

CSC 1051 Algorithms & Data Structures I

Introduction to Java



COMPUTER PROCESSING

Why program a computer?

Computer programming is the process of designing and building an executable computer program to accomplish a specific **computing result**.

To make it possible for humans to use computers, they need to be able to communicate information and instructions back and forth.

The **computer program** is how this communication is done.

Communication



Humans converse with each other using **natural languages**

Humans converse with computers using **programming languages**



Talking to the Computer

You tell the computer what to do using a sequence of step-by-step instructions called a **program**

Developing a program involves a few activities:

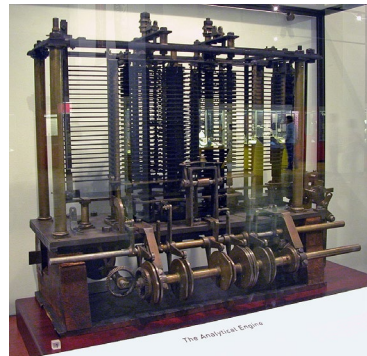
- Writing the program in a specific **high-level language** (such as Java)
- Translating the program into a **low-level language** or **machine code** that the computer can run

History of Computer Programming

The first computer program is generally dated to 1843



Ada Lovelace
(1815 -1852)



Analytical Engine
Designed by Charles Babbage



Bernoulli numbers

$$\sum_1^n r^p = \sum_0^p \frac{B_k}{k!} \frac{p!}{(p-1+k)!} n^{p+1-k}$$

More here: <https://twobithistory.org/2018/08/18/ada-lovelace-note-g.html>

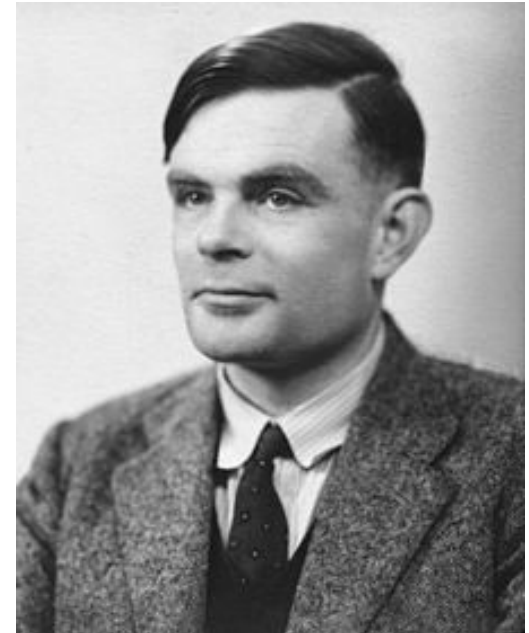
History of Computer Programming

Starting in the mid-20th century...

Colossus Mark I – first electronic computer to be programmable
(Alan Turing, England 1944)

Stored program and the fetch/
decode/execute cycle
(**John von Neumann**, USA 1945)

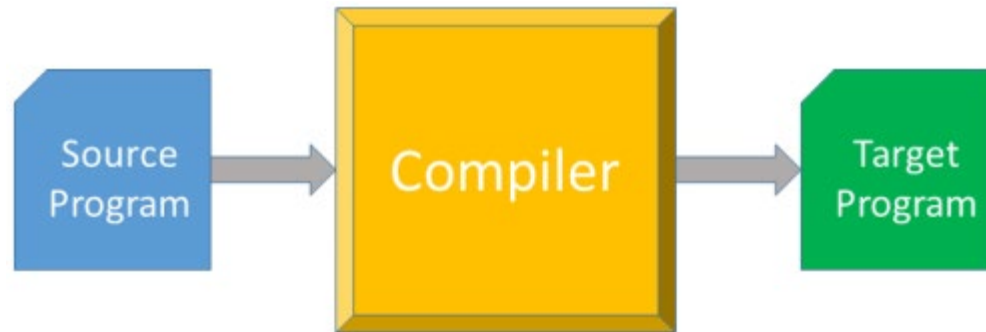
ENIAC - first fully electronic digital computer
(Eckert and Mauchley, University of Pennsylvania, 1946)



Alan Turing (1912 -1954)
The Imitation Game (2014)

Compiling the Program

The **source program** is written by a programmer in a high-level language, like Java, Python, or C.



