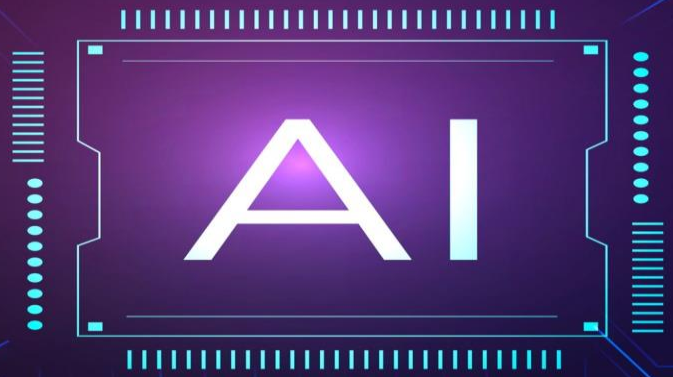


AP Computer Science A

Java Programming Essentials [Ver. 3.0]

Unit 1: Elementary Programming

CHAPTER 1: INTRODUCTION

DR. ERIC CHOU IEEE SENIOR MEMBER

Objectives

- Computer Hardware and Software
- Java Programming Languages
- Interpretation Levels
- Java Knowledge
- Goals of AP Computer Science A

Computer Hardware and Software

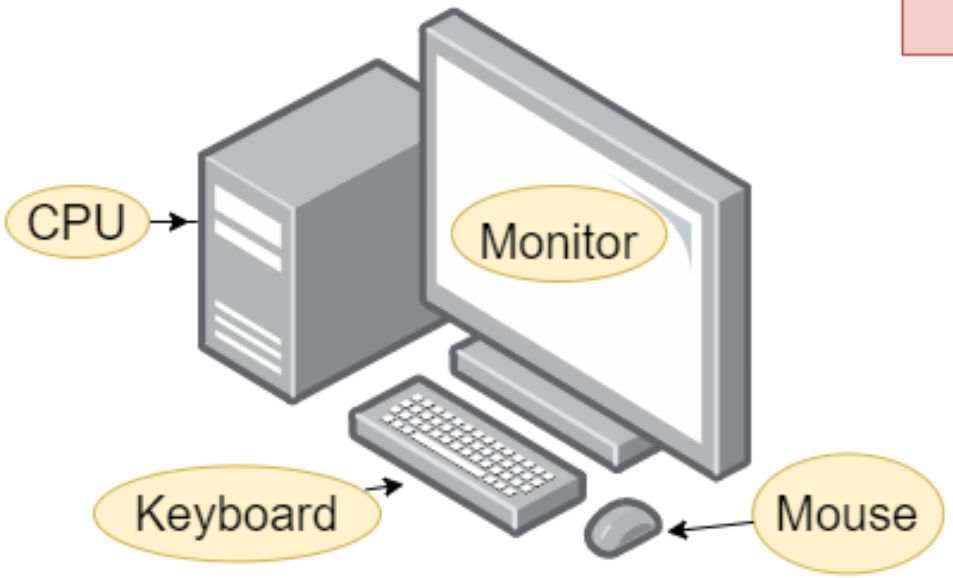
LECTURE 1

Computer Hardware and Software

- What is a computer?
- Personal Computer Hardware - Computer System Architecture
- Software - Operation System - BIOS
- Computer Science and Computer Engineering
- Video: Hidden Figure - IBM 4010, IBM 360

What is Computer

Input → Process → Output



Computer Stands for

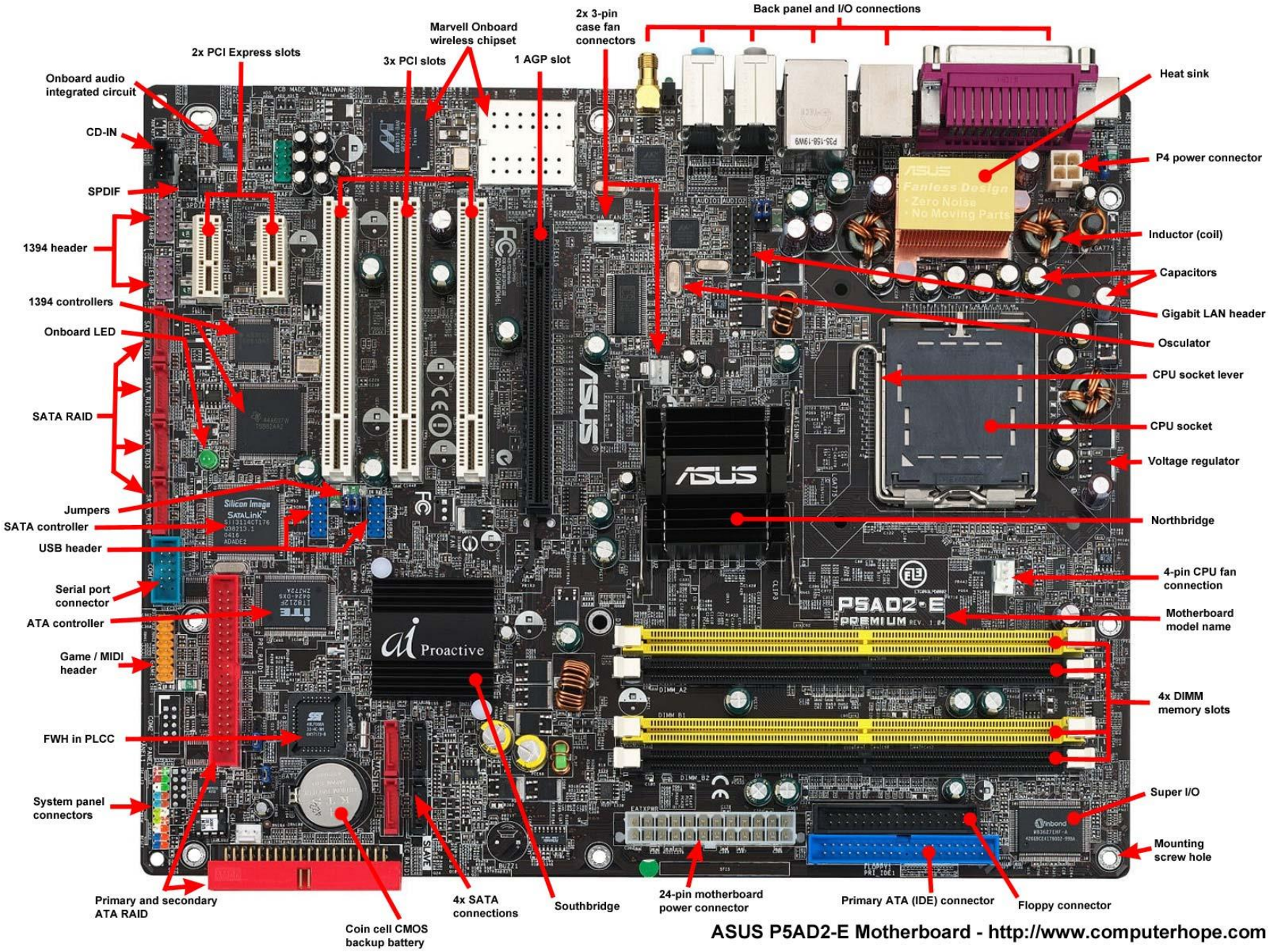
C: Common
O: Operating
M: Machine
P: Purposely
U: Used for
T: Technological
E: Educational
R: Research

