Java Basic Grammar 1

Java Programming

Byeongjoon Noh

powernoh@sch.ac.kr

SOON CHUN HYANG UNIVERSITY

Contents

Note

- ; (semicolon)
  - is used to mark the end of a statement, which can be a variable declaration, an assignment, or a control statement
  - DO NOT FORGET a semicolon at the end of each statement
    - forgetting a semicolon can lead to compilation errors
    - a common mistake but also an easy fix one you're aware of it

- { } (curly braces)
  - defines a block of code, for method or control structures like loops and conditionals

Note

- Comments
  - comments are used to explain what a certain part of the code does, to make the code easier to understand for humans
  - comments are ignored by the Java compiler
  - single-line comments: start with two forward slashes '//'

```
// This is comments
```

  - multi-line comments: enclosed between '/*' and '*/'

```
/*
* These are a multi-line comments
* This line also comment aprt
*/
```

Note

- The same name with FILE NAME and CLASS NAME
    - the name of the public class in a file MUST MATCH the name of the file itself
    - this is a fundamental rule enforced by the Java compiler
        - to ensure consistency and manageability in code organization

```
// In a file named HelloWorld.java

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

Note

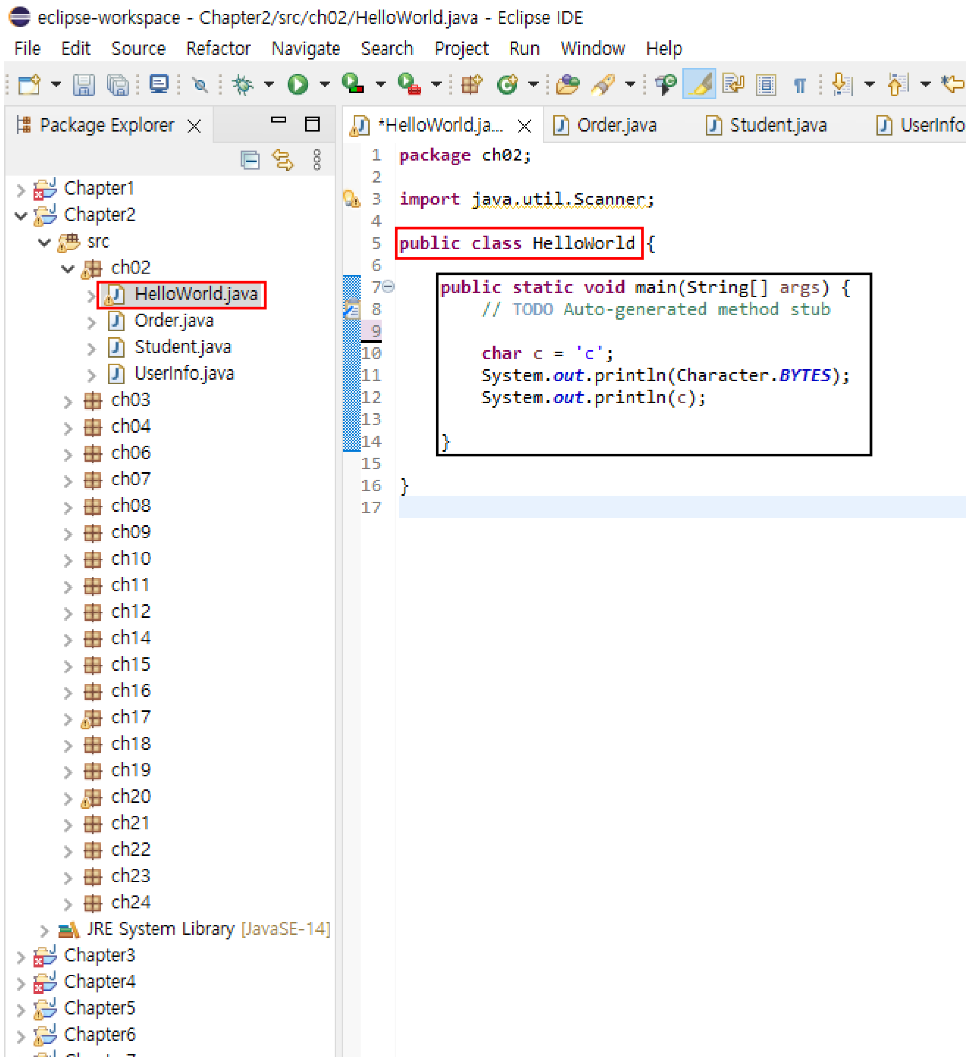
- Overall structure of Java project

```
/*
 * 소스 파일 : Hello.java
 */
public class Hello {

   public static int sum(int n, int m) {
      return n + m;
   }

   // main() 메소드에서 실행 시작
   public static void main(String[] args) {
      int i = 20;
      int s;
      char a;

      s = sum(i, 10); // sum() 메소드 호출
      a = '?';
      System.out.println(a); // 문자 '?' 화면 출력
      System.out.println("Hello"); // "Hello" 문자열 화면 출력
      System.out.println(s); // 정수 s 값 화면 출력
   }
}
```

메소드

클래스

```
?
Hello
30
```

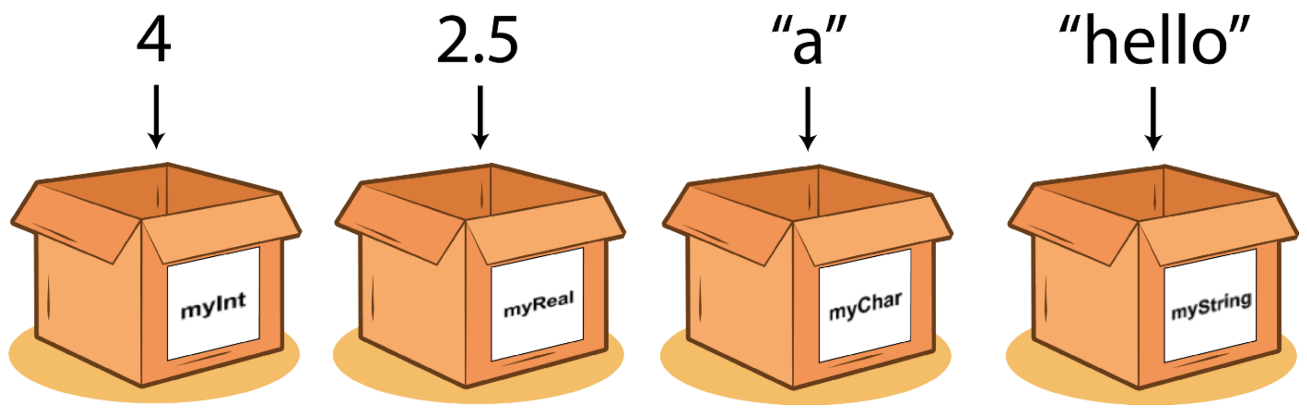


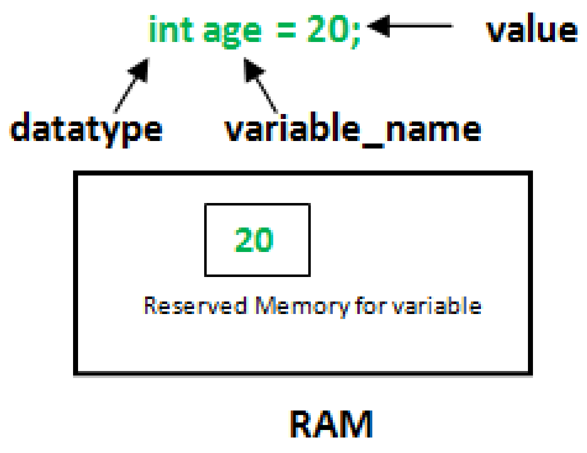
```
package ch02;

import java.util.Scanner;

public class HelloWorld {

    public static void main(String[] args) {
        // TODO Auto-generated method stub

        char c = 'c';
        System.out.println(Character.BYTES);
        System.out.println(c);

    }

}
```

1. Variables

What is a variable?

- A storage location paired with an associated symbolic name
  - contains some known or unknown quantity of information referred to as value
  - the value is stored in memory (RAM)





Variable declaration and assignment

- Declaration
  - variable must be declared before they can be used
    - this declaration typically specified the type of data the variable will hold
  - syntax