The **target program** comes out of the compiler in either a textual form called **assembly language** or a binary (1s and 0s) form called **machine code**.

A **compiler** translates from source to target language.

Types of Code

Source Code

```
int x = 5;
```

```
y = z * 8;
```

```
a = 3 * y;
```

Assembly Language

```
mov R0, 5
```

```
mul R1, R0, 8
```

```
mul R2, r1, 3
```

Machine Code

```
10000101001111
```

```
00100000110100
```

```
00110111011011
```

System Components

Hardware

- the physical, tangible parts of a computer
- keyboard, monitor, disks, wires, chips, etc.

Software

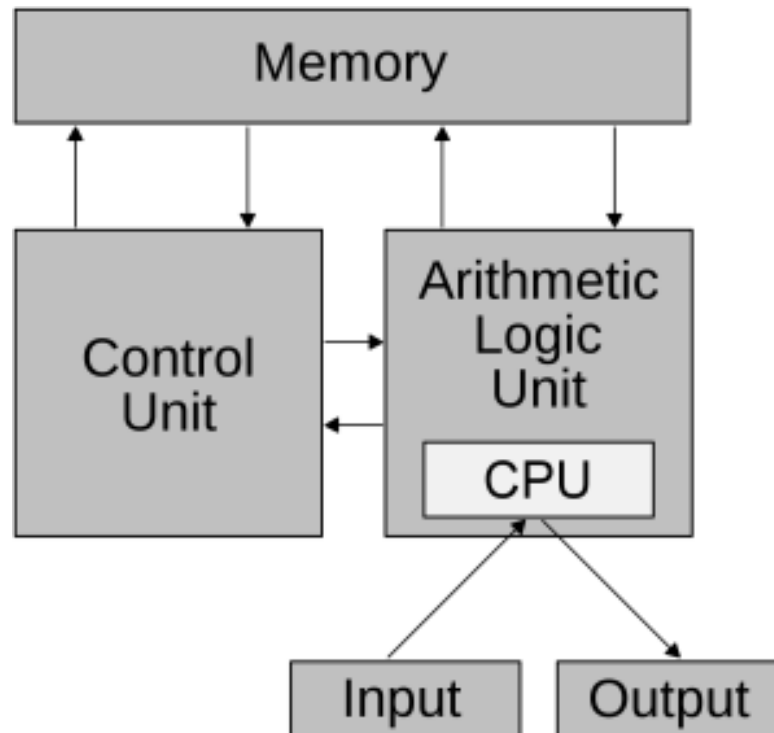
- programs and data
- a **program** is a series of step-by-step instructions

A computer needs both

- Requires both hardware and software
- Each is essentially useless without the other

The von Neumann Architecture

Modern computers all use a **computer architecture** created in 1945 by a mathematician and inventor named **John von Neumann** containing 5 essential computer components.



The 5 Components in Every Computer

- **Control Unit** – the **heartbeat** of the system, orchestrates execution of all instructions.
- **Arithmetic Logic Unit** – the **brain** of the system, uses the Central Processing Unit (**CPU**) to perform math, logic, and comparison operations.
- **Memory Unit** – stores programs and data, the way a human brain **remembers information**, in Random Access Memory or **RAM** (short-term) and **Hard Disk** (long-term).
- **Input Devices** – how data **gets in**, keyboard, mouse, camera, microphone, touchscreen, etc.
- **Output Devices** – how data **comes out**, monitor, speakers, headphone, printer, etc.