IBM®


DELL

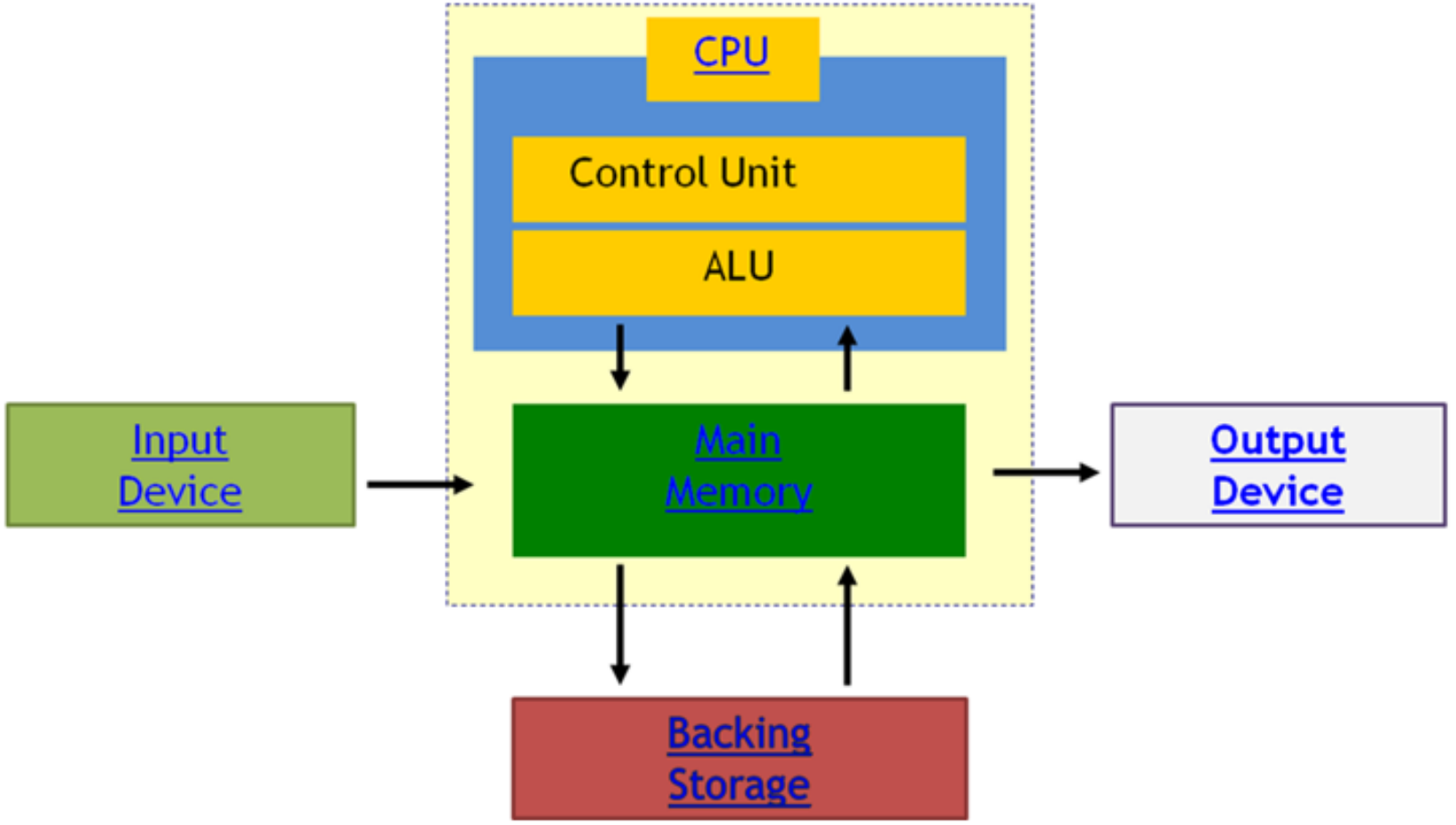

SMS

amazon
web services



Computer System Diagram

Hardware View



Input Devices of Computer

Touch screen

Camera

Scanner

Microphone

Joystick

Keyboard

Mouse

Web cam

Track ball

Output Devices of Computer

SPEAKER

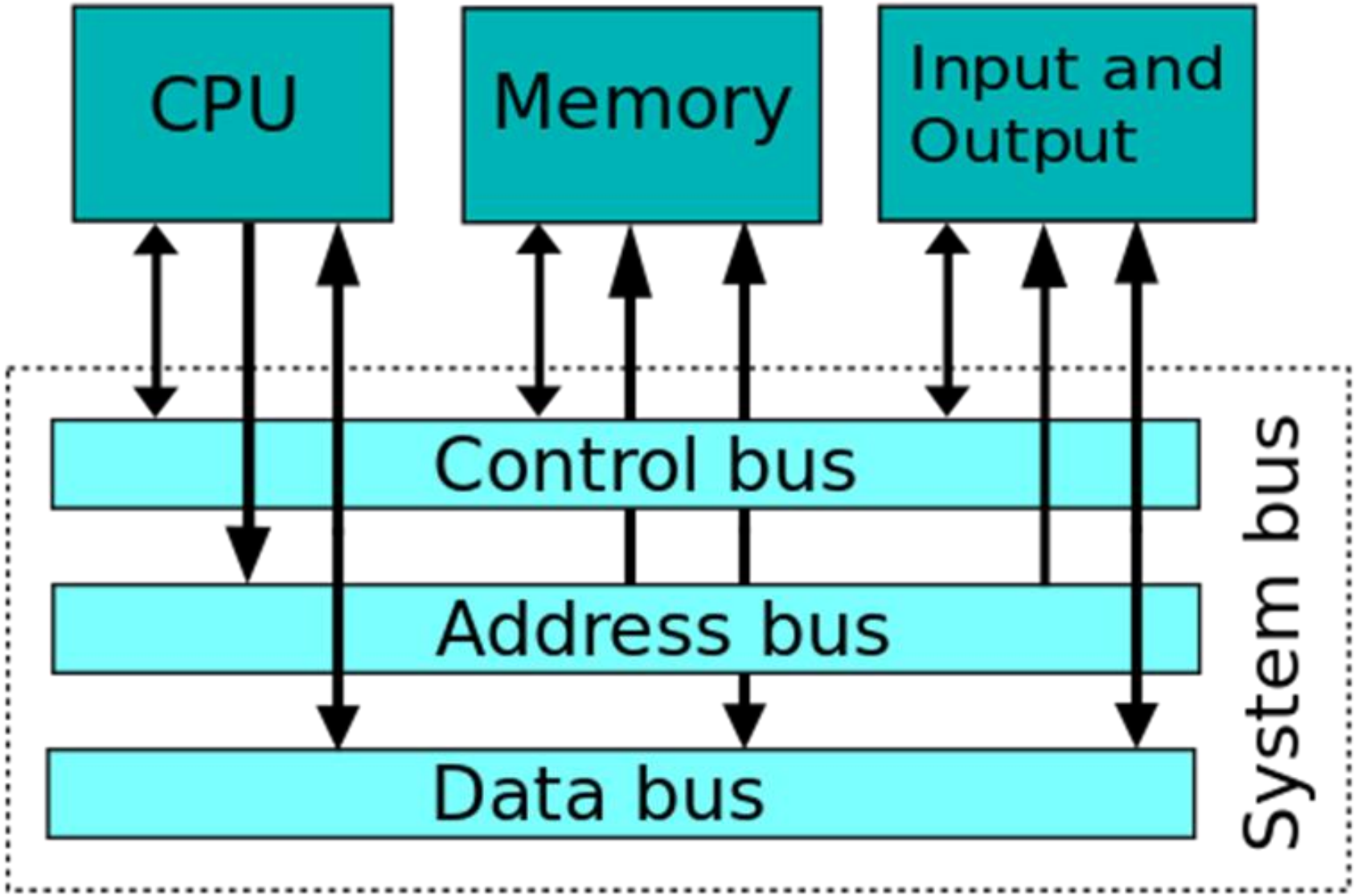
MONITOR

HEADPHONE

PLOTTER

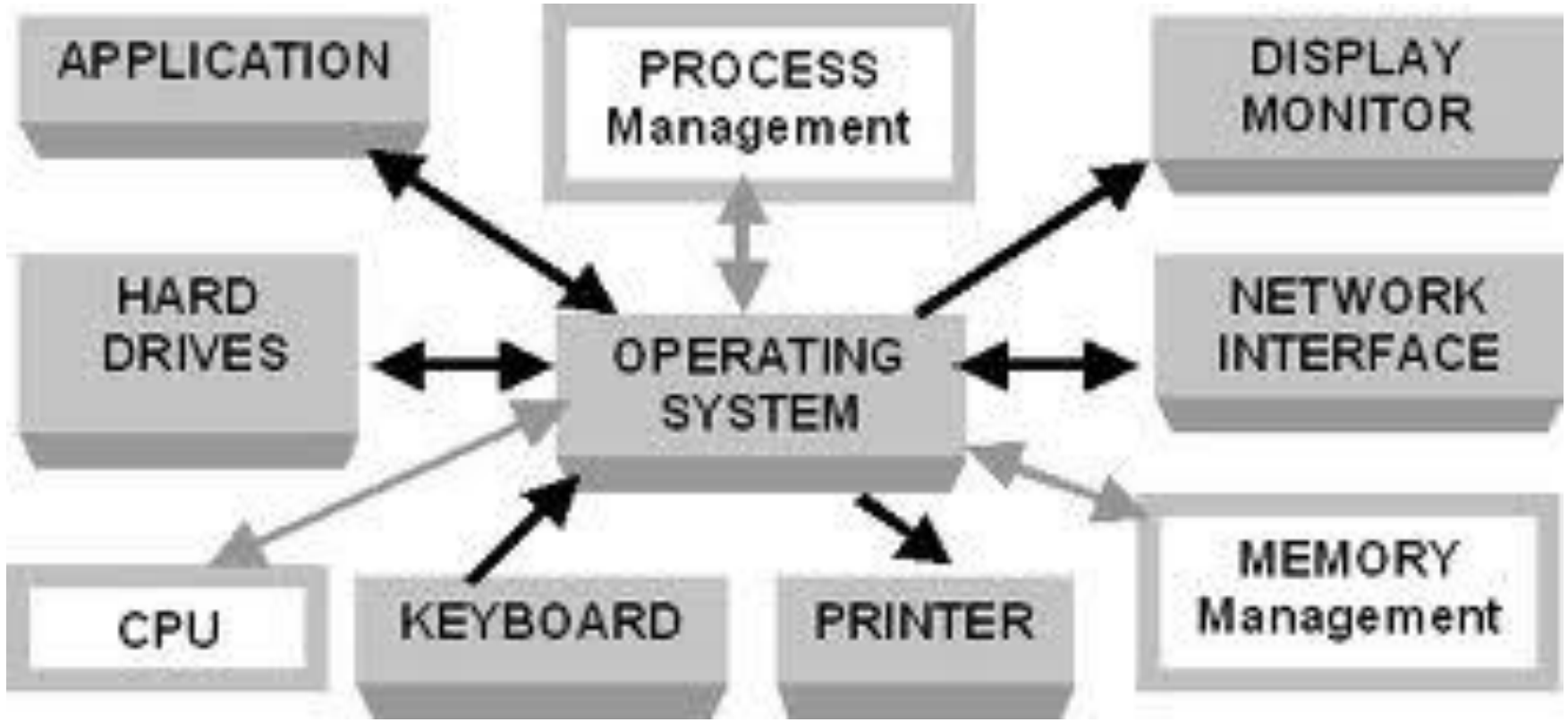
PROJECTOR

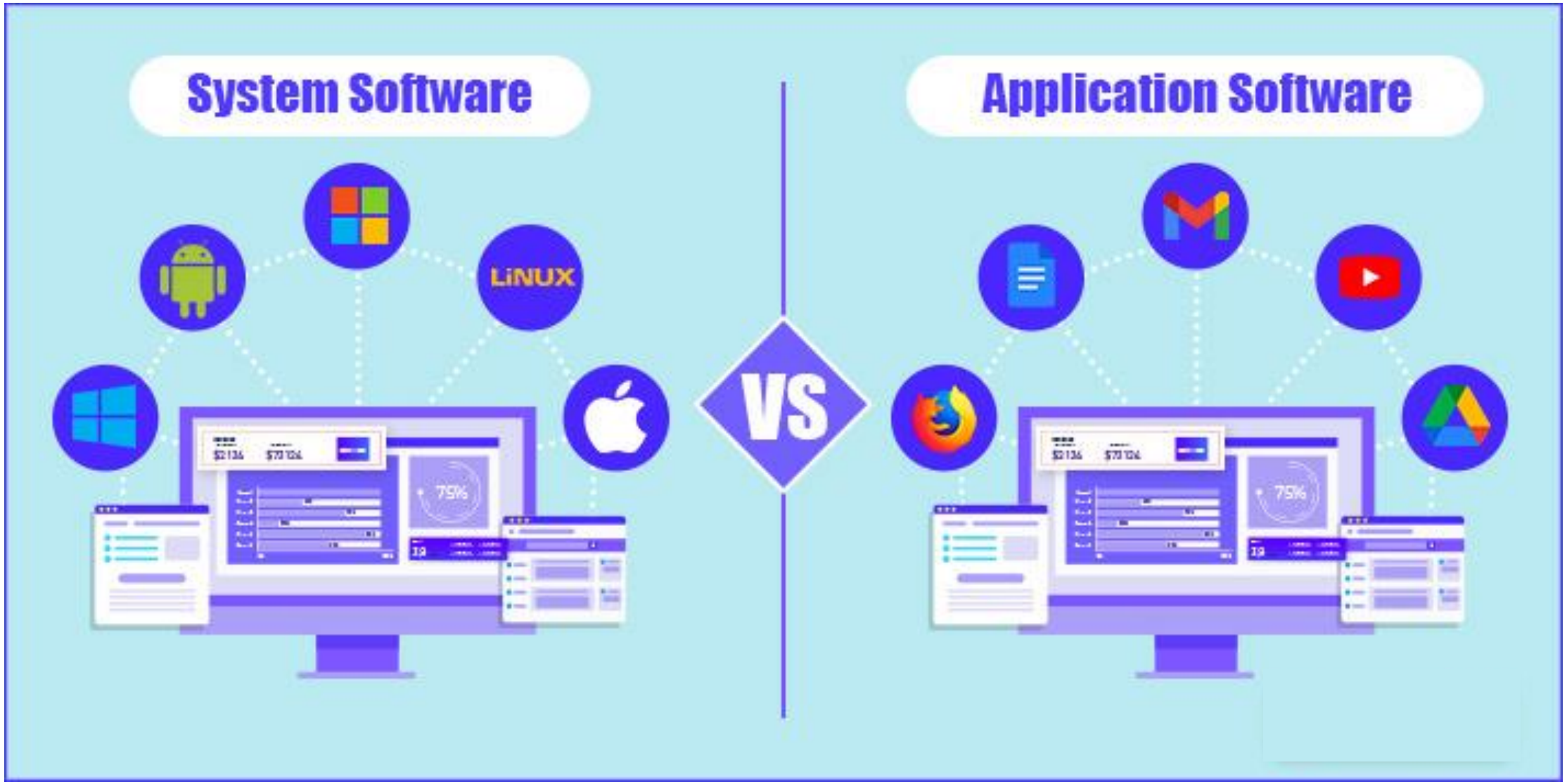
PRINTER


CPU
Memory
Input and Output
Control bus
Address bus
Data bus
System bus

Computer System Diagram

(Operation System/Software View)




