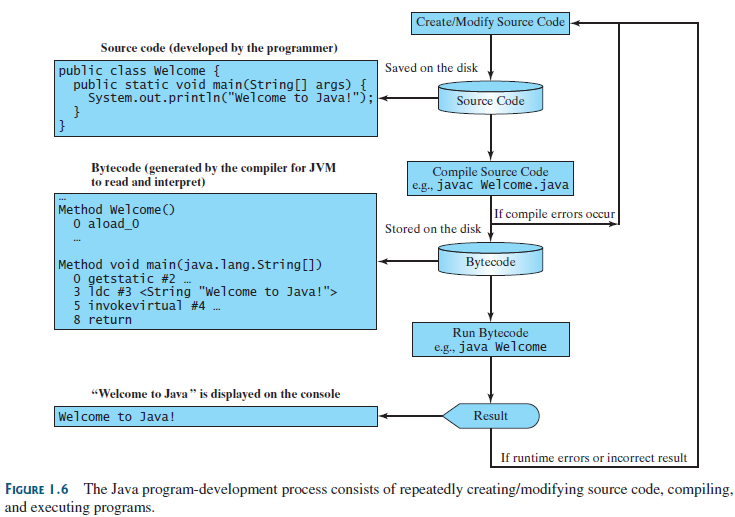
The JDK consists of a set of separate programs, each invoked from a command line, for developing and testing Java programs. Most IDEs have a version of the JDK built into them.

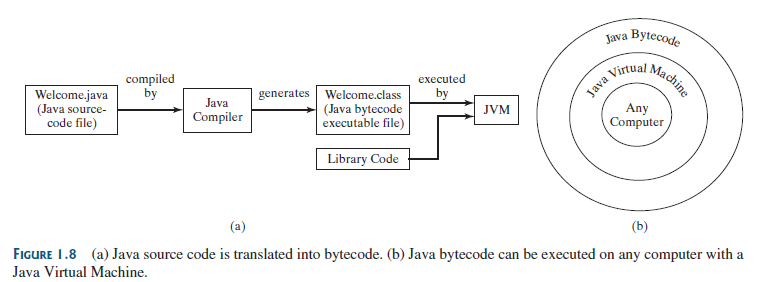
## IDE – Integrated developing environment

An IDE is -usually a graphical user interface (GUI), but not always - a software program that is used to write a high-level language program in, for editing, compiling, building, debugging, and has various help/assist tools.

Java uses *both* a compiler and an interpreter. Below are the series of steps taken when developing and executing Java programs.

1. You first write the Java statements in an IDE and save the file with the extension of .java.
2. Then you build/compile the program, this compiler will then generate a machine-level language program called *Java byte-code*, this is a different file – a new file is generated – and has the extension of .class but with the same file name. Java byte-code is different from other machine level languages in that it is platform neutral.
3. When you are ready to run/execute the program, the JVM, Java Virtual Machine, will execute the Java byte-code. The JVM is an interpreter.





# Writing Java Programs

When you are solving problems to be executed on a computer, you have several things to think about. Which language would be best? Who are the users? What kind of interaction will be needed? The system requirements? The program requirements? The list goes on and on. For us, we will be using Java, one end user (which will be us), the end user will be entering text from the keyboard and the program will display results to the screen. In chapter 5, we will introduce file I/O (input and output).

Since we are utilizing Java, we need to know about the syntax and nuances of Java.

## One Line at a Time Approach

Also, I highly/strongly suggest that when you are “writing” programs that you use this approach:

1. Think about what has to solved. Think about the features/structures of the programming language that can help/assist in solving the problem.
2. Jot is down on a scratch piece of paper.
3. Start the actual writing by **only** writing **one** line of code. Compile. Run.
4. Then add one more, and keep repeating.

Doing it this way, will stave off syntax errors, and help with logic errors as you can see what that one line of code will be doing right away.

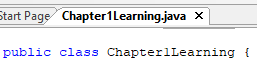
I can’t tell you how many times I have seen students write the entire code in one swoop, then compile. There are so many syntax errors, it is overwhelming. Hours will be wasted trying to figure out the syntax, only to realize that the logic isn’t working either.

Save yourself time and headaches by just doing one line at a time.

## Class name

The class name is the name of the file. Class name = the program name. This can be very confusing since the Java byte-code creates a .class file; remember, it is this file that is executed, not the .java file.

Class names start with upper case letters (they cannot start with numbers or special characters), and ***cannot*** have spaces.



The words of the file name and the words that are after class have to match verbatim.

## Main method

Main method is the method that the JVM automatically executes, it is the entry point where a program begins execution. The JVM executes each statement sequentially through the program until there are no more statements to execute, or a runtime error occurs.



This line never changes, it is always the same – memorize it verbatim.

## Statement terminator

All Java statements end with the semi-colon ( ; ). This is hard to remember to add when you are first learning – thank goodness for the compiler catching these errors.



## Reserved words / Keywords

These are words that are part of the Java language and can only be used in that context that is defined by the Java language. These words are a different color in the IDE that you use.

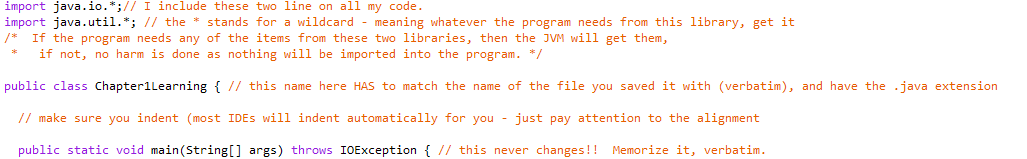
For example, in this IDE the reserved words/keywords are in blue:



## Scanner

The Scanner object allows a program to receive input from the user. The Scanner object is a reference data type (we also call those data types objects, ADT (Abstract Data Types), and class data types).