Software

Hardware needs **software** to tell it what to do. There are two categories of software:

Operating Systems

- Provides a user interface to the computer (console or GUI)
- Provides a program interface to the hardware
- Manages system resources like CPU and memory
- Examples: Windows, Mac OS, Linux, Unix

Application Programs

- Generic term for any type of software
- Examples: Word processor, browser, games, mobile apps

Counting Things

The natural way for humans to count things is with our **10 fingers**. This is why we our number system is **decimal** or **base 10**.



This approach is called **Place Value**. The place or position of each digit determines its **value**.

Binary Numbers

Unlike humans, computers only have **2 fingers** to count with in the form of electronic switches that are either **on** or **off**.

Computers use the **binary** or **base 2** number system.



A single **binary digit** (0 or 1) is called a **bit**

Devices that store and move information are cheaper and more reliable if they have to represent only two states

Permutations of bits are used to store values

Binary Numbers

Binary numbers use the position of each bit, just like decimal numbers do, to determine their values.

128	64	32	16	8	4	2	1
1	0	0	1	1	0	1	1
<hr/>							
128 + 0 + 0 + 16 + 8 + 0 + 2 + 1							
= 155							

Each position is a **power of two**. From right to left, the values are 1s, 2s, 4s, 16s, 32s, 64s, 128s, etc.

Bit Permutations

1 bit	2 bits	3 bits	4 bits
0	00	000	0000 1000
1	01	001	0001 1001
	10	010	0010 1010
	11	011	0011 1011
		100	0100 1100
		101	0101 1101
		110	0110 1110
		111	0111 1111

Each additional bit doubles the number of possible permutations

Bit Permutations

Each permutation can represent a particular item

There are 2^N permutations of N bits

Therefore, N bits are needed to represent 2^N unique items

**How many
items can be
represented by**

1 bit ?

$$2^1 = 2 \text{ items}$$

2 bits ?

$$2^2 = 4 \text{ items}$$

3 bits ?

$$2^3 = 8 \text{ items}$$

4 bits ?

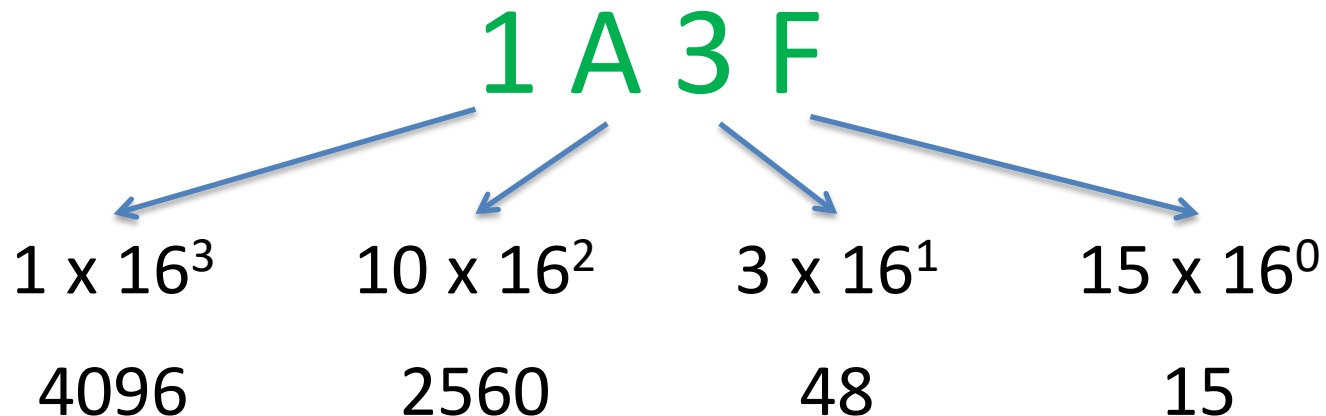
$$2^4 = 16 \text{ items}$$

5 bits ?

$$2^5 = 32 \text{ items}$$

Hexadecimal

To save space, the **hexadecimal** number system was devised using **base 16** numbers to represent numbers from 0 to 15 with a single hex digit. It uses decimal digits 0 to 9 followed by letters A (for 10), B (11), C (12), D (13), E (14), and F (15).



$$4096 + 2560 + 48 + 15 = 6719$$

Bit Permutations – Example

How many bits would you need to represent ten digits in decimal system?

