Chapter 2

Introduction to Java

References :

[1] รังสิพรรณ มฤคทัต, กระบวนทัศน์ในการเขียนโปรแกรม (บทที่ 2)

Chapter Objectives

At the end of this chapter, you should be able to:

- Install IDE (NetBeans) for Java programming
- Create Java project and set folder structure correctly
- Write Java programs with standard input/output streams
- Write Java programs that read and write files

Introduction to Java

- Developed by Sun Microsystems (now under Oracle)
- Support multiple paradigms such as object-oriented, event-driven, multithreading (concurrent)
- Similar to C / C++, but complex features are omitted
  - Global variables
  - Pointers & access to variable's address (referencing)
  - Memory de-allocation in Java is done automatically
- Strongly typed language
- Exception handling mechanism
- Platform independent (language)
  - Source code is compiled to platform-independent byte code
  - Byte code is interpreted by Java Virtual Machine (JVM)

Wikipedia, Java version history (https://en.wikipedia.org/wiki/Java_version_history)

Version	Release date	End of Free Public Updates[1][5][6][7]	Extended Support Until
JDK Beta	1995	?	?
JDK 1.0	January 1996	?	?
JDK 1.1	February 1997	?	?
J2SE 1.2	December 1998	?	?
J2SE 1.3	May 2000	?	?
J2SE 1.4	February 2002	October 2008	February 2013
J2SE 5.0	September 2004	November 2009	April 2015
Java SE 6	December 2006	April 2013	December 2018 December 2026 for Azul[8]
Java SE 7	July 2011	April 2015	July 2022
Java SE 8 (LTS)	March 2014	**January 2019 for Oracle (commercial)** December 2030 for Oracle (non-commercial) December 2030 for Azul May 2026 for IBM Semeru[9] At least May 2026 for Eclipse Adoptium At least May 2026 for Amazon Corretto	December 2030[10]

First EGCO 213 class

EGCO 213 textbook

AKA "Java Lambda"

LTS = long term support

Java SE 9	September 2017	March 2018 for OpenJDK	N/A
Java SE 10	March 2018	September 2018 for OpenJDK	N/A
Java SE 11 (LTS)	September 2018	September 2026 for Azul October 2024 for IBM Semeru[9] At least October 2024 for Eclipse Adoptium At least September 2027 for Amazon Corretto At least October 2024 for Microsoft[11][12]	September 2026 September 2026 for Azul[8]
Java SE 12	March 2019	September 2019 for OpenJDK	N/A
Java SE 13	September 2019	March 2020 for OpenJDK	N/A
Java SE 14	March 2020	September 2020 for OpenJDK	N/A
Java SE 15	September 2020	March 2021 for OpenJDK March 2023 for Azul[8]	N/A
Java SE 16	March 2021	September 2021 for OpenJDK	N/A
Java SE 17 (LTS)	September 2021	September 2029 for Azul At least September 2027 for Microsoft At least TBA for Eclipse Adoptium	September 2029 or later September 2029 for Azul
Java SE 18	March 2022	September 2022 for OpenJDK	N/A
Java SE 19	September 2022	March 2023 for OpenJDK	N/A
Java SE 20	March 2023	September 2023 for OpenJDK	N/A
Java SE 21 (LTS)	September 2023	TBA	September 2031[10]

Legend: Old version | Older version, still maintained | **Latest version** | Future release

Oracle Java SE Support Roadmap*†			
Release	**GA Date**	**Premier Support Until**	**Extended Support Until**
8 (LTS)**	March 2014	March 2022	December 2030*****
9 - 10 (non-LTS)	September 2017 - March 2018	March 2018 - September 2018	Not Available
11 (LTS)	September 2018	September 2023	January 2032*****
12 - 16 (non-LTS)	March 2019 - March 2021	September 2019 - September 2021	Not Available
17 (LTS)	September 2021	September 2026****	September 2029****
18 - 20 (non-LTS)	March 2022 - March 2023	September 2022 - September 2023	Not Available
21 (LTS)	September 2023	September 2028****	September 2031****
22 (non-LTS)	March 2024	September 2024	Not Available
23 (non-LTS)***	September 2024	March 2025	Not Available
24 (non-LTS)***	March 2025	September 2025	Not Available
25 (LTS)***	September 2025	September 2030	September 2033

