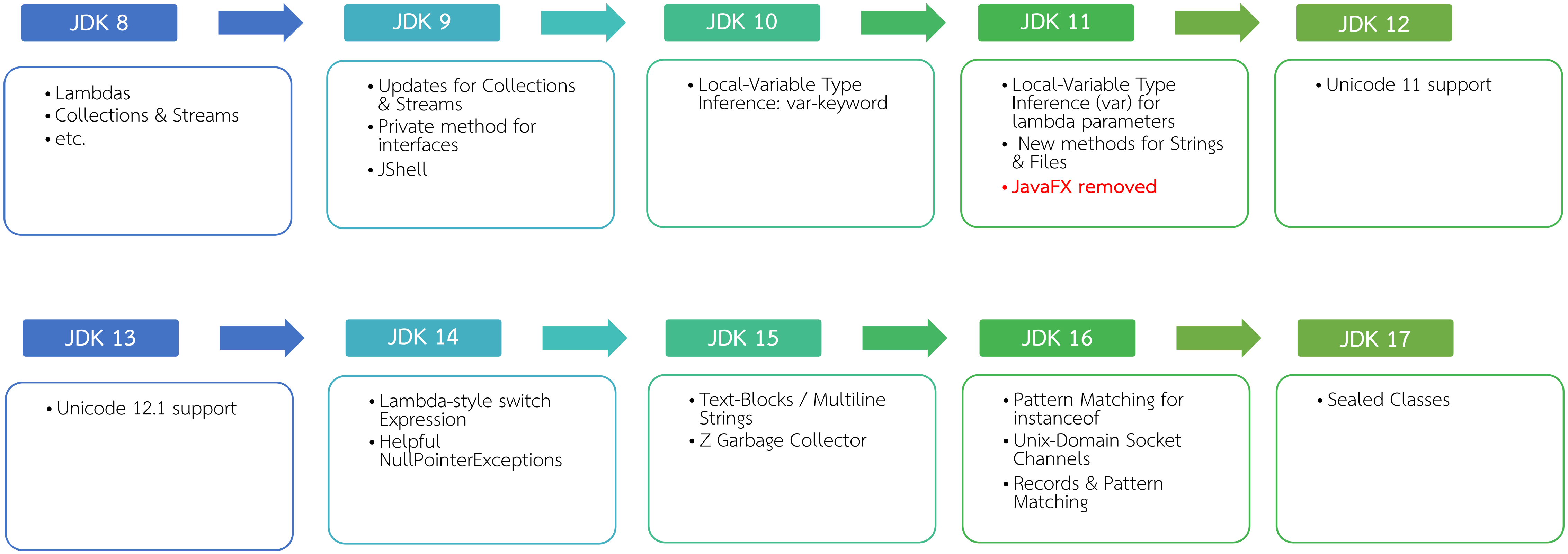
Introduction to Java Programming

Thammakorn Saethang

JDK Notable Features (ver.8-17)



JDK 8

- Lambdas
- Collections & Streams
- etc.

JDK 9

- Updates for Collections & Streams
- Private method for interfaces
- JShell

JDK 10

- Local-Variable Type Inference: var-keyword

JDK 11

- Local-Variable Type Inference (var) for lambda parameters
- New methods for Strings & Files
- JavaFX removed

JDK 12

- Unicode 11 support

JDK 13

- Unicode 12.1 support

JDK 14

- Lambda-style switch Expression
- Helpful NullPointerExceptions

JDK 15

- Text-Blocks / Multiline Strings
- Z Garbage Collector

JDK 16

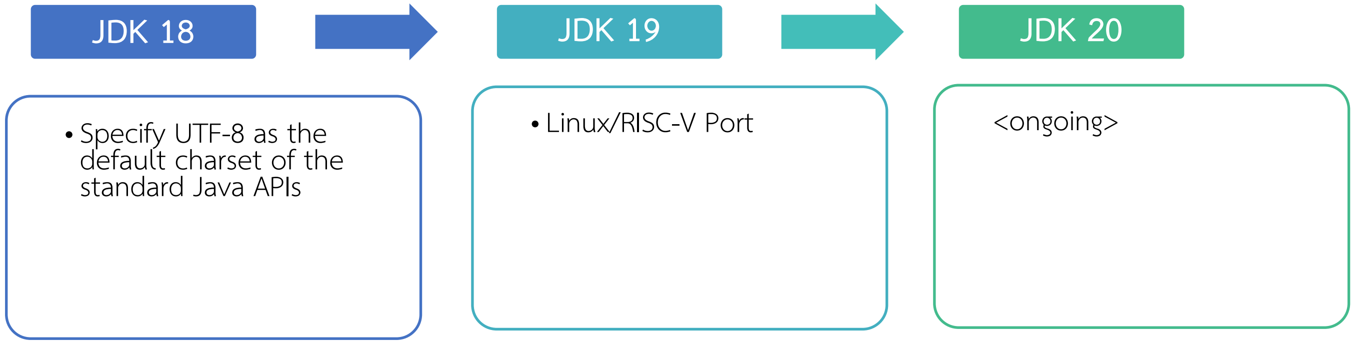
- Pattern Matching for instanceof
- Unix-Domain Socket Channels
- Records & Pattern Matching

JDK 17

- Sealed Classes

https://www.oracle.com/cis/java/technologies/downloads/

JDK Notable Features (ver.18-20)



JDK 18

- Specify UTF-8 as the default charset of the standard Java APIs

JDK 19

- Linux/RISC-V Port

JDK 20

<ongoing>

Getting Help

- https://docs.oracle.com/en/java/javase/17/index.html

Short History of Java

- Developed by James Gosling in 1991, the original name was "**Oak**".
  - Aimed for consumer electronics (TV, VCR, washing machine, etc.)
- Renamed to "Java" in 1994.
  - Java is a name of coffee.

- The first public implementation, Java 1.0, was released in 1995. It promised "Write Once, Run Anywhere".

ที่มา: https://en.wikipedia.org/wiki/James_Gosling

ที่มา: https://en.wikipedia.org/wiki/Java_(programming_language)

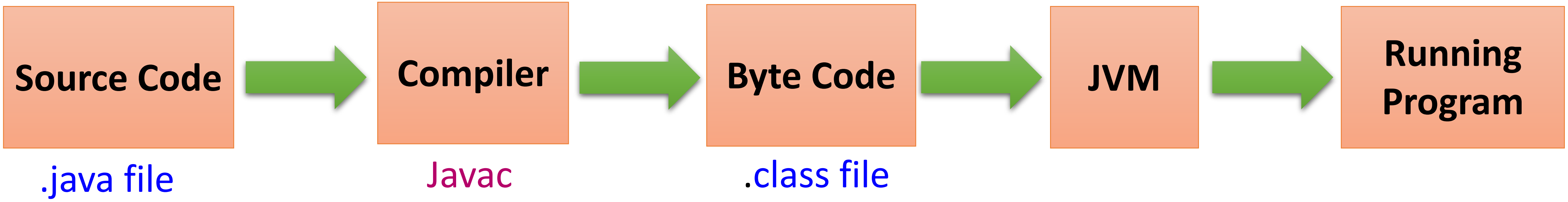
Java® Programming Language

- A general-purpose, concurrent, class- based, **object-oriented language**
- Strongly and statically typed programming language
- A relatively high-level language
  - Includes automatic storage management, typically using a garbage collector, to avoid the safety problems of explicit deallocation (as in C's free or C++'s delete).
  - High-performance garbage-collected implementations
  - Not include any unsafe constructs, such as array accesses without index checking
- Platform-neutral
  - The same program can run on any correctly implemented Java system

ที่มา: https://docs.oracle.com/javase/specs/jls/se11/

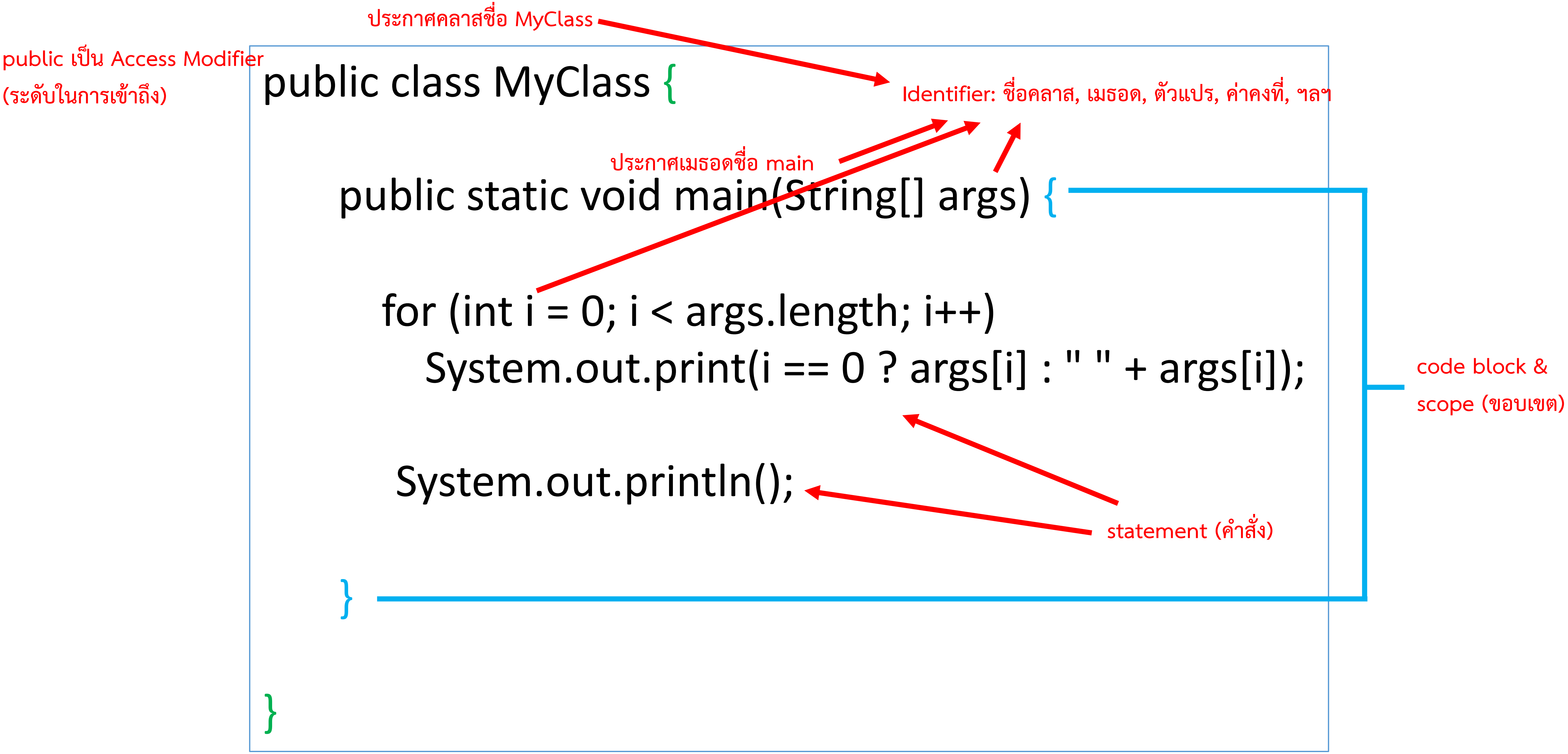
Running a Java Program

JVM: Java Virtual Machine



ภาพดัดแปลงจาก: https://www.oracle.com/java/technologies/security-in-java.html

MyClass.java



```
public class MyClass {

    public static void main(String[] args) {

        for (int i = 0; i < args.length; i++)
            System.out.print(i == 0 ? args[i] : " " + args[i]);

        System.out.println();

    }

}
```

ประกาศคลาสชื่อ MyClass

public เป็น Access Modifier

(ระดับในการเข้าถึง)

Identifier: ชื่อคลาส, เมธอด, ตัวแปร, ค่าคงที่, ฯลฯ

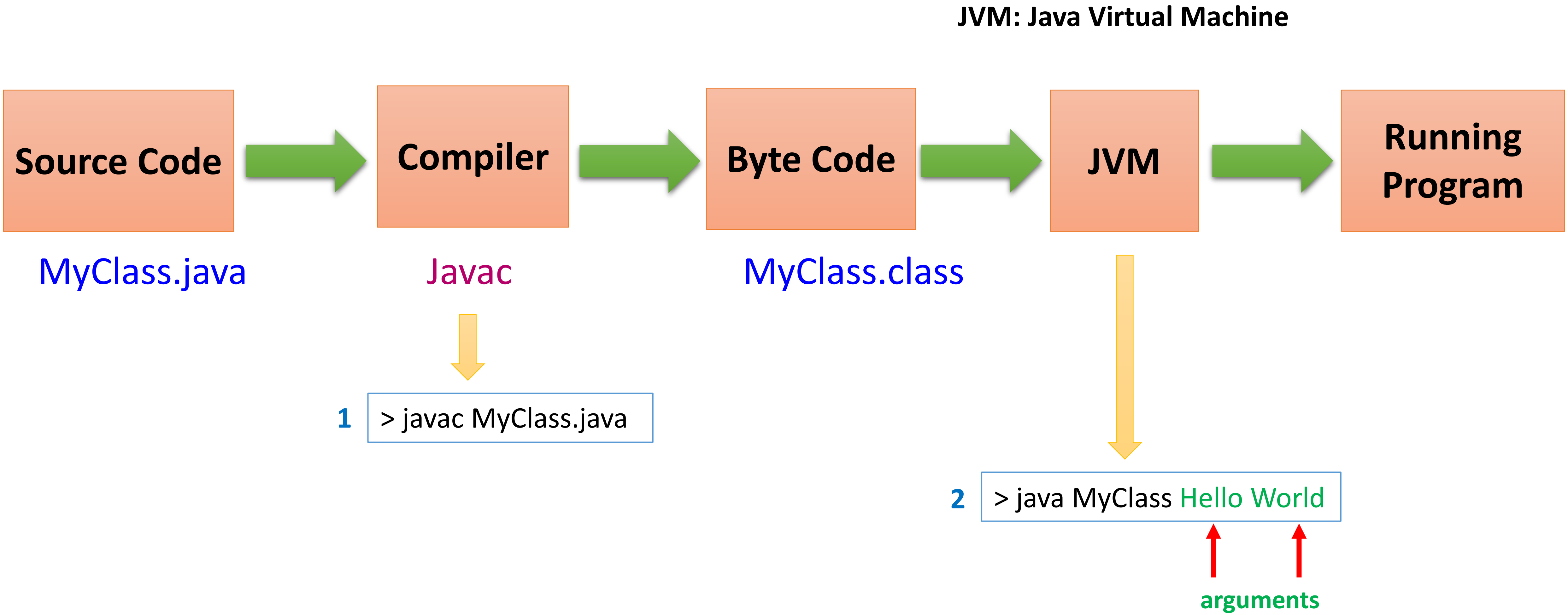
ประกาศเมธอดชื่อ main

code block &

scope (ขอบเขต)

statement (คำสั่ง)

