PÉCSI TUDOMÁNYEGYETEM
TERMÉSZETTUDOMÁNYI KAR

“Methodology of Programming I”

Spring 2025

PTE

Zahra Ahmadipour

Zahra@gamma.ttk.pte.hu

Java overview:

- Java is based on C++ but it was specifically designed to be smaller, simpler, and more reliable.
- All Java subprograms are methods and are defined in classes. Java is a class-based, object-oriented programming language.
- Java is now widely used in a variety of different applications areas; mobile apps, web apps, desktop apps, games, web servers, data connections and etc.

Some apps built with Java:

- Mat-lab
- Amazon web application
- Netflix

Why use Java?

- Open-source
- Cross platform (Windows, Mac and Linux)
- Most popular programming language with a huge community
- An object oriented language which gives a clear structure to programs
- Free compiler/interpreter system which can be easily obtained on the Web

Java editions:

- Standard Edition (SE): Core Java platform. Contains all libraries every Java developer must learn.
- Enterprise Edition (EE): for building very large scale and distributed systems.
- Micro Edition (ME): Subset of SE designed for mobile devices.
- Java Card: used in smart cards.

JDK:

The Java Development Kit (JDK) is a cross-platform software development environment that offers a collection of tools and libraries necessary for developing Java-based software applications and applets.

JDK = JRE (JVM + Library classes) + Development tools

Compilation and execution:

Java, being a platform-independent programming language, doesn’t work on the one-step compilation.

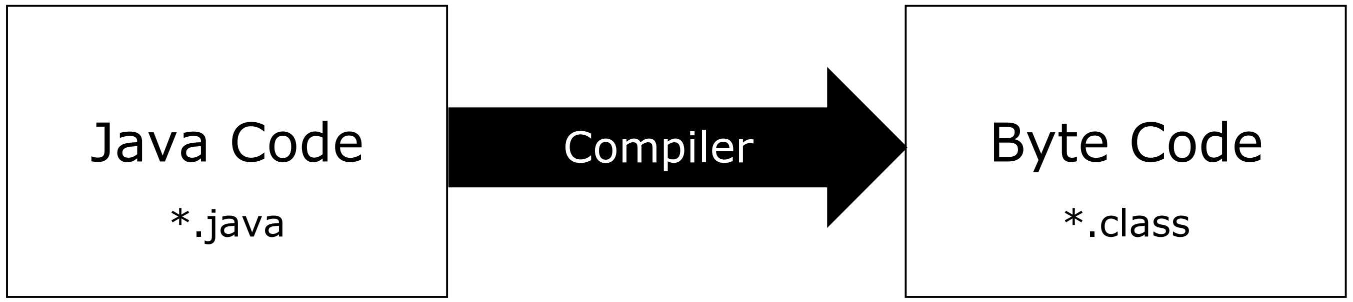
It involves a two-step execution:

First, through an OS-independent compiler;

Second, in a virtual machine (JVM) which is custom-built for every operating system.

1- Create a Java program

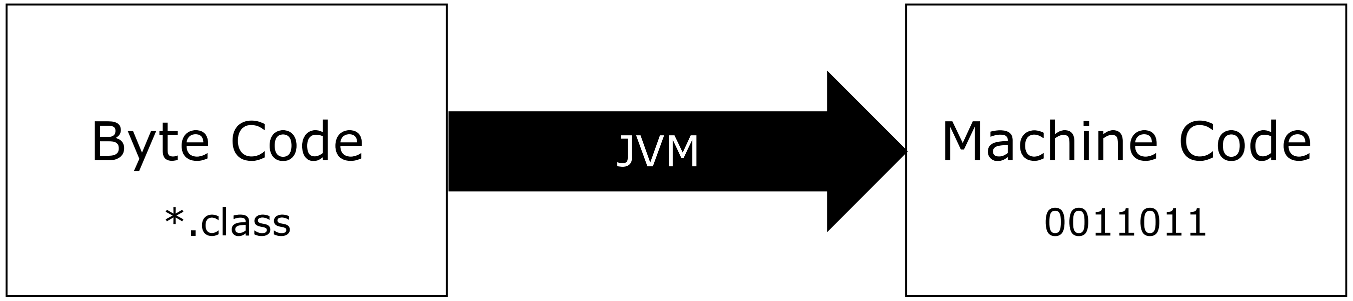
2- Compile the program



The class files generated by the compiler are independent of the machine or the OS, which allows them to be run on any system.

3- Execute the program

To run, the main class file (the class that contains the method main) is passed to the JVM and then the final machine code is executed.



