Language Basics

CSX3002/ITX2001 Object-Oriented Concepts and Programming
CS4402 Selected Topic in Object-Oriented Concepts
IT2371 Object-Oriented Programming

Kwankamol Nongpong, Ph.D.
Assumption University

Structure of Java Program

```
public class HelloWorld {
  public static void main(String[] args) {
    System.out.println("Hello World");
    System.out.println("How are you?");
    System.out.print("I'm fine.");
    System.out.print("Thank you");
  }
}
```

Standard I/O

Displaying Outputs

```
System.out.print(     );
```

Show the output specified in the parenthesis.

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

Show the output and start a new line.

Reading Inputs

- There are several ways to do so but we'll just use the following approach.
- Need to use Scanner class
- Scanner comes with Java Library
  - You need to import the class to your code before using it.

```
import java.util.scanner;

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

Put it before the class declaration

Tell the scanner to fetch the input from standard input (console)

Can we read inputs now?

- Not quite.
- You need space to store the values of those inputs.
- How to do so?
  - You need to declare a variable first.

Example

```
import java.util.Scanner;

public class HelloYou {

  public static void main(String[] args) {

    Scanner reader = new Scanner(System.in);

    String name = reader.nextLine();

    System.out.println("Hello " + name);

    System.out.println("How are you?");

    reader.close();

  }

}
```

Always close the Scanner when you are done.

Variables

Variable Declaration

```
<type> <name>;
```

Declaration without initialization

```
<type> <name> = <expression>;
```

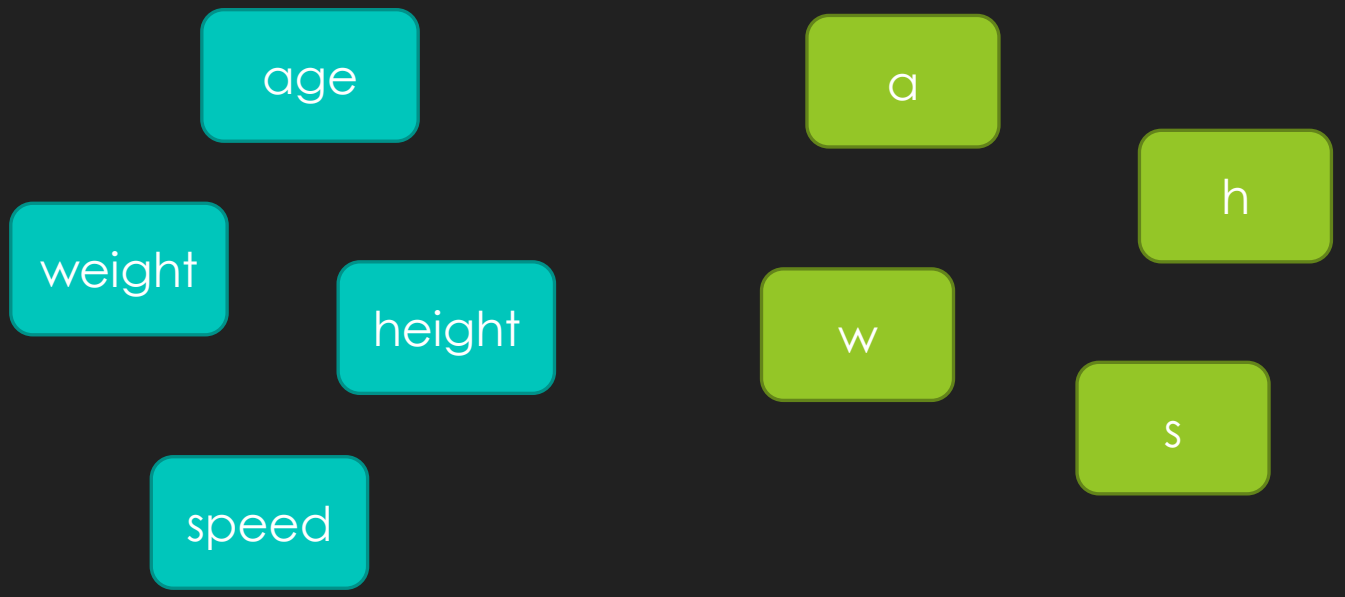
Declaration with an initialization

Variable Naming

- Case-sensitive
- Sequence of letters and digits, underscore (_), dollar sign ($)
- Begin with letters, underscore or dollar sign
  - Note that having a variable beginning with a dollar sign is not conventional
- Keywords and reserved words can't be used.
- White space is not allowed.

Notes on Variable Naming

- Always use full words not abbreviations
  - It makes your code self-documenting



Data Types

Types in Java

- Primitive Types
- Reference Types

Primitive Types

- Primitive types are special data types built into the language.
  - Predefined by the language and is named by a reserved keyword
  - Are not objects created from a class
- For instance,
  - int
  - float
  - double
  - boolean
  - char

Java Primitive Types

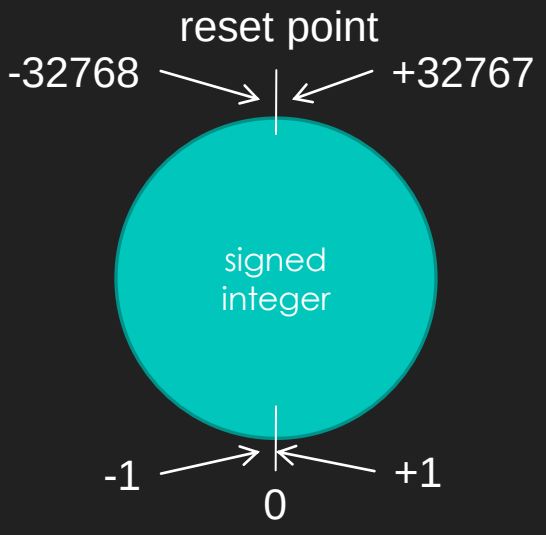
- Integer Types
- Floating-Point Numbers
- Boolean
- Character

Integer Types

- byte: 8-bit signed integer
- short: 16-bit signed integer
- int: 32-bit signed integer
- long: 64-bit signed integer

Arithmetic Overflow

- Suppose short is 16 bits
  - The value of a short variable ranges from -32768 to 32767



Example: Arithmetic Overflow