1. How many unique items?

- 10 digits (0 through 9)

2. How many bits at least?

- $2^3 = 8$
- $2^4 = 16$
- $8 < 10 < 16$
- Need **4** bits

0000	0	0101	5
0001	1	0110	6
0010	2	0111	7
0011	3	1000	8
0100	4	1001	9

The rest of the permutations are not needed

Exercise

How many bits would you need to represent each of the 50 states of U.S. using a unique permutation of bits?

Five bits wouldn't be enough, because 2^5 is 32.

Six bits would give us **64** permutations, and some wouldn't be used.

000000	Alabama
000001	Alaska
000010	Arizona
000011	Arkansas
000100	California
000101	Colorado
etc.	

Try Yourself

We have 7.8 Billion people in the world.

How many bits can be used to represent each person?

Try Yourself

How many bits can be used to represent a single COVID-19 test result?

A test result could be:

- Positive
- Negative
- No Contact
- Invalid Test

USING JGRASP

Install Java JDK

To use [jGRASP](#), you must first install the [Java JDK](#).

Step 1. Visit the [Java SE Download page](#) and click the link for **JDK Download**

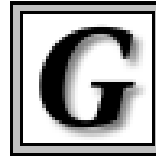
Step 2. Select and download the installer for your system

Step 3. Double-click the downloaded file to install it

Easier: read [Using jGRASP](#) in **Rephactor**

Install jGRASP

Visit the [jGRASP website](#) and click on the **Download jGRASP** link.



[Download jGRASP](#)

Fill out the **User Information** and optionally the **Email** and **Name** boxes. Find the latest [jGRASP](#) version for your system, download and install it. Avoid the "Beta" version as it is still under development.

btw

Mac users, make sure to move the jGRASP app into your Applications folder before trying to open it.

INTRODUCTION TO JAVA

The Java Programming Language



One of the most successful programming languages ever

Over 9 million developers in every major industry

The Java Programming Language

Designed by James Gosling and the "Green Team"



at Sun Microsystems, Inc.

The Java Programming Language

Originally intended to control consumer devices

special-purpose language

Soon changed to a much broader scope

general-purpose language

First public release: 1995

Sun Microsystems was purchased by Oracle in 2010

The Java Programming Language

Key features when Java was first released:

Platform Independence – not tied to one type of computer*

Applets – Programs that run in a web browser

Object-oriented – A effective programming paradigm

Garbage Collection – Java "cleaned up" memory by getting rid of unused objects

*Java's slogan: "Write Once, Run Anywhere."

Hello, World!

A **program** is a sequence of instructions expressed in a particular **programming language** such as Java

We'll start with a classic first example

Program **statement**:

```
System.out.println("Hello, world!");
```


Output:

```
Hello, world!
```


Hello, World!

System.out is an
object representing the
console window

println is the name
of a method



```
System.out.println("Hello, world!");
```

A method is a group of program statements that can be called (or invoked)

The println method is part of a big library of code that we can make use of in any Java program

Hello, World!

The **main method** is the starting point of any Java program

The **header** of the main method must be written like this

```
public static void main(String[] args)
{
    System.out.println("Hello, world!");
}
```

For now, just consider this the scaffolding necessary to write a main method

The code between the { and } is called the **method body**

Hello, World!

Finally, every Java method must be defined in a **class**

```
public class HelloWorld
{
    public static void main(String[] args)
    {
        System.out.println("Hello, world!");
    }
}
```

Similar to a method, a class has a **class header** and a **class body**

Consistent indentation makes the program more readable

Spaces and tabs in a program are called **white space**

Hello, World!

