Computer Language

Basic operator

Agenda

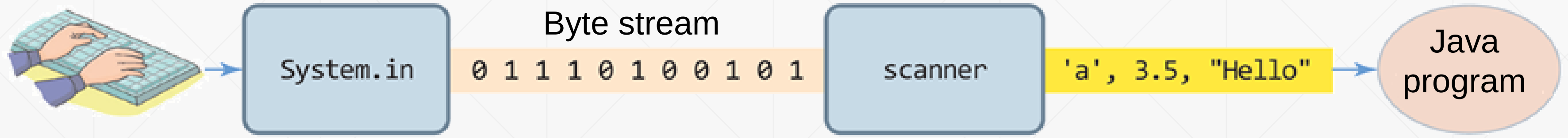
- Scanner & Print
- Basic Operators

Scanner & Print

Basic Operators

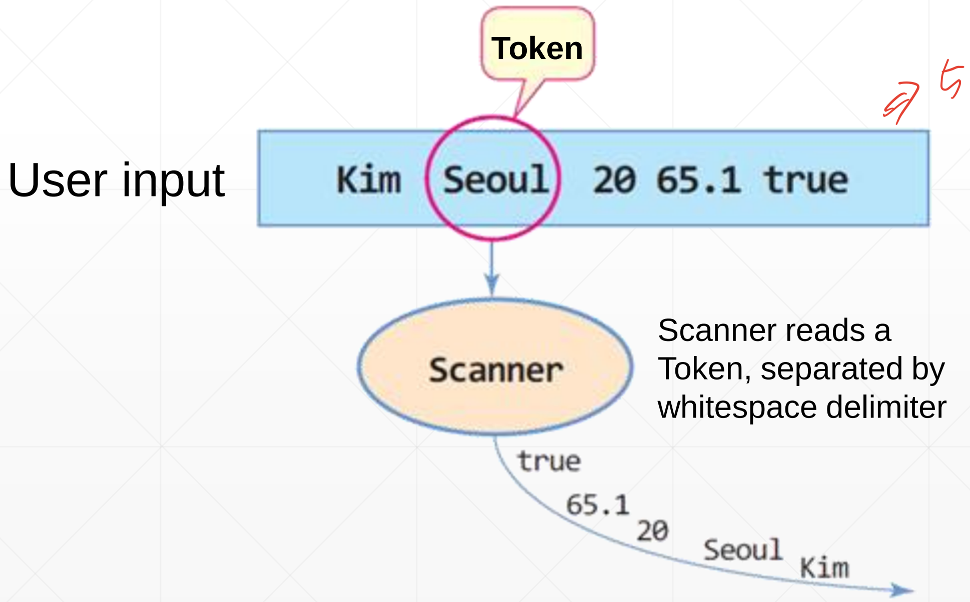
Scanner

- Let's take an input from the user!
  - System.in
    - Standard input stream in Java
    - Return byte-type data
    - Not developer-friendly
  - Scanner class
    - Need to import java.util.Scanner calss
    - Ask System.in to take a sequence of bytes from the user
    - Convert input bytes to data with an arbitrary type and then return!



Scanner (cont'd)

- Reading key inputs
  - Scanner reads an item based on the whitespace delimiter 구획문자
    - Whitespace character: '\t', '\f', '\r', '\n', ''
  - Scanner can read byte streams and convert it to various data types



Scanner reads a Token, separated by whitespace delimiter

5 tokens here

```
Scanner scanner = new Scanner(System.in);

String name = scanner.next();            // "Kim"
String city = scanner.next();  // "Seoul"
int age = scanner.nextInt();  // 20
double weight = scanner.nextDouble(); // 65.1
boolean single = scanner.nextBoolean(); // true
```

Scanner (cont'd)

- Scanner methods

Method	Description
next()	Reads a value as a String from the user
nextBoolean()	Reads a boolean value from the user
nextByte()	Reads a byte value from the user
nextDouble()	Reads a double value from the user
nextFloat()	Reads a float value from the user
nextInt()	Reads an int value from the user
nextLine()	Reads one line (before '\n') from the user
nextLong()	Reads a long value from the user
nextShort()	Reads a short value from the user
close()	Close a Scanner
hasNext()	Returns True if a token is given, otherwise waits for a new input. CTRL-Z will break this loop.