```
type variableName;
```

  - examples

```
int byeongjoon;
float coffee;
```

  - cannot be declared twice or more with the same variable's name (SyntaxError)

Variable declaration and assignment

- Assignment
  - after declaring a variable, you can assign a value to it using the assignment operator '='
  - values are actually stored in memory
  - syntax

```
variableName = value;
```

  - examples

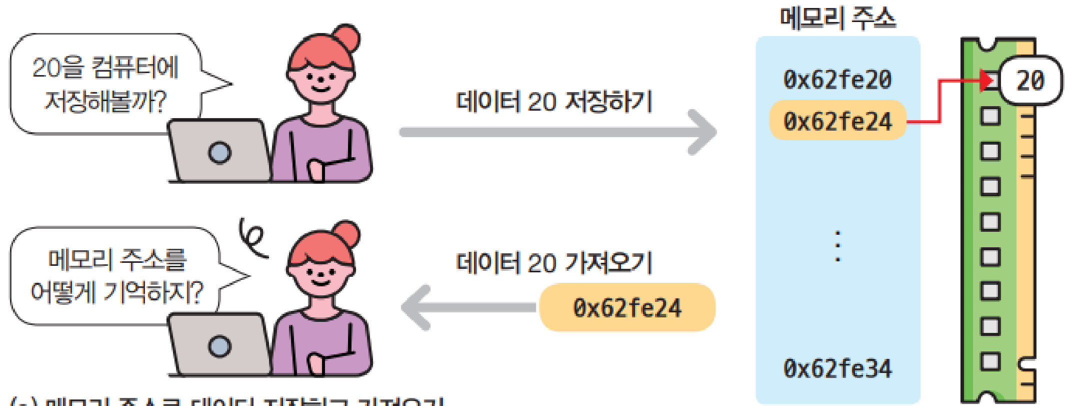
```
byeongjoon = 3;
coffee = 1.5;
```

- You can also declare and assig a value to a variable in a single statement:

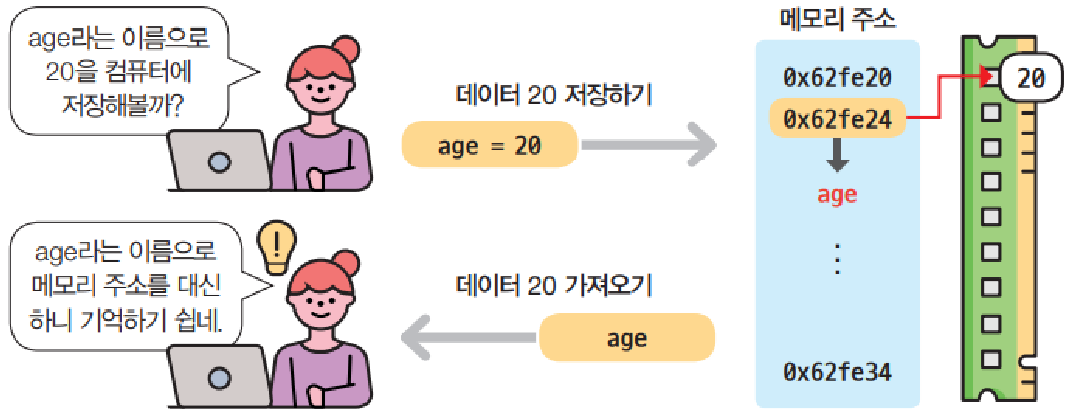
```
int byeongjoon = 3;
float coffee = 1.5;
```

Variable declaration and assignment

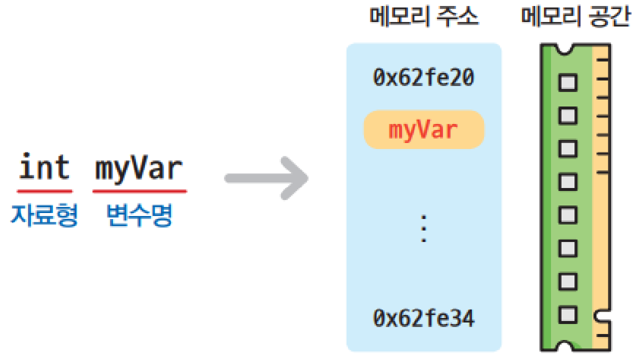
- 프로그래밍 속 변수 (variable)의 개념

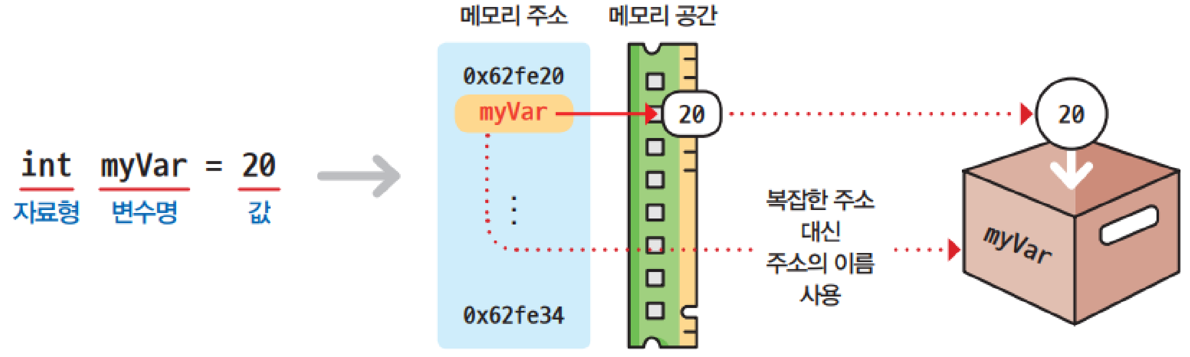


(a) 메모리 주소로 데이터 저장하고 가져오기


age라는 이름으로 20을 컴퓨터에 저장해볼까?
데이터 20 저장하기
age = 20
age라는 이름으로 메모리 주소를 대신 하니 기억하기 쉽네.
데이터 20 가져오기
age
메모리 주소
0x62fe20
0x62fe24
age
0x62fe34
20

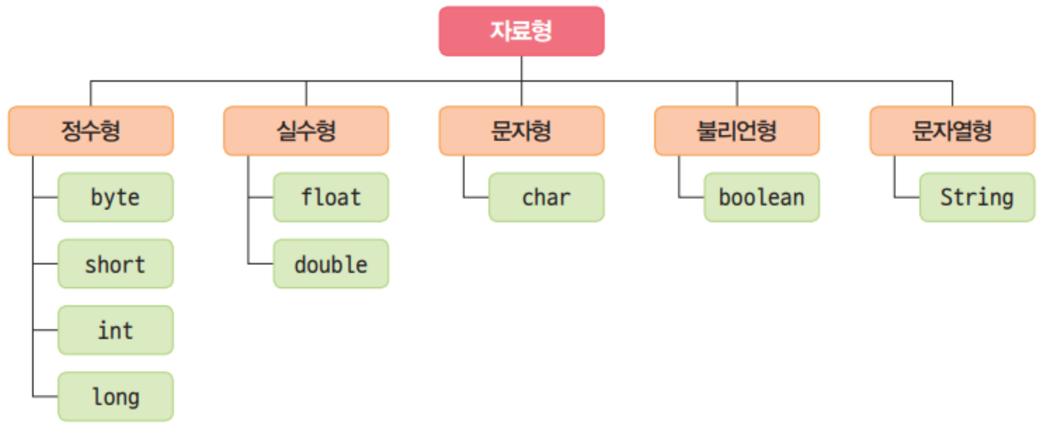
(b) 메모리 주소 이름으로 데이터 저장하고 가져오기

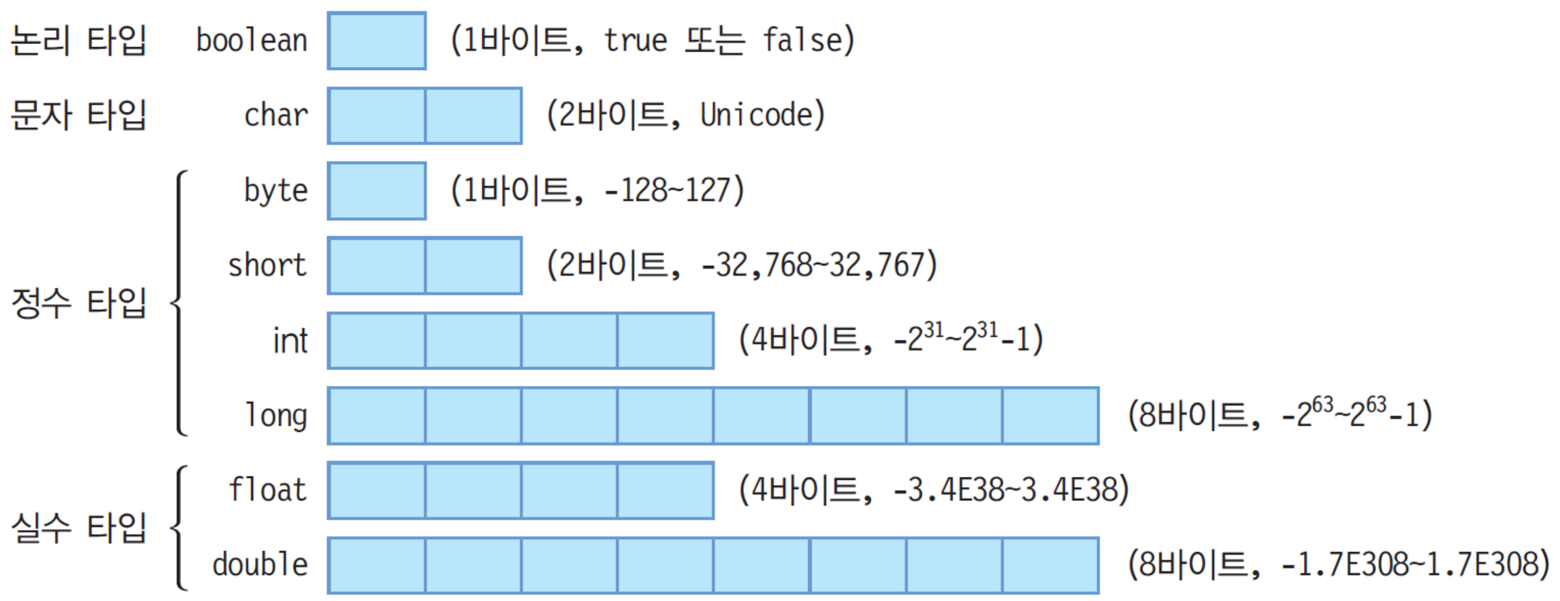




Primary data type (기본 자료형)

- Determines the size of layout of the variable's memory, the range of values that can be stored within that memory, and the set of operations that can be applied to the variable





논리 타입	boolean	(1바이트, true 또는 false)
문자 타입	char	(2바이트, Unicode)
정수 타입	byte	(1바이트, -128~127)
	short	(2바이트, -32,768~32,767)
	int	(4바이트, -2^{31}~2^{31}-1)
	long	(8바이트, -2^{63}~2^{63}-1)
실수 타입	float	(4바이트, -3.4E38~3.4E38)
	double	(8바이트, -1.7E308~1.7E308)

Reference data types

- To refer to objects, not a primitive type; such as classes, interfaces, and arrays
    - They store the memory address of the object they refer to, rather than the data itself
- Classes
    - ex) String, Integer, System, etc.
    - to create objects and define the data type of those objects
- Interfaces
    - ex) List, map, Set, etc.
    - to specify a set of methods that a class must implement
- Arrays
    - ex) int[], double[], String[], etc.
    - To store multiple values of the same type in a single variable

Examples of variable declaration and assignment

- Variable declaration