Oracle, Java SE Support Roadmap
(https://www.oracle.com/th/java/technologies/java-se-support-roadmap.html)

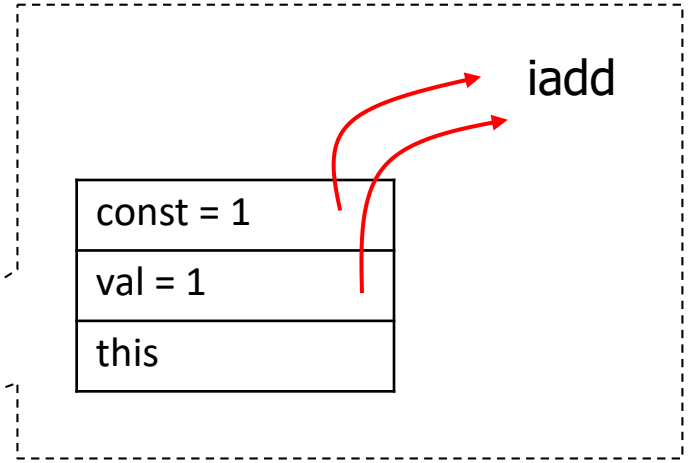
Java Virtual Machine

- Java Runtime Environment (JRE) contains JVM and library files (classes) used by JVM
- JVM interprets byte code
  - Generic machine-independent code
  - Opcode is 1-byte width ∴ 256 elementary instructions
  - Stack machine, not using registers
- Platform (HW+OS) dependent, but all JVM implementation must be in accordance with Sun's specification
- Program execution is slow especially class loading at the beginning of the execution

Example : byte code

- Source program

```
class A {
    int k;
    void f (int val) { k = val + 1; }
}
```