Scanner (cont'd)

- Example)

```
System.out.println("Input your name, city, age, and weight, separated by a single whitespace");
Scanner scanner = new Scanner(System.in);

String name = scanner.next(); // Read a string
System.out.print("My name is " + name + ", ");

String city = scanner.next(); // Read a string
System.out.print("city is " + city + ", ");

int age = scanner.nextInt(); // Read an integer value
System.out.print("age is " + age + "-years old, ");

double weight = scanner.nextDouble(); // Read a floating-point value
System.out.print("Weight is " + weight + "kg, ");

System.out.println("\nOk. Are you single?");
boolean single = scanner.nextBoolean(); // Read a boolean value
System.out.println(single);

System.out.println("\nAny comment?");
String comment = scanner.nextLine(); // Read a line
System.out.println("Your answer: " + comment);

scanner.close(); // Close the scanner
```

Print

- Basic functions to print out the contents

```
System.out.[print methods]
```

  - Print out something to the system's standard output

- Methods
  - println(contents): print out the contents and make a newline
  - print(contents): print out the contents
  - printf("formatting string", val1, val2, …): print out the values using the formatting string
  - Contents can be either literals or variables

Print (cont'd)

- printf("formatting string", val1, val2, …)
  - Prints the values using the formatting string

- Formatting string

```
% [argument_index$] [flags] [width] [.precision] conversion
```

  - Only %conversion is mandatory
  - argument_index$: the position of the argument in the argument list (e.g., 1$, 2$, …)
  - flags: controls the modification 수정 of output
  - width: the minimum number of characters to be written
  - precision: the digits after the radix point
  - conversion: determines how the argument should be formatted

Print (cont'd)

- Formatting string
  - Flags: controls the modification of output
    - '-' : left-justified 왼쪽정렬
    - '+' : includes sign, whether positive or negative
    - '0' : zero padding ⇒ width가 5인데, 123이 입력되었다면 00123 출력.
    - …
  - Conversion: determines how the argument should be formatted
    - 'd': integer
    - 'f', 'g': floating-point number f → float, g → double
    - 's': string
    - …

Print (cont'd)

- Example)

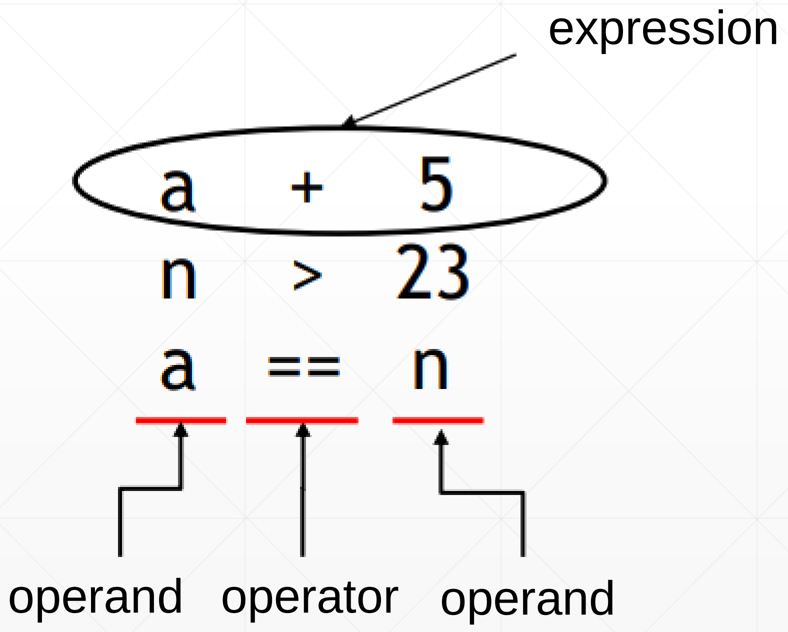
```
System.out.printf("%1$d %3$d %2$d\n", 10, 20, 30);
System.out.printf("%1$d %3$f %2$s\n", 10, "Hi", 20.5);
System.out.printf("%1$+5d %3$.2f %2$s\n", 10, "Hi", 20.5);
System.out.print("Hoy Hoy~");
System.out.println("Hey Hey~");
System.out.print("Hay Hay~");
```