Running a Java Program Using a Command Line Tool



JVM: Java Virtual Machine

Source Code → Compiler → Byte Code → JVM → Running Program

MyClass.java

Javac

MyClass.class

1

```
> javac MyClass.java
```

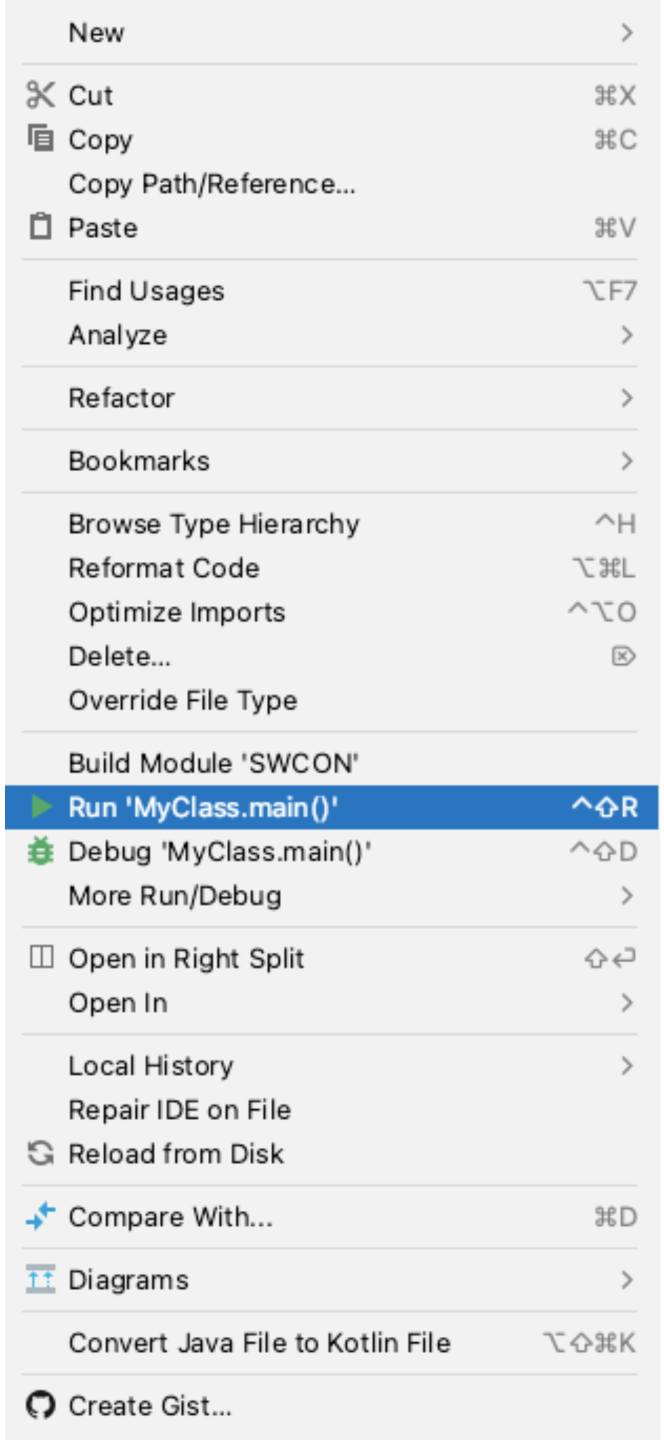
2

```
> java MyClass Hello World
```

arguments

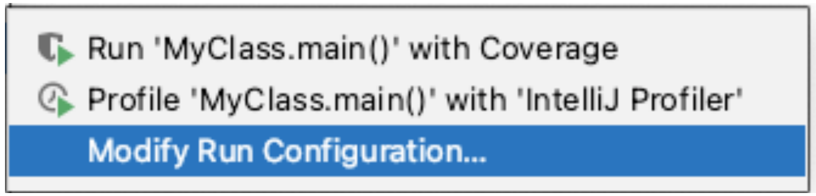
ภาพดัดแปลงจาก: https://www.oracle.com/java/technologies/security-in-java.html

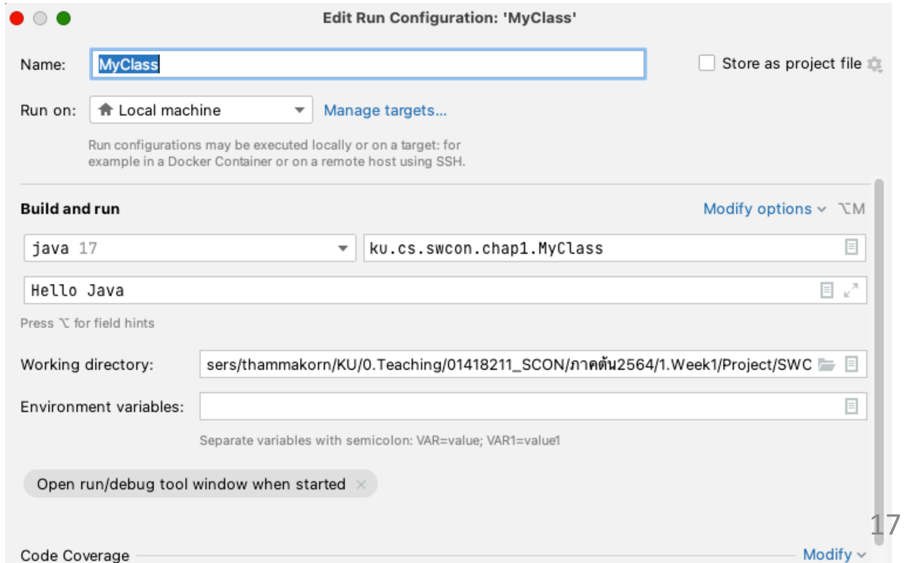
Running a Java Program Using IntelliJ IDEA®



Automatically compile and execute

For input arguments


Run 'MyClass.main()' with Coverage
Profile 'MyClass.main()' with 'IntelliJ Profiler'
Modify Run Configuration...



Edit Run Configuration: 'MyClass'

Name: MyClass

□ Store as project file

Run on: Local machine

Manage targets...

Run configurations may be executed locally or on a target: for example in a Docker Container or on a remote host using SSH.

Build and run

Modify options ⌥M

java 17

ku.cs.swcon.chap1.MyClass

Hello Java

Press ⌥ for field hints

Working directory: sers/thammakorn/KU/0.Teaching/01418211_SCON/ภาคต้น2564/1.Week1/Project/SWC

Environment variables:

Separate variables with semicolon: VAR=value; VAR1=value1

Open run/debug tool window when started

Code Coverage

Modify

JAR (Java ARchive)

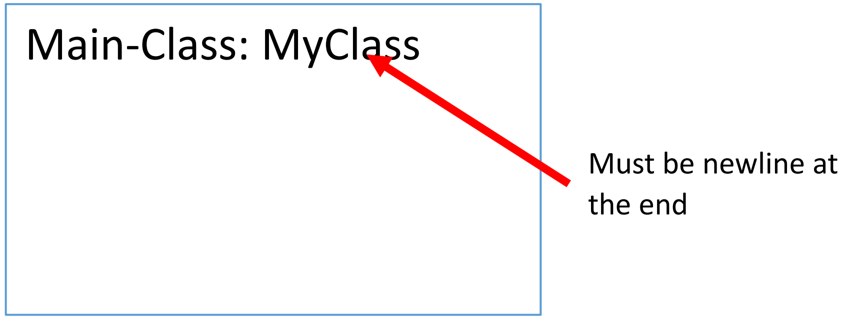
- A .jar file is a package file format typically used to aggregate many Java class files and associated metadata and resources into one file for distribution.

Create jar file using command line

Use custom MANIFEST file

```
> jar cvfm MyClass.jar MANIFEST.MF MyClass.class
```

MANIFEST.MF



```
Main-Class: MyClass
```

Must be newline at the end

OR

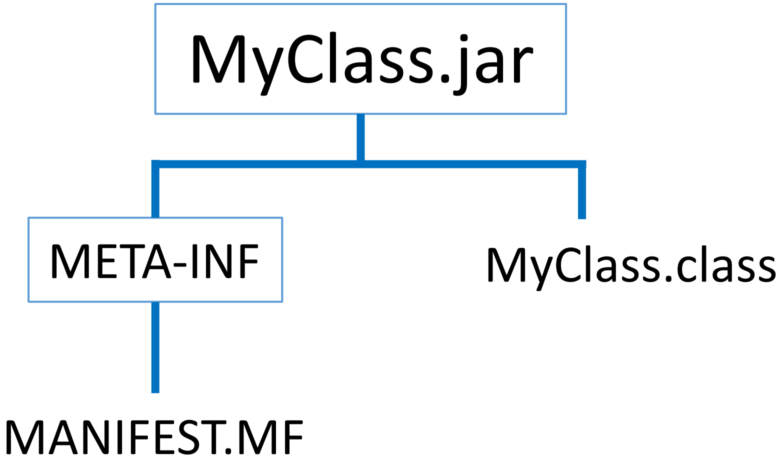
Use auto generated MANIFEST file

```
> jar cvfe MyClass.jar MyClass MyClass.class
```

Option	Description
v	Produces *verbose* output on stdout while the JAR file is being built. The verbose output tells you the name of each file as it's added to the JAR file.
c	Indicates that you want to create a JAR file.
f	Indicates that you want the output to go to a file rather than to stdout.
m	Used to include manifest information from an existing manifest file.

ที่มา: https://docs.oracle.com/javase/tutorial/deployment/jar/build.html

Inside jar file



MyClass.jar

META-INF

MyClass.class

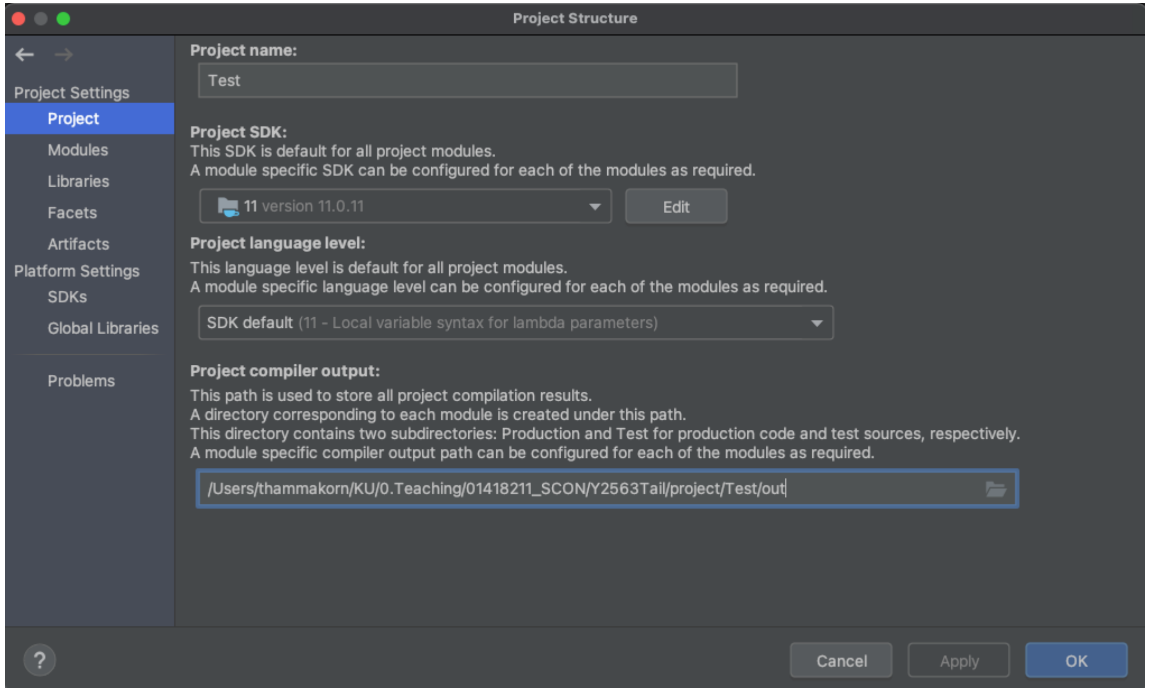
MANIFEST.MF

Running jar file

```
> java -jar MyClass.jar Hello World
```

Create jar file using IntelliJ IDEA®

Specify compilation output folders



Project name:

Test

Project SDK:

This SDK is default for all project modules.
A module specific SDK can be configured for each of the modules as required.

11 version 11.0.11

Project language level:

This language level is default for all project modules.
A module specific language level can be configured for each of the modules as required.