```
int radius;
char c1, c2, c3;
double weight;
```

- Declaration and assignment

```
int radius = 10;
char c1 = 'a', c2 = 'b', c3 = 'c';
double weight = 75.56;
```

- Value assignment

```
radius = 10 * 5;
c1 = 'r';
weight = weight + 5.0;
```

  - if the variable is already declared, the value is overwritten

Quiz

- What is the value in variable result?

```
int radius = 3.5;
int result = radius + 5;
```

- Which are grammatically correct sentences?

```
int a, b = 1, 2;  // 1
int a = 1, b = 2; // 2
int a = 1; int b = 2; // 3
int a = 1, b; // 4
```

Rules for variable names in Java

- Alphanumeric characters: Variable names can include letters, digits, underscores ('_'), and dollar sign ('$')
  - but cannot begin with a digit
- Case sensitivity: Variable names are case-sensitive
  - ex) 'variable', 'Variable', and 'VARIABLE' are considered different identifiers
- No reserved words: You cannot use Java reserved words (keywords) as variable names
  - ex) 'int', 'float', 'class', etc.
- Meaningful names
  - not enforced by the language
  - it is a best practice to use meaningful variable names to make your code more readable and maintainable
  - ex) 'temperatureCelsius', 'userName', 'user_name', 'strInput', 'nCnt', etc.
    - 'temperatureCelsius' is more descriptive than 'temp' or 't'

Java keywords

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

Rules for variable names in Java

- Example of correct variable names

```
int age;
double salary;
String firstName;
boolean isEmployee;
int whatsyournamemynameisbyeongjoon;
float Monster3;
char _kkk;
```

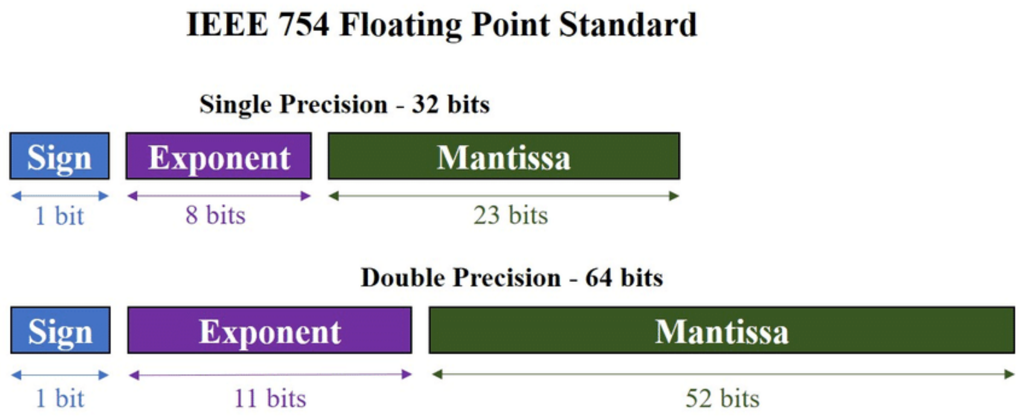
- Example of incorrect variable names

```
int 1stPlace;
float –amount;
String class;
boolean is Employee;
double %calc;
```

Note: Floating-point numbers in Java

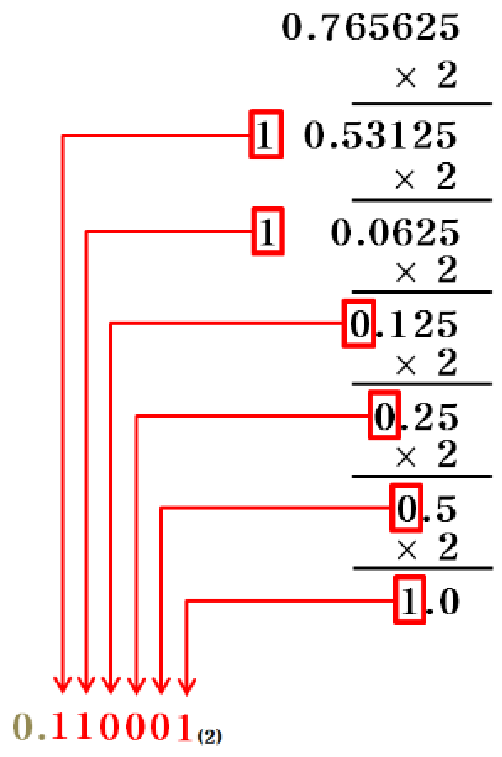
- Real number = integer (decimal) part + fractional (mantissa) part
  - 23.1519 = 23 + 0.1519

- Floating-point (부동소수점) by IEEE 754 floating point standard
  - a method to represent real numbers in computer
  - data types in Java: 'float' and 'double'
  - why are the real numbers represented by floating-point?
    - floating-point numbers cannot precisely represent all real numbers
      - ➔ precision and rounding errors
      - ex) 0.1 (decimal) ➔ 0.0011001100110011..... (binary)


IEEE 754 Floating Point Standard
Single Precision - 32 bits
Sign
Exponent
Mantissa
1 bit
8 bits
23 bits
Double Precision - 64 bits
Sign
Exponent
Mantissa
1 bit
11 bits
52 bits

Note: How to convert the real number into floating point?

- Example for 11.765625 (decimal) (only float type)
  - 1) representation in binary ➔ 1011.110001 (binary)
    - integer part: 11 (decimal) ➔ 1011 (binary)
    - fractional part: 0.765625 (decimal) ➔ 0.110001 (binary)
  - 2) normalize the binary number
    - 1011.110001 ➔ 1.01110001 * 2^3
  - 3) determine the exponent
    - the exponent is 3 ➔ 130 (=127 + 3) = 10000010
      - 127 is is the bias for float type; double is 1023
  - 4) encode the fraction
    - 01110001 (ignoring the leading 1)



exponent (8bit)

01000001001110001000000000000000

sign (1bit) mantissa (23bit)

Note: How to convert the real number into floating point?

- Example for 0.15625 (decimal)
  - 1) representation in binary ➔ 0.00101 (binary)
  - 2) normalize the binary number
    - 0.00101 ➔ 1.01 * 2^(-3)
  - 3) determine the exponent
    - the exponent is -3 ➔ 124 (=127 - 3) = 01111100
  - 4) encode the fraction
    - 01 (ignoring the leading 1)

exponent (8bit)

float type 00111110001000000000000000000000

sign (1bit) mantissa (23bit)

double type 0010000000000000000000000010000000000000000000000000000000000000000000000000

sign (1bit) exponent (11bit) mantissa (52bit)

Size of data type

- There two types of fields to obtain the sizes of primitive data types
  - .BYTES: the size of the data type in bytes
  - .SIZE: the size of data types in bits