Scanner & Print

Basic Operators

Operator

- First step to do something with values!
- Operators are special symbols that perform specific operations on one, two, or three *operands (literals or variables)*, and then return a result



Type	Operator	Type	Operator
In/decrement	++ -- unary	Bit	& | ^ ~
Arithmetic	+ - * / %	Conditional	&& || !
Shift	>> << >>>	Assignment	= *= /= += -= &= ^= |= <<= >>= >>>=
Relational	> < >= <= == !=		

Operator: Arithmetic

- Arithmetic computation

Description	Operator	Example	Result
Additive	+	25.5 + 3.6	29.1
Subtraction	-	3 - 5	-2
Multiplication	*	2.5 * 4.0	10.0
Division	/	5/2	2
Remainder	%	5%2	1

몫

나머지 1

- Ex) check if x is an odd or not

```
int x = n % 2;        // if x is 1, n is an odd number,  otherwise even
```

Operator: Arithmetic (cont’d)

- String concatenation
  - When one of operands for ‘+’ operation is String type
  - Example)

```
System.out.println("30"+5);
System.out.println(30+5);
System.out.println("Java "+11.0);
```

Operator: Increment/Decrement

- Unary increment and decrement operators
  - A single operand is required
  - Increase or decrease the value by 1

- Prefix operators (++a, --a)
  - In/decrease the value by 1 first, and then return the value

출력, 계산 순서 다름!

- Postfix operators (a++, a--)
  - Return the value first, and then in/decrease the value by 1

Operator: Increment/Decrement (cont’d)

- Example)

```
int myNum = 1;

System.out.println(myNum++);

System.out.println(++myNum);

System.out.println(--myNum);

System.out.println(myNum--);

System.out.println(myNum);
```

Operator: Relational

- Used to determine if one operand is greater than, less than, equal to, or not equal to another operand
  - Returns true or false

Description	Operator	Example	Result
Equal to	==	1 == 3	False
Not equal to	!=	1 != 3	True
Greater than	>	3 > 5	False
Greater than or equal to	>=	10 >= 10	True
Less than	<	3 < 5	True
Less than or equal to	<=	1 <= 0	False

Operator: Conditional

- Used to determine the logic between variables or values
  - Returns true or false

Description	Operator	Example	Result
Conditional AND $tt \to t$ $ff \to f$ $tf \to f$	&&	$(3\overset{t}{<}5)\ \&\&\ (1\overset{t}{==}1)$	True
(returns true if both operands are true)		$(3\overset{f}{>}5)\ \&\&\ (1\overset{t}{==}1)$	False
Conditional OR $tt \to t$ $ff \to f$ $tf \to t$	||	$(3\overset{t}{<}5)\ \|\|\ (1\overset{t}{==}1)$	True
(returns true if one of the statements is true)		$(3\overset{f}{>}5)\ \|\|\ (1\overset{t}{==}1)$	True
Complement	!	$!(3\overset{t}{<}5)$	False
(inverts the value of a Boolean)		$!(3\overset{f}{>}5)$	True

Operator: Relational + Conditional

- Example)

```
// true if one is 20s
(age >= 20) && (age < 30)

// what about 20 <= age < 30 ?



// true if a character c is a capital letter
(c >= 'A') && (c <= 'Z')

// true if (x,y) is inside the rectangle from (0,0) to (50,50)
(x>=0) && (y>=0) && (x<=50) && (y<=50)
```

Operator: Relational + Conditional (cont'd)

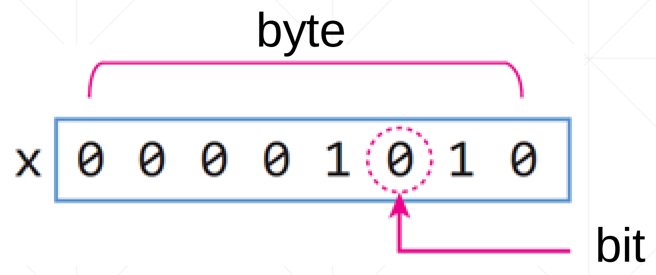
- Example)

```
System.out.println('a' > 'b');

System.out.println(3 >= 2);

System.out.println(-1 < 0);

System.out.println(3.45 <= 2);

System.out.println(3 == 2);

System.out.println(3 != 2);

System.out.println(!(3 != 2));

System.out.println((3 > 2) && (3 > 4));

System.out.println((3 != 2) || (-1 > 0));
```

