# Computer Language

Variables & Types

# Agenda

- Variables
- Types

# Variables

Types

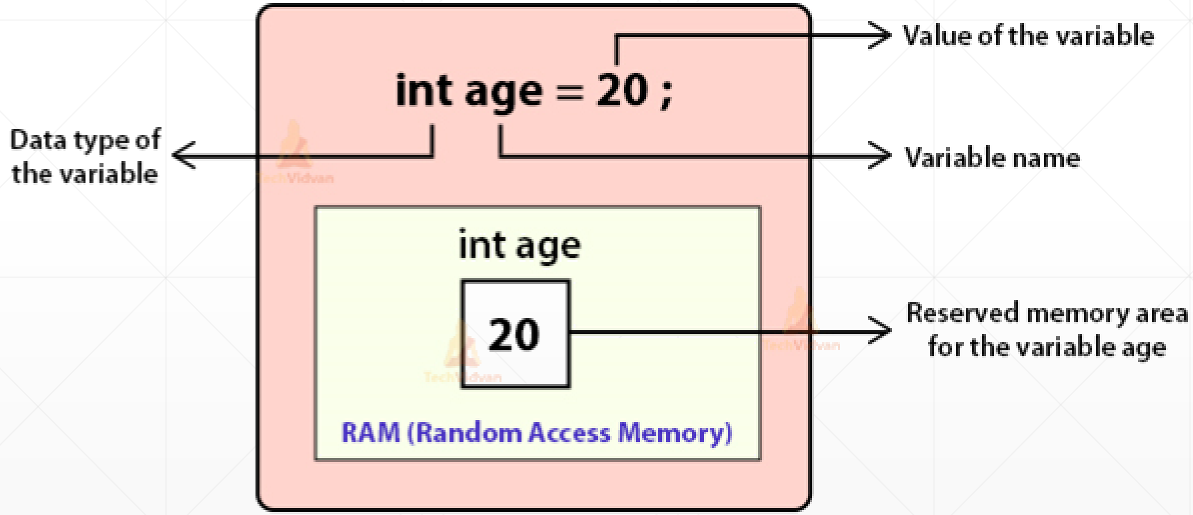
# Variables

- A way to store data in a computer
  - Declaration of variables
    - What kind (type) of data?
    - How to call it? (name)

type Variable name

int age ;

double value ;


int age = 20 ;
Value of the variable
Data type of the variable
Variable name
int age
20
Reserved memory area for the variable age
RAM (Random Access Memory)

  - The contents of a variable can be changed (variable)

# Variables (cont’d)

- Naming rule
  - Variable names are case-sensitive `int age ;` != `int AGE ;`
  - Unlimited-length sequence of Unicode letters and digits
  - Can begin with a letter, the dollar sign "$", or the underscore character "_“
  - Special characters like “@”, “!”, “#”, and whitespaces are not allowed
    - Example) price, $price, _price (possible)
    - Example) 1v, @speed, $#value (impossible)
  - Java keywords are not allowed
  - Boolean literal (true/false), null literals are not allowed

# Variables (cont'd)

- Naming rule
  - Java keywords are not allowed

| | | | | |
|---|---|---|---|---|
| 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 |

# Variables (cont’d)

- Naming convention
  - Begin your variable names with a letter, not "$" or "_“
  - Use full words instead of cryptic abbreviations
    - Speed, gear, age are more intuitive than *s*, *g*, and *a*
  - If your variable name consists of only one word, spell that word in all lowercase letters
    - Example) age, grade
  - If your variable name consists of more than one word, capitalize the first letter of each subsequent word
    - Example) myAge, myFinalGrade

# Variables (cont’d)

- Valid variable names

```
int     name;
char    student_ID;
int     whatsYourNameMyNameIsKitae;
int     barChart;    int barchart;
int     가격;
```

- Invalid variable names

```
int 3Chapter;                  // use of number for the first character
double  if;                    // java keyword (if)
char   false;                  // java keyword (false)
float   null;                  // java keyword (null)
short ca%lc;                   // special character (%)
```

# Variables (cont’d)

## Initialization

- Variables needs to be initialized before being used
- Use assignment operator (‘=’)  to assign/initialize a value to the variable

```
int radius;
radius = 10;
```

// declaration
// initialization

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

// declaration & initialization

# Variables (cont’d)

- Access
  - Variables needs to be initialized before being used
  - Variables can be accessed by its name
  - The value of variables can be used for printing, calculation, etc
  - The value of a variable can be copied to another variable