```
System.out.println("Size of byte     ==> " + Byte.BYTES+ ", " + Byte.SIZE);
System.out.println("Size of short    ==> " + Short.BYTES + ", " + Short.SIZE);
System.out.println("Size of int      ==> " + Integer.BYTES+ ", " + Integer.SIZE);
System.out.println("Size of long     ==> " + Long.BYTES+ ", " + Long.SIZE);
System.out.println("Size of float    ==> " + Float.BYTES+ ", " + Float.SIZE);
System.out.println("Size of double   ==> " + Double.BYTES+ ", " + Double.SIZE);
```

```
Size of byte     ==> 1, 8
Size of short    ==> 2, 16
Size of int      ==> 4, 32
Size of long     ==> 8, 64
Size of float    ==> 4, 32
Size of double   ==> 8, 64
```

Constant

- A constant is a variable whose value cannot be changed once it has been assigned
  - defined using the ‘final’ keyword, which can be applied to primitive data types

```
final int MAX_USERS = 100; // Constant declaration
System.out.println("Maximum users allowed: " + MAX_USERS);

final String WELCOME_MESSAGE = "Welcome to Java Programming!";
System.out.println(WELCOME_MESSAGE);

final double PI = 3.14159;
PI = 3.14; // Error
```

- the convention for naming constants is to use all uppercase letters with underscores (‘_’)

2. Literals

What is a literal?

- A literal in programming refers to a fixed value that appears directly in the source code
    - representing constant values assigned to variables and are not altered during the execution of the program

```
int age = 30;
System.out.println(age);
```

- Java literals
    - integers, floating-point numbers, characters, booleans, strings

Integer literals

- Integer literals can be expressed in decimal, hexadecimal (base 16), octal (base 8), or binary (base 2) form
- Decimal: Any standard numeric value without a leading zero
  - e.g. 123, -456
- Hexadecimal: Prefixed with '0x'
  - e.g., 0xFF, 0x7a
- Octal: Prefixed with '0'
  - e.g., 077, 023
- Binary: Prefixed with '0b' or '0B'
  - e.g., 0b1011, 0B10010
- (Note) Long-typed literal: Postfixed with 'L'

```
int n = 15;
int m = 015;
int k = 0x15;
int b = 0b0101;
long g = 24L;
System.out.println(n);
System.out.println(m);
System.out.println(k);
System.out.println(b);
System.out.println(g);
```

```
15
13
21
5
24
```

Floating-point literals

- Floating-point literals represent integer part and fractional part
  - standard notation: Decimal numbers with a dot (/)
    - e.g., 3.14, -0.001
  - scientific notation: Expressed with an 'e' or 'E' indicating the power of 10
    - e.g., 1.5e2 (which is 150.0), 6.022E23

```
double d = 0.1234;
double e = 1234E-4;
float f = 0.1234f;
double w = .1234D;
System.out.println(d);
System.out.println(e);
System.out.println(f);
System.out.println(w);
```

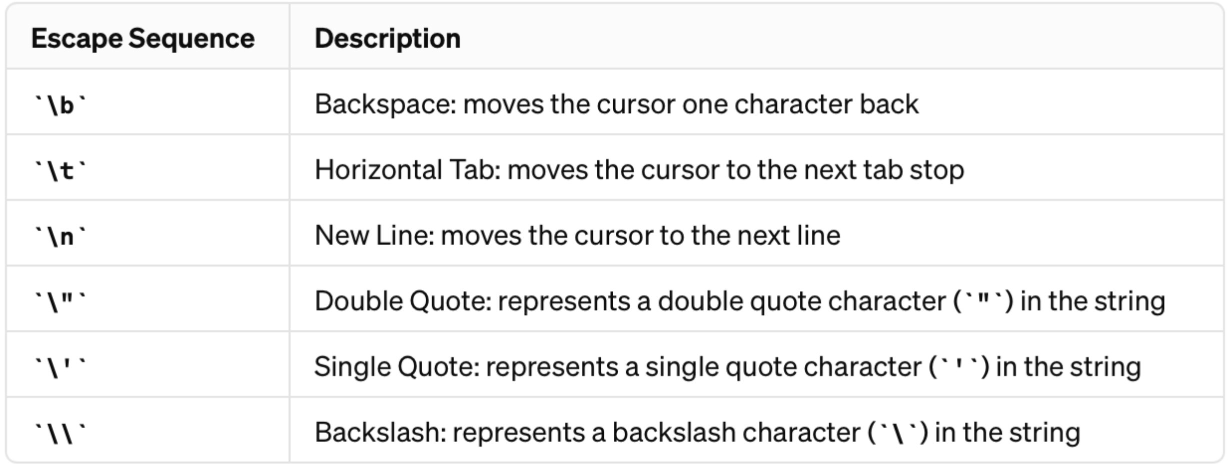
```
0.1234
0.1234
0.1234
0.1234
```

Character literals

- Character literals represents a single character
  - enclosed in single quotes (‘’)
    - (Note) double quotes (“”): String
- Regular characters
  - e.g., ‘a’, ‘Z’, ‘글’
- Escape sequences: Special characters represented with a backslash (‘₩’)
  - ₩n (newline)
  - ₩t (tab)
  - ₩’ (single quote)
  - ₩₩ (backslash)

```java
char p = 'W';
char h = '글';
char i = '\uae00';
System.out.println(p);
System.out.println(h);
System.out.println(i);
```

```
W
글
글
```



Escape Sequence	Description
`\b`	Backspace: moves the cursor one character back
`\t`	Horizontal Tab: moves the cursor to the next tab stop
`\n`	New Line: moves the cursor to the next line
`\"`	Double Quote: represents a double quote character (`"`) in the string
`\'`	Single Quote: represents a single quote character (`'`) in the string
`\\`	Backslash: represents a backslash character (`\`) in the string

String literals

- String literals is sequences of characters enclosed in double quotes
    - can include any characters, including escape sequences
    - e.g., “Hello, World!”, “Java₩nProgramming”, “ㄴㅏㄴㅏㄴㅏㄱㅏㅁ”

```
String str = "Hello, World!";
String nation = "Korea";
String version = "11.2";
System.out.println(str);
System.out.println(nation);
System.out.println(version);
```

```
Hello, World!
Korea
11.2
```

- Operation for string (literals)

```
str = "Hello" + "World!" + "\t" + "123";
System.out.println(str);
```

```
HelloWorld!	123
```

Other literals

- Boolean (logical) literals
    - Boolean type has two literals: ‘true’ and ‘false’

```
boolean isJavaFun = true;
boolean isFlag = false;
System.out.println(isJavaFun);
System.out.println(isFlag);
```

```
true
false
```

- Null literal
    - a special literal in Java that represents a reference that points to no object

```
String myString = null;
System.out.println(myString);
```

```
null
```

Quiz

- What are the outputs of the following code?