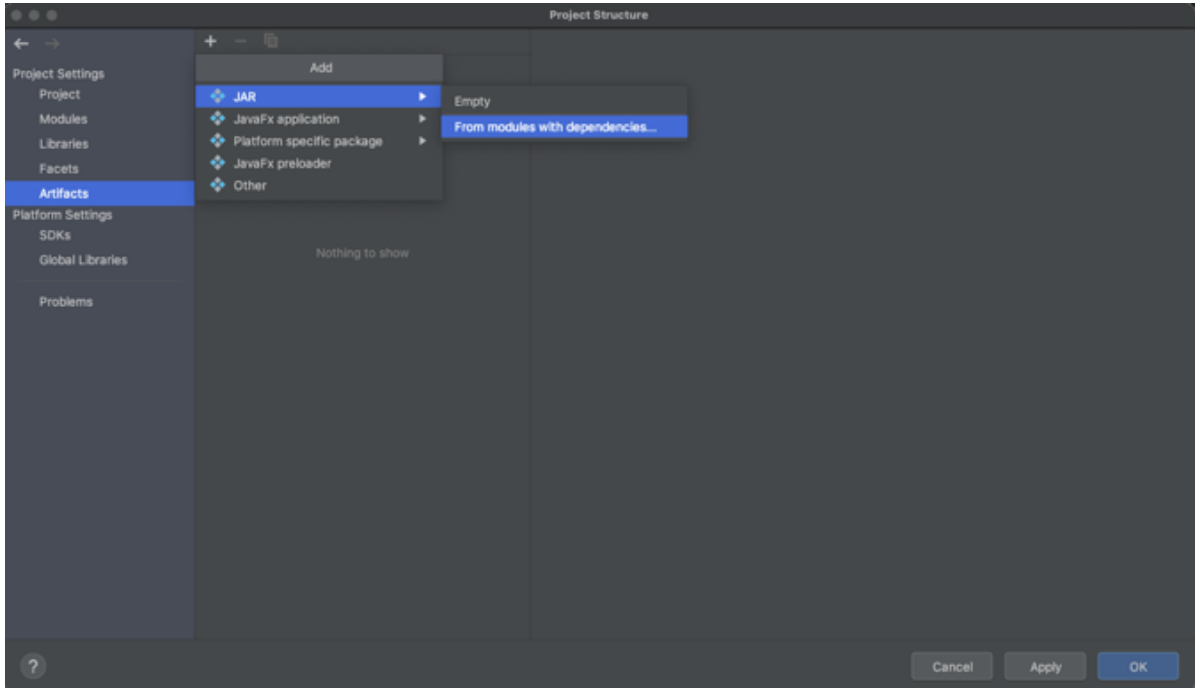
SDK default (11 - Local variable syntax for lambda parameters)

Project compiler output:

This path is used to store all project compilation results.
A directory corresponding to each module is created under this path.
This directory contains two subdirectories: Production and Test for production code and test sources, respectively.
A module specific compiler output path can be configured for each of the modules as required.

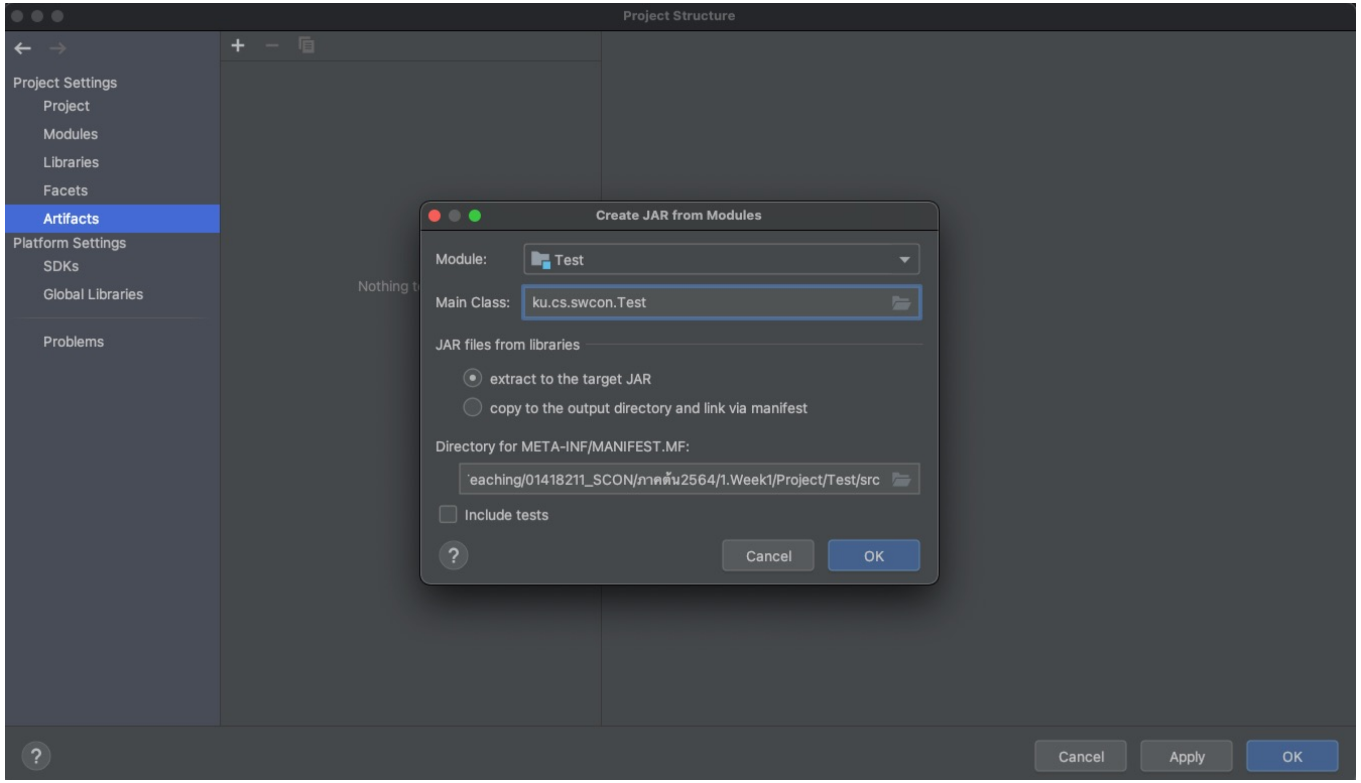
/Users/thammakorn/KU/0.Teaching/01418211_SCON/Y2563Tail/project/Test/out

Set artifact to jar



Create jar file using IntelliJ IDEA® (cont.)

Specify main class



Module: Test

Main Class: ku.cs.swcon.Test

JAR files from libraries

● extract to the target JAR

○ copy to the output directory and link via manifest

Directory for META-INF/MANIFEST.MF:

'eaching/01418211_SCON/ภาคต้น2564/1.Week1/Project/Test/src

□ Include tests

Basic Java Programming Components

- Classes are the fundamental building blocks of Java programs:

  ```
  public class MyClass { … }
  ```

- Java applications contain a class with a **main method**

  When the application starts, the instructions in the main method are executed

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

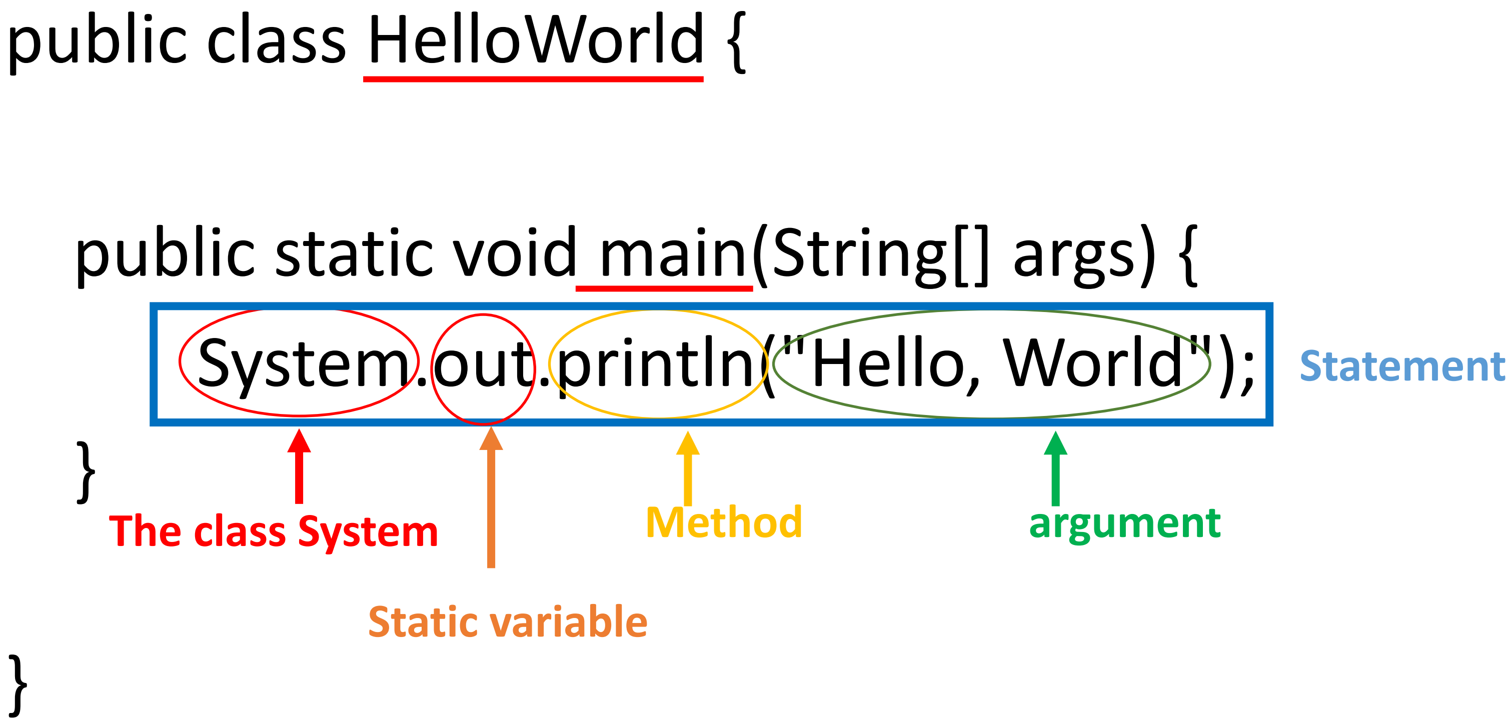
Hello World!

C++

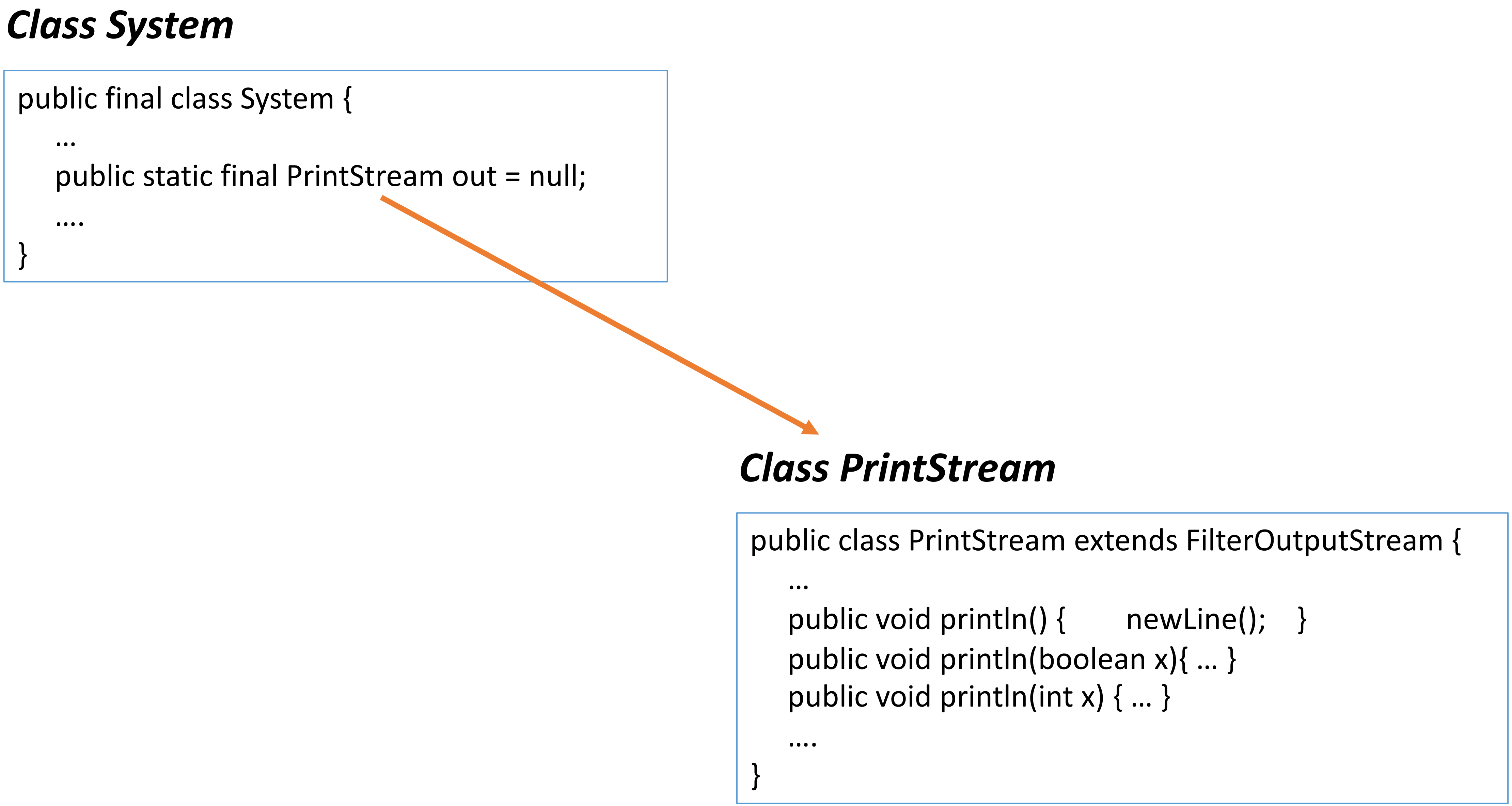
```cpp
#include <iostream>
using namespace std;

int main()
{
    cout << "Hello, World!";
    return 0;
}
```

Java

```java
public class HelloWorld {

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

}
```



```
public class HelloWorld {

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

}
```