Primary components of a computer:

- Internal memory: stores programs and data
- Processor: a collection of circuits that provides a realization of a set of primitive operations, or machine instructions, such as those for arithmetic and logic operations

The machine language of the computer is its set of instructions.

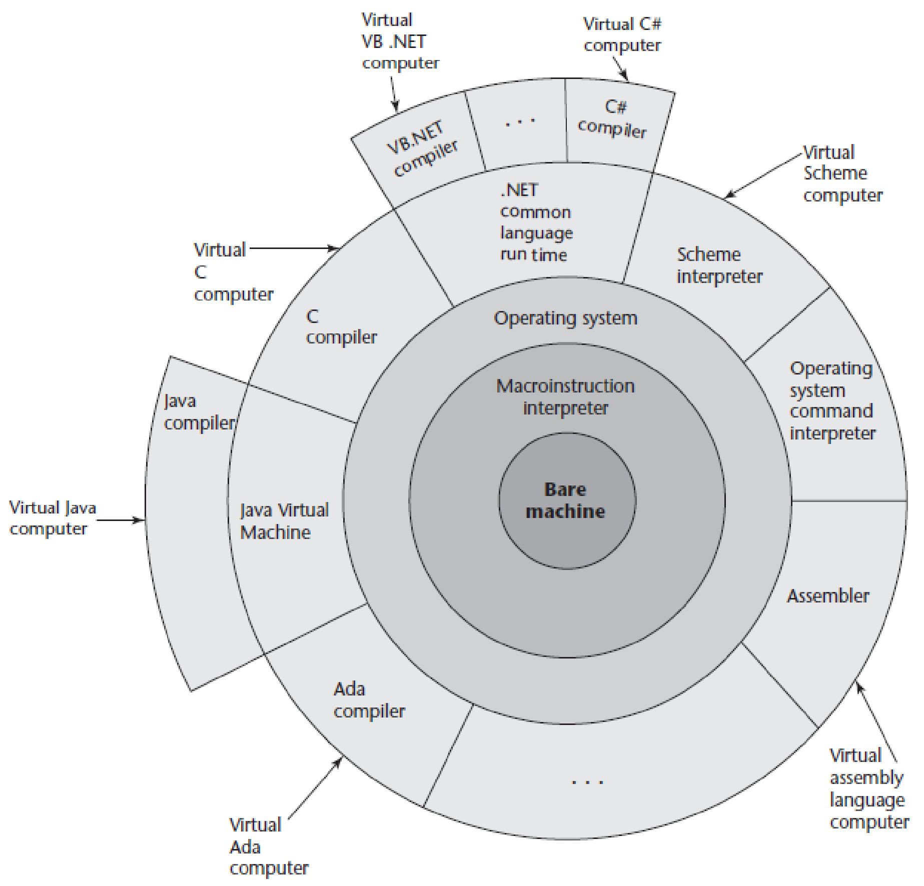
In the absence of other supporting software, its own machine language is the only language that most hardware computers “understand”.

To supply higher-level primitives than those of the machine language, a large collection of programs, called the operation system is required.

Language implementation systems need many of the operating system facilities, they interface with the operating system rather than directly with the processor in machine language.

The operating system and language implementations are layered over the machine language interface of a computer.

These layers can be thought of as virtual computers, providing interfaces to the user at higher levels.



Implementation Methods:

- Compilation
- Pure Interpretation
- Hybrid Interpretation

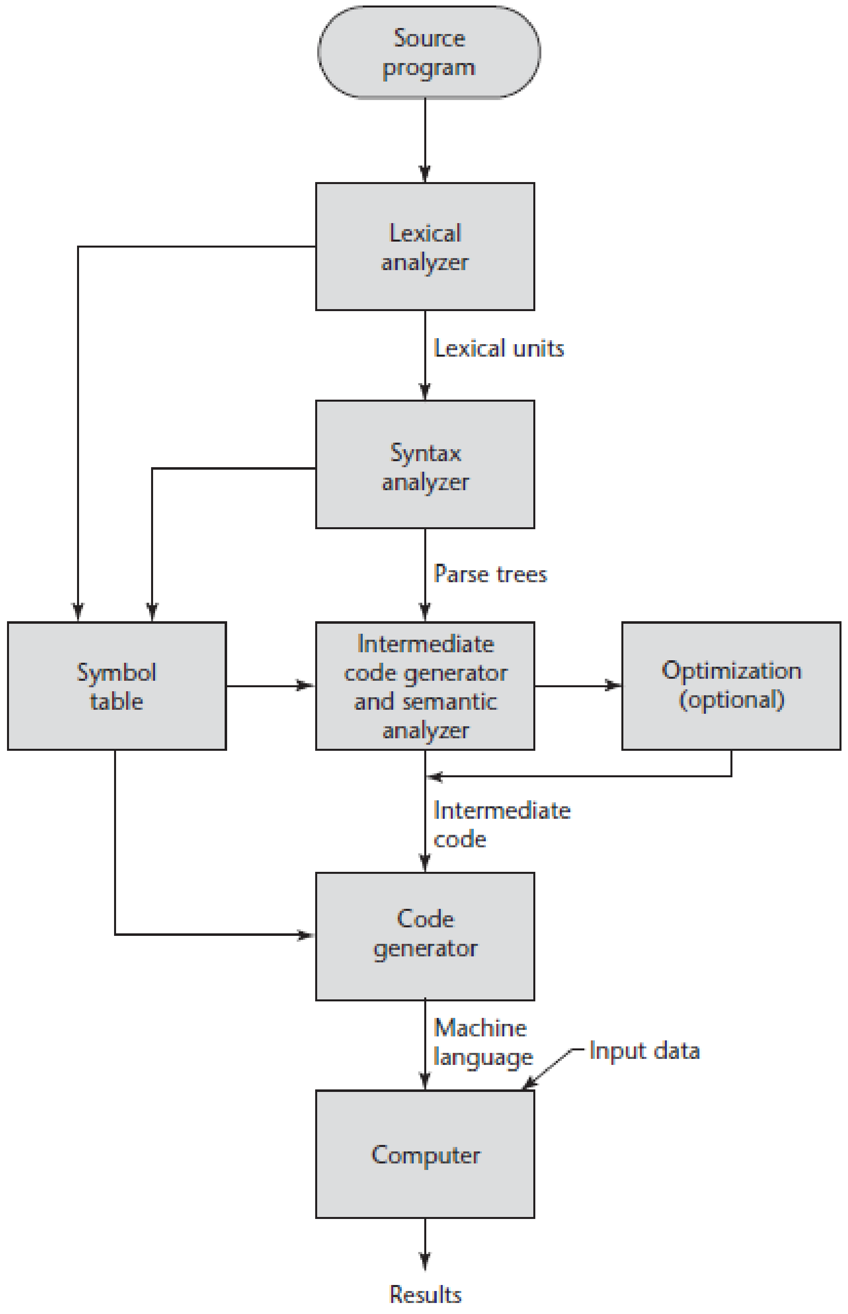
Compilation:

Programming languages can be implemented by any of three general methods.

At one extreme, programs can be translated into machine language, which can be executed directly on the computer.

This method is called a compiler implementation and has the advantage of very fast program execution, once the translation process is complete.

Most production implementations of languages, such as C, COBOL, C++, and Ada, are by compilers.


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Symbol table
Intermediate code generator and semantic analyzer
Optimization (optional)
Intermediate code
Code generator
Machine language
Input data
Computer
Results

Pure Interpretation:

Pure interpretation lies at the opposite end (from compilation) of implementation methods. With this approach, programs are interpreted by another program called an interpreter, with no translation whatever.

The interpreter program acts as a software simulation of a machine whose fetch-execute cycle deals with high-level language program statements rather than machine instructions. This software simulation provides a virtual machine for the language.