```
// Chap02Example/StringLiteralQuiz01.java

System.out.println("She said, \"Hello! How are you?\"");
System.out.println('A' + "\'s grade is 90.");
System.out.println("The file path is C:\\Users\\User\\Documents"); System.out.println("First line.\nSecond line.");
System.out.println("Column 1\tColumn 2\tColumn 3");
System.out.println("abcdef\b\bghijkl");
```

Print the value

```
System.out.println(argument)
```

- to print the data to the console
- a part of *java.lang* package, automatically imported into every Java program
- argument: any data type or object you wish to print to the console
- example

```java
int age = 30;
System.out.println(age);
System.out.println("Age is " + age);
```

```
30
Age is 30
```

- use '+' symbol to concatenate the values
  - "Age is " ➔ string literal
  - age ➔ integer-typed variable

Print the value

- Various types of print methods

```
System.out.println(argument)
System.out.print(argument)
System.out.printf(argument)
```

  - *println()*
    - automatically append a newline character at the end of the output, moving the cursor to the beginning of the next line on the console
  - *print()*
    - prints its argument without appending a newline at the end
    - the cursor remain the end of the output text
  - *printf()*
    - used for formatted output, allowing to specify a format string and then provide the corresponding values

Examples and practices for printing values

- 1) 다양한 데이터 타입에 대한 변수를 선언하고, 값을 할당하고, 출력하는 프로그램을 작성해보세요.
    - file path and name: Chap02Example/PrintPractice01.java

- 2) 주어진 변수를 사용하여 다음의 결과가 나타나도록 *System.**out**.println()*의 내부를 작성해보세요.
    - file path and name: Chap02Example/PrintPractice02.java

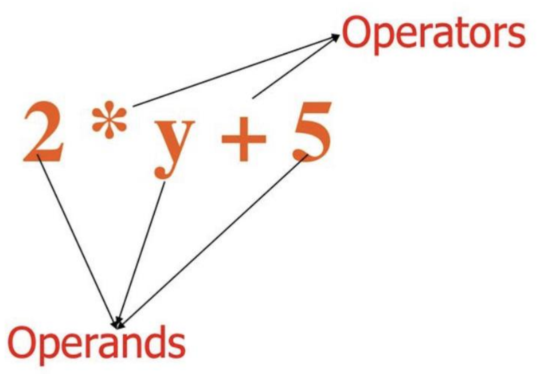
```java
// Chap02Example/PrintPractice02.java

int age = 30;
String name = "Hong";
System.out.println([Fill it out]);

int x = 3, y = 6;
System.out.println([Fill it out]);
```

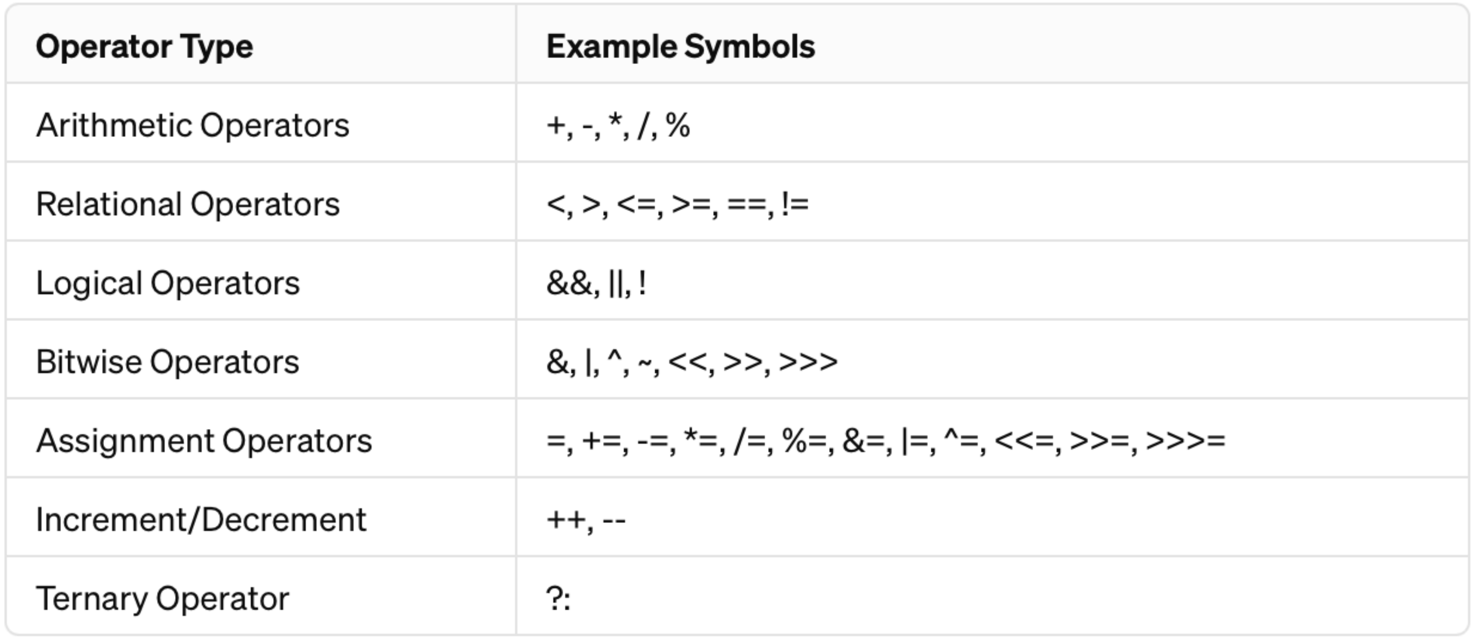
```
My name is Hong, and I am 30 years old
The point is (3, 6)
```

3. Operators

Getting started

- Java supports a wide range of operations that can be performed on various types of data
- These operations can be broadly classified into several categories:
  - arithmetic, bitwise, relational, logical, assignment, and special operations

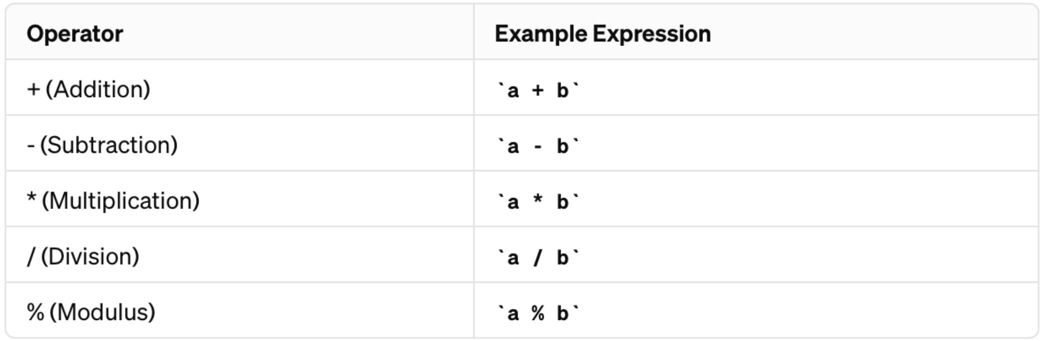




Operator Type	Example Symbols
Arithmetic Operators	+, -, *, /, %
Relational Operators	<, >, <=, >=, ==, !=
Logical Operators	&&, ||, !
Bitwise Operators	&, |, ^, ~, <<, >>, >>>
Assignment Operators	=, +=, -=, *=, /=, %=, &=, |=, ^=, <<=, >>=, >>>=
Increment/Decrement	++, --
Ternary Operator	?:

Arithmetic operations

- Arithmetic operations are used for performing mathematical calculations



Operator	Example Expression
+ (Addition)	`a + b`
- (Subtraction)	`a - b`
* (Multiplication)	`a * b`
/ (Division)	`a / b`
% (Modulus)	`a % b`

```
int a = 7, b = 5;
System.out.println("a + b = " + (a + b));
System.out.println("a - b = " + (a - b));
System.out.println("a * b = " + (a * b));
System.out.println("a / b = " + (a / b));
System.out.println("a % b = " + (a % b));
```

```
a + b = 12
a - b = 2
a * b = 35
a / b = 1
a % b = 2
```

- division operation
  - the result of integer division will always truncate any decimal part
    - ex) 7/3 = 2, not 2.3333..
  - if you want to obtain the real number form, one or both of the denominator and numerator must be a floating-point type (double of float)

Note: Type casting

- Type casting is the process of converting a variable from one type to another; two types of casting
  - widening casting (implicit) – 암시적 캐스팅 (자동으로 처리됨)
    - occurring when data from a smaller type is automatically converted into a larger type size
  - narrow casting (explicit) – 명시적 캐스팅 (개발자가 직접 명시함)
    - this happens when data from a larger type is converted into a smaller type size
    - this type casting must be explicitly done by the programmer, as it can lead to loos of information

Note: Type casting

- Example of the widening casting (implicit)