Class System

```
public final class System {
    …
    public static final PrintStream out = null;
    ….
}
```

Class PrintStream

```
public class PrintStream extends FilterOutputStream {
    …
    public void println() {      newLine();    }
    public void println(boolean x){ … }
    public void println(int x) { … }
    ….
}
```

Input Elements in Java Programming

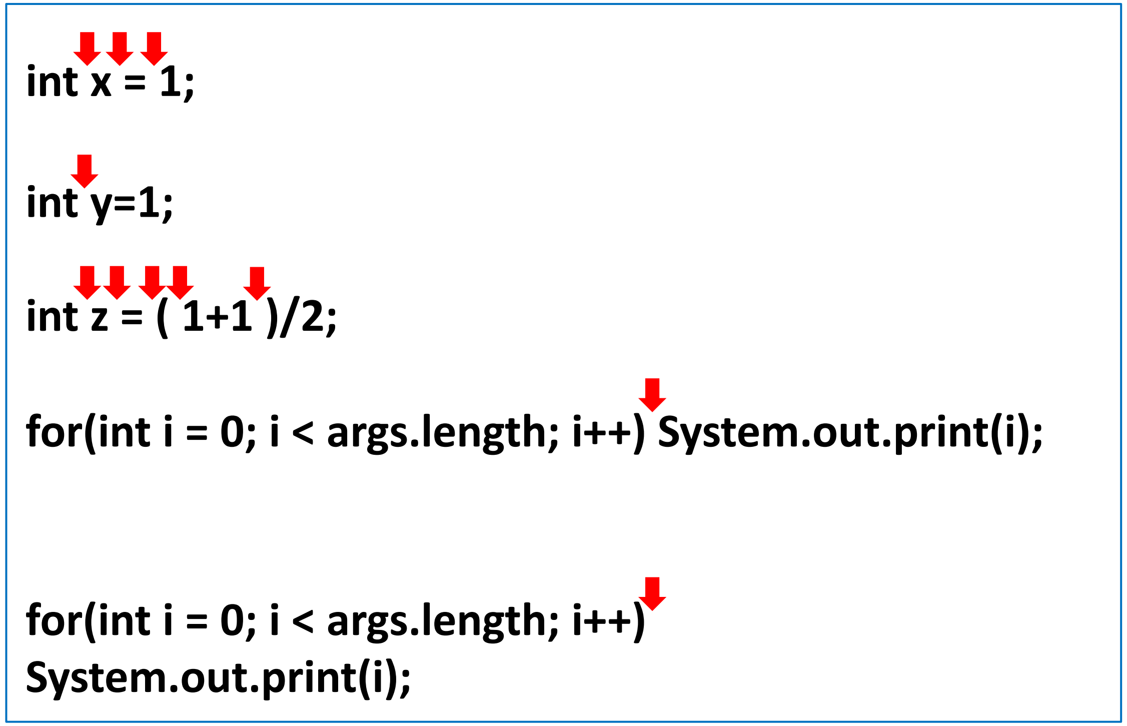
- Input Elements
  - Token
  - White Space
  - Comment

White Space

- “space”

White Space	Ex. Unicode
space	\u0020
horizontal tab	\u0009
line feed	\u000A

etc.



```
int x = 1;

int y=1;

int z = ( 1+1 )/2;

for(int i = 0; i < args.length; i++) System.out.print(i);


for(int i = 0; i < args.length; i++)
System.out.print(i);
```

Comment

- Single-line Comments
  - All the text from the characters // to the end of the line is ignored (as in C++).

```
// comment text
```

- Multi-line Comments
  - All the text from the characters /* to the ASCII characters */ is ignored (as in C and C++).

```
/* comment text line 1
   comment text line 2 */
```

```
//declare variable x
int x = 1;

int y=1; //declare variable y

/*
declare variable z
and initialize value using x and y
*/
int z = ( x+y )/2;
```

Statement

- A statement forms a complete unit of execution
- Java statements appear inside of methods and classes (appear within a code block ({ })
  - Describe all activities of a Java program such as variable declarations and assignments.
  - Each statement usually ends with semicolon (;)

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

    // variable declaration statement
    int x ;

    // variable assignment statement
    x = 1;

    // variable increment statement
    x++;

    // method invocation statement
    System.out.println("Hello, World");

   // object creation statement
   Object o = new Object();
}
```

ที่มา: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html
https://www.oreilly.com/library/view/learning-java/1565927184/ch04s04.html

Statement (cont.)

There are various kinds of statements in the Java programming language

- Declaration Statement
- Labeled Statement
- If Statement
- If-Else Statement
- While Statement
- Do Statement
- For Statement
- Break Statement
- Continue Statement
- Return Statement
- Synchronized Statement
- Throw Statement
- Try Statement
- Block Statement
- Empty Statement
- Expression Statement
- Assert Statement
- Switch Statement
- etc.

Expression

- An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language.
- An expression produces a result, or value, when it is evaluated
  - A numeric type, as in an arithmetic expression
  - A reference type, as in an object allocation
  - A special type void, which is the declared type of a method that doesn’t return a value

ที่มา: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html
https://www.oreilly.com/library/view/learning-java/1565927184/ch04s04.html

Expression (cont.)

- While variable initialization (i.e., declaration and assignment together) is considered a statement, with no resulting value, variable assignment alone is also an expression:

```
public class ExampleClass{
   public static void main(String[] args) {

      int i,j;            // statement
      j = 5;   1 Ac       // both expression and statement
      j += (i = (j-3)); //?
      System.out.println("j = " + j);   //? → Both

   }
}
```

ทุกตัวที่ให้ค่า ออกมา = Expression Ex. j=5 j มีค่า 5

ถ้ามี ; อยู่ท้าย = Statement

j-3

i = (j-3)

also expression

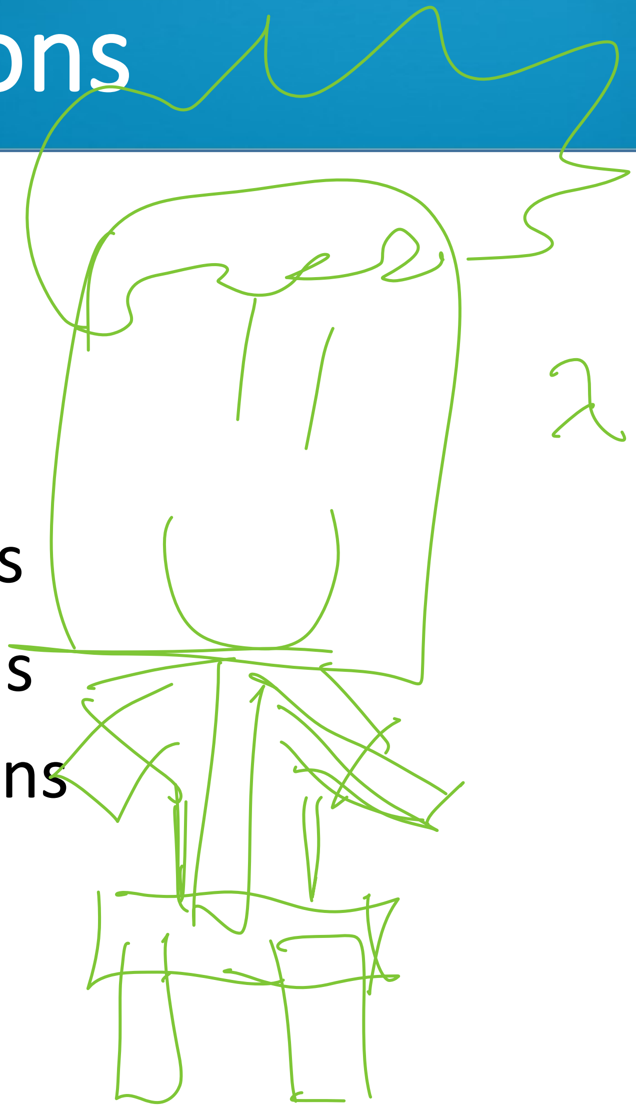
```
public class ExampleClass{
  public static void main(String[] args) {
      int x = 1;
      if(x>0){  // x>0 is boolean expression
         System.out.println("x = " + x);
      }

  }
}
```

```
if ( condition )
   statement;
[ else
   statement; ]
```

ที่มา: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html
https://www.oreilly.com/library/view/learning-java/1565927184/ch04s04.html

Forms of Expressions

- Expression names
- Primary expressions*
- Unary operator expressions
- Binary operator expressions
- Ternary operator expressions
- Postfix / Prefix expressions
- Lambda expressions



ที่มา: https://docs.oracle.com/javase/specs/jls/se11/

Expression Statements

Certain kinds of expressions may be used as statements by following them with semicolons.

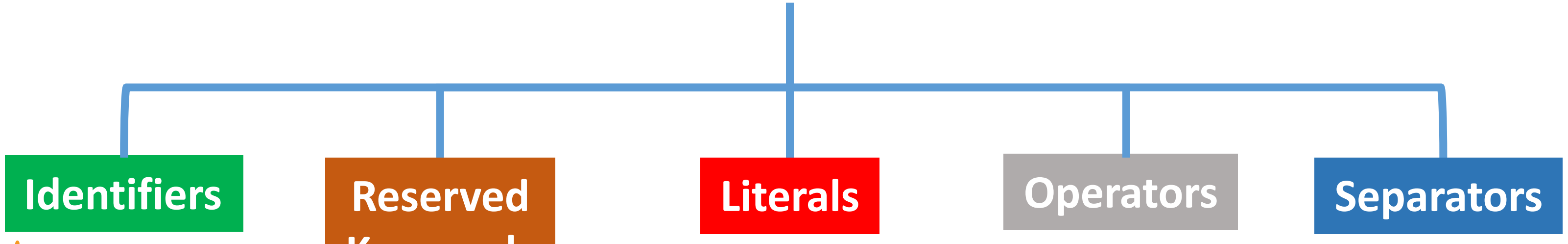
An expression statement is executed by evaluating the expression; if the expression has a value, the value is discarded.

- Assignment Expression
  - x = 1;
- Pre-Increment Expression
  - ++x;
- Pre-Decrement Expression
  - --x;
- Post-Increment Expression
  - x++;
- Post-Decrement Expression
  - x--;
- Method Invocation Expression
  - Math.sqrt(4); Class ของ Math แล้ว Method sqrt ค่าตอบได้ 2.0
- Class Instance Creation Expression
  - new Object();
  - new String("ABC");

Tokens in Java Programming

- Token
  - Identifier
  - Keyword
  - Literal
  - Separator
  - Operator

Tokens in Java Statement



Identifiers: ใช้ชื่อตัวแปร, Method และ Class

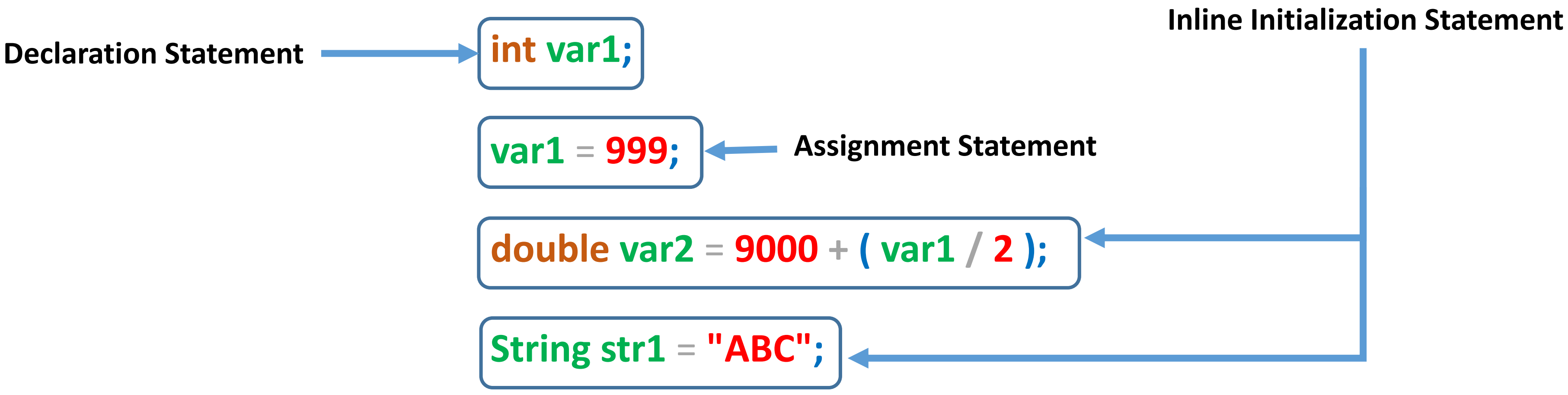
Reserved Keywords: ใช้คำสั่งๆที่สงวน เช่น Int ฯลฯ เอามา ทำเป็นตัวแปร

Literals: ใช้ค่าคงที่ เช่น ตอน Assign ค่า

Operators: ใช้ดำเนินการ

Separators: ใช้ตัวอักษรพิเศษ ประโยคระหว่าง
Ex. (;) (,) (:)

*every statement in Java ends with ;



Declaration Statement