- Example)

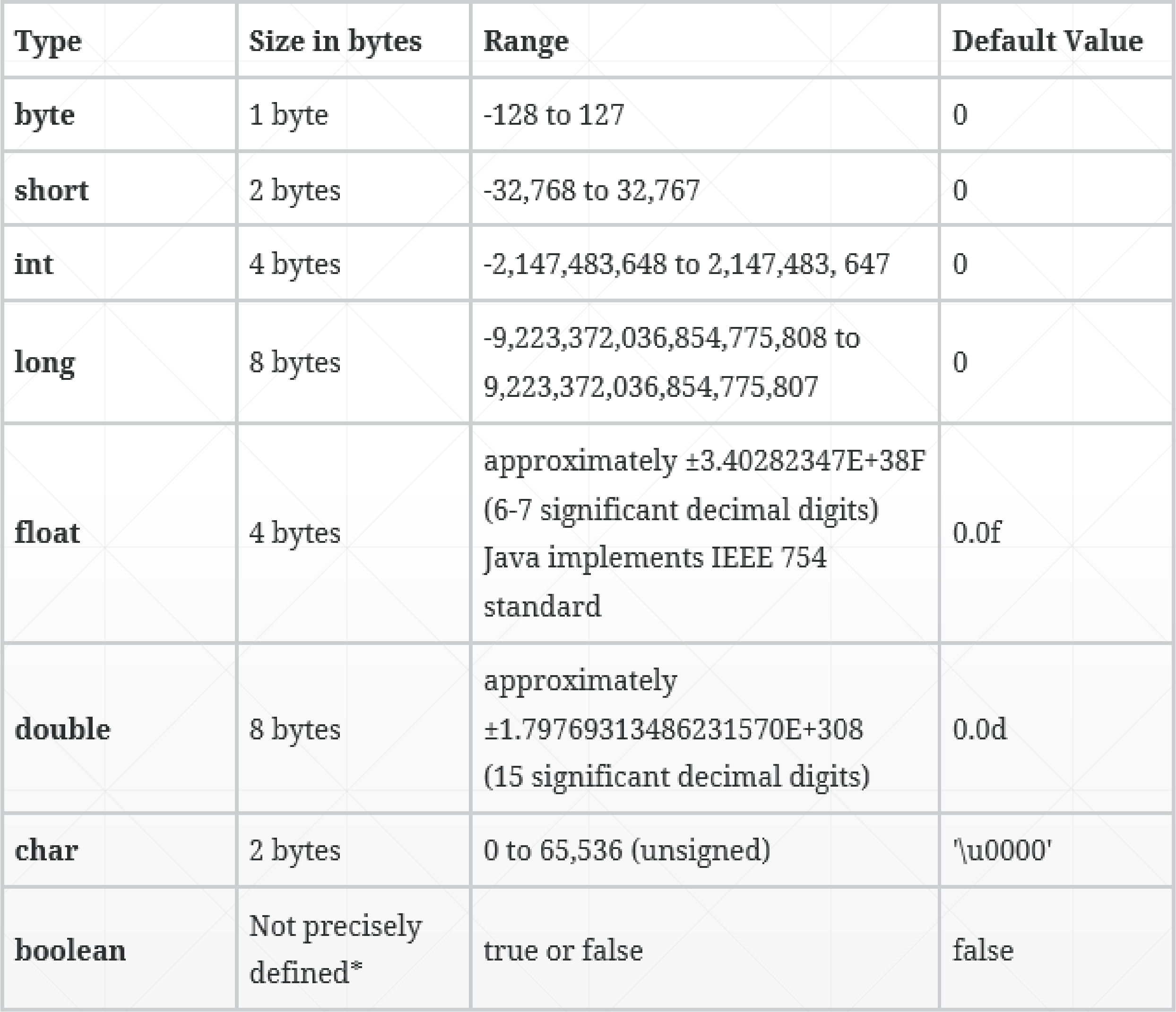
```
int hour = 3, minute = 5;
System.out.println(hour + "h" + minute + "m");
System.out.println(hour * 60 + minute + "m");
int totalMinute = hour * 60 + minute;
System.out.println(totalMinute);
```

Variables

# Types

# Types

- Java data types
  - Primitive types
    - Types for number, character, boolean
  - Non-primitive types
    - String, array, class, etc.