```
int myInt = 9;
double myDouble = myInt;
System.out.println(myInt);
System.out.println(myDouble);
```

```
9
9.0
```

- Example of the narrow casting (explicit)

```
double myDouble = 9.78;
int myInt = (int) myDouble;
System.out.println(myDouble);
System.out.println(myInt);
```

```
9.78
9
```

Note: Type casting

- Division operations with type casting
  - division without casting

```java
int a = 5;
int b = 2;
double result = a / b;
System.out.println(result); // Outputs 2.0
```

  - division with casting

```java
int a = 5;
int b = 2;
double result = (double) a / b; // or a / (double) b
System.out.println(result); // Outputs 2.5
```

  - mixed data types

```java
double a = 5.5;
int b = 2;
double result = a / b;
System.out.println(result); // Outputs 2.75
```

Arithmetic operations

- Modulus operator
  - a fundamental arithmetic operator in Java and many other programming languages
  - returns “the remainder” of the division, instead of returning the quotient

```
int result1 = 10 % 3;
int result2 = 10 % 4;
double result3 = 10.5 % 3;
```

  - useful in various programming scenarios
    - e.g., determining whether a number is even or odd (`number % 2` is zero or one?)
    - e.g., checking for multiples of 3 (`number % 3` is zero or not?)

Arithmetic operations

- Division and modulus combination
    - this can be used to solve problems that involve dividing numbers into parts
        - e.g., extracting digits from a number of converting units

```java
int number = 12345;

int ones = number % 10; // Extracts the 'ones' digit
int tens = (number / 10) % 10; // Extracts the 'tens' digit
int hundreds = (number / 100) % 10; // Extracts the 'hundreds' digit
int thousands = (number / 1000) % 10; // Extracts the 'thousands' digit
int tenThousands = (number / 10000); // Extracts the 'ten thousands' digit

System.out.println("Ones: " + ones);
System.out.println("Tens: " + tens);
System.out.println("Hundreds: " + hundreds);
System.out.println("Thousands: " + thousands);
System.out.println("Ten thousands: " + tenThousands);
```

Examples and practices for math expressions in Java

- How to represent the real-world mathematical expressions in Java
  - File path and name: Chap02Example/MathExpressionPractice01.java

  - $3a$
  - $-1 + 4a$
  - $5a + 5bc$
  - ${}^{2a}/_{3b}$
  - $\frac{4c-1}{4a+1}$
  - $ad - bc$
  - $b + 7 * 12 - \mathrm{a}/7$

Arithmetic operations

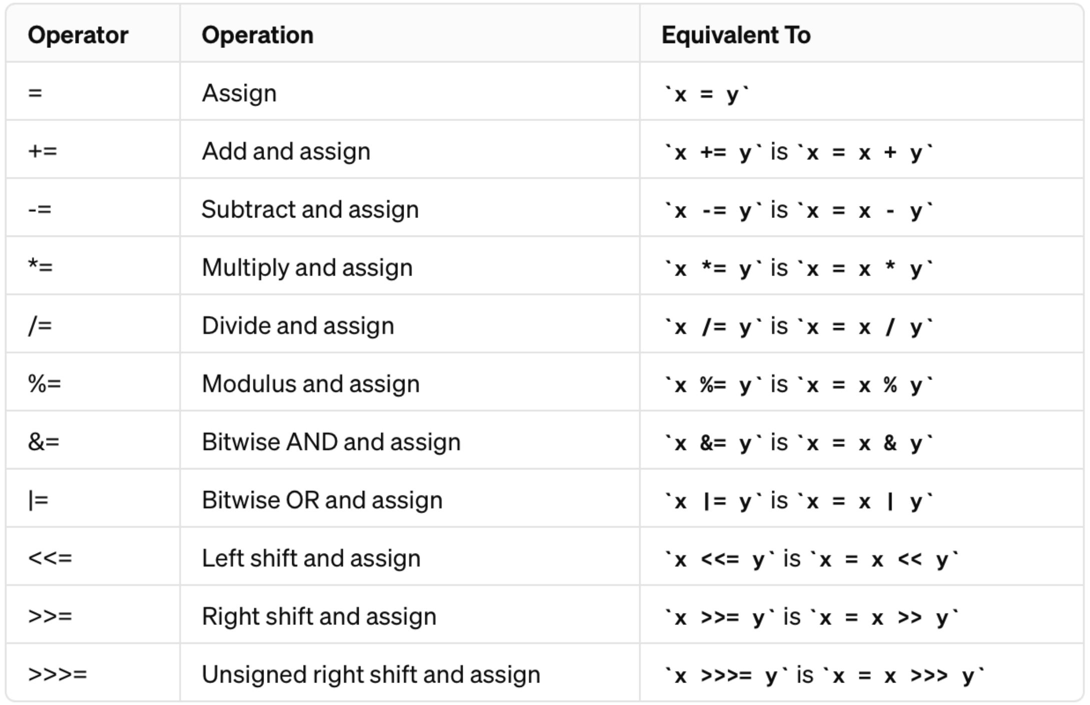
- How can we calculate the power operation and square root operation?
  - e.g., 2^3, $\sqrt{3}$
  - power and square root operations in 'Math' class; standard Java library

```
double result1 = Math.pow(2, 3);
System.out.println("2 to the power of 3 is " + result1); // Outputs 8.0

double result2 = Math.sqrt(16);
System.out.println("The square root of 16 is " + result2); // Outputs 1.7320508075688772
```

  - how to represent $\frac{-b+\sqrt{b^2-4ac}}{2ac}$ in Java expression?

Assignment operations

- Assignment operations assign a value to a variable
  - not the equal symbol in programming language
- There are also compound assignment operators that combine an arithmetic operation with assignment
  - add and assign (‘+=‘): add the right operand to the left operand and assigns the result to the left operand



Operator	Operation	Equivalent To
=	Assign	\`x = y\`
+=	Add and assign	\`x += y\` is \`x = x + y\`
-=	Subtract and assign	\`x -= y\` is \`x = x - y\`
*=	Multiply and assign	\`x *= y\` is \`x = x * y\`
/=	Divide and assign	\`x /= y\` is \`x = x / y\`
%=	Modulus and assign	\`x %= y\` is \`x = x % y\`
&=	Bitwise AND and assign	\`x &= y\` is \`x = x & y\`
|=	Bitwise OR and assign	\`x |= y\` is \`x = x | y\`
<<=	Left shift and assign	\`x <<= y\` is \`x = x << y\`
>>=	Right shift and assign	\`x >>= y\` is \`x = x >> y\`
>>>=	Unsigned right shift and assign	\`x >>>= y\` is \`x = x >>> y\`

```java
int g = 10;
g += 5; // g = g + 5
System.out.println("g = " + g); // 15
g *= 5;
System.out.println("g = " + g); // 75
```

Relational operations

- Relational (comparison) operations compare two values and return a boolean result
  - ‘==’ (equal to): Checks if two values are equal
  - ‘!=’ (not equal): Checks if two values are not equal
  - ‘>’ (not equal): Checks if the left value is greater than the right value
  - ‘<’ (not equal): Checks if the left value is less than the right value
  - ‘>=’ (not equal): Checks if the left value is greater than or equal to the right value
  - ‘<=’ (not equal): Checks if the left value is less than or equal to the right value

```
int c = 10, d = 20;
System.out.println("c == d = " + (c == d)); // false
System.out.println("c != d = " + (c != d)); // true
System.out.println("c > d = " + (c > d)); // false
System.out.println("c < d = " + (c < d)); // true
System.out.println("c >= d = " + (c >= d)); // false
System.out.println("c <= d = " + (c <= d)); // true
```

Relational operations

- It can also work for other data types; not numbers