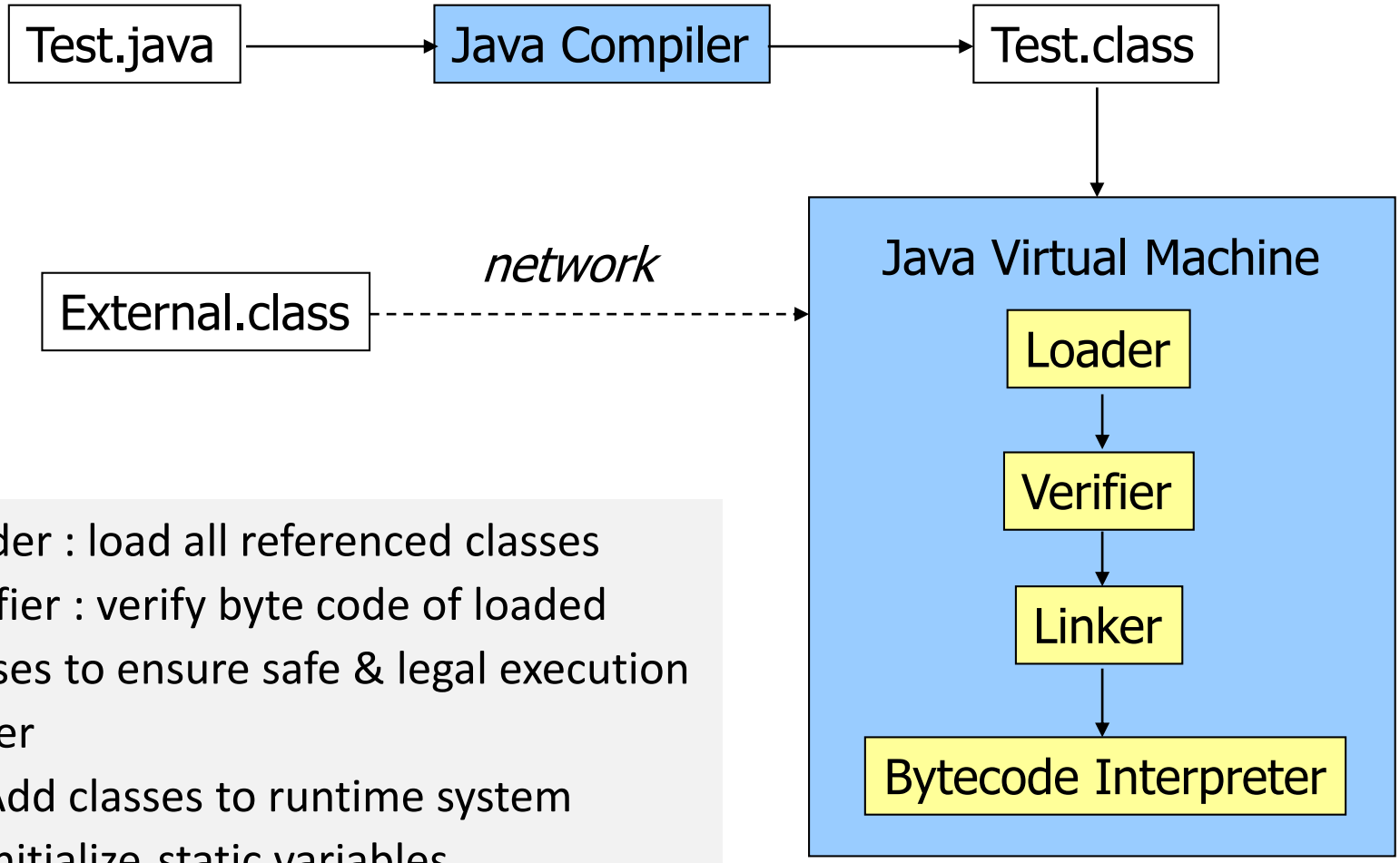
- Bytecode

```
Method void f (int)
    aload 0                 // push a ref (“this”) from local var at index 0
    iload 1                 // push an int (“val”) from local val at index 1
    iconst 1
    iadd
    putfield #4 <Field int k>       // pop a value & assign it to “k”
    return
```

Compiling + Executing Java Program

- SET PATH=$PATHS;C:\Java\jdk1.7.0_03\bin
  - Path to JDK tools such as javac, java
- SET CLASSPATH=.;C:\Java\jdk1.7.0_03\
  - Path to classes that need to be loaded for program execution
- Compile : javac Test.java ➔ Test.class
  - javac first compiles current class
  - Classes that are referenced by this current class will also be compiled automatically
- Execute : java Test (interpret Test.class)
  - When program execution starts, the interpreter loads all JRE and user-defined classes automatically