```
int var1;
```

Assignment Statement

```
var1 = 999;
```

Inline Initialization Statement

```
double var2 = 9000 + ( var1 / 2 );
String str1 = "ABC";
```

```
System.out.println("Hello, World");
```

Identifiers

- Naming of:
  - class
  - variable
  - constant
  - method
  - interface
  - package
  - enum
- Are case sensitive

- *Must not* be
  - Keywords or boolean literal (true / false) or null literal (null)
  - single underscore (_ , \u005f)
- *Should not* be
  - var (type inference)
    - Also, it is illegal to declare a class named var
  - $ (involve in source code generation)

ที่มา: https://docs.oracle.com/javase/specs/jls/se11/

Identifiers (cont.)

- More rules for identifiers in Java:
  - *Can be made up of letters, digits, and the underscore* (_)

    **Cannot be single underscore*
  - *Cannot start with a digit*
  - *Cannot use other symbols such as ? or %*
  - *White spaces are not permitted inside identifiers*

- By convention …
  - Variable/method names start with a lowercase letter

    *"**Camel Case**": Capitalize the first letter of a word in a compound word such as*

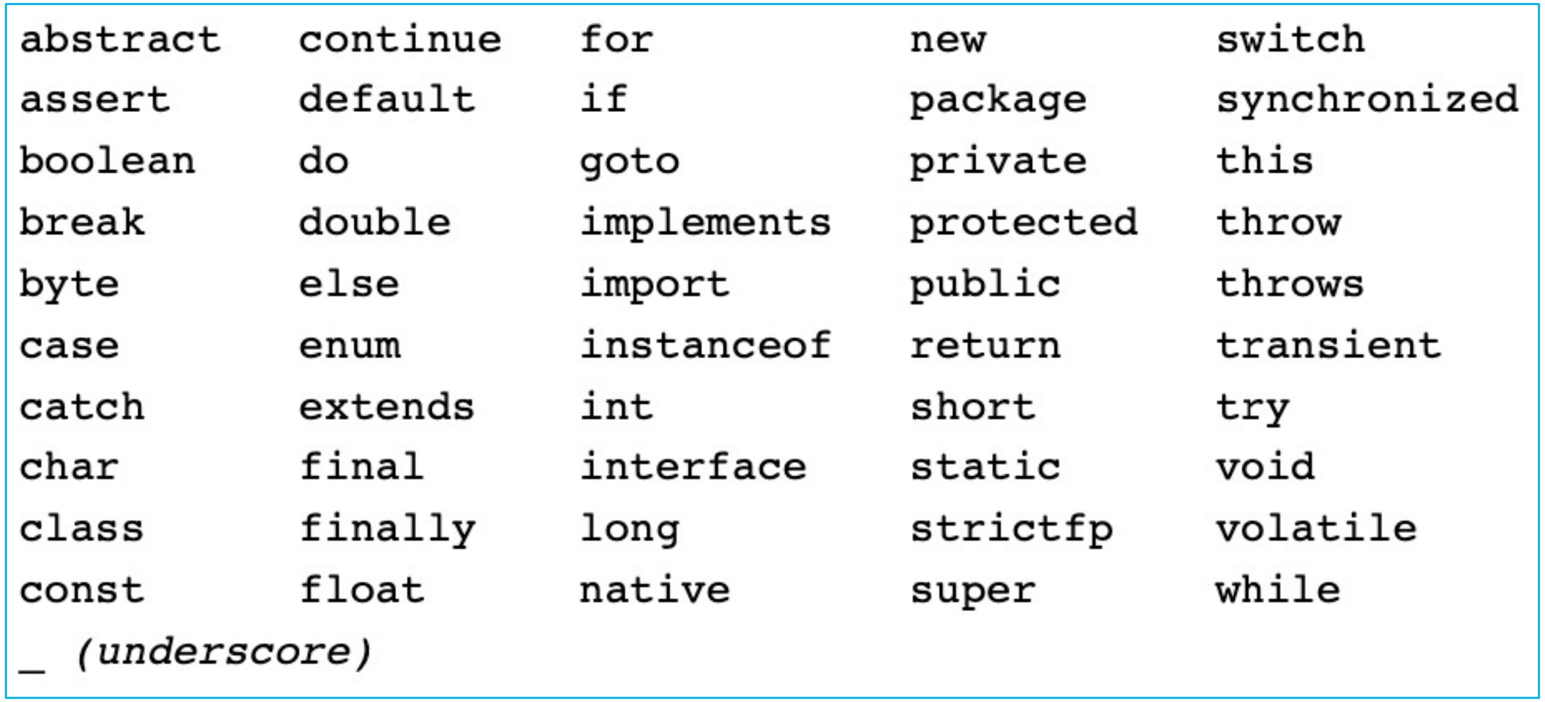
    *account**N**ame, job**S**tatus,* farewell**M**essage
  - Class names start with an uppercase letter

    ***H**ello**W**orld*
  - Capitalize all letters for constant

    public static final double **PI** = 3.141592;

Java Reserved Keywords

***keywords cannot be used as identifiers



```
abstract   continue   for          new         switch
assert     default    if           package     synchronized
boolean    do         goto         private     this
break      double     implements   protected   throw
byte       else       import       public      throws
case       enum       instanceof   return      transient
catch      extends    int          short       try
char       final      interface    static      void
class      finally    long         strictfp    volatile
const      float      native       super       while
_ (underscore)
```

- true and false are not keywords, but rather boolean literals
- null is not a keyword, but rather the null literal
- var* is not a keyword, but rather an identifier with special meaning as the type of a local variable declaration and the type of a lambda formal parameter
  *ไม่ควรใช้เป็น identifier

Avoid using open, module, requires, transitive, exports, opens, to, uses, provides, or with as an identifier.

ที่มา: https://docs.oracle.com/javase/specs/jls/se11/

ข้อใดตั้งชื่อ **class** ไม่เหมาะสม

ก) User ข) Room ค) 2BedRoom ง) Letter

มีตัวเลขข้างหน้า
ห้ามใช้

ข้อใดตั้งชื่อ ตัวแปร ไม่เหมาะสม

เป็นคำพิเศษ

ก) username ข) numRoom ค) Count ง) super จ) return

พิมพ์ใหญ่

เป็น reserved keyword

Literals

- The source code representation of a value of a primitive type, the String type, or the null type

- Integer literal
  - e.g., 0, 1, -1258, 9865745L, 0X001, 0x7fff_ffff, 0x31, 1_000_000

long int

ฐาน 16

1 ล้าน

- Floating point literal
  - e.g., 0.5, -0.0, 123.59999, 0.0000001, 0.3F

เพราะมีจุด เลยเรียก float

ถ้า 0.3 = double / 0.3F คือ float แต่ Java จะมองว่าเป็น double

- Boolean literal
  - true, false

- Character literal
  - e.g., 'A', 'a', '1', '%', \u0000, \uffff

Java มอง Type char ว่า int

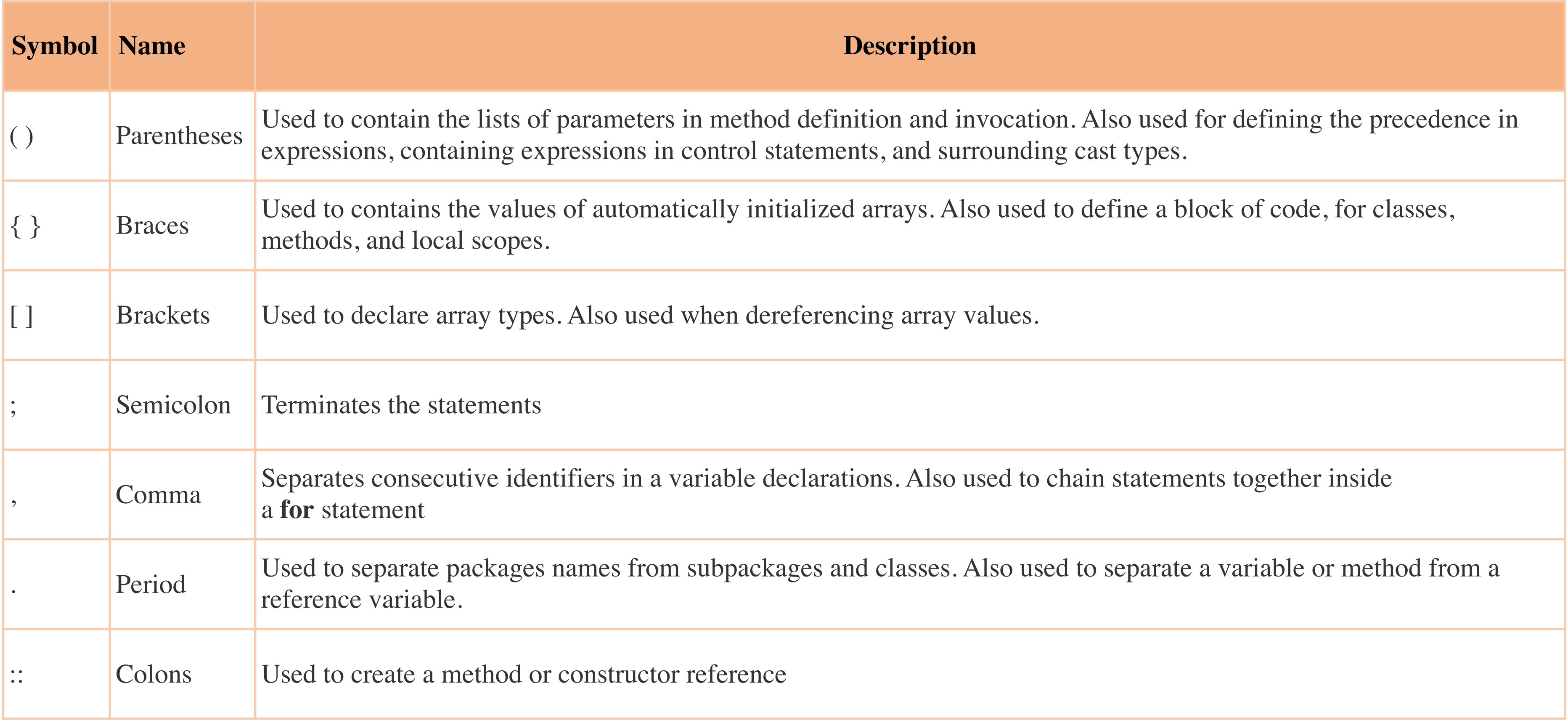
- StringLiteral
  - e.g., "Text", "version", "123", "", "\t", "\n"

ก็ string ว่า

- Null literal
  - null

see primitive type

Separators



Symbol	Name	Description
( )	Parentheses	Used to contain the lists of parameters in method definition and invocation. Also used for defining the precedence in expressions, containing expressions in control statements, and surrounding cast types.
{ }	Braces	Used to contains the values of automatically initialized arrays. Also used to define a block of code, for classes, methods, and local scopes.
[ ]	Brackets	Used to declare array types. Also used when dereferencing array values.
;	Semicolon	Terminates the statements
,	Comma	Separates consecutive identifiers in a variable declarations. Also used to chain statements together inside a **for** statement
.	Period	Used to separate packages names from subpackages and classes. Also used to separate a variable or method from a reference variable.
::	Colons	Used to create a method or constructor reference

ที่มา: https://codescracker.com/java/java-separators.htm

Operators

Simple Assignment Operator

= Simple assignment operator

Arithmetic Operators

+ Additive operator (also used for String concatenation)

- Subtraction operator

* Multiplication operator

/ Division operator

% Remainder operator

Unary Operators

+ Unary plus operator; indicates positive value (numbers are positive without this, however)

- Unary minus operator; negates an expression

++ Increment operator; increments a value by 1

-- Decrement operator; decrements a value by 1

! Logical complement operator; inverts the value of a boolean

Equality and Relational Operators

== Equal to

!= Not equal to

> Greater than

>= Greater than or equal to

< Less than

<= Less than or equal to

Conditional Operators

&& Conditional-AND

|| Conditional-OR

?: Ternary (shorthand for if-then-else statement)

Type Comparison Operator

instanceof Compares an object to a specified type

Bitwise and Bit Shift Operators

~ Unary bitwise complement

<< Signed left shift

>> Signed right shift

>>> Unsigned right shift

& Bitwise AND

^ Bitwise exclusive OR

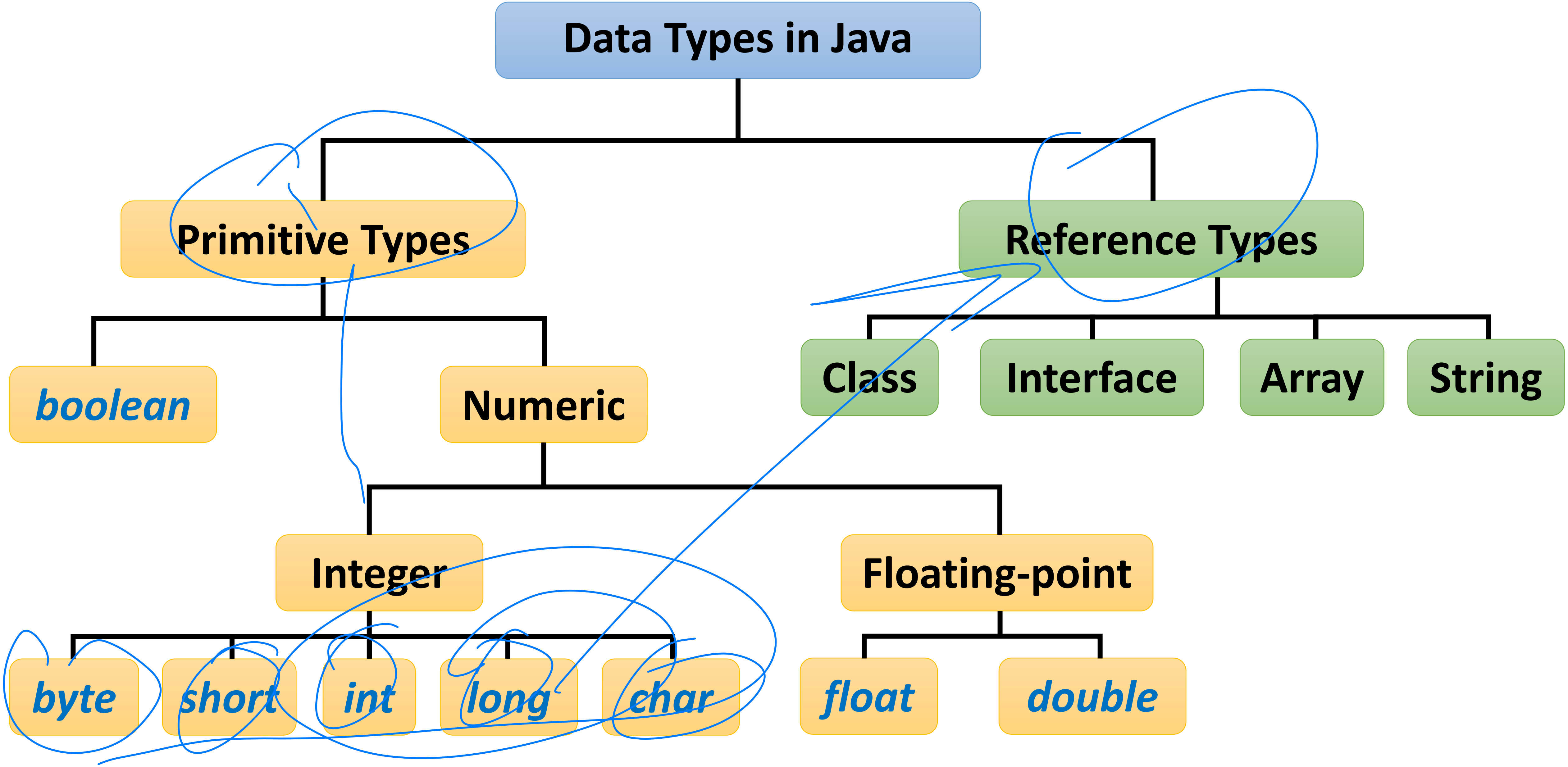
| Bitwise inclusive OR

ที่มา: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html

Types, Values, and Variables

- Java programming language is a *strongly typed* language
  - limit the values that a variable can hold or that an expression can produce,
  - limit the operations supported on those values and determine the meaning of the operations.
  - Strong static typing helps detect errors at compile time.
- The types of the Java programming language are divided into two categories: primitive types and reference types
  - The primitive types are the boolean type and the numeric types.
  - The reference types are class types, interface types, and array types
- There is also a special null type
  - In practice, the programmer can ignore the null type and just pretend that null is merely a special literal that can be of any reference type.

ที่มา: https://docs.oracle.com/javase/specs/jls/se11/


Data Types in Java
Primitive Types
Reference Types
boolean
Numeric
Class
Interface
Array
String
Integer
Floating-point
byte
short
int
long
char
float
double

Primitive & Reference Types

- Two kinds of data values that can be
    - Stored in variables
    - Passed as arguments
    - Returned by methods
    - Operated on.

Java Primitive Data Types

1 B > 8

Type	Description	Size
int	The integer type, with range -2,147,483,648 . . . 2,147,483,647	4 bytes
byte	The type describing a single byte, with range -128 . . . 127	1 byte (8 bit)
short	The short integer type, with range -32768 . . . 32767	2 bytes
long	The long integer type, with range -9,223,372,036,854,775,808 . . . -9,223,372,036,854,775,807	8 bytes
double	The double-precision floating-point type, with a range of about ±10^{308} and about 15 significant decimal digits	8 bytes
float	The single-precision floating-point type, with a range of about ±10^{38} and about 7 significant decimal digits	4 bytes
char	The character type, representing code units in the Unicode encoding scheme from '\u0000' to '\uffff'	2 bytes 16bit
boolean	The type with the two truth values false and true	1 bit

see literals

ที่มา: Horstmann CS. Big Java. John Wiley & Sons; 2016

Default Values for Primitive Types

It's not always necessary to assign a value when an **attribute / field / Instance variable** is declared.

Attributes that are declared but not initialized will be set to a reasonable default by the compiler.

Data Type	Default Value (for fields)
byte	0
short	0
int	0
long	0L
float	0.0f
double	0.0d
char	'\u0000'
String (or any object)	null
boolean	false

Not for local variable!!!

ที่มา: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/datatypes.html

Assign null to primitive & reference types