Operator: Bit & Shift

- Operators for the bits of operands
  - Bitwise conditional operators
    - AND, OR, XOR, NOT operation on bits
  - Bit shift operators
    - Operations to shift the bits to the left/right

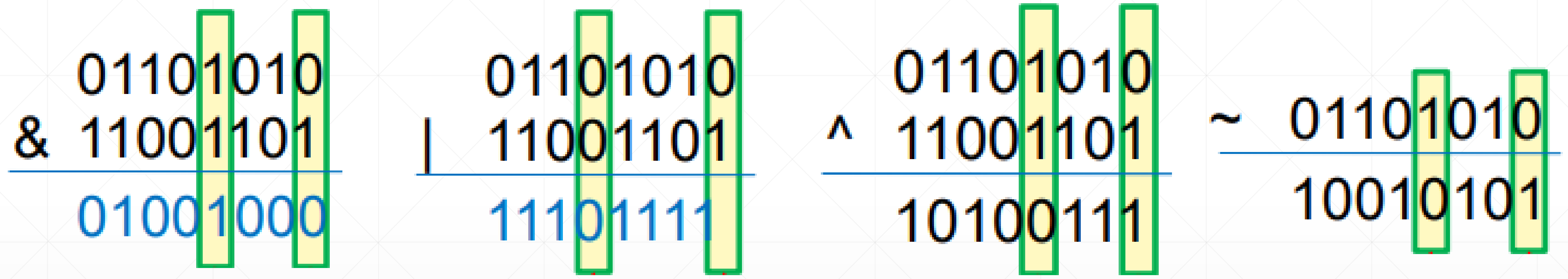
```
byte x = 10;
```



Description	Operator
AND (returns true if both bits are 1)	a & b
OR (returns true if one of the bits is 1)	a | b
NOT (inverts a bit pattern)	~ a
XOR (returns true if two bits are different)	a ^ b

Operator: Bit & Shift (cont'd)

- Operators for the bits of operands



Operator: Bit & Shift (cont'd)

- Operators for shifting the bits of operands

Description	Operator
Arithmetic Left shift When shifting left, the most-significant bit is lost, and a 0 bit is inserted on the other end	a << 1
Arithmetic Right shift When shifting right with an **arithmetic right shift**, the least-significant bit is lost, and the most-significant bit is *copied*	a >> b
Logical Right shift When shifting right with a **logical right shift**, the least-significant bit is lost, and a 0 bit is inserted on the other end	a >>> b

$\underbrace{00001111}_{=15} \rightarrow \underbrace{00011110}_{=30} \rightarrow \underbrace{00111100}_{=60}$

$\rightarrow \underbrace{01111000}_{=60}$

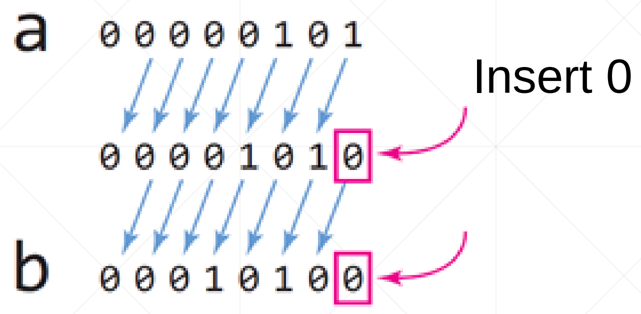
$\underbrace{1000\ 1000}_{=-120} \rightarrow \underbrace{0100\ 0100}_{=68}$

Operator: Bit & Shift (cont'd)

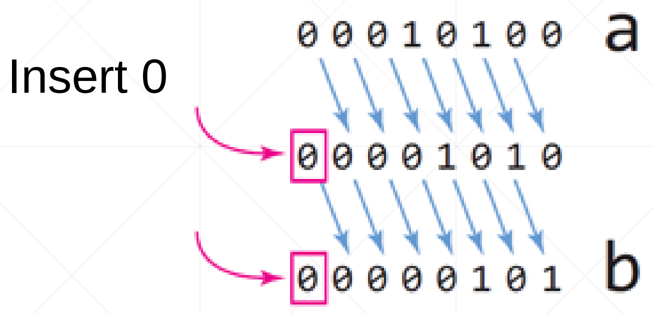
MSB ⇒ 최상위비트　　LSB ⇒ 최하위비트

- Example)