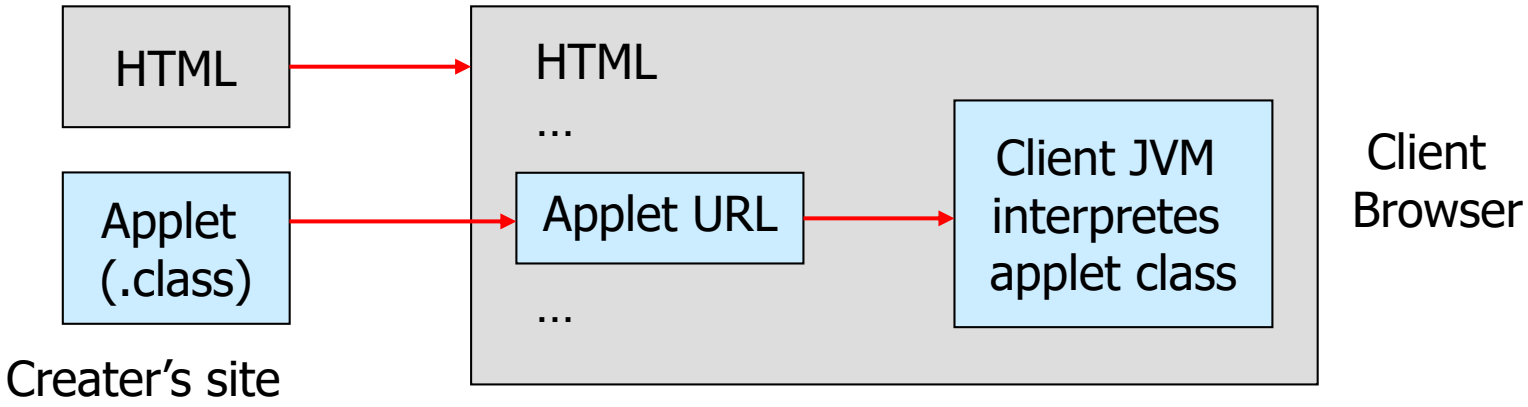
JVM Architecture



- Loader : load all referenced classes
- Verifier : verify byte code of loaded classes to ensure safe & legal execution
- Linker
  - Add classes to runtime system
  - Initialize static variables
  - Resolve names + references

Application *vs.* Applet

- Application in this course is stand-alone, stored & executed on user's PC
- Applet : mini-application incorporated into a web page
  - Class file is downloaded and executed in client browser
  - User interacts with applet without leaving the browser
  - Multithreaded & GUI programming



Basic Java Syntax

- Java program is encapsulated in class – usually one class in a source file
  - Only one class in a source file can have **main** method
  - When the program is executed ➔ main is called first (similar to C, C++)

General program structure

```
import javax.swing.*;
public class <class name> {
    <variable declarations>
    <method declarations>

    public static void main ( String[ ] args ) {
        …
    }
}
```

HelloWorld.java

```
public class HelloWorld {
    public static void main(String[ ] args) {
        System.out.println("HELLO WORLD");
    }
}
```

Main method must always have this parameter

* Source file is named after class with main(…)
* If there are multiple classes in a file – there can be only one public class
  - The file must be named after public class
  - Compiler generates 1 class file (.class) per class
* If there are only default classes in a file – any file name is allowed

Language Overview

Primitive data type

- boolean, byte, char, short, int, long, float, double
- Unicode format for character ➔ 2 bytes : 1 char
- Use reference data type instead of pointer

Type wrapper class

- Boolean, Byte, Character, Short, Integer, Long, Float, Double
- Correspond to primitive type
  - With MIN_VALUE, MAX_VALUE constants
  - Also POSITIVE_INFINITY, NEGATIVE_INFINITY, NaN constants in Long, Float, Double
  - Type conversion methods

Variable

- Must be declared before used
- No global variable – all variables must be local vars inside method or be class members
- Modifiers e.g.
  - Visibility scope : private, public, protected
  - “final” is equivalent to “const” in C : variable can be assigned a value only once (in declaration statement)
  - “static” variable is bound to class, not object
- Access to variable’s address is not allowed
- But reference variable has its own addressing mechanism

Operator, Expression, Assignment

- Similar to C, C++
- Primitive and reference operands are processed differently
  - Value processing (primitive)
  - Address processing (reference)
- Elementary operators, such as math & logical operators, cannot be overloaded

Statement

- Similar to C, C++ but without “goto”

Method

- Methods can be put in any order
- Overloading is allowed
- Modifiers e.g.
  - Visibility scope : private, public, protected
  - “static” method is bound to class – it can only call static methods or access static variables
- Argument passing : same syntax for value and reference data, but different mechanisms when passing occurs