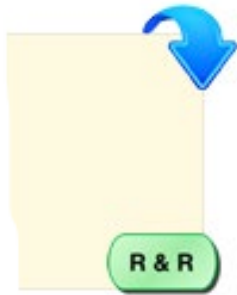
Another example:

```
public class Proverb
{
    public static void main(String[] args)
    {
        System.out.println("Tell me and I forget.");
        System.out.println("Show me and I remember.");
        System.out.println("Involve me and I understand.");
    }
}
```

```
Tell me and I forget.
Show me and I remember.
Involve me and I understand.
```

Hello, World!

Make use of the download and R & R buttons on code in the textbook



← Download the program to your computer

← Run & Revise the code right in the browser

Comments

Comments explain a program's purpose and processing

They are intended for the human reader – they have no effect on a program

A **single-line comment** begins with a `//` and continues until the end of the line

```
// This is a comment
```

It might be put on the end of a line of code

```
balance = balance - fees; // deduct monthly fees
```

Comments

A **multi-line comment** begins with a `/*` and ends with a `*/`

It might span multiple lines

```
/*  A multi-line comment that only spans one line  */  
  
/*  
    A multi-line comment that spans multiple lines  
    might be formatted like this.  
*/
```

A variation of the multi-line comment begins with `/**` and is called a **JavaDoc comment**

JavaDoc comments are used to generate online documentation

Comments

```
/*
 * Demonstrates the use of comments.
 */
public class SimpsonQuote
{
    /*
     * Prints a quote from The Simpsons TV show.
     */
    public static void main(String[] args)
    {
        // From the episode "Bart vs. Thanksgiving"
        System.out.println("Operator, what's the number " +
            "for 9-1-1?"); // season 2, ep 7
        System.out.println("    - Homer Simpson");
    }
}
```


Program Style

The term **program style** refers to the way your program is formatted and the conventions you follow

Program conventions are guidelines that technically don't have to be followed, but should be to make your program easy to read

For example, use appropriate spacing (**white space**) between symbols in an expression:

```
System.out.println("Total:" + count * unitCost);
```

A single space on either side of an operator

Program Style

Consistent indentation is important, and is related to the **block style** used (where braces are placed)

Aligned:

```
public static void main(String[] args)
{
    System.out.println("Java");
}
```

End-of-line:

```
public static void main(String[] args) {
    System.out.println("Java");
}
```

Compiling and Executing a Java Program

A **software development environment** is software that helps you develop software

create, run, organize, modify, test, debug

There are many options, some free and some not

It's important to get comfortable with whatever development environment you use

There are only a few crucial tools you need initially – learn the rest over time

Two categories: command-line and integrated environments

Compiling and Executing a Java Program

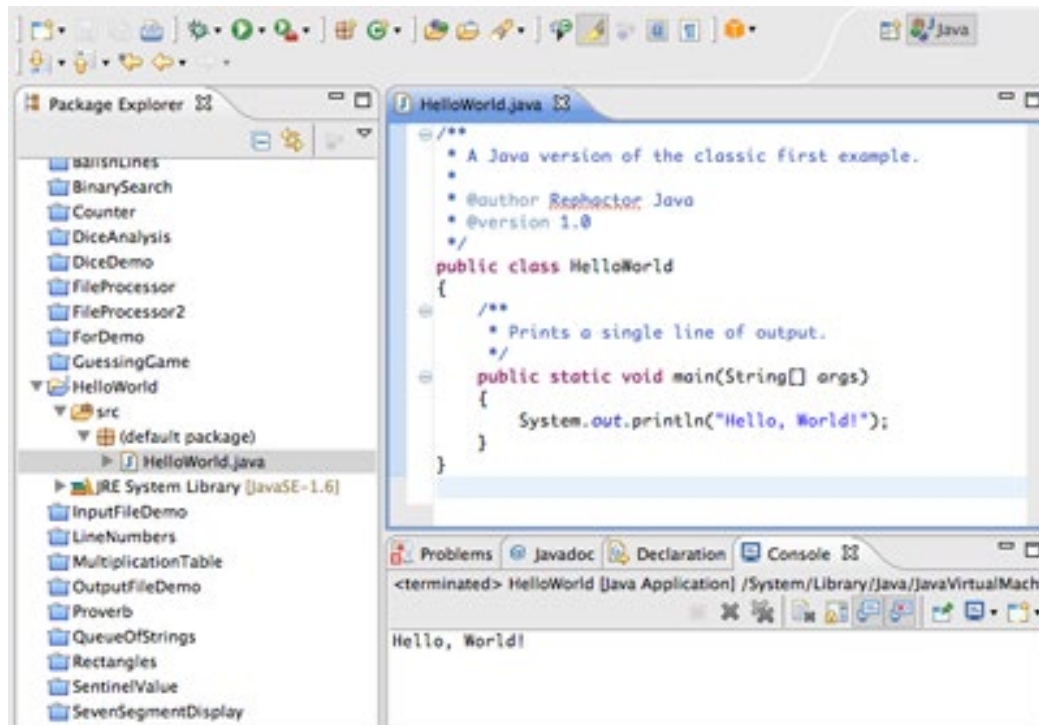
A [command-line environment](#) is a suite of separate tools executed as individual commands in a console window

```
examples >
examples > ls
HelloWorld.java
examples > javac HelloWorld.java
examples > ls
HelloWorld.class      HelloWorld.java
examples > java HelloWorld
Hello, World!
examples >
examples >
```