System Software
LiNUX
VS
Application Software

Java Programming Languages

LECTURE 2

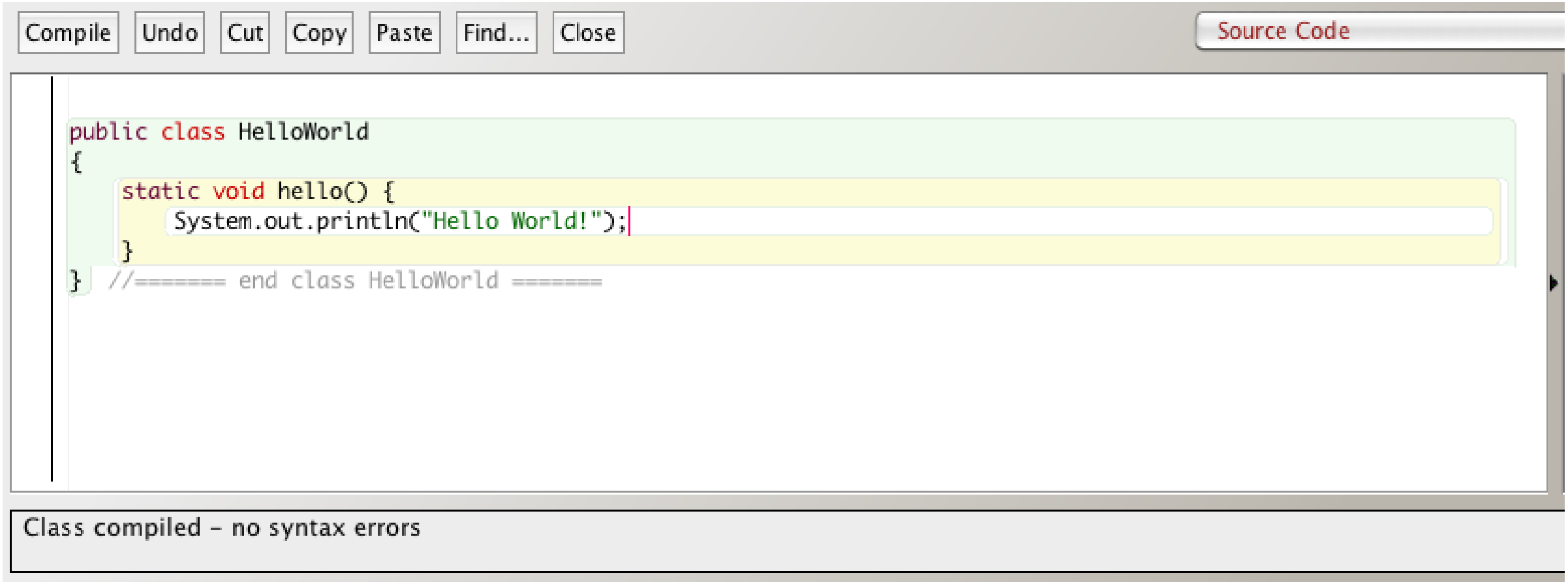

django
python
php
maven
spring
Java
Gradle
iOS
C#
ASP.NET
Ruby
SQL
mongo
C++
jQuery
JS
npm
ember
Scala

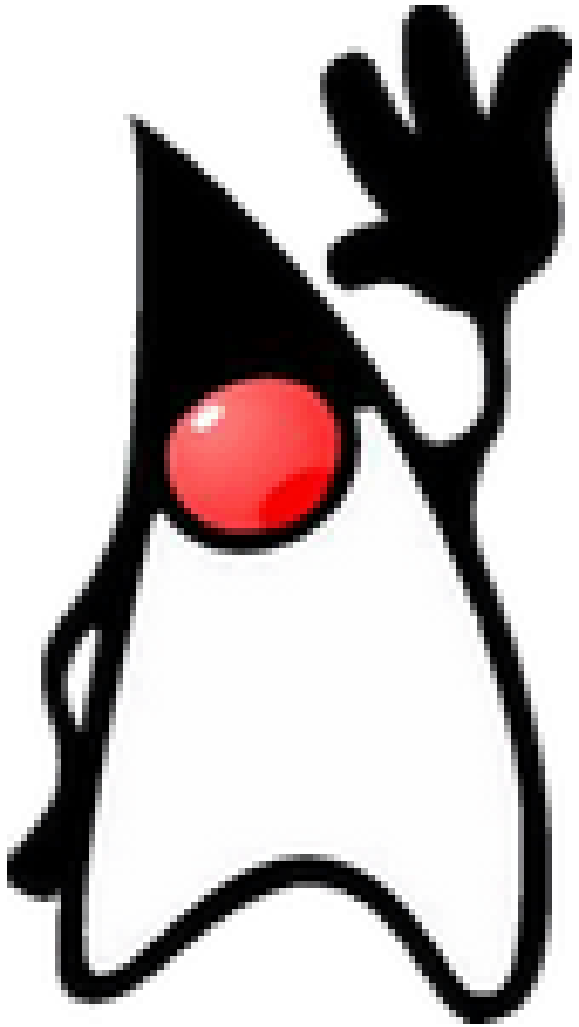
Java Programming Languages

- First Program: HelloWorld!
- Programming Languages
- IDE: BlueJ, DrJava, IntelliJ IDEA, Eclipse
- The Java Compilation Flow
- Command line compilation
- Java Virtual Machine
- Java APIs, Android O.S. and Java

Demonstration Program

HELLOWORLD.JAVA

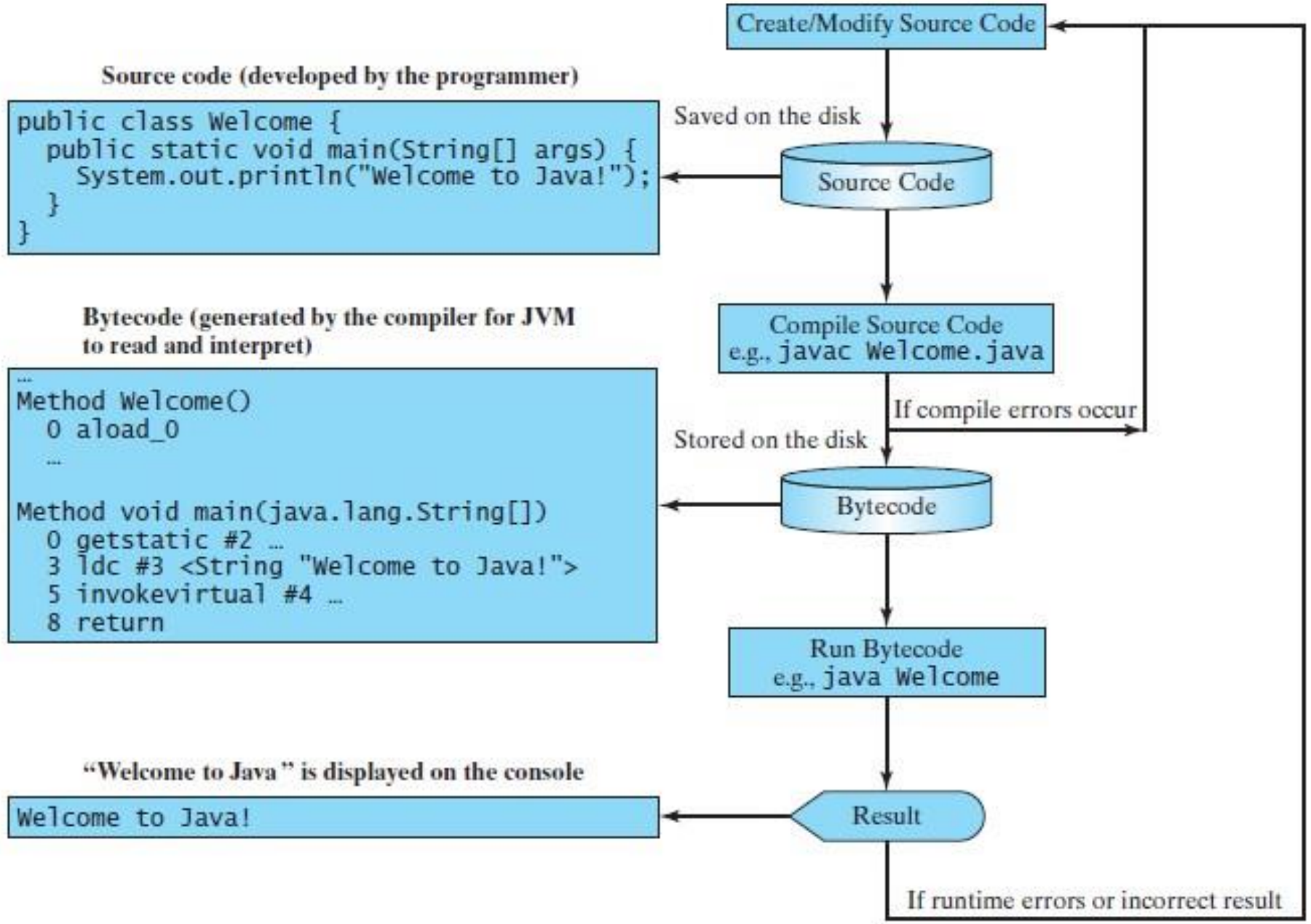

Compile
Undo
Cut
Copy
Paste
Find...
Close
Source Code
public class HelloWorld
{
static void hello() {
System.out.println("Hello World!");
}
} //======= end class HelloWorld =======
Class compiled - no syntax errors