```
char c = 'a', d = 'b';
System.out.println("c == d = " + (c == d)); // false
System.out.println("c != d = " + (c != d)); // true
System.out.println("c > d = " + (c > d)); // false
System.out.println("c < d = " + (c < d)); // true
```

Logical operations

- Logical operations operate on boolean values and return a boolean result
  - ‘&&’ (AND): Returns true if both operands are true

A (Input)	B (Input)	A AND B (Result)
false	false	false
false	true	false
true	false	false
true	true	true

  - ‘!’ (NOT): Inverts the value of a boolean

A (Input)	NOT A (Result)
false	true
true	false

  - ‘||’ (OR): Returns true if at least one of the operands is true

A (Input)	B (Input)	A OR B (Result)
false	false	false
false	true	true
true	false	true
true	true	true

  - ‘^’ (XOR): Returns true if and only if one of the operands is true but not both

A (Input)	B (Input)	A XOR B (Result)
false	false	false
false	true	true
true	false	true
true	true	false

Logical operations

- Examples of logical operations

```
int a = 5, b = 10, c = 5;
boolean result;

result = (a == c) && (b > a);
System.out.println("Result of (a == c) && (b > a): " + result); // Output: true
result = (a > b) && (a == c);
System.out.println("Result of (a > b) && (a == c): " + result); // Output: false
result = (a == c) || (b < a);
System.out.println("Result of (a == c) || (b < a): " + result); // Output: true
result = (a > b) || (a != c);
System.out.println("Result of (a > b) || (a != c): " + result); // Output: false
result = !(a == b);
System.out.println("Result of !(a == b): " + result); // Output: true
result = !(a > b);
System.out.println("Result of !(a > b): " + result); // Output: true
result = (a>b)^(b<=c);
System.out.println("Result of (a>b)^(b<=c): " + result); // Output: false
```

Logical and relational operations

- Combination of logical and relational operations to check if a variable falls within a specific range
  - Java does not support the direct chaining of relational operation; 20 < age < 30 (not valid in Java)
    - instead, need to break down the condition into two parts and then combine them using the logical operator (‘&&’ in this case)
  - examples
    - 20 < age < 30

```
age > 20 && age < 30
```

- age < 20 or age > 30

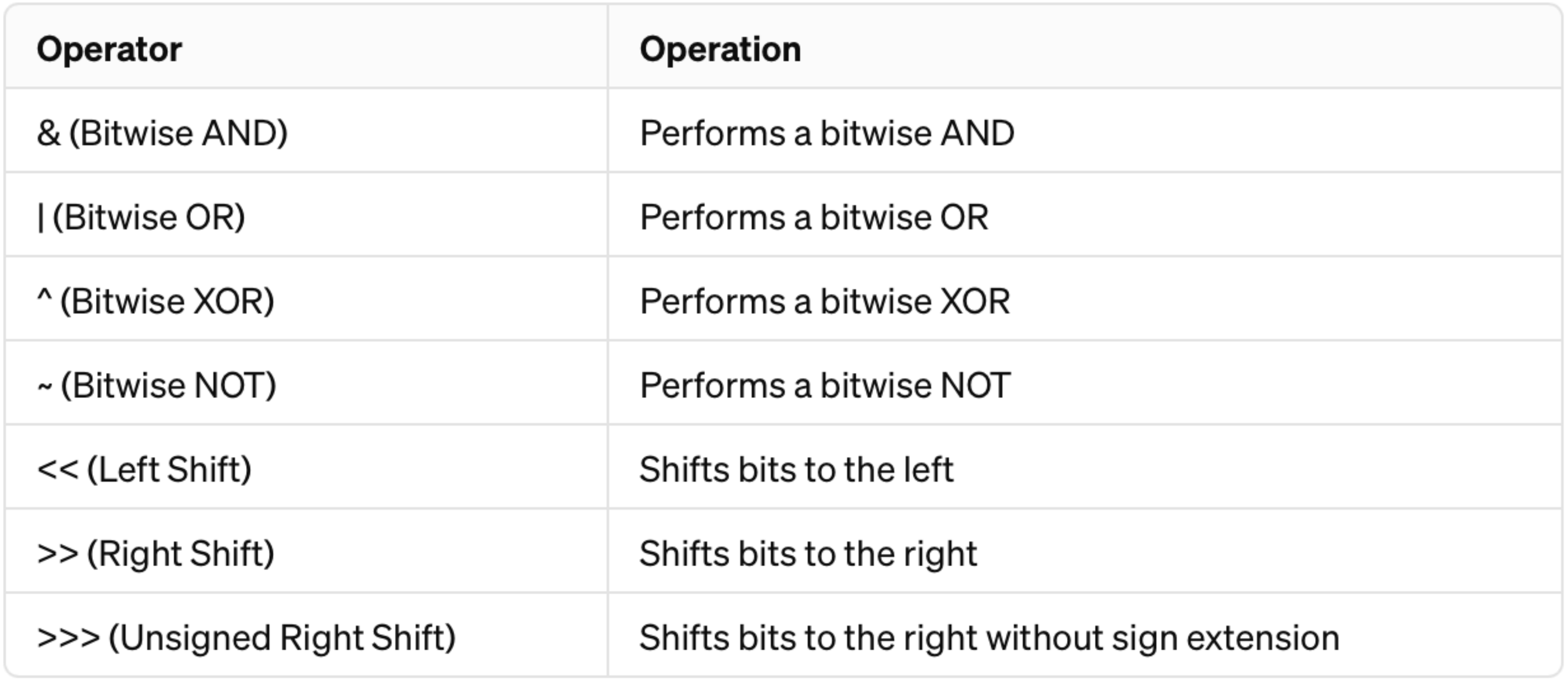
```
(age < 20) || (age > 30)
```

- height is equal to 150 or 200