Stream I/O

- There are 3 basic stream objects in Java
  - System.out
  - System.err

  } PrintStream object
  - System.in — InputStream object
- Show output on screen : System.out or System.err
  - System.out.print (<expression>)
  - System.out.println (<expression>)

```
float x = 10.5;
System.out.println("x = " + x);

// C-style printf
System.out.println( String.format("x = %.4f", x) );    OR
System.out.printf("x = %.4f \n", x);
```

- System.out : buffered stream
  - Characters are buffered & only printed out once the stream accumulates enough data
  - Few calls to I/O ➔ better performance
  - The stream can be crashed before results are printed
- System.err : unbuffered stream
  - Each character is printed immediately once being put the stream
  - Many calls to I/O
  - But in case of errors, we want to inform the user asap

- Repeating strings by repeat(n)
  - String separator = ”-+ ”;
  - System.out.println(separator.repeat(40));
  - System.out.printf(”%s \n”, separator.repeat(40));
- Formatting character & string
  - **% [*align*] [*width*] { c | C | s | S }**

–	= left alignment
c, s	= as-is
C, S	= all upper case

char a = ‘a’;	
%4c	□□□a
%4C	□□□A
%-4c	a□□□
%-4C	A□□□

String name = ”John”;	
%8s	□□□□John
%8S	□□□□JOHN
%-8s	John□□□□
%-8S	JOHN□□□□
%3s %-3s	John

Left alignment for insufficient width

Formatting integer

- **% [*align*] [*sign*] [0] [,] [*width*] d**

Width includes sign, zeros, and comma
In case of sufficient width, alignment will be on the left

–	= left alignment
+	= always add sign
0	= add leading zeros
,	= comma

int x = 11111;	
%8d	□□□11111
%+8d	□□+11111
%08d	00011111
%+08d	+0011111
%,8d	□□11,111
%+,8d	□+11,111
%0,8d	0011,111
%+0,8d	+011,111

int y = -22222;	
%-8d	-22222□□
%-+8d	-22222□□
%-,8d	-22,222□
%-+,8d	-22,222□

Leading zeros are added for right alignment only

Formatting double & float

- **% [*align*] [*sign*] [0] [,] [*width*] [*.precision*] f**

Width includes sign, zeros, comma, dot, and precision

double u = 10000.456;	
%12.2f	□□□□10000.46
%12.4f	□□10000.4560
%012.2f	000010000.46
%+012.2f	+00010000.46
%,12.2f	□□□10,000.46
%+,12.2f	□□+10,000.46
%0,12.2f	00011,000.46
%+0,12.2f	+0011,000.46

float v = 20000.456f;	
%-12.2f	20000.46□□□□
%-12.4f	20000.4551□□
%-,12.2f	20,000.46□□□
%-+,12.2f	+20,000.46□□

Float supports up to 6-7 digits including mantissa + decimals
20000.456 has 8 digits ➔ imprecise decimals from rounding/truncation

Using Simple Dialog Box

- **JOptionPane** → import javax.swing.*
- Read input via showInputDialog(prompt_string)
  - Dialog box can read 1 input at a time
  - Return a string which must be properly casted before used
- Show output in dialog box : slower than System.out

showMessageDialog(null, message_string, title, icon_type)

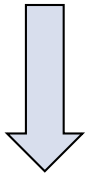
JOptionPane.ERROR_MESSAGE
JOptionPane.INFORMATION_MESSAGE
JOptionPane.WARNING_MESSAGE
JOptionPane.QUESTION_MESSAGE
JOptionPane.PLAIN_MESSAGE = no icon

```
import javax.swing.*;
class Test
{
      public static void main(String[ ] args)
      {
            String buf = JOptionPane.showInputDialog("Enter input");

            // convert to numbers
            int ibuf = Integer.parseInt(buf);
            double dbuf = Double.parseDouble(buf);

            System.out.println("Output = " + buf);

            JOptionPane.showMessageDialog( null,  buf,  "Output",
                              JOptionPane.INFORMATION_MESSAGE );

      }
}
```