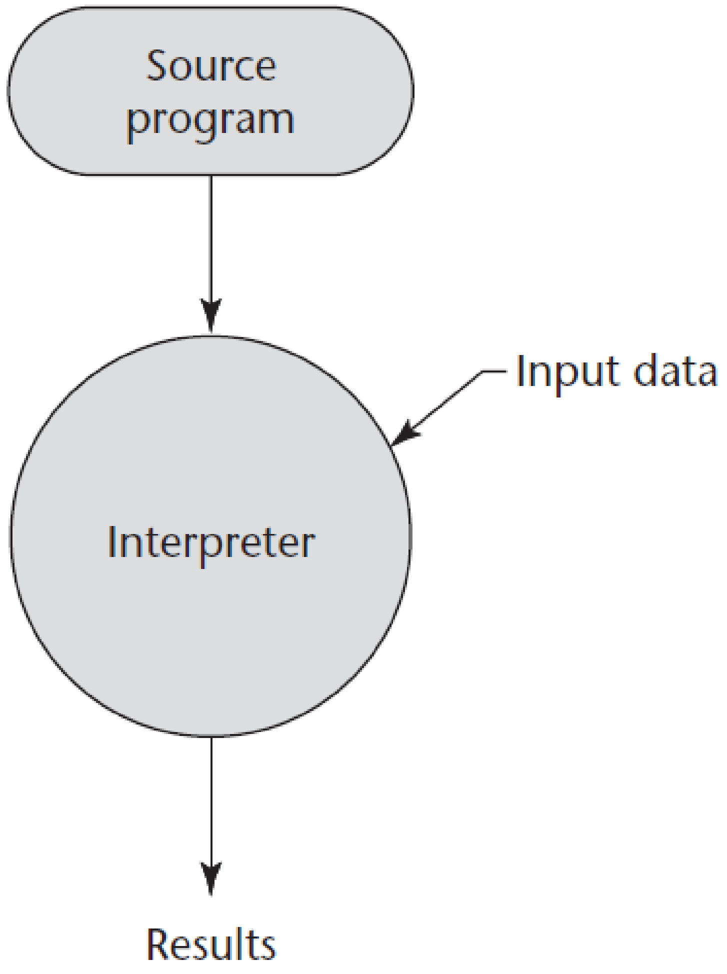
This approach is rarely used on high level languages.

Advantages:

- allowing easy implementation of many source-level debugging operations (e.g. the error message can easily indicate the source line)

Disadvantages:

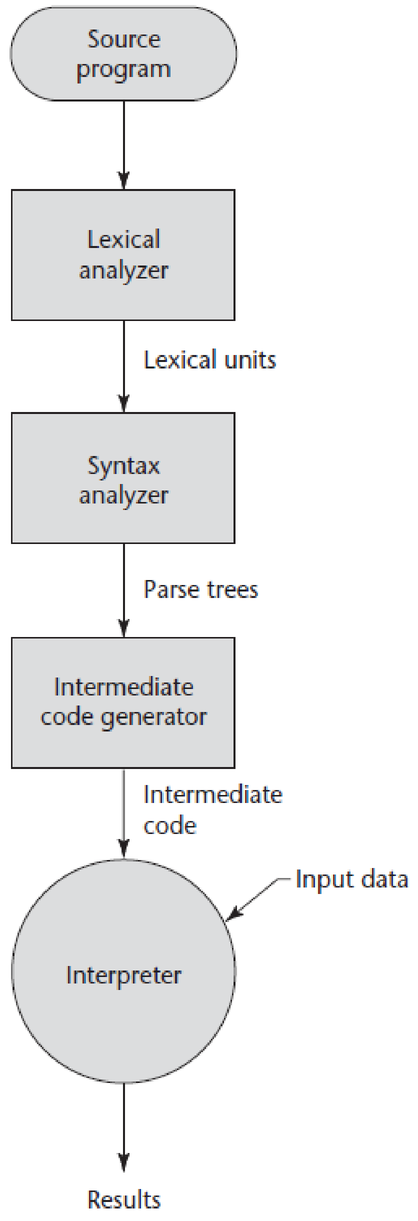
- Slow execution caused by decoding of the high-level language statements which are far more complex than machine language instructions
- regardless of how many times a statement is executed, it must be decoded every time
- requires more space


Source program
Input data
Interpreter
Results

Hybrid Interpretation:

Some language implementation systems are a compromise between compilers and pure interpreters; they translate high-level language programs to an intermediate language designed to allow easy interpretation.

This method is faster than pure interpretation because the source language statements are decoded only once. Instead of translating intermediate language code to machine code, it simply interprets the intermediate code.


Source program
Lexical analyzer
Lexical units
Syntax analyzer
Parse trees
Intermediate code generator
Intermediate code
Input data
Interpreter
Results

Initial implementations of Java were all hybrid. Its intermediate form, called byte code, provides portability to any machine that has a byte code interpreter and an associated run-time system. Together, these are called the Java Virtual Machine. There are now systems that translate Java byte code into machine code for faster execution.

A Just-in-Time (JIT) implementation system initially translates programs to an intermediate language. Then, during execution, it compiles intermediate language methods into machine code when they are called. The machine code version is kept for subsequent calls.

Preprocessor:

A preprocessor is a program that processes a program immediately before the program is compiled. Preprocessor instructions are embedded in programs.

The preprocessor is essentially a macro expander. Preprocessor instructions are commonly used to specify that the code from another file is to be included.

For example, the C preprocessor instruction:

#include "myLib.h"

causes the preprocessor to copy the contents of myLib.h into the program at the position of the #include.