```
int i = null;
```

```
Integer i = null;
```

```
String s = null;
```

```
Object o = null;
```

Attribute/Field/Instance vs. Local variable

```
public class LocalAttr {
    int attr;

    public static void main(String[] args) {
        LocalAttr la = new LocalAttr();
        System.out.println(la.attr);


        int local;
        System.out.println(local);
    }
}
```

int attr & int local

ต่างกัน!!!

** บทเรียนช่วงแรกจะเน้น local variable **

Primitive Type Variable Declaration

<data type> <identifier> [= value]

```
int x;
x = 1;
```

```
int x = 1;  //inline initialization
```

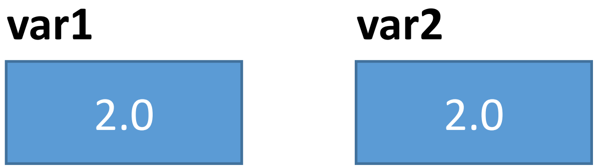
```
int a,b,c;
```

```
int a=1, b=2, c=3; //inline initialization
```

Primitive Type Variable

Primitive type variable: store values

double var1 = 2.0;
double var2 = var1



var1

2.0

var2

2.0

Var2 = 4.0
Var1 = ?

Reference Type Variable

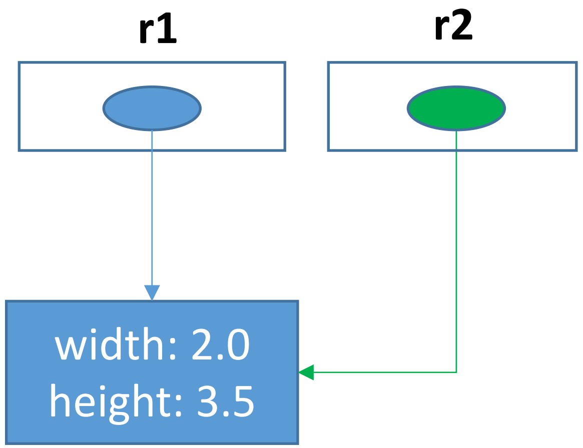
- **Object reference:** describes the location of an object
- The new operator returns a reference to a new object:

```
Rectangle r1 = new Rectangle(2.0, 3.5);
```

- Multiple object variables can refer to the same object:

```
Rectangle r2 = r1;
r2.setWidth(4.0);
System.out.println(r1.getWidth());
```

- Primitive type variables ≠ reference variables



ที่มา: Horstmann CS. Big Java. John Wiley & Sons; 2016

```
public class Rectangle {
    private double width;
    private double height;

    public Rectangle(double width, double height) {
        this.width = width;
        this.height = height;
    }
    public void setWidth(double width) {
        this.width = width;
    }
    public double getWidth() {
        return width;
    }
}
```

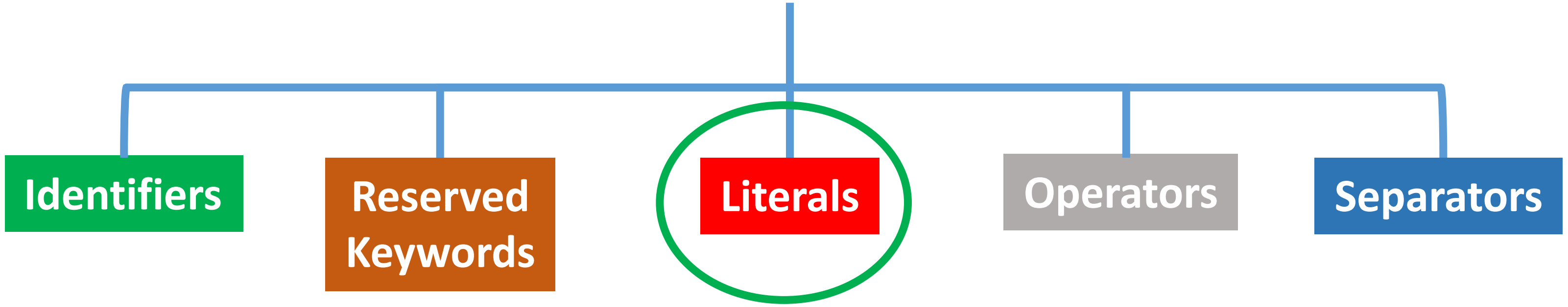
1. (2 คะแนน) เลือกทุกข้อที่เป็น primitive type (เลือกผิดตัดข้อละ 1 คะแนน จน 0)

a. System ✗
b. Double ✗
c. byte ✓
d. Short ✗
e. Class ✗
f. String ✗
g. Char ✗
h. long ✓
i. float ✓
j. primitive ✗
k. Object ✗
l. boolean ✓

Literals & Primitive Type

- In Java programming language, literals can be of primitive data types. The way each literal is represented depends upon it type.

Tokens in Java Statement



Identifiers

Reserved Keywords

Literals

Operators

Separators

*every statement in Java ends with ;

```
int var1;

var1 = 999;

double var2 = 9000 + ( var1 / 2 );

String str1 = "ABC";

System.out.println("Hello, World");
```

Types of Literals

- **Integer literals:** these are used to represent integer values, and can be written in decimal, hexadecimal, or octal notation (e.g. 1, 0, -100).
- **Floating-point literals:** these are used to represent real numbers, and can be written in decimal or scientific notation (e.g. 0.5, -100.0, 2e-1).
- **Boolean literals:** these represent boolean values of either "true" or "false".
- **Character literals:** these are used to represent a single character, and are enclosed in single quotes (e.g. 'A').
- **String literals:** these are used to represent a sequence of characters, and are enclosed in double quotes (e.g. "Hello, world!").

float x = 0.0f;
(บอกว่า x เป็น float)

float x = 0.0f+1;
(ผลลัพธ์เป็น float)

double x = 0.0+1;
(ผลลัพธ์คือ double = 1.0)

Java Int/Int = Int
(หารลงตัว)

(ถ้าไม่ลงตัว เช่น 1/2 จะได้ 0)

(double/Int = double)
1.0 / 2 = 0.5

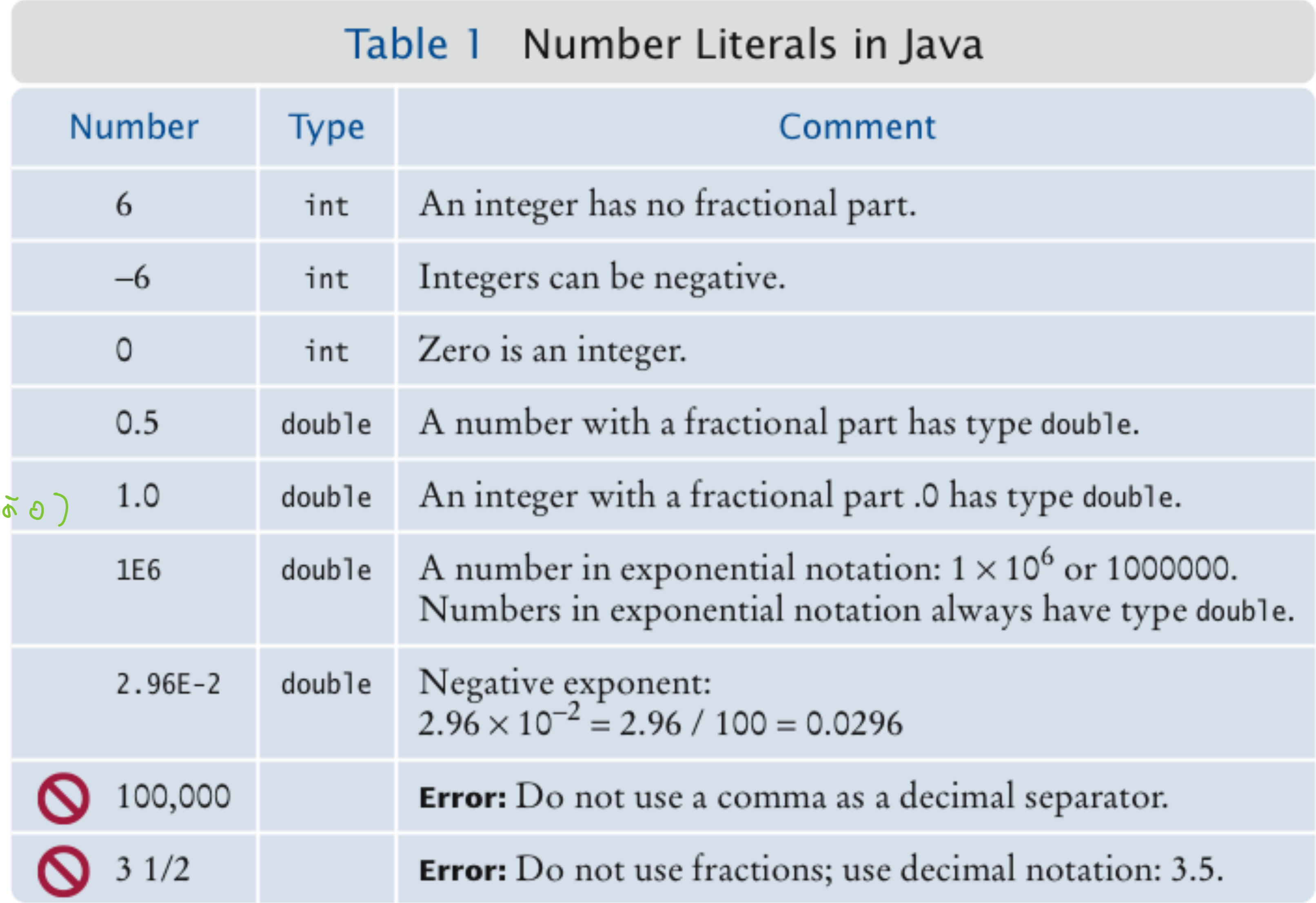


Table 1 Number Literals in Java

Number	Type	Comment
6	int	An integer has no fractional part.
–6	int	Integers can be negative.
0	int	Zero is an integer.
0.5	double	A number with a fractional part has type double.
1.0	double	An integer with a fractional part .0 has type double.
1E6	double	A number in exponential notation: 1×10^6 or 1000000. Numbers in exponential notation always have type double.
2.96E-2	double	Negative exponent: $2.96 \times 10^{-2} = 2.96 / 100 = 0.0296$
🚫 100,000		**Error:** Do not use a comma as a decimal separator.
🚫 3 1/2		**Error:** Do not use fractions; use decimal notation: 3.5.

ที่มา: Horstmann CS. Big Java. John Wiley & Sons; 2016