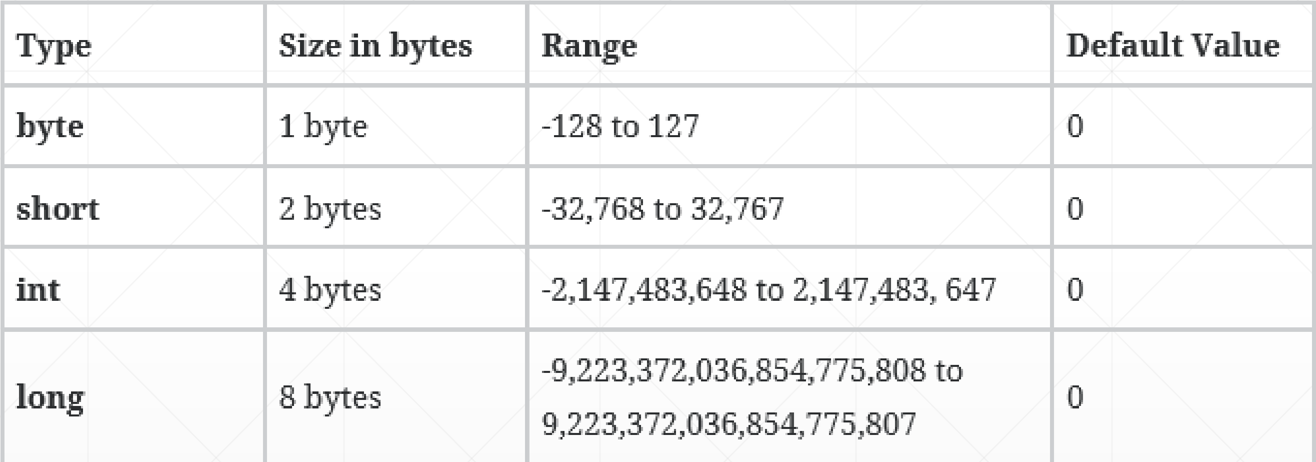


| Type | Size in bytes | Range | Default Value |
|---|---|---|---|
| **byte** | 1 byte | -128 to 127 | 0 |
| **short** | 2 bytes | -32,768 to 32,767 | 0 |
| **int** | 4 bytes | -2,147,483,648 to 2,147,483, 647 | 0 |
| **long** | 8 bytes | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0 |
| **float** | 4 bytes | approximately ±3.40282347E+38F (6-7 significant decimal digits) Java implements IEEE 754 standard | 0.0f |
| **double** | 8 bytes | approximately ±1.79769313486231570E+308 (15 significant decimal digits) | 0.0d |
| **char** | 2 bytes | 0 to 65,536 (unsigned) | '\u0000' |
| **boolean** | Not precisely defined* | true or false | false |

- Literal
  - Source code representation of a fixed value
  - Represented directly in the code without requiring computation
  - Can be assigned to a variable

# Types: Number

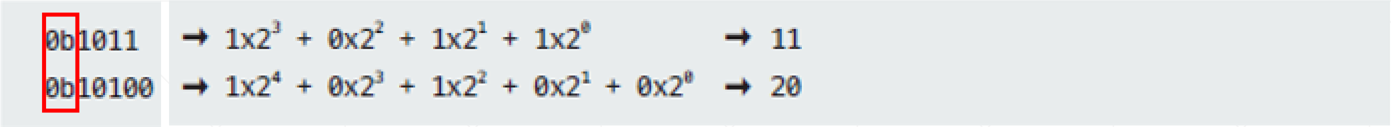
- Integer types
  - Stores whole numbers without decimals (fraction)
    - Includes positive and negative



| Type | Size in bytes | Range | Default Value |
|---|---|---|---|
| **byte** | 1 byte | -128 to 127 | 0 |
| **short** | 2 bytes | -32,768 to 32,767 | 0 |
| **int** | 4 bytes | -2,147,483,648 to 2,147,483, 647 | 0 |
| **long** | 8 bytes | -9,223,372,036,854,775,808 to 9,223,372,036,854,775,807 | 0 |