Variables

A program variable is an abstraction of a computer memory cell or collection of cells.

Programmers often think of variable names as names for memory locations, but there is much more to a variable than just a name.

A variable can be characterized by a collection of properties, or attributes::

(name, address, value, type, lifetime, and scope).

Most variables have **names**.

The **address** of a variable is the machine memory address with which it is associated.

The **type** of a variable determines the range of values the variable can store and the set of operations that are defined for values of the type.

The **value** of a variable is the contents of the memory cell or cells associated with the variable. It is convenient to think of computer memory in terms of abstract cells, rather than physical cells.

Name:

Name is a string of characters used to identify some entity in a program.

Names in Java, and C# have no length limit, and all characters in them are significant.

Names in most programming languages have the same form:

> a letter followed by a string consisting of letters, digits, and underscore characters (_). It cannot contain whitespace.

All Java variables must be identified with unique names.

In many languages, notably Java, uppercase and lowercase letters in names are distinct; that is, names in these languages are case-sensitive.

Special words in programming languages are used to make programs more readable by naming actions to be performed. They also are used to separate the syntactic parts of statements and programs.

In most languages, special words are classified as reserved words, which means they cannot be redefined by programmers

A reserved word (like Java keywords, such as int or boolean) is a special word of a programming language that cannot be used as a name.

The Java programming language is statically-typed, which means that all variables must first be declared before they can be used.

The “equal sign” is used to assign values to the variable.

You can assign a value to a variable, when declaring it or later.

If you assign a new value to an existing variable, it will overwrite the previous value.

If you don't want to overwrite existing values, use the final keyword (Which makes the variable unchangeable and read-only).

Data Types

A data type defines a collection of data values and a set of predefined operations on those values.

The type system of a programming language defines how a type is associated with each expression in the language and includes its rules for type equivalence and type compatibility.

One of the most important parts of understanding the semantics of a programming language is understanding its type system.

1. Primitive data types:

   Data types that are not defined in terms of other types are called primitive data types. Nearly all programming languages provide a set of primitive data types.

   Some of the primitive types are merely reflections of the hardware—for example, most integer types. Others require only a little non-hardware support for their implementation.

   A primitive data type specifies the size and type of variable values, and it has no additional methods.

2. Non-primitive data types:

   Non-primitive data types are called reference types because they refer to objects; and they all have the same size.

Primitive data types:

There are eight primitive data types in total in Java.

- Numbers
- Booleans
- Characters

Type	Size	Description	Range
byte	1 byte	Whole numbers	-128 to 127
short	2 bytes	Whole numbers	-32,768 to 32,767
int	4 bytes	Whole numbers	-2,147,483,648 to 2,147,483,647
long	8 bytes	Whole numbers (Value ends with an "L")	-9,223,372,036,854,775,808 to 9,223,372,036,854,775,807
float	4 bytes	Fractional numbers (Value ends with an "f")	6 to 7 decimal digits
double	8 bytes	Fractional numbers (Value ends with a "d")	15 decimal digits
boolean	1 bit	true/false	
char	2 bytes	Single character/letter/ASCII (Surrounded by single quotes)	

Non-primitive data types:

- Strings
- Arrays
- Classes
- Interfaces

Strings:

String data type is used to store a sequence of characters (text).

String values must be surrounded by double quotes.

String in Java is actually a class, which contain methods that can perform certain operations on strings.

String methods:

• length()			returns	int
• toUpperCase()			returns	String
• toLowerCase()			returns	String
• indexOf()	String	parameter;	returns	int
• charAt()	int	parameter;	returns	char
• contains()	String	parameter;	returns	boolean
• equals()	String	parameter;	returns	boolean
• equalsIgnoreCase()	String	parameter;	returns	boolean
• replace()	String	parameter;	returns	String
• trim()			returns	String

Escape Sequences:

The backslash (\) escape character; turns special characters into string characters. Some common escape sequences in Java:

Escape character	Result
➢ \’	‘
➢ \”	“
➢ \\	\
➢ \n	New Line
➢ \t	Tab

Arrays:

Arrays are used to store multiple values in a single variable.

To declare an array, define the variable type with square brackets.

Place the values in a comma-separated list, inside curly braces.

Arrays have fixed size and items cannot be added or removed after creation.

References:

Concepts of Programming Languages By Robert W. Sebesta

https://docs.oracle.com/javase/tutorial/tutorialLearningPaths.html

https://www.w3schools.com/

https://www.geeksforgeeks.org/