```
int x = 0;

int x = 0.0;   ❌

double x = 0;

double x = 0.0;
```

```
float f = 2.5;   ❌
```

```
float f = 2.5f;
```

```
double d = 2.5;
float y = d/2;  ❌
```

```
int x  = 2147483648;
```
❌ over flow

```
int x  = 2147483647+1;
```
over flow

```
System.out.println(x);  //what value will be printed
```

byte && Short

สามารถ Assigned Int

ได้

```
long x  = 2147483647+1;
```
ตัวค่ามันยังเป็น int อะ ตอนบวก ก็ int อยู่

```
System.out.println(x); //and this ?
```
จึงเกิด Overflow

```
long x  = 2147483647L+1;
```
อันนี้ได้ เพราะว่า Type เป็น long + int

```
System.out.println(x); //and this ?
```
ผลลัพธ์ได้ long

```
long x = 100;
long x = 7890000000000;
```
❌ เพราะว่าเลข เกิน int ไม่ได้เติม L ข้างหลังด้วย

ก็ เรียบร้อย แตก Over flow

```
long x = 7890000000000L;
```

Java Primitive Data Types

Type	Description	Size
int	The integer type, with range -2,147,483,648 . . . 2,147,483,647	4 bytes
byte	The type describing a single byte, with range -128 . . . 127	1 byte (8 bit)
short	The short integer type, with range -32768 . . . 32767	2 bytes
long	The long integer type, with range -9,223,372,036,854,775,808 . . . -9,223,372,036,854,775,807	8 bytes
double	The double-precision floating-point type, with a range of about $\pm 10^{308}$ and about 15 significant decimal digits	8 bytes
float	The single-precision floating-point type, with a range of about $\pm 10^{38}$ and about 7 significant decimal digits	4 bytes
char	The character type, representing code units in the Unicode encoding scheme from '\u0000' to '\uffff'	2 bytes
boolean	The type with the two truth values false and true	1 bit

see literals

ที่มา: Horstmann CS. Big Java. John Wiley & Sons; 2016

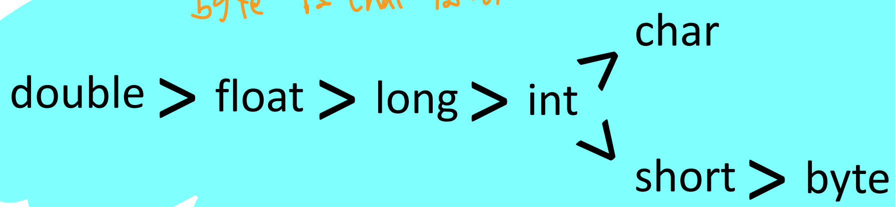
```
float a = 5.59f;

           double
int b = 5.59; ❌

int c = (int) 5.9;  //keep 5



double balance = 13.75;

int amount1 = balance;❌ double 转 int

int amount2 = (int) balance;  // OK ⇒ 13
```

Subtyping Among Primitive Types

* เหตุผลที่ char ต้องแยกจาก short และ byte
เพราะ UTF-8 (ไปถาม Dev) เอา

Type	Subtype
double	float
float	long
long	int
int	char
int	short
short	byte

byte จะ char ไม่ได้



```
char c1 = 'c';
short s1 = 1;
byte b1 = 0;

int int1 = c1;
int int2 = s1;
int int3 = b1;

char c1 = s1;  ✗
short s1 = c1; ✗

short s2 = b1;
char c2 = b1;  ✗
```

ที่มา: https://docs.oracle.com/javase/specs/jls/se11/

Cautions When Performing Operations on Integers

```
public class Test {
        public static void main(String[] args) {
          int i = 1000000;
          System.out.println(i * i);        ⟵  too large for int

          long l = i;
          System.out.println(l * l);
          System.out.println(20296 / (l - i));   ⟵  ArithmeticException
        }                                  error เพราะหารด้วย 0
}
```

Cautions When Performing Operations on Floating-Points

- Overflow

```
public class Test {
   public static void main(String[] args) {

      double d = 1e308;
      System.out.print("overflow produces infinity: ");
      System.out.println(d + "*10 == " + d * 10);

   }
}
```

1×10^{308}

ผลลัพธ์ Infinity

- Underflow

```
public class Test {
   public static void main(String[] args) {

      double d = 1e-305 * Math.PI;
      System.out.print("gradual underflow: " + d + "\n   ");
      for (int i = 0; i < 4; i++)
         System.out.print(" " + (d /= 100000));
       System.out.println();
   }
}
```

เมื่อหารจนเกินขอบเขตของ double แบบน้อยๆๆ มากๆ

จนเกินขอบเขต จะได้ 0.0

Cautions When Performing Operations on Floating-Points (cont.)

- Inexact results with float

```
public class Test {
   public static void main(String[] args) {

      for (int i = 1; i < 100; i++) {
         float z = 1.0f / i;
         if (z * i != 1.0f)
            System.out.print(" " + i);
      }
      System.out.println();
   }
}
```

(1.0 x49)/49

- Inexact results with double

```
public class Test {
   public static void main(String[] args) {

      for (int i = 1; i < 100; i++) {
         double z = 1.0 / i;
         if (z * i != 1.0)
            System.out.print(" " + i);
      }
      System.out.println();
   }
}
```

Overflow in evaluation

```
public static void main(String[] args) {
   double d = 8e+307;
   System.out.println(4.0 * d * 0.5); infinity
   System.out.println(4.0 * (d * 0.5)); print ได้
   System.out.println(2.0 * d); ได้
}
```

Underflow in evaluation

```
public static void main(String[] args) {
      double d = 8e-307;
      System.out.println(1e-20 * d * 1e+20); ⇒ 0
      System.out.println(1e-20 * (d * 1e+20)); ⇒ 1
}
```

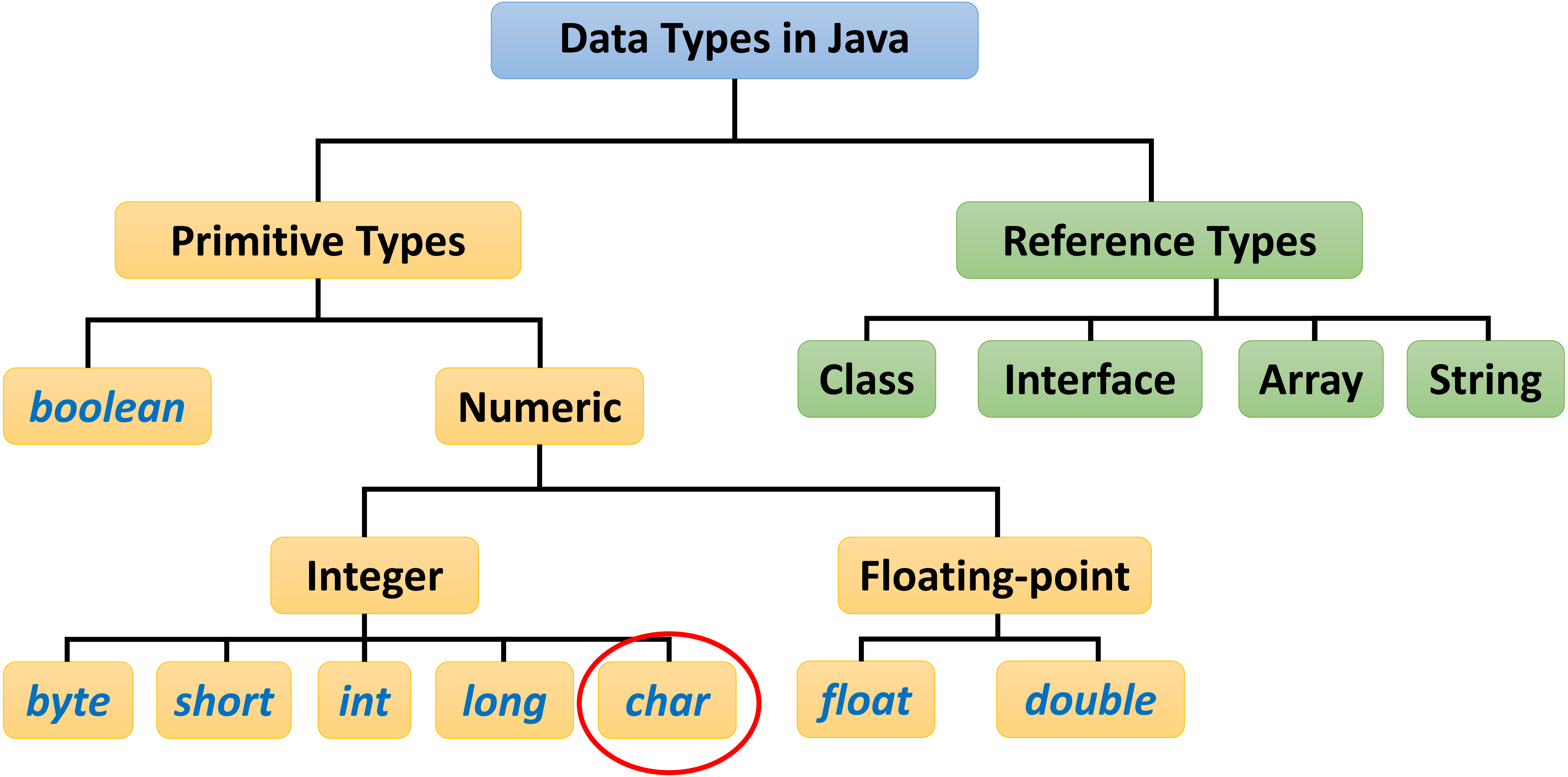
Cautions When Performing Operations on Floating-Points (cont.)

- Not-a-Number (NaN)

```
public class Test {
   public static void main(String[] args) {

      double d = 0.0 / 0.0;
      System.out.println(d); ⇒ NaN

   }
}
```

- Cast to int rounds toward 0

```
public class Test {
   public static void main(String[] args) {

      double d = 12345.6;
      System.out.println((int) d + " " + (int) (-d));

   }
}
```



- Data Types in Java
  - Primitive Types
    - *boolean*
    - Numeric
      - Integer
        - *byte*
        - *short*
        - *int*
        - *long*
        - *char*
      - Floating-point
        - *float*
        - *double*
  - Reference Types
    - Class
    - Interface
    - Array
    - String

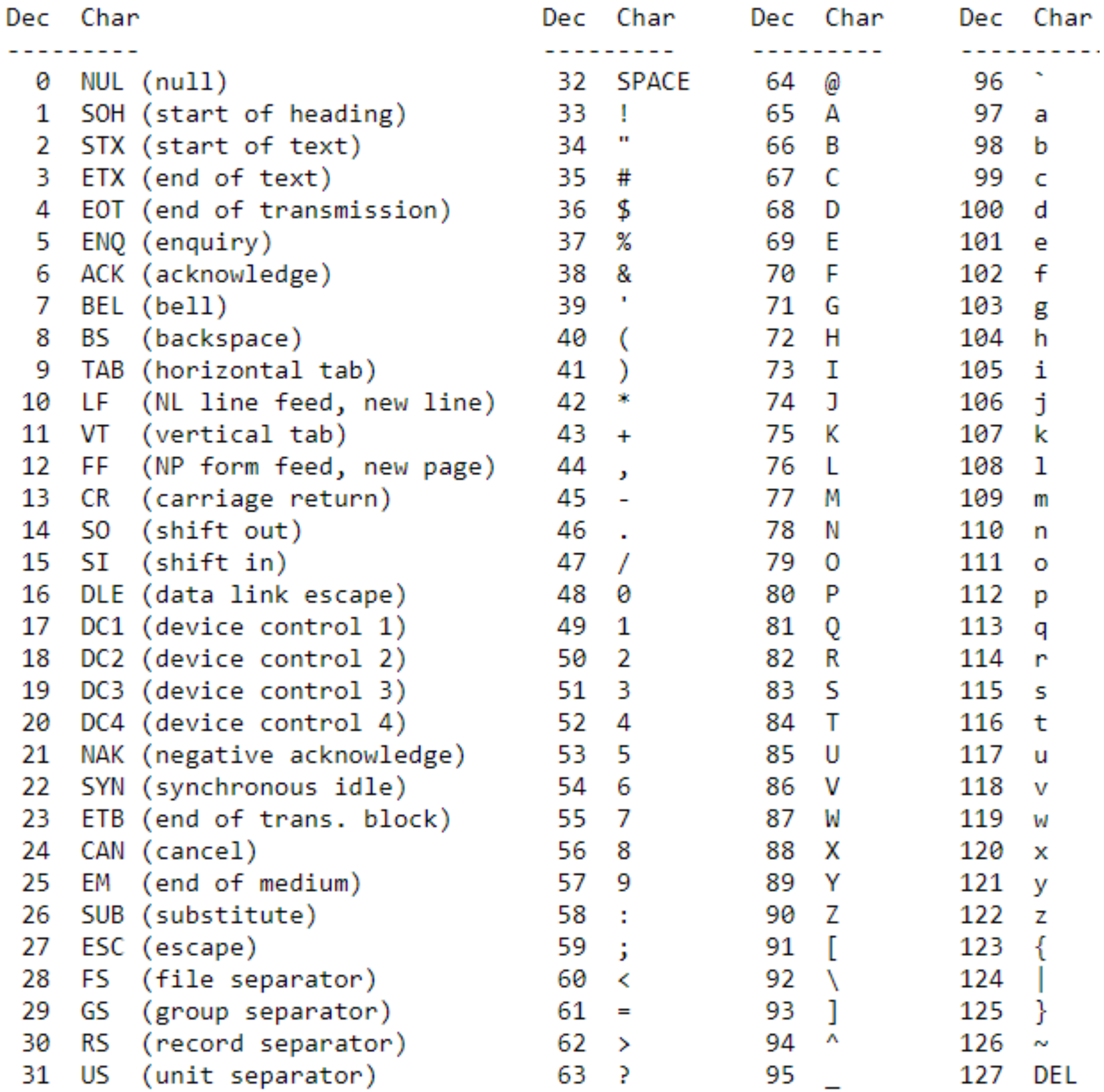
```
char c = 'A';
System.out.println(c+0); //65
```