# Types: Number (cont'd)

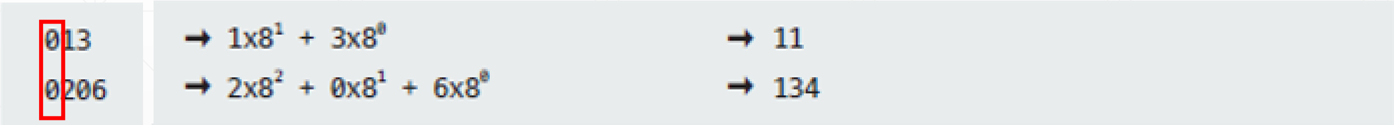
- Integer types
  - Integer literal
    - Type of ‘int’ unless the literal ends with the letter ‘L’ or ‘l’
    - Binary system (stating with 0b or 0B, 0/1 representation)



0b1011 → $1x2^3 + 0x2^2 + 1x2^1 + 1x2^0$ → 11

0b10100 → $1x2^4 + 0x2^3 + 1x2^2 + 0x2^1 + 0x2^0$ → 20

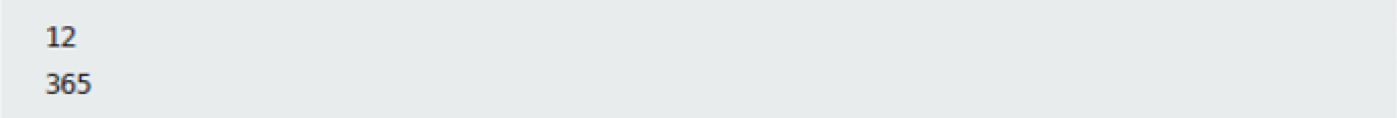
- Octal system (stating with 0, 0-7 representation)



013 → $1x8^1 + 3x8^0$ → 11

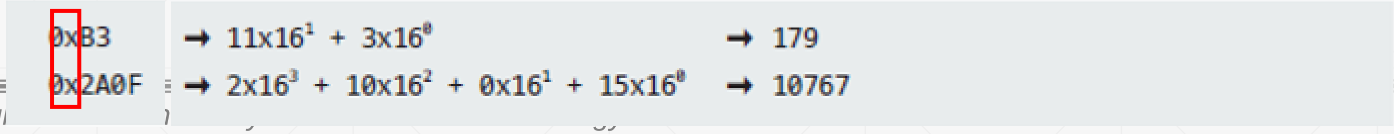
0206 → $2x8^2 + 0x8^1 + 6x8^0$ → 134

- Decimal system



```
12
365
```

- Hexadecimal system (starting with 0x, 0-F representation)



0xB3 → $11x16^1 + 3x16^0$ → 179

0x2A0F → $2x16^3 + 10x16^2 + 0x16^1 + 15x16^0$ → 10767

# Types: Number (cont'd)

- Integer types
  - Use of underscore ('_') character in Integer literal
    - Use of underscore to separate groups of digits is allowed
    - Use of underscore can improve the readability of the code