Class java.util.Scanner

- Create Scanner objects from System.in or file
  - Scanner input = new Scanner(System.in);
  - Scanner input = new Scanner(new File("temp.txt"));
    - When working with file, exceptions must be caught
- Methods to scan input from input stream or file
  - public boolean hasNext()
  - public boolean hasNextLine()
  - public String next() // read only 1 token
  - public boolean nextBoolean()
  - public int nextInt()
  - public float nextFloat()
  - public double nextDouble()
  - public String nextLine() // read the whole line

```
import java.util.*;
class Test
{
    public static void main(String[ ] args)
    {
        int x;  double y;  String s;
        Scanner input = new Scanner(System.in);

        System.out.println("Enter integer = ");    x = input.nextInt();
        System.out.println("Enter float = ");      y = input.nextDouble();
        System.out.println("Enter string = ");     s = input.next();
        …
    }
}
```



In new Java version, prompt message must end with \n,
so that input cursor is at the beginning of the next line

Consuming new line character

- Methods : next, nextBoolean, nextInt, nextFloat, nextDouble
  read only input value excluding \r (when user presses ENTER)
- Method nextLine reads everything until \r and return the whole line before \r

```
int x = input.nextInt();                    // type 100 + \r, but read only 100
String s = input.nextLine();                // read leftover \r
                                            // proceed to next instruction
System.out.printf("input = %d  %s \n");     // input = 100
```

- To fix this

```
int x = input.nextInt();                    // type 100 + \r, but read only 100
input.nextLine();                           // read leftover \r
String s = input.nextLine();                // type XXX + \r, read XXX + \r
System.out.printf("input = %d  %s \n");     // input = 100 XXX
```

```
import java.util.*;
import java.io.*;
class Test
{
    public static void main(String[ ] args)
    {
        int x;  double y;  String s;
        try {
            Scanner input = new Scanner( new File("temp.txt") );
            while ( input.hasNext() ) {
               x = input.nextInt();   y = input.nextDouble();   s = input.next();
            }
            input.close();
        }
        catch (Exception e) { System.err.println(e); }
    }
}
```

temp.txt

```
10   100.5   Susan
20   200.6   John
...
```

Class java.io.File

- File creation must be done inside try-catch
- When creating file ➔ use relative path, not absolute path that is valid only on one particular PC
  - Also make sure all folder names don't contain Thai or special characters
- Some methods
  - public boolean exists()
  - public String getPath() // check relative path
  - public String getAbsolutePath() // check absolute path
  - public void delete()

Class java.io.PrintWriter

- Create PrintWriter objects from System.out or file
  - PrintWriter write = new PrintWriter(System.out);
  - File outFile = new File(new File("temp.txt"));
  - PrintWriter write = new PrintWriter(outFile);
    - Overwrite existing file by default
  - PrintWriter write = new PrintWriter(new FileWriter(outFile), true);
  - PrintWriter write = new PrintWriter(new FileWriter(outFile), false);
    - If arg2 = true ➔ append mode
    - If arg2 = false ➔ overwrite mode
  - When working with file, exceptions must be caught

- Methods to write string to output stream or file
  - Similar to print, println, printf in System.out
  - public void print(String message)
  - public void println(String message)
  - public void printf(String format, *arguments*)
  - public void flush()

```
import java.io.*;
class Test
{
    public static void main(String[ ] args)
    {
        try {
            // If file already exists, new file will replace the old one
            PrintWriter write = new PrintWriter( new File("temp.txt") );

            // use \r\n to force line break when printing to file
            write.printf("%d   %.2f   %s \r\n", 10, 100.5, "Susan");
            write.printf("%d   %.2f   %s \r\n", 20, 200.6, "John");

            write.flush();
            write.close();      // auto flush before closing file
        }
        catch (Exception e) { System.err.println(e); }
    }
}
```

temp.txt

```
10   100.5   Susan
20   200.6   John
...
```