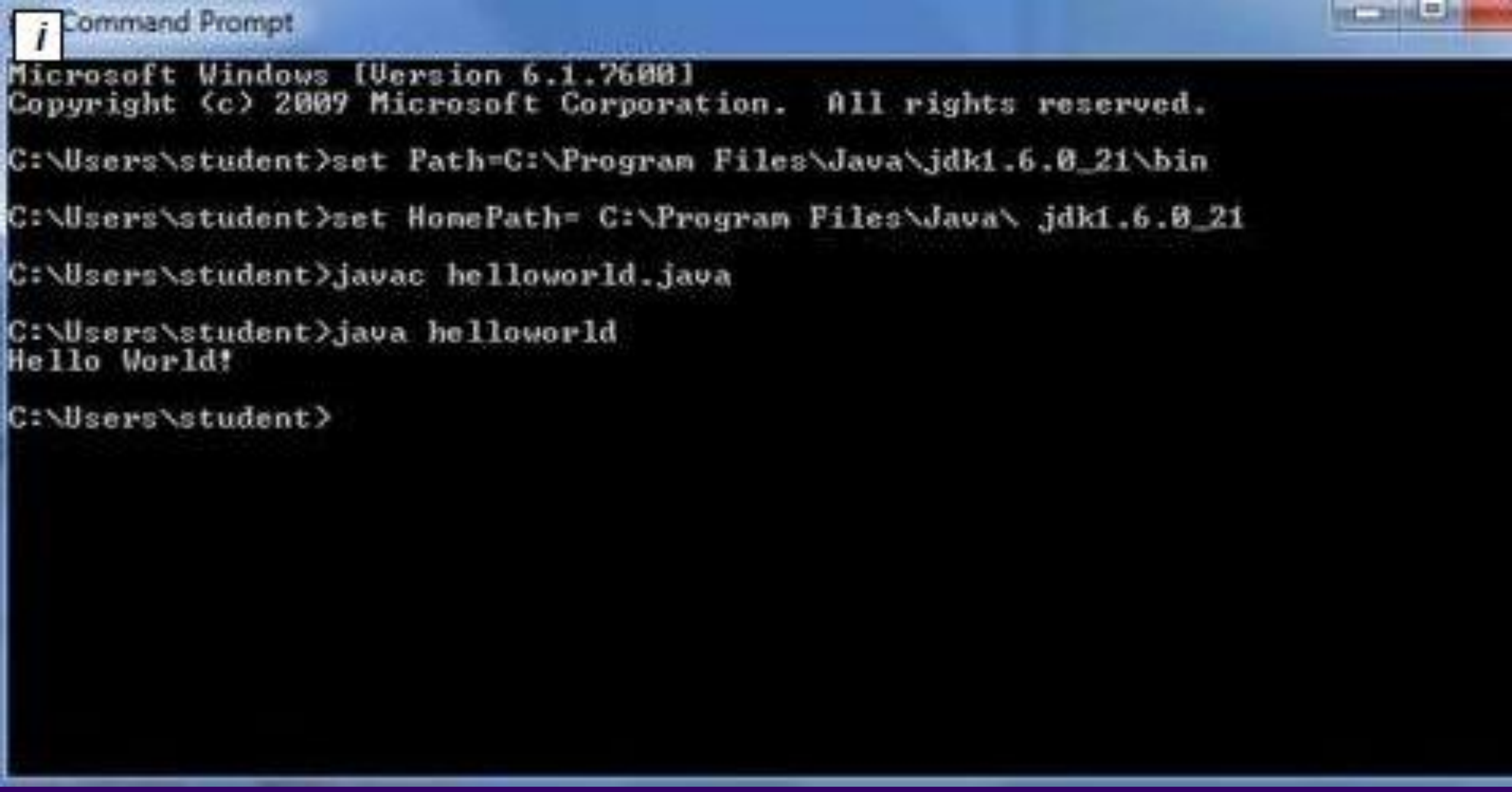


Java 20

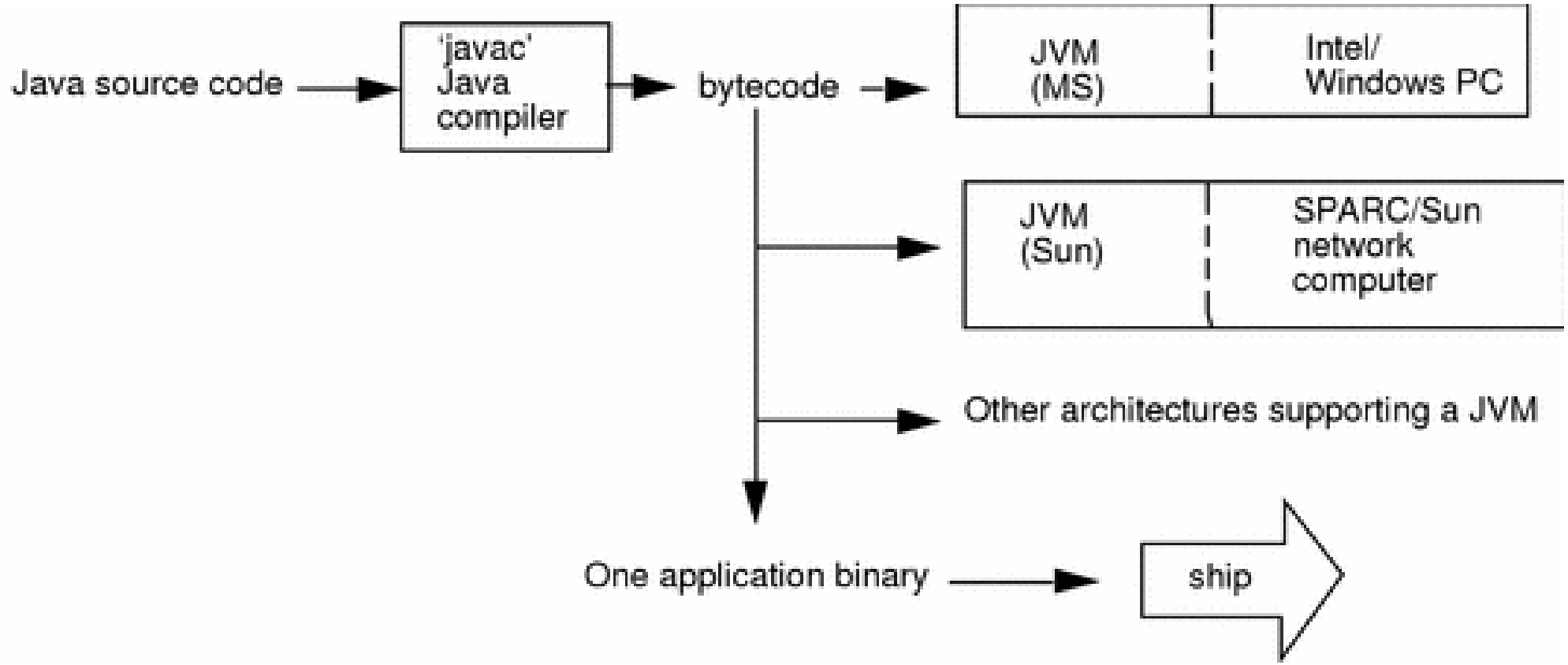
What's your favorite IDE for Java Development?

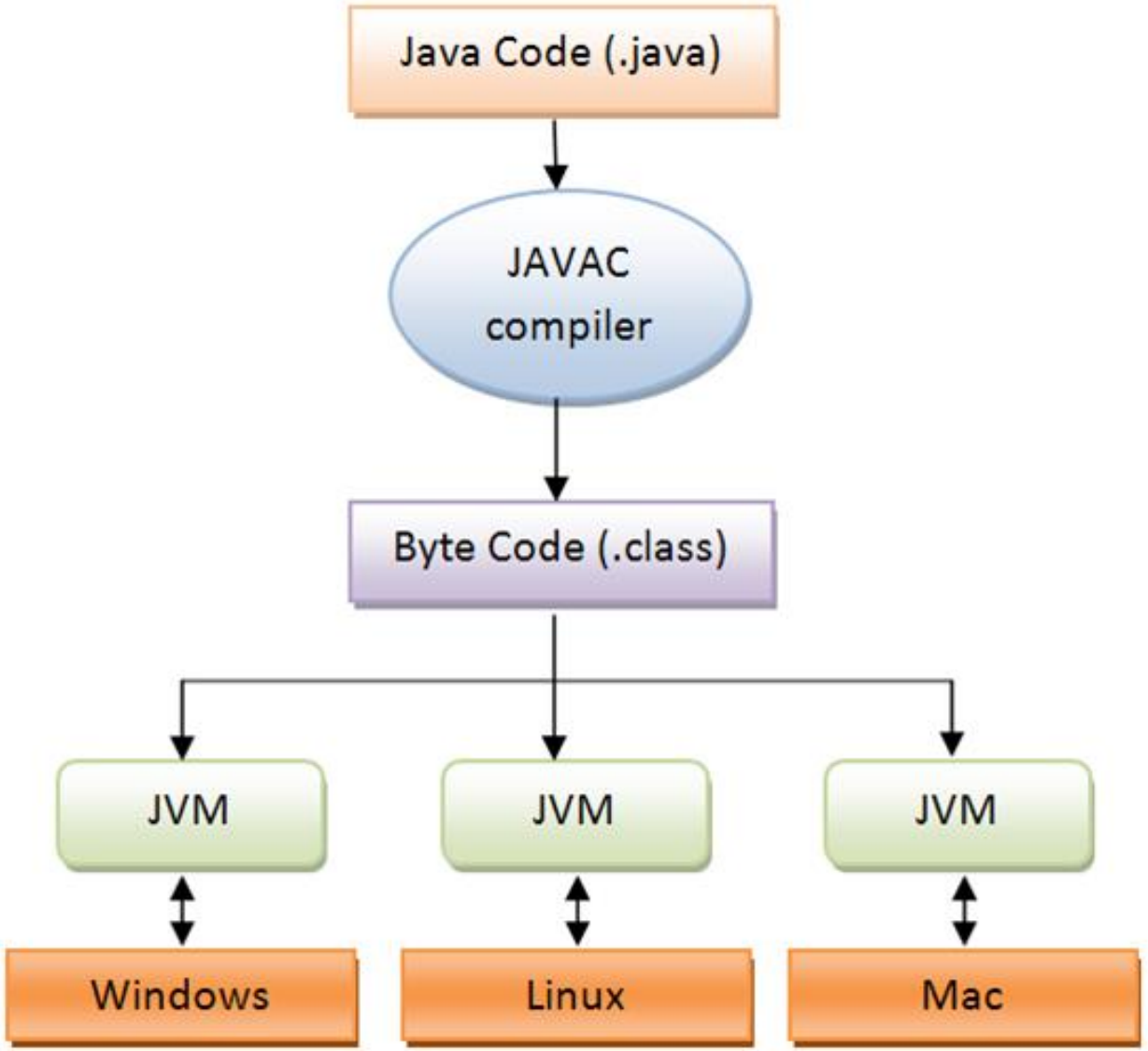
- Eclipse
- IntelliJ IDEA
- NetBeans
- BlueJ
- JDeveloper
- DrJava
- Android Studio
- Other


Create/Modify Source Code
Source code (developed by the programmer)
public class Welcome {
public static void main(String[] args) {
System.out.println("Welcome to Java!");
}
}
Saved on the disk
Source Code
Compile Source Code
e.g., javac Welcome.java
If compile errors occur
Bytecode (generated by the compiler for JVM to read and interpret)
...
Method Welcome()
0 aload_0
...
Method void main(java.lang.String[])
0 getstatic #2 ...
3 ldc #3 <String "Welcome to Java!">
5 invokevirtual #4 ...
8 return
Stored on the disk
Bytecode
Run Bytecode
e.g., java Welcome
"Welcome to Java" is displayed on the console
Welcome to Java!
Result
If runtime errors or incorrect result

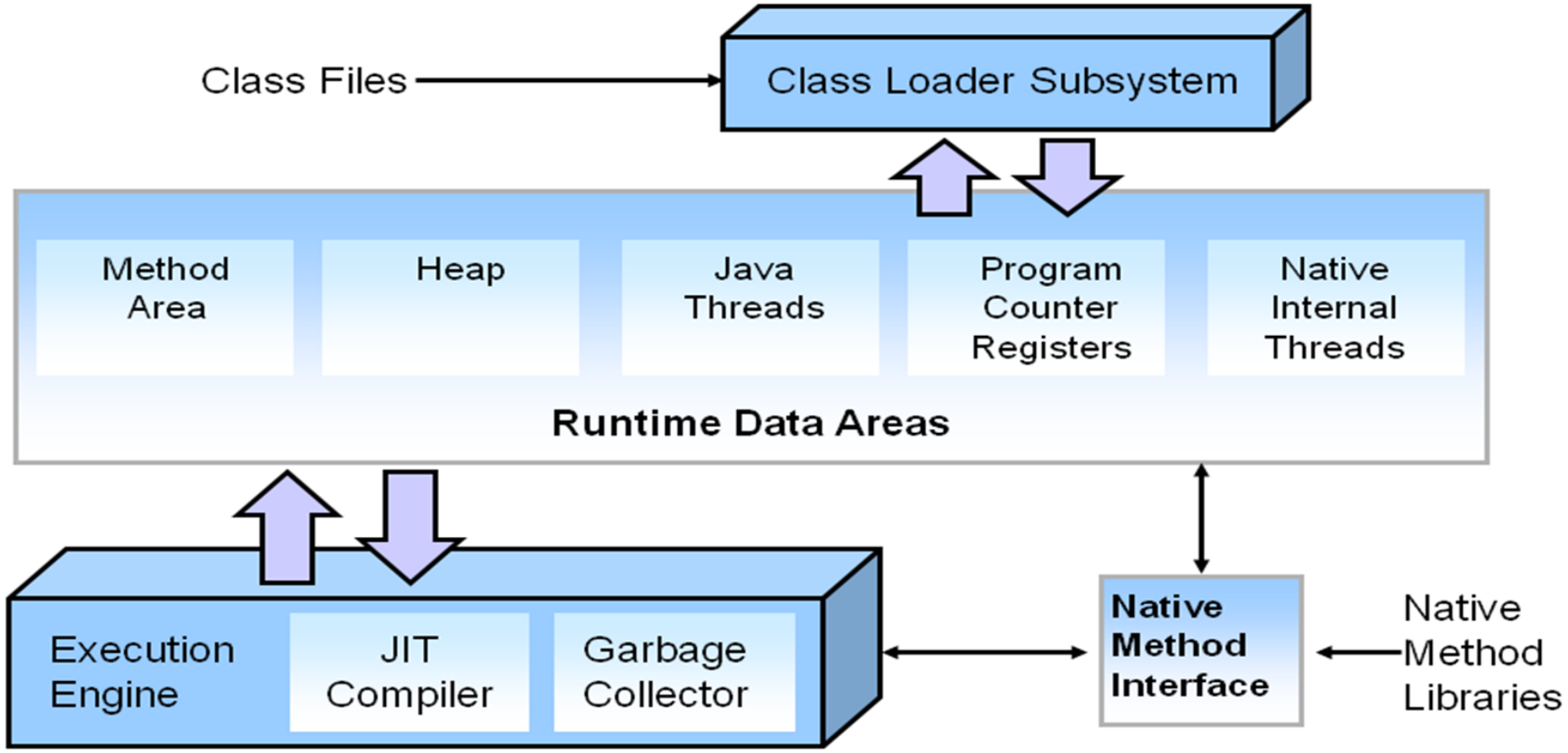


Cross-platform by JVM and bytecode




Java Code (.java)
JAVAC compiler
Byte Code (.class)
JVM
JVM
JVM
Windows
Linux
Mac