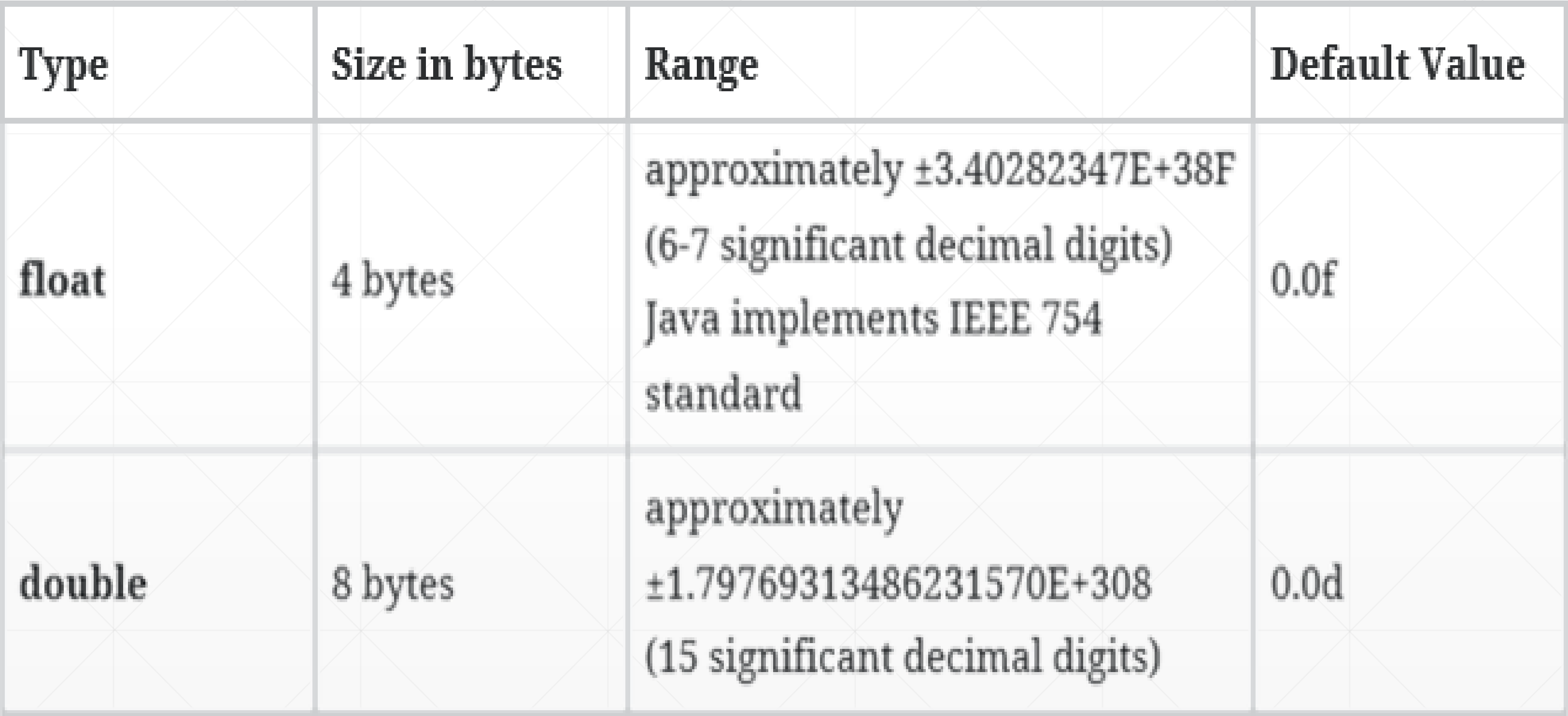
```
int price = 20_100;                                    // 20100
long cardNumber = 1234_5678_1357_9998L;                // 1234567813579998L
long controlBits = 0b10110100_01011011_10110011_111110000;
long maxLong = 0x7fff_ffff_ffff_ffffL;
int age = 2______5;                                    // 25
```

  - Underscores cannot be used in the following places:
    - At the beginning or end of a number
    - Adjacent to a decimal point in a floating-point literal
    - Prior to an F or L suffix
    - Inside prefix 0b and 0x
    - In positions where a string of digits is expected

```
int x = 15_;                   // error. At the end of a number
double pi = 3_.14;             // error. Adjacent to a decimal point
long idNum = 981231_1234567_L; // error. Prior to an F or L suffix
int y = 0_x15;                 // error. Inside the prefix '0x'
```

# Types: Number (cont'd)

- Floating point types
  - Represents numbers with a fractional part, containing one or more decimals



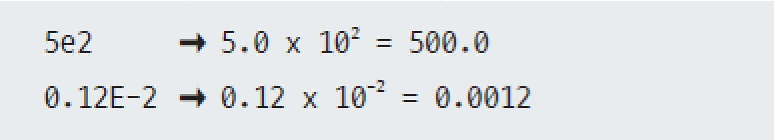
| Type | Size in bytes | Range | Default Value |
|---|---|---|---|
| **float** | 4 bytes | approximately ±3.40282347E+38F (6-7 significant decimal digits) Java implements IEEE 754 standard | 0.0f |
| **double** | 8 bytes | approximately ±1.79769313486231570E+308 (15 significant decimal digits) | 0.0d |

# Types: Number (cont'd)

## Floating point types

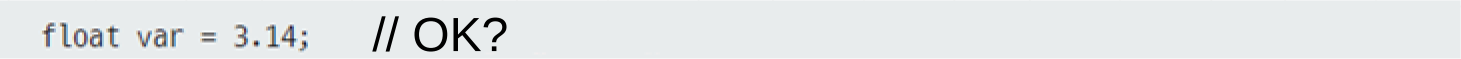
### Floating point literal

- Basically, type of ‘double’ and it can optionally end with the letter ‘D’ or ‘d’
- Type of ‘float’ if the literal ends with the letter ‘F’ or ‘f’
- Can represent scientific (floating-point) number with an “e”



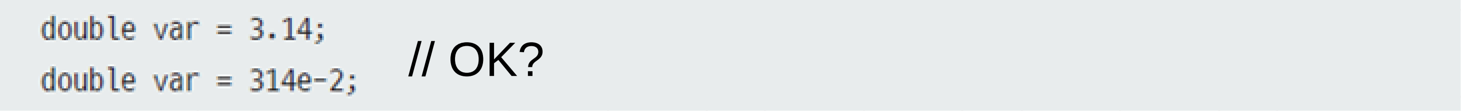
```
5e2       → 5.0 x 10^2 = 500.0
0.12E-2 → 0.12 x 10^-2 = 0.0012
```

- Example)



```
float var = 3.14;
```

// OK?