```
for (char i = 'A'; i <= 'Z'; i++) {
    System.out.print(i);
}
//ABCDEFGHIJKLMNOPQRSTUVWXYZ
```

16 | 65

4 1

41 (Hex)



Dec	Char	Dec	Char	Dec	Char	Dec	Char
0	NUL (null)	32	SPACE	64	@	96	`
1	SOH (start of heading)	33	!	65	A	97	a
2	STX (start of text)	34	"	66	B	98	b
3	ETX (end of text)	35	#	67	C	99	c
4	EOT (end of transmission)	36	$	68	D	100	d
5	ENQ (enquiry)	37	%	69	E	101	e
6	ACK (acknowledge)	38	&	70	F	102	f
7	BEL (bell)	39	'	71	G	103	g
8	BS (backspace)	40	(	72	H	104	h
9	TAB (horizontal tab)	41	)	73	I	105	i
10	LF (NL line feed, new line)	42	*	74	J	106	j
11	VT (vertical tab)	43	+	75	K	107	k
12	FF (NP form feed, new page)	44	,	76	L	108	l
13	CR (carriage return)	45	-	77	M	109	m
14	SO (shift out)	46	.	78	N	110	n
15	SI (shift in)	47	/	79	O	111	o
16	DLE (data link escape)	48	0	80	P	112	p
17	DC1 (device control 1)	49	1	81	Q	113	q
18	DC2 (device control 2)	50	2	82	R	114	r
19	DC3 (device control 3)	51	3	83	S	115	s
20	DC4 (device control 4)	52	4	84	T	116	t
21	NAK (negative acknowledge)	53	5	85	U	117	u
22	SYN (synchronous idle)	54	6	86	V	118	v
23	ETB (end of trans. block)	55	7	87	W	119	w
24	CAN (cancel)	56	8	88	X	120	x
25	EM (end of medium)	57	9	89	Y	121	y
26	SUB (substitute)	58	:	90	Z	122	z
27	ESC (escape)	59	;	91	[	123	{
28	FS (file separator)	60	<	92	\	124	|
29	GS (group separator)	61	=	93	]	125	}
30	RS (record separator)	62	>	94	^	126	~
31	US (unit separator)	63	?	95	_	127	DEL

char & print()

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

    char building = 'A';
    int floor = 1;
    int room = 1;
    System.out.println(building+floor+room); // 65
                      65   +  1  +  1

}
```

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

    char building = 'A';
    int floor = 1;
    int room = 1;

    System.out.println("roomID: "+building+floor+room);

}
```

Operators

Simple Assignment Operator

=	Simple assignment operator

Arithmetic Operators

+	Additive operator (also used for String concatenation)
-	Subtraction operator
*	Multiplication operator
/	Division operator
%	Remainder operator

Unary Operators

+	Unary plus operator; indicates positive value (numbers are positive without this, however)
-	Unary minus operator; negates an expression
++	Increment operator; increments a value by 1
--	Decrement operator; decrements a value by 1
!	Logical complement operator; inverts the value of a boolean

Equality and Relational Operators

==	Equal to
!=	Not equal to
>	Greater than
>=	Greater than or equal to
<	Less than
<=	Less than or equal to

Conditional Operators

&&	Conditional-AND
||	Conditional-OR
?:	Ternary (shorthand for if-then-else statement)

Type Comparison Operator

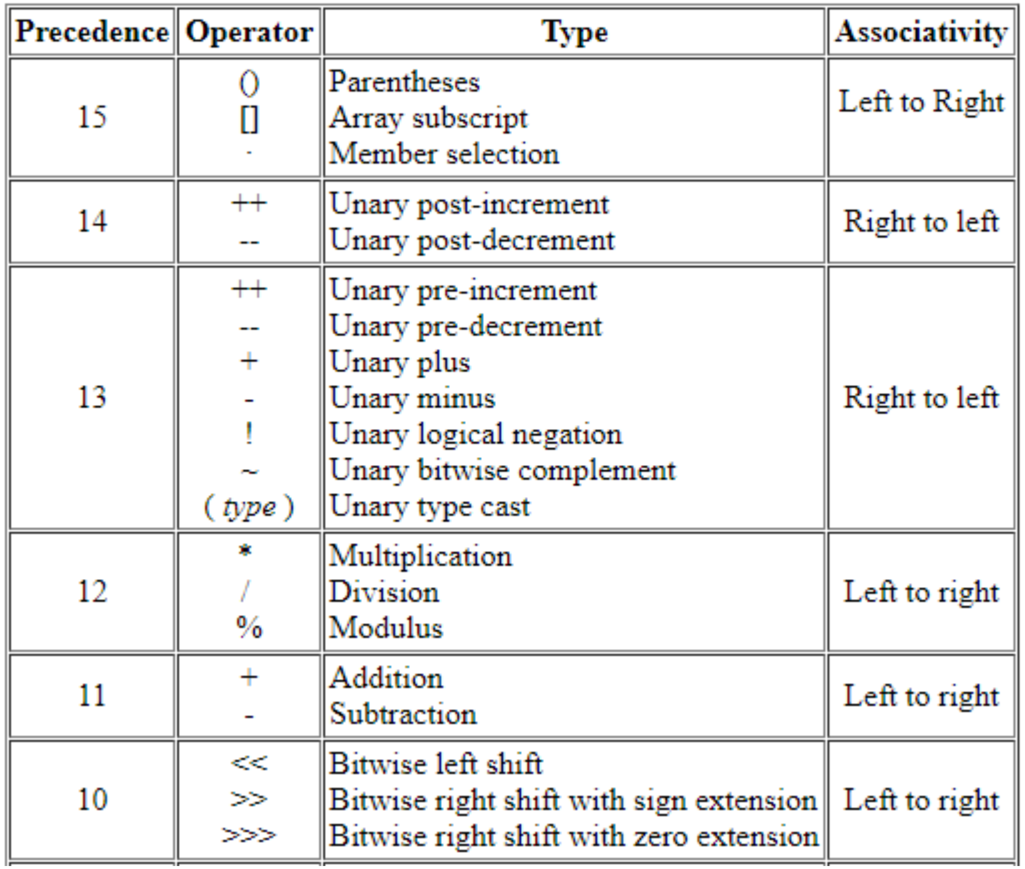
instanceof	Compares an object to a specified type

Bitwise and Bit Shift Operators

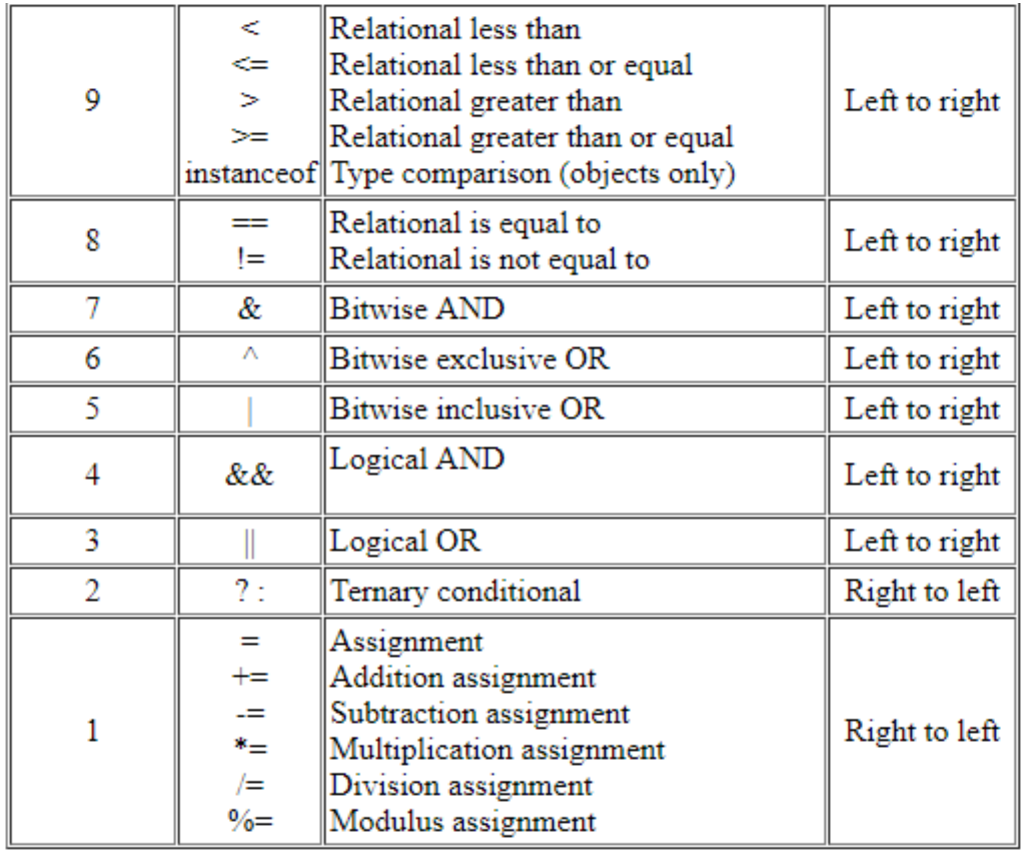
~	Unary bitwise complement
<<	Signed left shift
>>	Signed right shift
>>>	Unsigned right shift
&	Bitwise AND
^	Bitwise exclusive OR
|	Bitwise inclusive OR

ที่มา: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/opsummary.html

Operators (cont.)



Precedence	Operator	Type	Associativity
15	() [] .	Parentheses Array subscript Member selection	Left to Right
14	++ --	Unary post-increment Unary post-decrement	Right to left
13	++ -- + - ! ~ (*type*)	Unary pre-increment Unary pre-decrement Unary plus Unary minus Unary logical negation Unary bitwise complement Unary type cast	Right to left
12	* / %	Multiplication Division Modulus	Left to right
11	+ -	Addition Subtraction	Left to right
10	<< >> >>>	Bitwise left shift Bitwise right shift with sign extension Bitwise right shift with zero extension	Left to right



9	< <= > >= instanceof	Relational less than Relational less than or equal Relational greater than Relational greater than or equal Type comparison (objects only)	Left to right
8	== !=	Relational is equal to Relational is not equal to	Left to right
7	&	Bitwise AND	Left to right
6	^	Bitwise exclusive OR	Left to right
5	|	Bitwise inclusive OR	Left to right
4	&&	Logical AND	Left to right
3	||	Logical OR	Left to right
2	? :	Ternary conditional	Right to left
1	= += -= *= /= %=	Assignment Addition assignment Subtraction assignment Multiplication assignment Division assignment Modulus assignment	Right to left

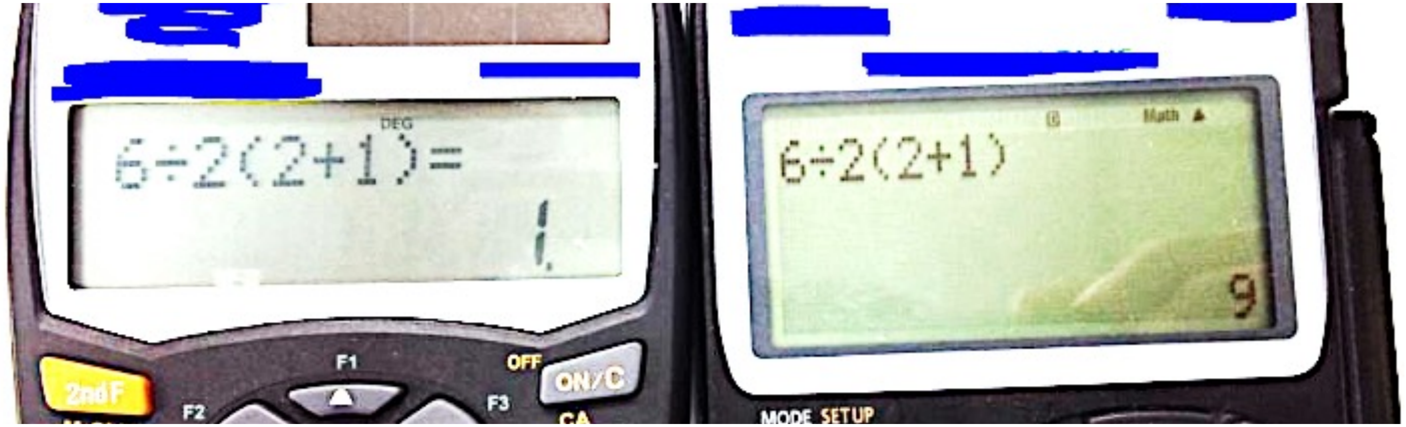
Larger number means higher precedence.

ที่มา: http://www.cs.bilkent.edu.tr/~guvenir/courses/CS101/op_precedence.html

Expression

- An expression is a construct made up of variables, operators, and method invocations, which are constructed according to the syntax of the language.
- An expression produces a result, or value, when it is evaluated
  - A numeric type, as in an arithmetic expression
  - A reference type, as in an object allocation
  - A special type void, which is the declared type of a method that doesn’t return a value

ที่มา: https://docs.oracle.com/javase/tutorial/java/nutsandbolts/expressions.html
https://www.oreilly.com/library/view/learning-java/1565927184/ch04s04.html



```
System.out.println(6/2*(2+1));
```

Left-hand operand is evaluated first

```
public static void main(String[] args) {
   int i = 2;
   int j = (i=3) * i;
   System.out.println(j); // ? → 9
}
```

the * operator has a left-hand operand that contains an assignment to a variable and a right-hand operand that contains a reference to the same variable. The value produced by the reference will reflect the fact that the assignment occurred first.

```
public static void main(String[] args) {
   int i = 2;
   int j = i = 3 * i;  → 6
   System.out.println(j); // ?
}
```

Implicit left-hand operand In operator of compound assignment

```
public static void main(String[] args) {
   int a = 9;
   a += (a = 3);  // ?
   System.out.println(a);

   int b = 9;
   b = b + (b = 3);  // ?  12
   System.out.println(b);
}
```

9 + 3

a = a + (a = 3)

9 + (3) = 12