HotSpot JVM: Architecture



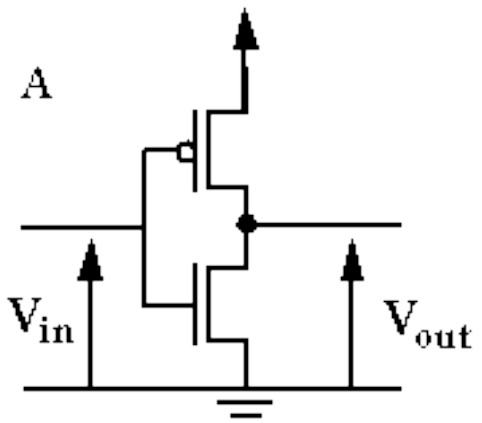
Interpretation Levels

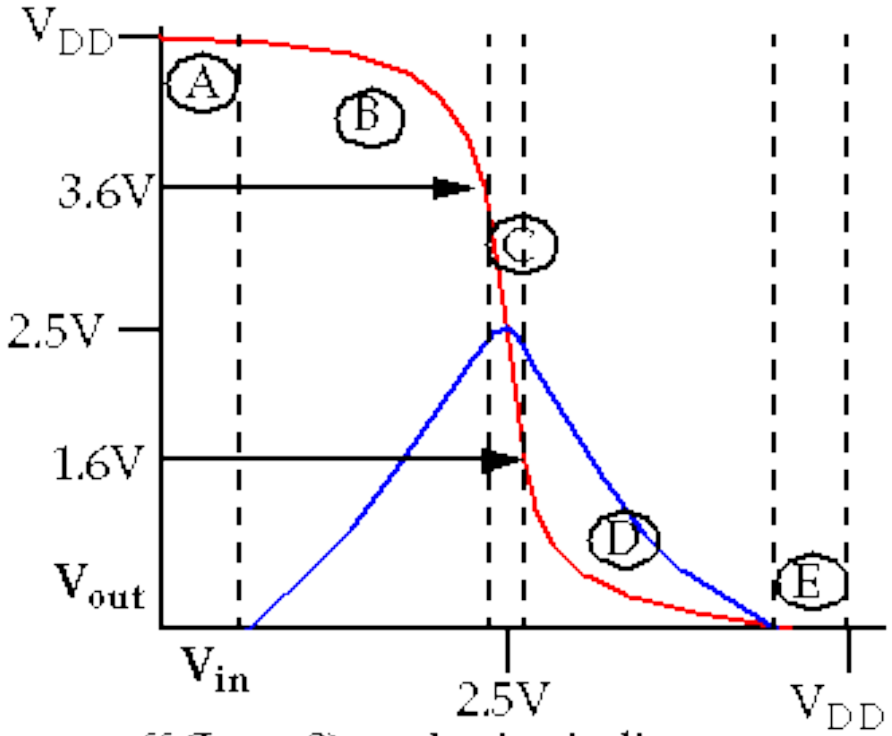
LECTURE 3

Interpretation Levels

- Compiler versus Interpreter
- Machine Code/Assembly/High Level Languages
- Electronics and H/L for binary bits

Binary Digit (Bit)	Electronic Charge	Electronic System
1		ON
0		OFF





(A) $0 \le V_{in} \le V_{tn}$;n-device is cut off (I_{dsn}=0), p-device in linear.

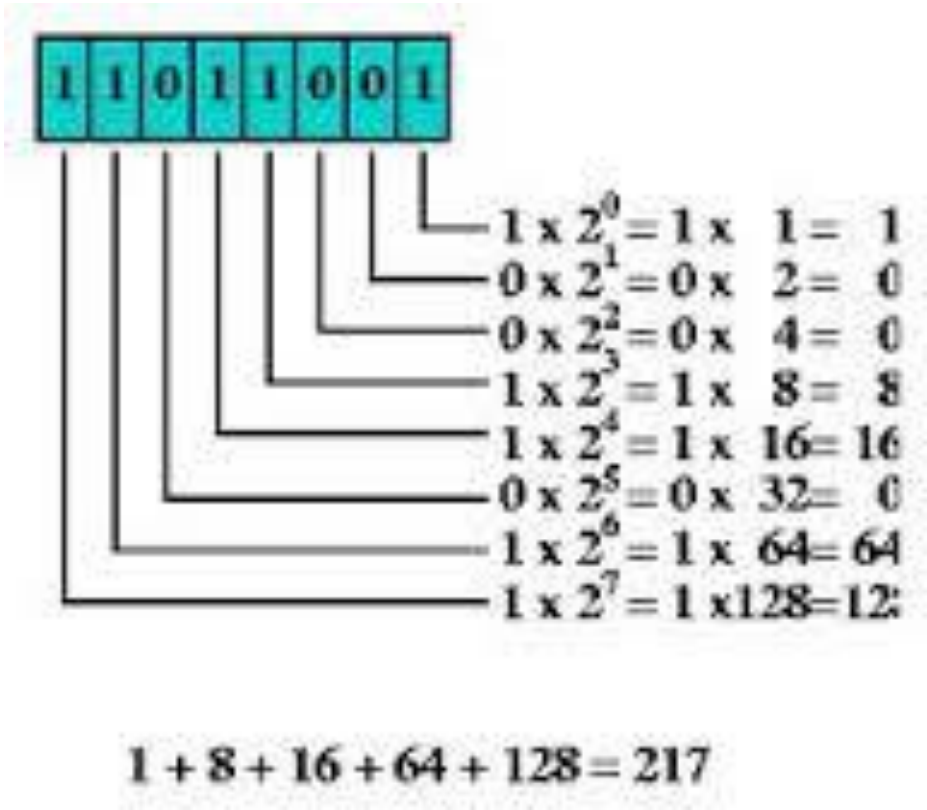
(B) $V_{tn} < V_{in} < V_{DD}/2 - \Delta$;n-device is in sat., p-device in linear.

(C) $V_{DD}/2 - \Delta \le V_{in} \le V_{DD}/2 + \Delta$;n-device is in sat., p-device in sat.

(D) $V_{DD}/2 + \Delta < V_{in} < V_{DD} + V_{tp}$;n-device is in linear, p-device in sat.

(E) $V_{DD} + V_{tp} \le V_{in} \le V_{DD}$;n-device is in linear, p-device in cut off (I_{dsp}=0).