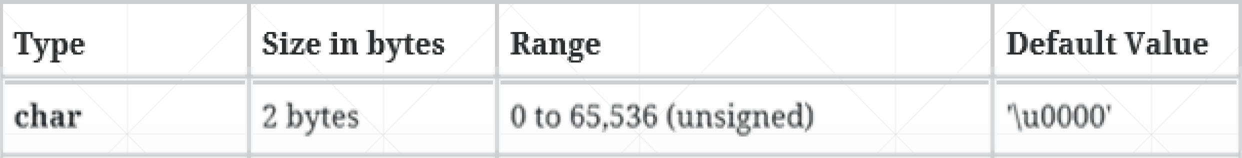


```
double var = 3.14;
double var = 314e-2;
```

// OK?

# Types: Character

- Char type
  - Used to store a single character (0000 to FFFF)
    - Unicode (utf-16) character
    - Unicode table: https://unicode-table.com/en/blocks/
    - The character must be surrounded by single quotes, like 'A' or 'c'



| Type | Size in bytes | Range | Default Value |
|---|---|---|---|
| **char** | 2 bytes | 0 to 65,536 (unsigned) | '\u0000' |

```
char a = 'A';
char b = '글';
char c = ‘₩u0041’; // Unicode of 'A’
char d = ‘₩uae00’; // Unicode of '글'
```

# Types: String

- String type
  - Non-primitive type
  - Used to store a sequence of characters (i.e., string)
  - The string must be surrounded by double quotes, like “Hello, Java!”
  - String literal can be assigned to a String object

```
String str = “Good";
```

- Escape character
  - A character starting with backslash ('\’)
  - Can be used to represent special character
  - Can be used to control printing of a string

# Types: String (cont’d)

- Escape character

| Escape character | Purpose | Escape character | Purpose |
|---|---|---|---|
| \b | Backspace | \n | Line feed |
| \r | Carriage return | \t | Tab |
| \' | Print ' | \” | Print ' |
| \\ | Print \ | \u(Unicode) | Print character based on Unicode |

```
System.out.println("I love ₩"Java₩"");
System.out.println("Name ₩tsID ₩tAge");
System.out.println("Computer₩nLanguage₩u2661");
```