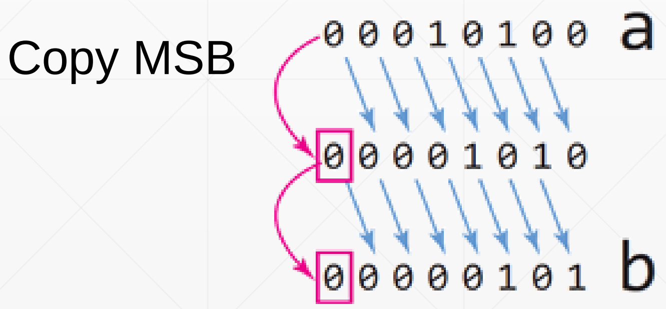
```
byte a = 5; // 5
byte b = (byte)(a << 2); // 20
```



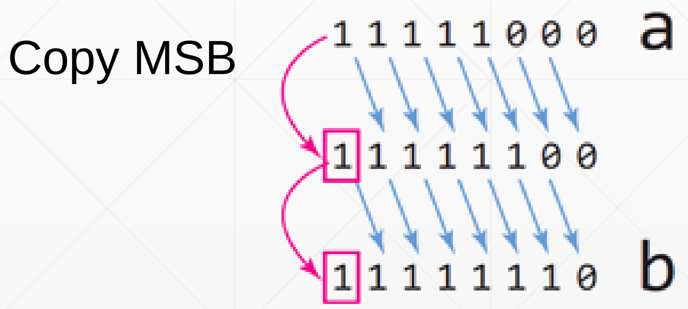
```
byte a = 20; // 20
byte b = (byte)(a >>> 2); // 5
```



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



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



Operator: Bit & Shift (cont'd)

- Example)

```
short a = (short)0b0101010111111111;
short b = (short)0x00ff;
                 0b 00000000 1111 1111

System.out.printf("%04x\n", (short)(a & b)); // bitwise AND
System.out.printf("%04x\n", (short)(a | b)); // bitwise OR
System.out.printf("%04x\n", (short)(a ^ b)); // bitwise XOR
System.out.printf("%04x\n", (short)(~a)); // bitwise NOT


int c = 20;
int d = -8;


System.out.println(c <<2);
System.out.println(c >>2); // arithmetic right shift
System.out.println(d >>2); // arithmetic right shift
System.out.println(d >>>2); // logical right shift
```

printf("%04x"): print a 4-digit number with a hexadecimal (0~f) format

Operator: Assignment

- Operators to assign values to variables

- Simple assignment (=)
  - Ex) myValue = 5; // assign 5 to the variable *myValue*

Operator: Assignment (cont'd)

- Operators to assign values to variables
- Compound assignment

Operator	Example	Same As
+=	x += 3	x = x + 3
-=	x -= 3	x = x - 3
*=	x *= 3	x = x * 3
/=	x /= 3	x = x / 3
%=	x %= 3	x = x % 3
&=	x &= 3	x = x & 3
|=	x |= 3	x = x | 3
^=	x ^= 3	x = x ^ 3
>>=	x >>= 3	x = x >> 3
<<=	x <<= 3	x = x << 3

Operator: Precedence

- Operators with higher precedence are evaluated before operators with relatively lower precedence

- Top priority: ( )

- Associativity
  - A rule for the operators with equal precedence
  - All binary operators except for the assignment operators are evaluated from left to right
    - Assignment operators are evaluated right to left

Operator: Precedence (cont'd)

high

Operators	Precedence	Associativity
postfix	expr++ expr--	←
unary	++expr --expr +expr -expr ~ !	
multiplicative	* / %	→
additive	+ -	
shift	<< >> >>>	
relational	< > <= >= instanceof	
equality	== !=	
bitwise AND	&	
bitwise XOR	^	
bitwise OR	|	
logical AND	&&	
logical OR	||	
ternary	? :	
assignment	= += -= *= /= %= &= ^= |= <<= >>= >>>=	←

low

Q&A

- Next week (eClass video)
  - Conditions & Loop