Binary Number System



$$1 + 8 + 16 + 64 + 128 = 217$$

FASTER CPU

NEW 5-CORE GPU

NEW DISPLAY ENGINE

FASTER NEURAL ENGINE

ProRes

encode and decode

Thunderbolt 4

Secure Enclave

Support for two external displays

Up to

32GB

Unified memory

33.7 billion

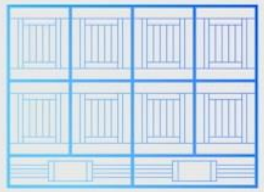
Transistors

M1 PRO

16-core

Neural Engine

11 trillion operations per second



Up to

10-core

CPU

Up to

16-core

GPU

Industry-leading performance per watt

5 nm process

200GB/s

Memory bandwidth

ProRes

encode and decode

Thunderbolt 4

Secure Enclave

Support for four external displays

64GB

Unified memory

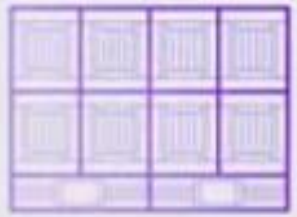
57 billion

Transistors

16-core

Neural Engine

11 trillion operations per second



10-core

CPU

Up to

32-core

GPU

Industry-leading performance per watt

5 nm process

400GB/s

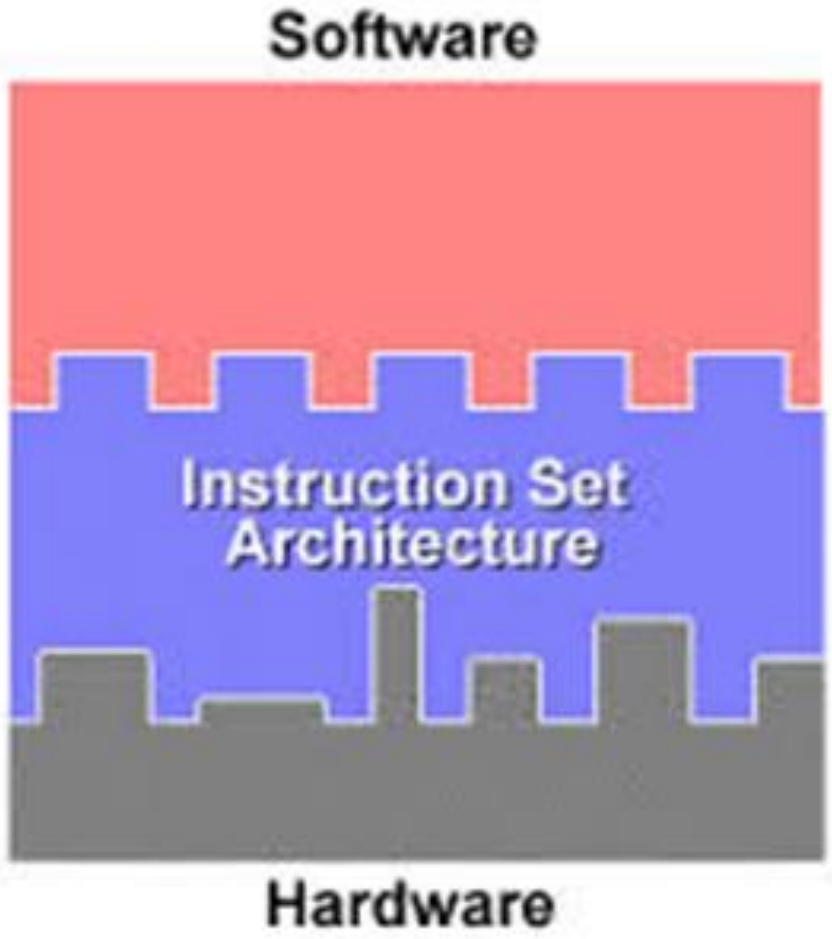
Memory bandwidth

Programming Languages

Machine Language Assembly Language High-Level Language

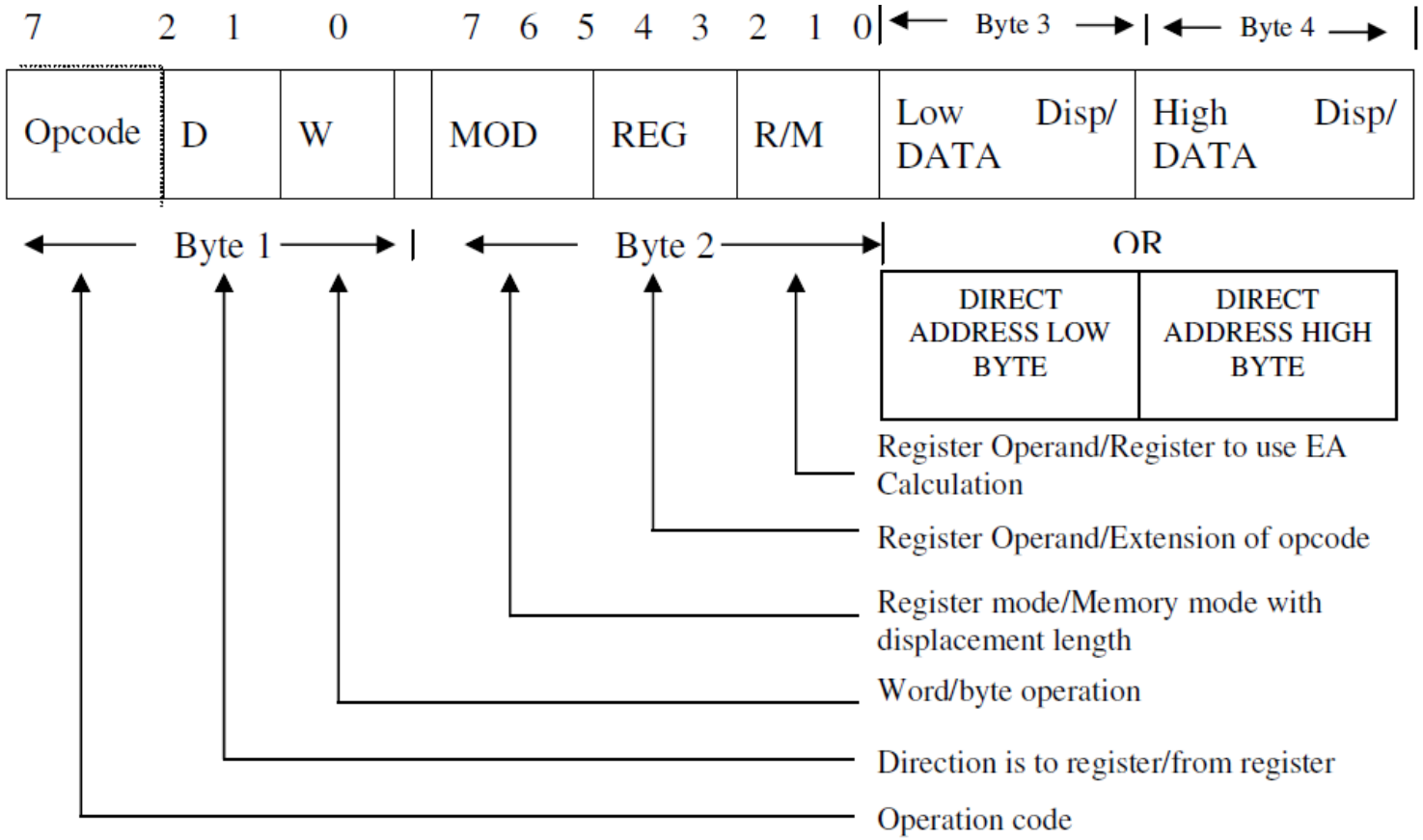
Machine language is a set of primitive instructions built into every computer. The instructions are in the form of binary code, so you have to enter binary codes for various instructions. Program with native machine language is a tedious process. Moreover the programs are highly difficult to read and modify. For example, to add two numbers, you might write an instruction in binary like this:

```
1101101010011010
```


Software
Instruction Set
Architecture
Hardware

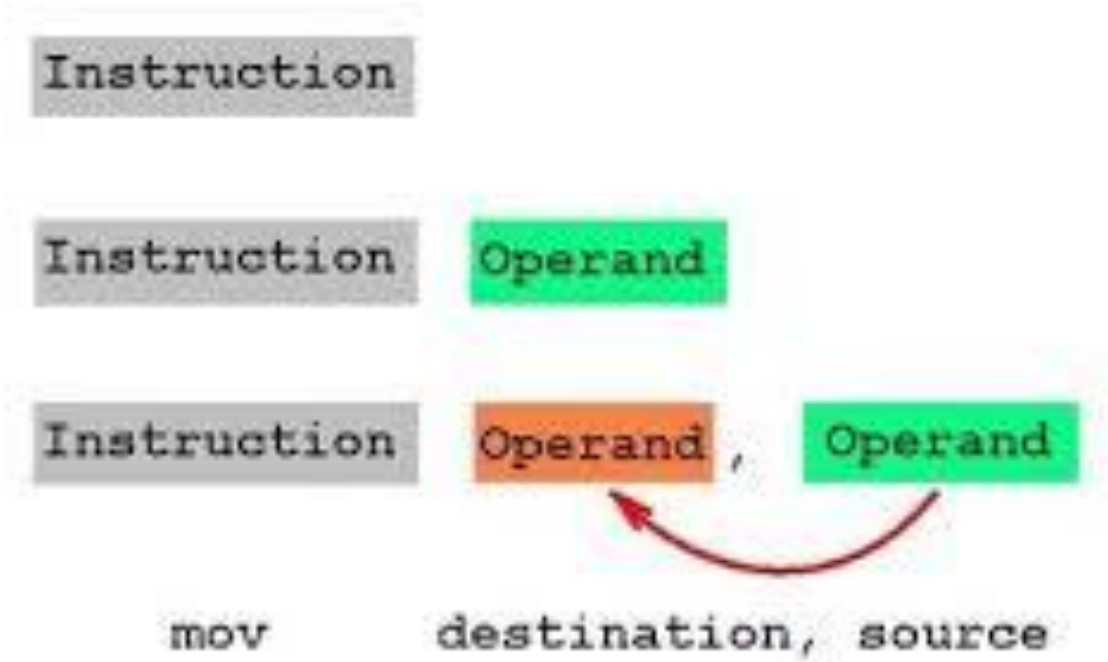
Machine Code (Instructions)

The whole legal collection of instructions is called instruction set



Machine Code (Instructions)

The whole legal collection of instructions is called instruction set

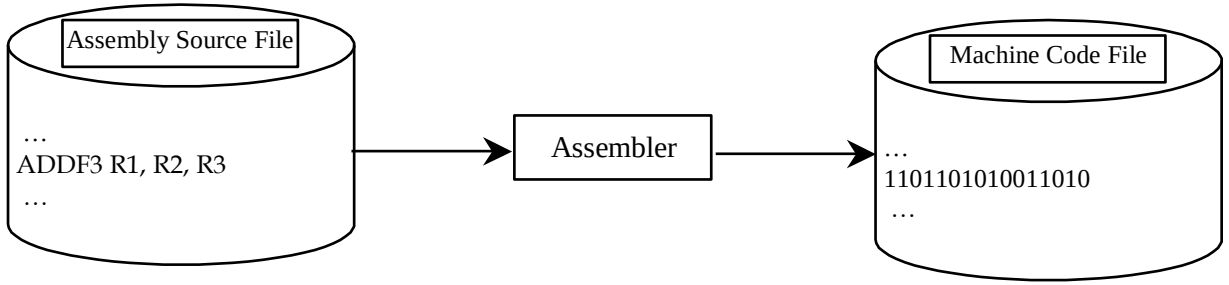


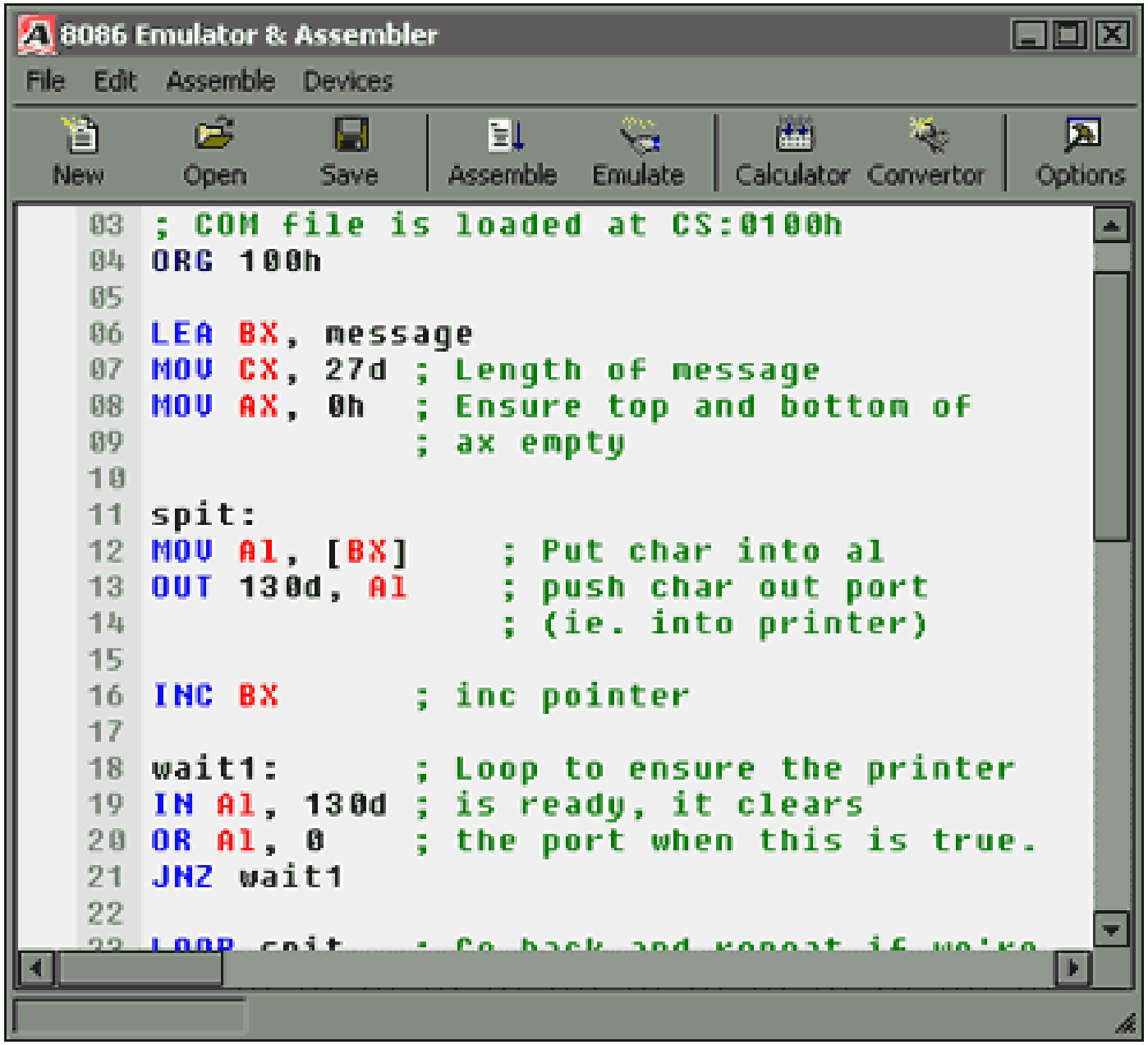
Programming Languages

Machine Language **Assembly Language** High-Level Language

Assembly languages were developed to make programming easy. Since the computer cannot understand assembly language, however, a program called assembler is used to convert assembly language programs into machine code. For example, to add two numbers, you might write an instruction in assembly code like this:

ADDF3 R1, R2, R3




8086 Emulator & Assembler
File Edit Assemble Devices
New Open Save Assemble Emulate Calculator Convertor Options
; COM file is loaded at CS:0100h
ORG 100h
LEA BX, message
MOV CX, 27d ; Length of message
MOV AX, 0h ; Ensure top and bottom of
; ax empty
spit:
MOV Al, [BX] ; Put char into al
OUT 130d, Al ; push char out port
; (ie. into printer)
INC BX ; inc pointer
wait1: ; Loop to ensure the printer
IN Al, 130d ; is ready, it clears
OR Al, 0 ; the port when this is true.
JNZ wait1