# Types: Boolean

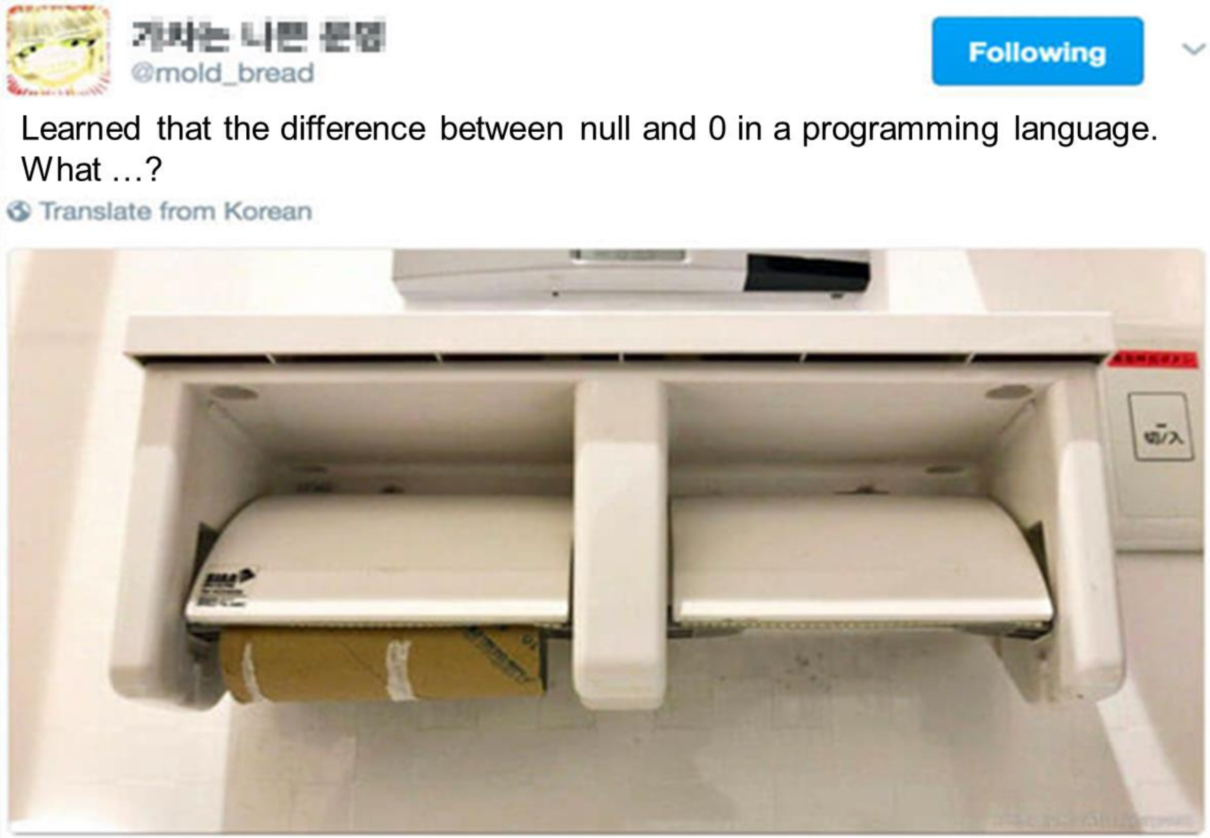
- Boolean type
  - Represents *true* or *false*
  - Can be stored in a Boolean type variable or used with condition statements

```
boolean myValue = true;
System.out.println(myValue);
myValue = 10 < 15;
System.out.println(myValue);
myValue = 10 == 15;
System.out.println(myValue);
```

# Types: Null

- Null literal
  - Represents *“not existent”*
  - Can be used for a reference type (will be discussed later)

```
int n = null;            // error!
String str = null;
```



@mold_bread

Learned that the difference between null and 0 in a programming language. What …?

Translate from Korean

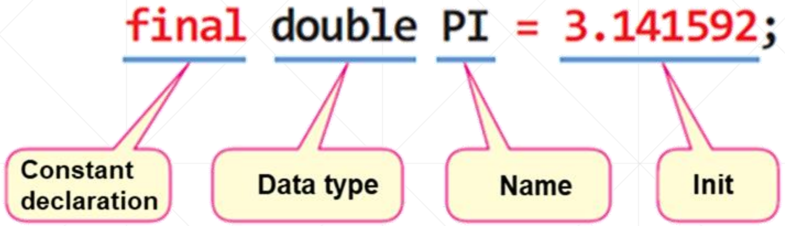
# Constant

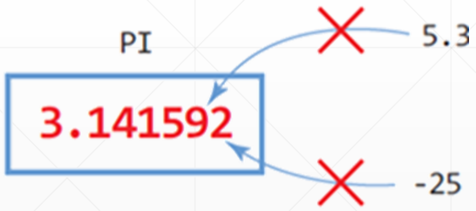
- Final variable (constant)
  - Unchangeable, read-only variable
  - Can be declared by adding *final* keyword

```
final double PI = 3.141592;

System.out.println(PI);

PI = 5.00;
```





  - Naming convention
    - All uppercase with words separated by underscores ("_")

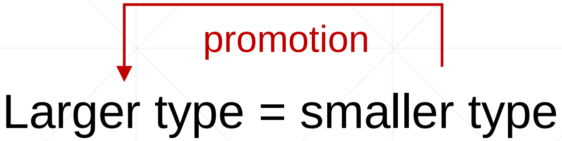
```
static final int MIN_WIDTH = 4;
static final int MAX_WIDTH = 999;
static final int GET_THE_CPU = 1;
```

# Type Conversion: Promotion

- Automatic conversion
  - Converting a smaller size type to a larger size type

```
byte -> short -> int -> long -> float -> double
```

  - Done automatically when,
    - Passing a smaller size type to a larger size type


promotion

Larger type = smaller type

    - Performing an arithmetic operation with integer-type values
      - Byte, short, char type values are automatically converted to int type values
    - Performing an arithmetic operation with different types of values
      - Arithmetic operation is only performed with the same type operands
      - Smaller type value is automatically converted to a larger type value