The [Java Development Kit \(JDK\)](#) from Oracle is a free command-line environment

Compiling and Executing a Java Program

An **integrated development environment** is one large program that combines the various tools



This one is
Eclipse

Others

BlueJ

DrJava

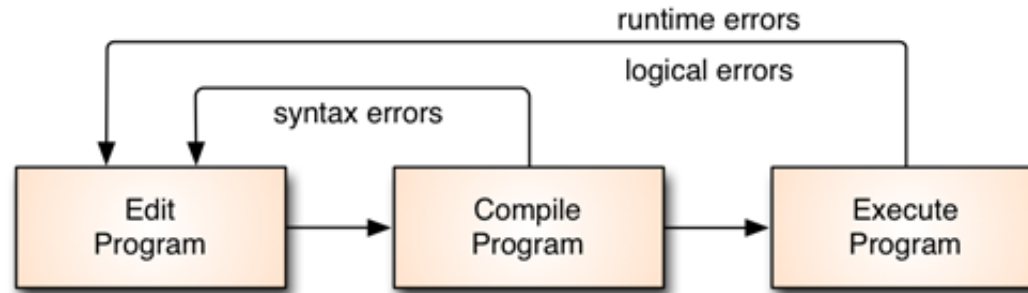
JEdit

jGRASP

NetBeans

Compiling and Executing a Java Program

No matter what environment you use, there are common activities:



editor – used to write and modify Java **source code**

compiler – checks for errors and translates the program into an executable version (**bytecode**)

interpreter – runs the program on the **Java Virtual Machine (JVM)**, a conceptual platform

Programming Errors

Natural languages, like English or Spanish, are often ambiguous

I'll ride my bike tomorrow if it looks nice in the morning.

I saw her duck.

Time flies like an arrow.

Fruit flies like a banana.

A sentence in a natural language can have many meanings

So, we use **programming languages** to write computer instructions

A programming language statement can only be interpreted one way

Programming Errors

The key is the relationship between two language aspects:

syntax – the rules that determine how words and symbols can be combined

semantics – the meaning of a syntactic element

An English sentence can be syntactically valid and have multiple semantic interpretations

In a programming language, *syntax determines semantics*

If a programming statement is syntactically valid, it has only one interpretation

Programming Errors

Programming errors occur when we violate the language syntax rules or when the semantics we set up are not what we intended

There are three types of programming errors:

syntax errors

runtime errors

logic errors

Errors happen. They are a part of programming. Don't sweat it. Just learn how to deal with them.