To use the Scanner, you have to import the class into your program, you accomplish this by putting the following statement prior to the class declaration statement:



You will learn more on the Scanner object in chapter 2.

## String

String is a programming term meaning a sequence of characters, including the space character. Strings are also a different color in the IDEs.

In this example, the Strings are in purple:

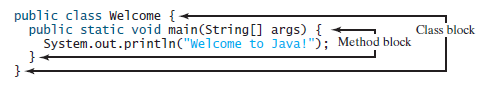


## Comments

Comments are a documentation/explanation regarding the program, to communicate and understand the program. They are not programming statements and are completely ignored by the compiler. Comments are useful to the programmers, they help identify what is going on, what the variables are used for, and so forth.

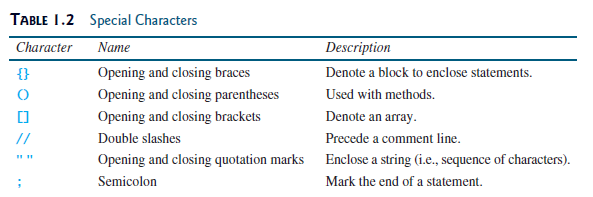
## Block

A block is a notation of grouping statements together, in Java the { } brackets denote blocks.



I suggest that you start and end the brackets as 1 step, then label the end bracket. That way you know what it matches up to and you don’t end up with dangling brackets or too few of brackets.

## Special Characters



Since the { }, the ( ), the [ ], and the “” all need a start/open and an end/close, it helps to keep the following mantra in mind: open – close, and type in between.

## Case sensitive

Java is case sensitive! Pay attention to your characters, j and J are two different characters.

## Syntax Rules

Java, just like all languages have syntax rules. In English, if we say they instead of she or he, we still understand and it’s ok. In programming languages, it is not. It has to be very specific, defined in the language, and clear and precise.

# Programming Style and Documentation

Good programming style and proper documentation make a program easy to read, maintain and help programmers prevent errors.

## Programming style

Programming style deals with what a program looks like. Programs can be written on one long line, but then you have to be able to read that one long line, find bugs, errors, modify, update, etc. that one long line. That is a headache (and more) that nobody wants to deal with. Programs can also be messy, with some indention, some mismatched indention, poor word choices for variable identifiers, repetitive statements instead of loops, no methods, etc., in general what is referred to as spaghetti code = sloppy code and usually full of errors. Another type of headache nobody wants to deal with.

Good programming style is consistent throughout the program and uses indentions that line up, loops, uses methods, good word choices for variable identifiers, in other words clean, easy to read code.

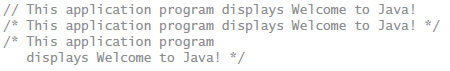
## Documentation

Documentation deals with providing an explanation of what the code is doing. Documentation is a big part of software development; without it, programs are hard to maintain (as one has to reverse engineer the sloppy code in order to find the spots that need to be modified).

For some examples of good documentation, look at the Java code of some of the classes, like the ArrayList (<https://hg.openjdk.java.net/jdk8/jdk8/jdk/file/tip/src/share/classes/java/util/ArrayList.java>).

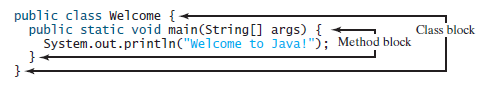
## Comments

A single line comment can be achieved by using //, in most IDEs comments are green colored text. Multiple lines of code can be achieved by using /\* at the beginning and ending with \*/.



## Indentation

Blocks of code should indented on the same level, nested blocks should be indented further – the more the block is embedded, the more the indentation should be.



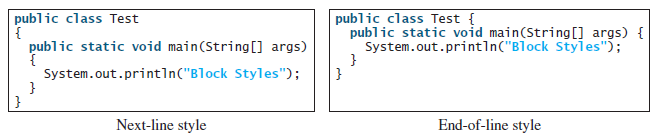
## Spacing

Provide spacing between the operators and the operands.



## Block Styles

A block is a group of related statements to a task, and are enclosed in { } . There are 2 popular block styles, next-line and end-of-line. Pick one of these styles and stick with that one style, do not mix the two styles in one program.



FYI: I am an end-of-line style, when you look at my program examples, you will see that style.

# Programming Errors

Programming errors are categorized in three types: syntax, runtime, and logic errors.

## Syntax Errors

These are the errors that the compiler catches for us (thank goodness too – as we make them all the time, especially when we are first learning to program). These errors are the typing errors, the forgotten ; at the end of the statement, the off } braces, etc.

Remember, it helps to type one line at a time to help with these errors.

## Runtime Errors

Runtime errors cause the program to terminate abnormally, the program is executing and then exits with a description of the runtime error (which usually does not provide very much information).

## Logic Errors

Logic errors are the hard ones to find. The program runs/executes and ends, it provides an output, but the output was not a valid output, or what was intended to happen.

Remember, it helps to type one line at a time to identify the error in your logic. It also helps to print variables/results/etc. to the screen to see if you are getting the result that you think you are.