# Type Conversion: Promotion (cont’d)

- Automatic conversion
  - Example) Passing a smaller size type to a larger size type

```
long longValue = 500000L;
double doubleValue = longValue;
System.out.println(longValue);
System.out.println(doubleValue);
```

```
char chValue = 'A';
int intValue = chValue;
System.out.println(intValue);
short shortValue = 10;
char chValue2 = shortValue;
```

# Type Conversion: Promotion (cont’d)

- Automatic conversion
  - Example) Performing an arithmetic operation with integer-type values

```
short x= 10;
short y = 20;

short total = x + y;
System.out.println(total);
```

  - Example) Performing an arithmetic operation with different types of values

```
int intValue = 10;
int anotherValue = 3;
double doubleValue = 3;

System.out.println(intValue / anotherValue);
System.out.println(intValue / doubleValue);
```

# Type Conversion: Casting

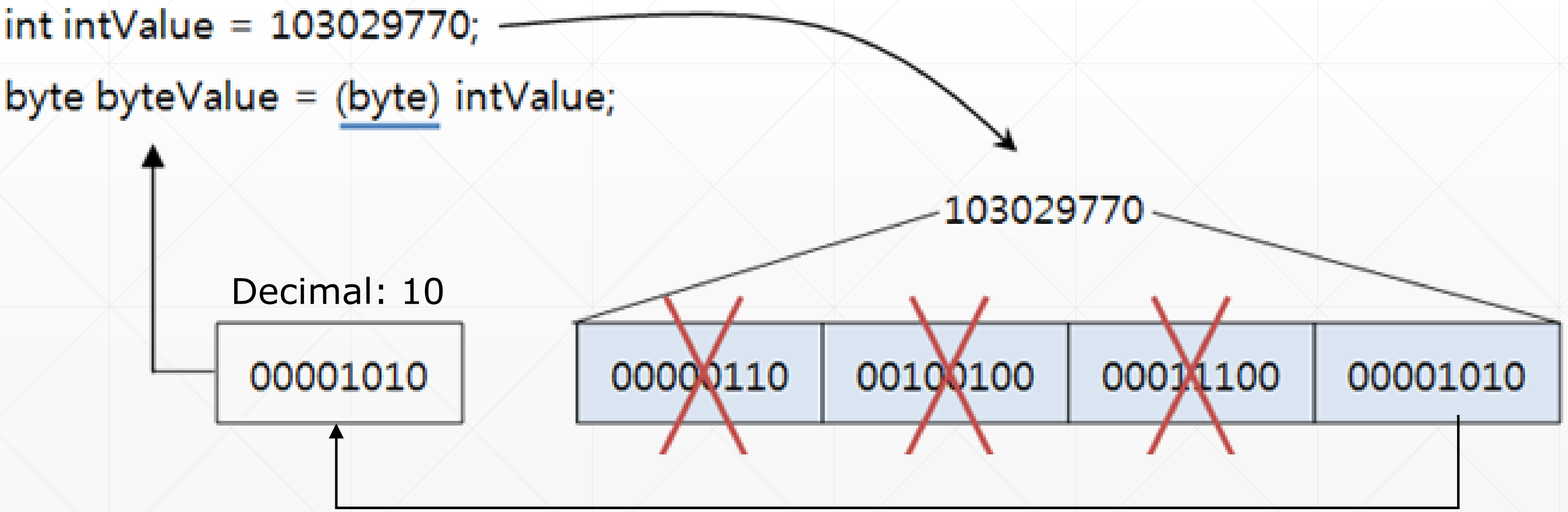
- Manual conversion
  - Converting a larger size type to a smaller size type

    `byte -> short -> int -> long -> float -> double`

  - Done manually by casting operation '(type)'
  - May result in the loss of a value

casting

Smaller type = (smaller type) larger type



```
int intValue = 103029770;
byte byteValue = (byte) intValue;
```

# Type Conversion: Casting (cont’d)

- Manual conversion
  - Example) casting from double to int

```
double myDouble = 11.50;
int myInt = (int) myDouble;

System.out.println(myDouble);
System.out.println(myInt);
```

  - Example) casting from int to char (to print a character!)

```
int myInt = 67;
char myChar = (char) myInt;

System.out.println(myInt);
System.out.println(myChar);
```

# Q&A

- Next week
  - Basic operators