8086 Assembly language.

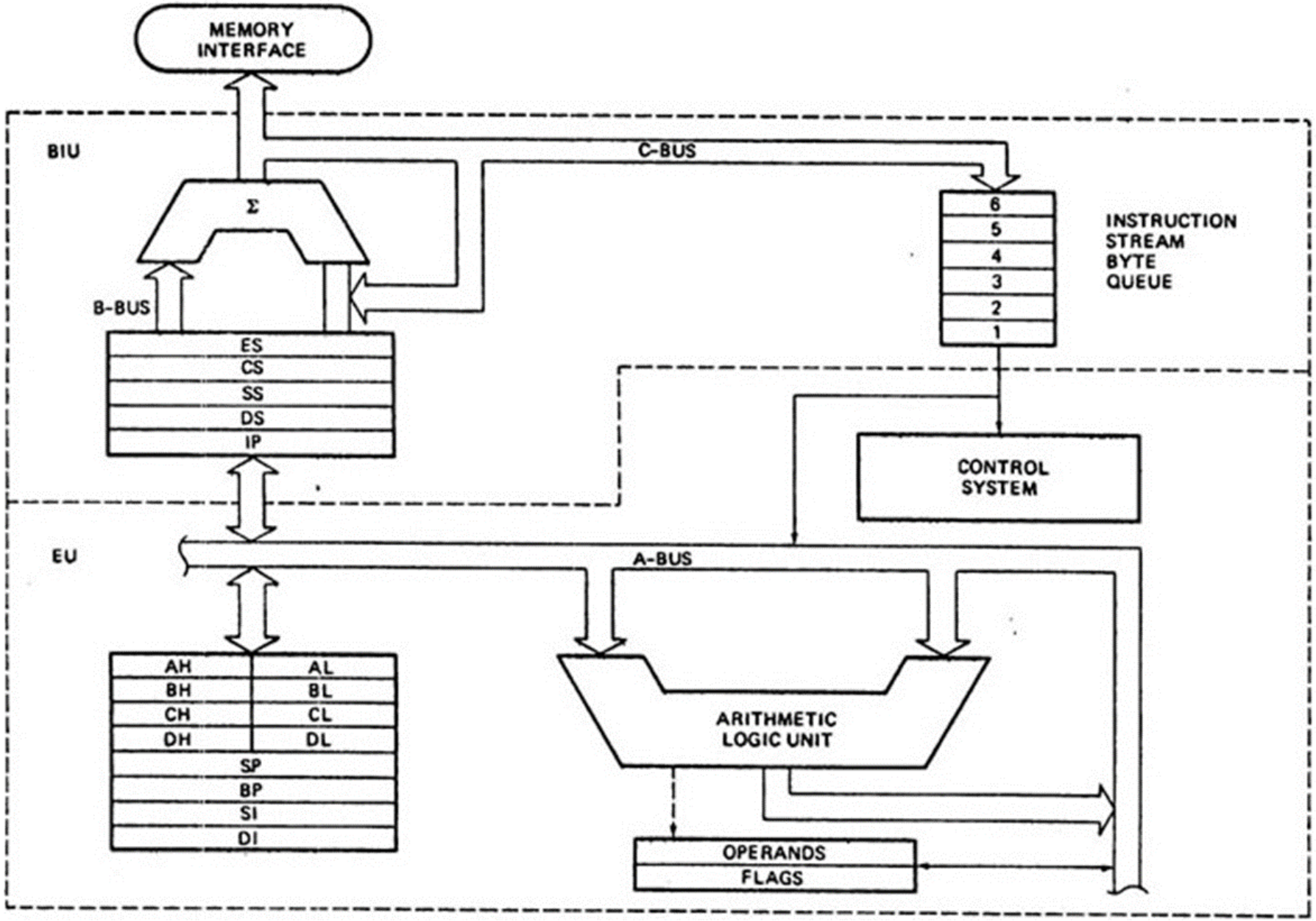
- The language that can be directly translated to machine code.
- LEA is an opcode. BX is a register name. Message is a pointer to a string.
- MOV is another opcode. CX is another register name. 27d is decimal 27.
- Assembly code is emulated by an emulator. And, there is debugger to help remove coding errors.
- Assembly code is used for device driver or O.S. Kernels.

Assembler and Debugger

Assembler is the software to convert Assembly Language into Machine code.

Emulator is to emulate the assembly code on real hardware to know about what will be the outcome.

Debugger is a software to show the error and contents for each registers and memory.



8086 Instruction Set Architecture

- Machine code is the real code for machine. It is use to control the Arithmetic and Logic Unit (ALU) and the register file and the memory.
- All programs are eventually executed in machine code.
- No one programs on Machine code.

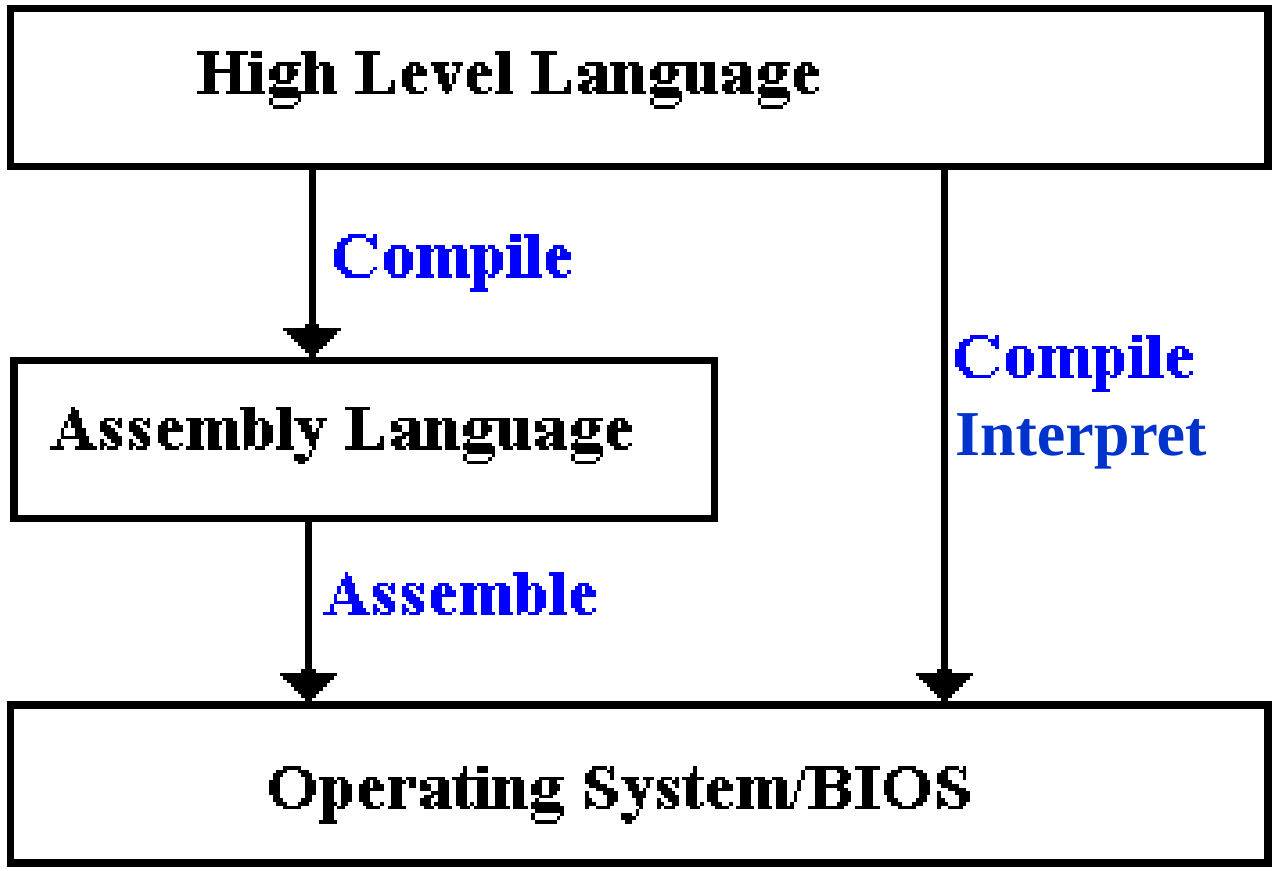
Programming Languages

Machine Language Assembly Language **High-Level Language**

The high-level languages are English-like and easy to learn and program. For example, the following is a high-level language statement that computes the area of a circle with radius 5:

```
area = 5 * 5 * 3.1415;
```

High Level Languages



Java Knowledge

LECTURE 4

Java Knowledge

- Comments
- Javadoc
- Java Subset
- Java Coding Habits
- Java Naming Conventions

Java Comments

Demonstration Program

ONEPLUSONE.JAVA

Program Structure

- Demonstrate basic Java program parts using OnePlusOne.java

Basic Program Structure:

```
public static void main(String[] args){
    // Input -------------------------
          // code for input here.
    // Processing --------------------
         // code for processing here.
    // Output ------------------------
        // code for output here.
}
```

Java comments and code block marks

Characters	Name	Description
//	Double Slash	Line comment
/* */	Slash star and star slash	Opening and closing of comment text
/** */	Slash double-star and star-slash	Opening and closing of Javadoc comment text Javadoc comment can be extracted into HTML file using the JDK's Javadoc command (Use to describe a module, a method or a variable)
{ }	Braces	For a code block.
[ ]	brackets	For the index variable
( )	parenthesis	For the boundary of an expression or a logic conditions
" "	double quotes	For boundary of a string of text data

Java Doc

What is JavaDoc?

- *Doc comments* (also known informally as *Javadoc comments*, although this technically violates trademark usage) document your APIs for other programmers who use your classes and for maintenance programmers.
- Doc comments standardize the way source code is documented.
- Documentation is kept in the source file, so it's easier for developers to keep it in sync with the code.
- You can document packages, classes, constructors, fields, and methods.

Java™ Platform, Standard Edition 8 API Specification

This document is the API specification for the Java™ Platform, Standard Edition.

See: Description

Profiles

- compact1
- compact2
- compact3

Packages

Package	Description
java.applet	Provides the classes necessary to create an applet and the classes an applet uses to communicate with its

Defining JavaDoc

- Use **/** */** comments right before the entities that are to be documented.
- You can have whitespace between the doc comment and the declaration, but no other code. For example, don't put import statements between your doc comment and a class declaration.
- If a doc comment line begins with a * preceded by optional whitespace, those characters are ignored.
- As of Java 1.4, leading whitespace is preserved if a line does not begin with a * character. This allows you to include formatted code fragments (wrapped with HTML <pre> tags) in your documentation.

Defining JavaDoc

- A doc comment consists of an optional main description followed by an optional tag section.
- **A doc comment can contain HTML markup**, but keep it simple (as in, keep it simple).
- The first sentence of the main description (ending in a period followed by a space, tab, line terminator, or first block tag) is used as the summary description for the declared entity.

Doc Comment Tags

Block tags have the format `@tag description`. There are many block tags available, but the more commonly used ones are:

@author name Author Name (class/interface only)
@version major.minor.patch Version number (class/interface only)
@param name description Description of parameter (method only)
@return description Description (method only)
@throws Throwable description Description of exception (exceptions are discussed in the next module)
@deprecated explanation Explanation (method only)
@see package.class#member label A hyperlink to a reference package/class/member or field.