```
(height == 150) || (height == 200)
```

Bitwise operations

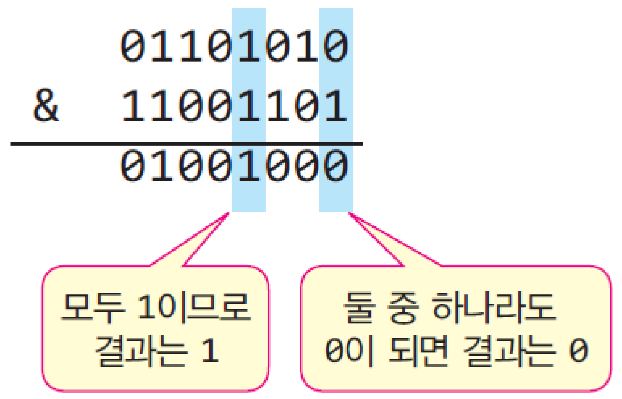
- Bitwise operations directly manipulate bits of integer types

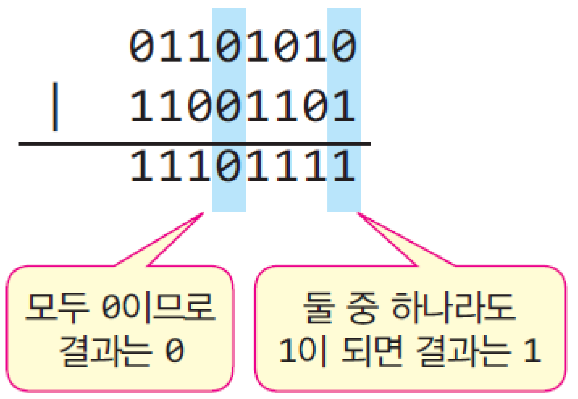


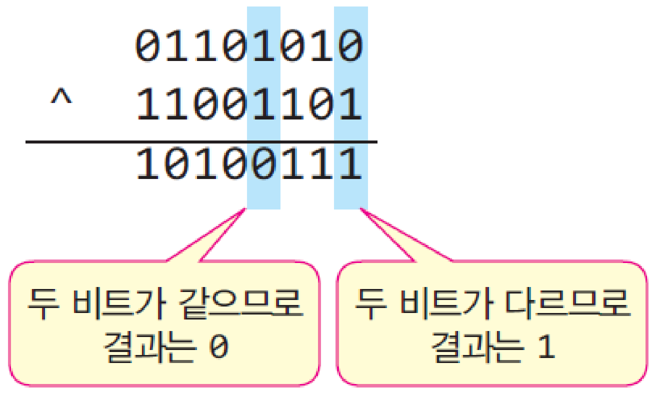
Operator	Operation
& (Bitwise AND)	Performs a bitwise AND
| (Bitwise OR)	Performs a bitwise OR
^ (Bitwise XOR)	Performs a bitwise XOR
~ (Bitwise NOT)	Performs a bitwise NOT
<< (Left Shift)	Shifts bits to the left
>> (Right Shift)	Shifts bits to the right
>>> (Unsigned Right Shift)	Shifts bits to the right without sign extension

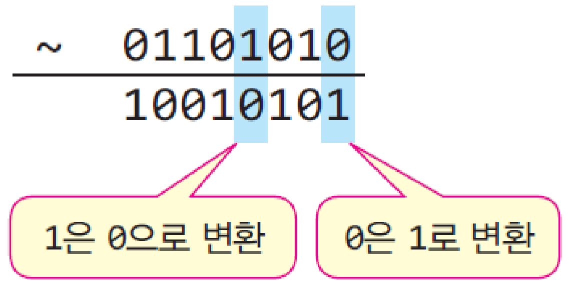
Bitwise operations

- Example for bitwise operations





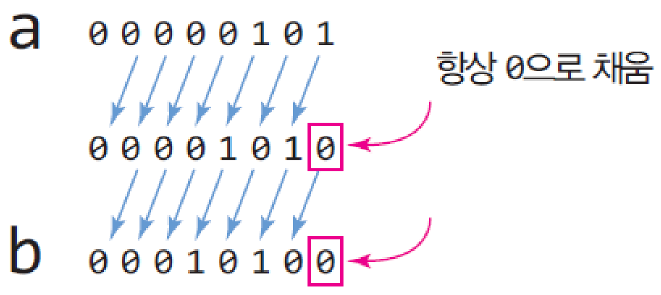




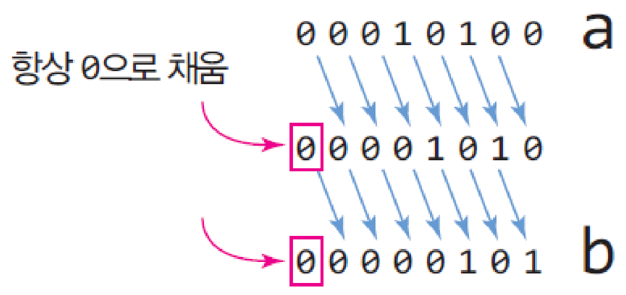
Bitwise operations

- Example for bitwise operations for bit-shift

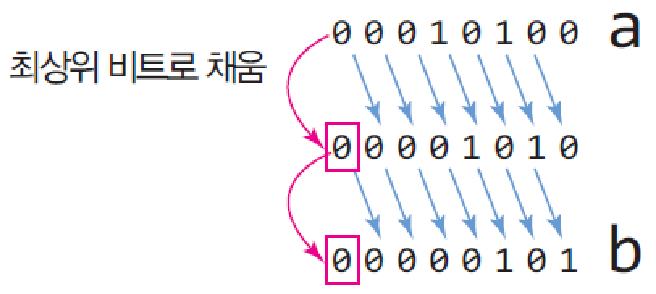
```
byte a = 5; // 5
byte b = (byte)(a << 2); // 20
```



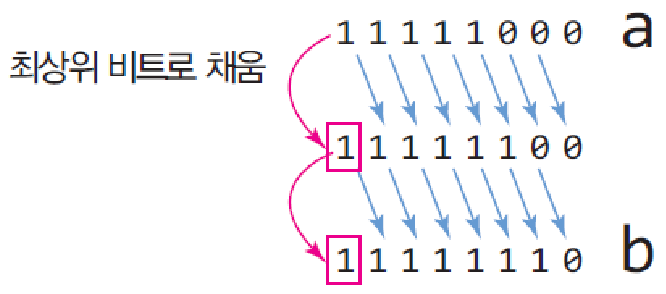
```
byte a = 20; // 20
byte b = (byte)(a >>> 2); // 5
```



```
byte a = 20; // 20
byte b = (byte)(a >> 2); // 5
```

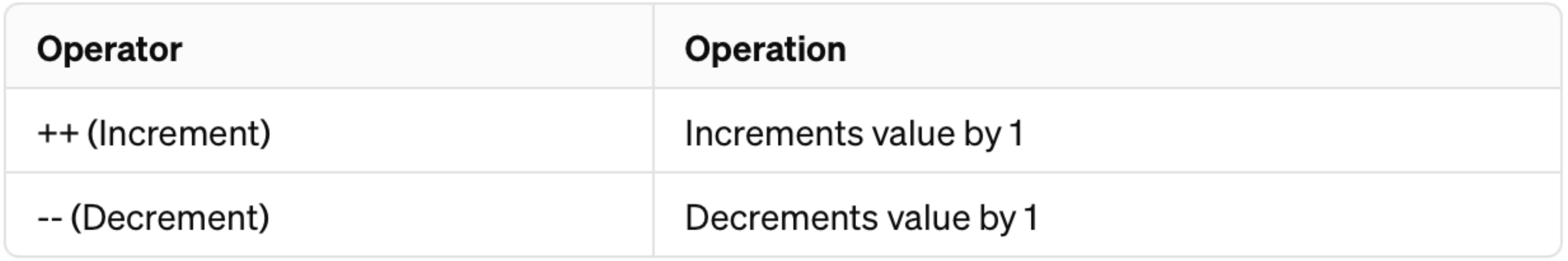


```
byte a = (byte)0xf8; // -8
byte b = (byte)(a >> 2); // -2
```



Increment/decrement operations

- Increment and decrement operators are unary operators
  - increase or decrease the value of a variable by 1, respectively
  - very useful in programming, especially for iterating through loops or simply modifying the value



Operator	Operation
++ (Increment)	Increments value by 1
-- (Decrement)	Decrements value by 1

- Two types of increment and decrement operators:
  - postfix increment (decrement)
    - when used, the variable is first used in the expression and then incremented (decremented)
  - prefix increment (decrement)
    - the variable is incremented (decremented) first and then used in the expression

Increment/decrement operations

- Examples

```
int a = 5;
int b = a++;
System.out.println(a + ", " + b); // 6, 5
```

```
int a = 5;
int b = ++a;
System.out.println(a + ", " + b); // 6, 6
```

```
int i = 1;
System.out.println(i++); // 1
System.out.println(++i); // 3

int j = 10;
System.out.println(j--); // 10
System.out.println(--j); // 8
```

Increment/decrement operations

- Examples

```
int d = 3;

a = d++;
System.out.println(a + ", " + d);
a = ++d;
System.out.println(a + ", " + d);
a = d--;
System.out.println(a + ", " + d);
a = --d;
System.out.println(a + ", " + d);
```

Quiz

- What is the output of the following code?

```
int x = 3;
int y = x-- + 5 + --x;

System.out.println(x);
System.out.println(y);
```

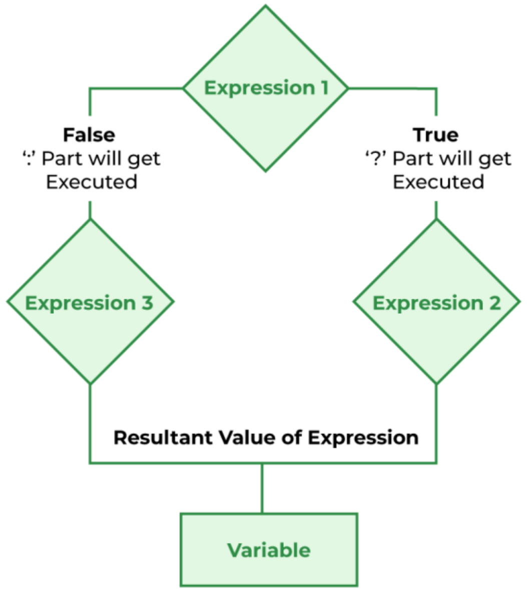
Ternary operation

- The ternary operation, a.k.a the conditional operator, is represented by '? :'
  - a shorthand for the 'if-else' statement and used to assign a value to a variable based on a condition

```
condition ? expression1 : expression2
```

- condition: This is a boolean expression that evaluate to either true or false
- expression1: This expression is evaluated and returned if the condition is true
- expression2: This expression is evaluated and returned if the condition is false

- flowchart of ternary operation


Expression 1
False
':' Part will get Executed
True
'?' Part will get Executed
Expression 3
Expression 2
Resultant Value of Expression
Variable

Ternary operation

- Example

```
int a = 5, b = 10;
int max = (a > b) ? a : b;
System.out.println("Maximum value is: " + max);

String result = (a % 2 == 0) ? "even" : "odd";
System.out.println("a is " + result);
```

```
int a = 30, b = 50;
System.out.println("The difference is" + ((a>b)?(a-b):(b-a)));
```

- How can we find the maximum number among three numbers (a, b, and c) by using ternary operation

End of slide