Programming Errors

The **compiler** analyzes a program and reports any **syntax errors**

```
public class Mistakes
{
    pulbic static void main(String[] args)
    {
        System.out.println("No mistakes, just lessons.)
    }
}
```

public
spelled
wrong

missing
semicolon

missing
quote
mark

Programming Errors

Syntax errors are reported in various ways depending on the development environment

```
Mistakes.java:3: <identifier> expected
    pulbic static void main(String[] args)
      ^
Mistakes.java:5: unclosed string literal
    System.out.println("No mistakes, just lessons.)
                        ^
Mistakes.java:5: ';' expected
    System.out.println("No mistakes, just lessons.)
                                                                ^
Mistakes.java:7: reached end of file while parsing
}
^
4 errors
```

Programming Errors

A **runtime error** occurs when an operation can't be carried out for some reason

The program terminates (crashes) immediately

In Java, a runtime error is usually represented by an **exception**

```
System.out.println(123 / 0)
```

```
Exception in thread "main" java.lang.ArithmeticException: / by zero  
at DivisionAttempt.main(DivisionAttempt.java:5)
```

line number

Programming Errors

A **logic error** occurs when the program doesn't produce the desired results

```
System.out.println("The perimeter of a square with " +  
    "3 inch sides is ");  
System.out.println(4 * 4 + "feet.");
```

The perimeter of a square with 3 inch sides is
16feet

perimeter is spelled wrong

wrong answer (16)

wrong output units (feet)

missing space between 16 and feet

Programming Errors

A **programming specification** (spec) is a set of requirements that must be satisfied by a program

A logic error may also occur if a program violates a requirement of its specification

For example, the program spec of the perimeter program might indicate that the output should be on one line

Unlike an incorrect answer, violating program spec may not be obvious

Compiling and Executing a Java Program

Bytecode is a "low-level" version of a Java program – it's not something you edit directly

Bytecode is not associated with any particular type of computer

That's what makes Java **platform-independent**

The Java Virtual Machine is implemented in software designed to execute the bytecode

A Java program can be run on any computer with a JVM

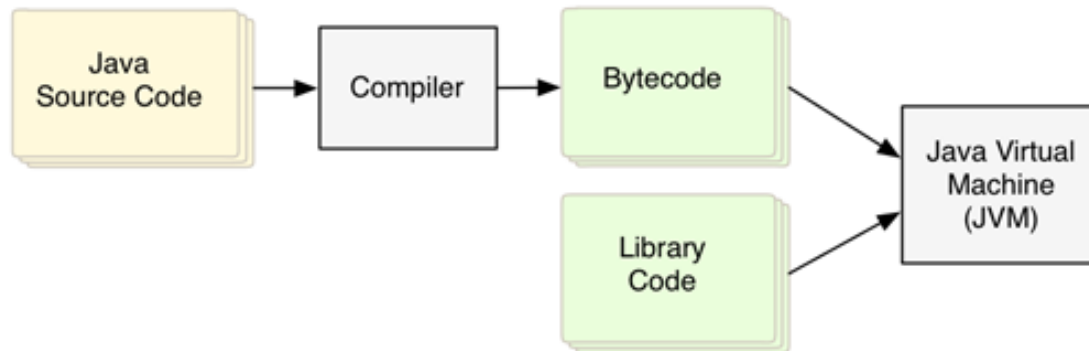
Java's original slogan:

Write once, run anywhere.

Compiling and Executing a Java Program

Java source code is stored in files with a .java extension (HelloWorld.java)

The compiler produces bytecode files with a .class extension (HelloWorld.class)



The JVM will load any library code (also in bytecode) that is referenced by the program

Try Yourself

- Copy & paste or download **ThreeErrors.java** (below)
- Run it in **jGRASP**
- How many **errors** can you find? syntax, logic, runtime

```
public class ThreeErrors
{
    public static void main (String[] args)
    {
        int value = 3; // original value
        double doubleValue = value / 2.0;

        System.out.println("original value: " + value);
        System.out.println("double value: " + duobleValue);
        System.out.println("infinity: " + (value / 0));
    }
}
```