See demo and command line demo …

AP Subset

Tested in the AP CS A Exam	Notes	Not tested in the AP CS A Exam, but potentially relevant/useful
Comments `/* */`, `//`, and `/** */` Javadoc `@param` and `@return` comment tags		Javadoc tool
Primitive Types `int`, `double`, `boolean`		`char`, `byte`, `short`, `long`, `float`

Operators
Arithmetic: `+`, `-`, `*`, `/`, `%`
Increment/Decrement: `++`, `--`
Assignment: `=`, `+=`, `-=`, `*=`, `/=`, `%=`
Relational: `==`, `!=`, `<`, `<=`, `>`, `>=`
Logical: `!`, `&&`, `||`
Numeric casts: `(int)`, `(double)`
String concatenation: `+`
	1, 2, 3, 4, 5	`&`, `|`, `^`
		`(char)`, `(float)`
		`StringBuilder`
		Shift: `<<`, `>>`, `>>>`
		Bitwise: `~`, `&`, `|`, `^`
		Conditional: `?:`

Object Comparison object identity `(==, !=)` vs. object equality `(equals)`, `String compareTo`		implementation of `equals` `Comparable`
Escape Sequences `\"`, `\\`, `\n` inside strings		`\'`, `\t`, `\unnnn`
Input / Output `System.out.print`, `System.out.println`	6	`Scanner`, `System.in`, `System.out`, `System.err`, `Stream` input/output, GUI input/output, parsing input: `Integer.parseInt`, `Double.parseDouble` formatting output: `System.out.printf`

Tested in the AP CS A Exam	Notes	Not tested in the AP CS A Exam, but potentially relevant/useful

Exceptions
`ArithmeticException`,
`NullPointerException`,
`IndexOutOfBoundsException`,
`ArrayIndexOutOfBoundsException`,
`IllegalArgumentException`
		`try/catch/finally`
		`throw`, `throws`
		`assert`

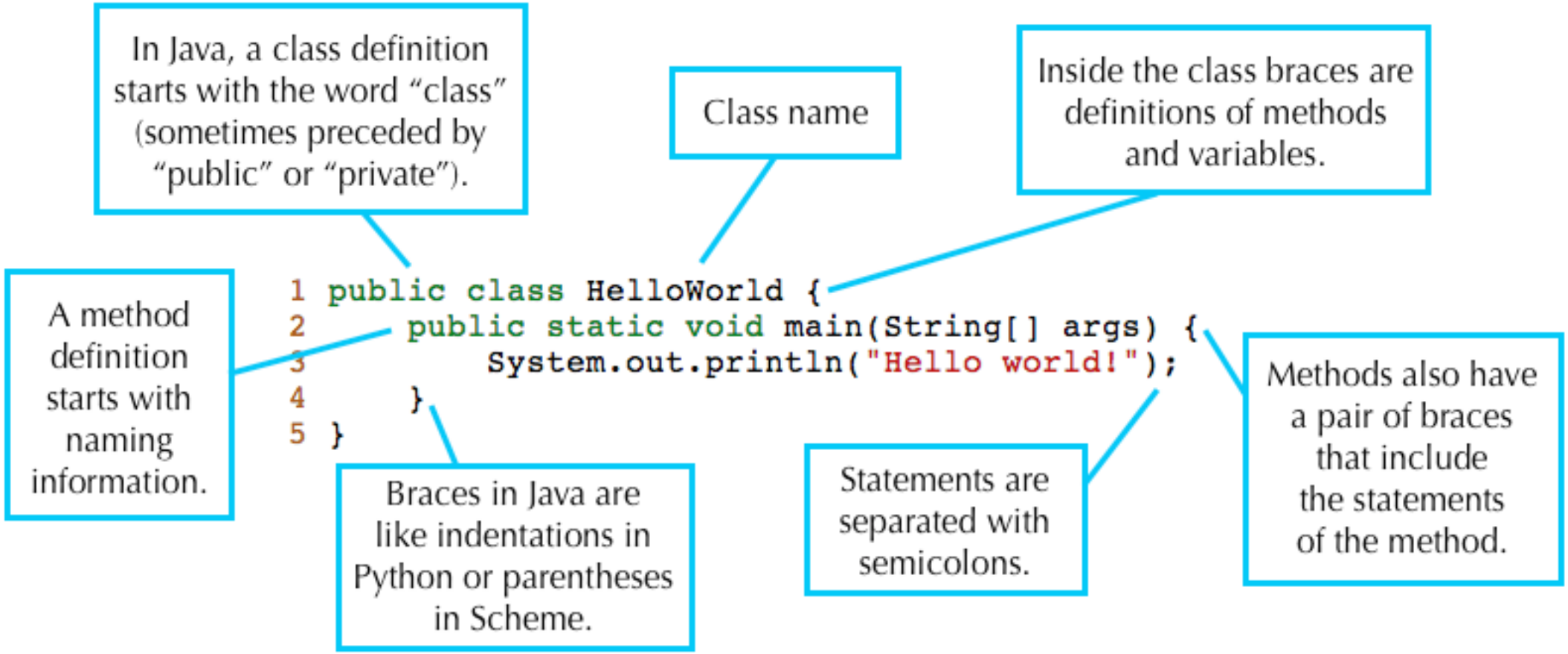
Arrays 1-dimensional arrays, 2-dimensional rectangular arrays, initializer list: `{ ... }`, row-major order of 2-dimensional array elements	7, 8	`new` *type*`[]{ ... }`, ragged arrays (non-rectangular), arrays with 3 or more dimensions
Control Statements `if`, `if/else`, `while`, `for`, enhanced `for` (for-each), `return`		`switch`, `break`, `continue`, `do-while`
Variables parameter variables, local variables, `private` instance variables: visibility `(private)` `static` (class) variables: visibility `(public, private)`, `final`		`final` parameter variables, `final` local variables, `final` instance variables

Methods
visibility `(public, private)`,
`static`, non-static,
method signatures,
overloading, overriding,
parameter passing
	9, 10	visibility `(protected)`,
		`public static void`
		`main(String[] args)`,
		command line arguments,
		variable number of parameters,
		`final`

Constructors `super()`, `super(`*args*`)`	11, 12	default initialization of instance variables, initialization blocks, `this(`*args*`)`

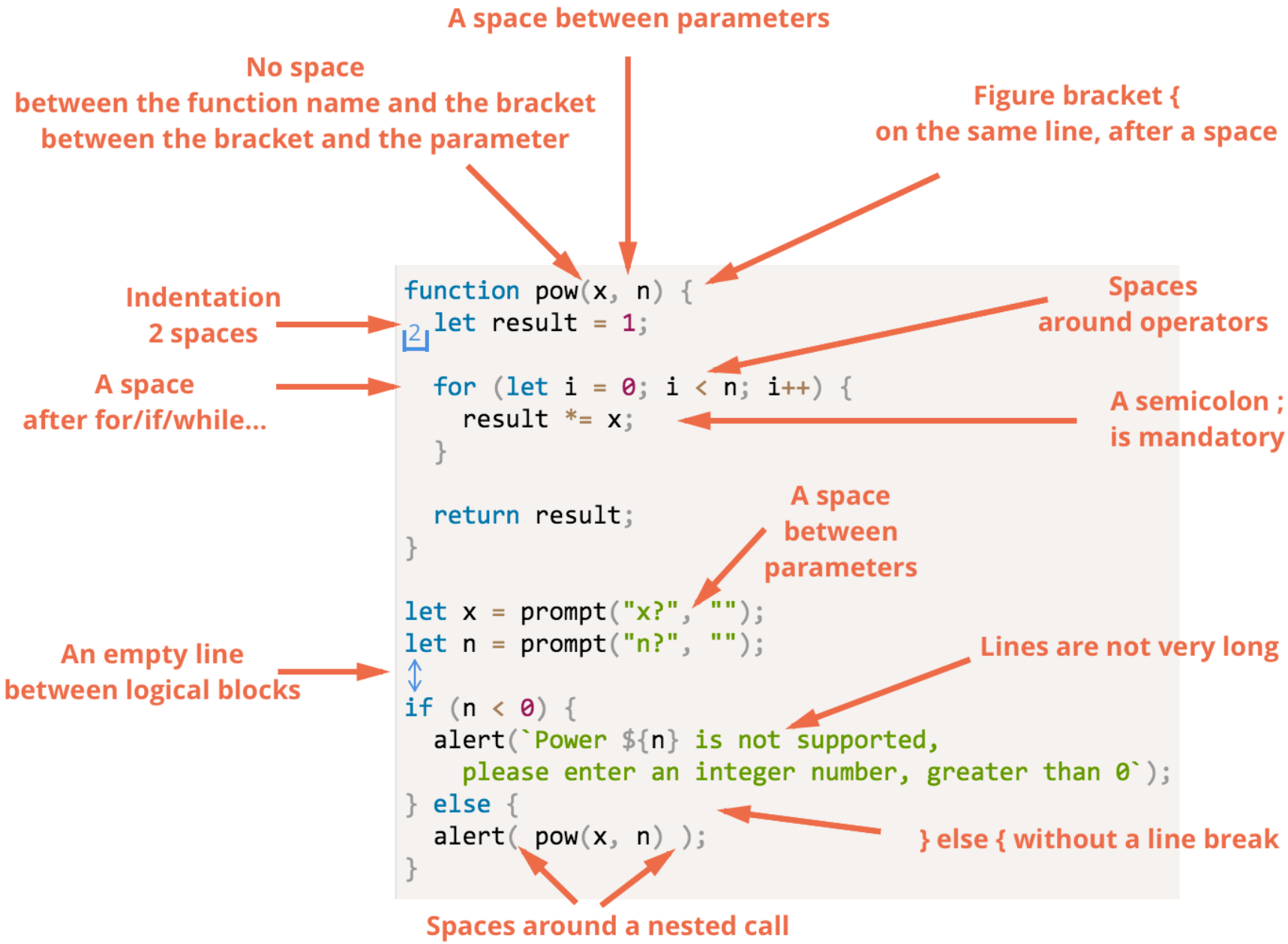
Tested in the AP CS A Exam	Notes	Not tested in the AP CS A Exam, but potentially relevant/useful
Classes `new`, visibility `(public)`, accessor methods, modifier (mutator) methods Design/create/modify class. Create subclass of a superclass `(abstract, non-abstract)`. Create class that implements an interface.	13, 14	`final`, visibility `(private, protected)`, nested classes, inner classes, enumerations
Interfaces Design/create/modify an interface.	13, 14	
Inheritance Understand inheritance hierarchies. Design/create/modify subclasses. Design/create/modify classes that implement interfaces.		
Packages `import` *packageName.className*		`import` *packageName*`.*` , `static` import, `package` *packageName* , class path
Miscellaneous OOP "is-a" and "has-a" relationships, `null`, `this`, `super.`*method(args)*	15, 16	`instanceof` *(class)* cast `this.`*var*, `this.`*method* `(`*args*`)`,
Standard Java Library `Object`, `Integer, Double`, `String`, `Math`, `List<E>, ArrayList<E>`	17, 18	`clone`, autoboxing, `Collection<E>`, `Arrays, Collections`

Java Coding Habits


In Java, a class definition starts with the word "class" (sometimes preceded by "public" or "private").
Class name
Inside the class braces are definitions of methods and variables.
A method definition starts with naming information.
1 public class HelloWorld {
2 public static void main(String[] args) {
3 System.out.println("Hello world!");
4 }
5 }
Braces in Java are like indentations in Python or parentheses in Scheme.
Statements are separated with semicolons.
Methods also have a pair of braces that include the statements of the method.

```
public void RefreshCatalog()
{
    if (IsCacheValid)
    {
        ResetFilterToDefaults();
    }
    else
    {
        RepopulateCatalogFromService();
    }
}
```

```
public void RefreshCatalog() {
    if (IsCacheValid) {
        ResetFilterToDefaults();
    }
    else {
        RepopulateCatalogFromService();
    }
}
```


A space between parameters
No space
between the function name and the bracket
between the bracket and the parameter
Figure bracket {
on the same line, after a space
Indentation
2 spaces
Spaces
around operators
A space
after for/if/while...
A semicolon ;
is mandatory
A space
between
parameters
An empty line
between logical blocks
Lines are not very long
} else { without a line break
Spaces around a nested call
function pow(x, n) {
let result = 1;
for (let i = 0; i < n; i++) {
result *= x;
}
return result;
}
let x = prompt("x?", "");
let n = prompt("n?", "");
if (n < 0) {
alert(`Power ${n} is not supported,
please enter an integer number, greater than 0`);
} else {
alert(pow(x, n));
}

Java Standard Naming Conventions

Package Name - A package should be named in lowercase characters.

Class Name - Class names should be nouns in UpperCamelCase.

Interface Name - Interface name should start with an uppercase letter and be an adjective.

Method Name - Methods should be verbs and in lowerCamelCase.

Variable Name - Variable name should in lowerCamelCase.

Constant Variable - Constant variable names should be written in upper characters separated by underscores.

Abstract Class Name - Abstract class name must start with Abstract or Base prefix.

Exception Class Name - Exception class name must end with Exception suffix

Code Listing 48: Naming Convections in Java

Avoid

```
1.    class icecream{
2.     int flavourtype;
3.     final int size=2;
4.     void
      getflavourtype(){
5.      return
      flavourtype;
6.     }
7.    }
```

Preferable

```
1.    class IceCream{
2.      int flavourType;
3.      final int SIZE=2;
4.      void
      getFlavourType(){
5.        return
      flavourType;
6.      }
7.    }
```

Goals of AP Computer Science A

LECTURE 5

Goals of AP Computer Science A

- Basic programming skills
- Program analytical skills
- Software development flow
- Hardware and number system knowledge