```
public class ArithmeticOverflowSample {

    public static void main(String[] args) {
        short positiveShort = 32767;
        positiveShort++;
        System.out.println(positiveShort);
        short negativeShort = -32768;
        negativeShort--;
        System.out.println(negativeShort);
    }
}
```

short is 16-bit signed integer

Floating-Point Numbers

- float: 32-bit floating point
- double: 64-bit floating point

Boolean

- boolean: has two possible values
  - true
  - false
- What is the size of boolean data type?

Character

- char: 16-bit Unicode character
- For instance,
  - '1' has the Unicode of '\u0031' (49)
  - 'A' has the Unicode of '\u0041' (65)
  - 'Z' has the Unicode of '\u005A' (90)
  - 'a' has the Unicode of '\u0061' (97)
  - 'z' has the Unicode of '\u007A' (122)

Literals

- A *literal* is the source code representation of a fixed value
- They are represented directly in your code without requiring computation.

Examples of Using Literals

- It's possible to assign a literal to a variable of a primitive type:

```
boolean result = true;
char capitalC = 'C';
byte b = 100;
short s = 10000;
int i = 100000;
```

Integer Literals

- For general-purpose programming, the decimal system is likely to be the only number system you'll ever use.
- However, if you need to use another number system, use the prefix to indicate the base.
  - 0x indicates hexadecimal
  - 0b indicates binary

Example: Integer Literals

```
int decVal = 26;
```

Number 26 in decimal

```
int hexVal = 0x1a;
```

Number 26 in hexadecimal

```
int binVal = 0b11010;
```

Number 26 in binary

Feature for Readability

- In Java SE 7 and later, any number of underscore characters (_) can appear anywhere between digits in a numerical literal.
- This feature enables you to separate groups of digits in numeric literals, which can improve the readability of your code.
- You can use an underscore character to separate digits in groups of three, similar to how you would use a punctuation mark like a comma, or a space, as a separator.

Example: Using Underscores as Separators

```
long creditCardNumber = 1234_5678_9012_3456L;
long socialSecurityNumber = 999_99_9999L;
float pi = 3.14_15F;
long hexBytes = 0xFF_EC_DE_5E;
long hexWords = 0xCAFE_BABE;
long maxLong = 0x7fff_ffff_ffff_ffffL;
byte nybbles = 0b0010_0101;
long bytes =
0b11010010_01101001_10010100_10010010;
```

What about Reference Types?

- We will talk about them later.

Now…
Let's go back to reading Inputs

Methods to Read Inputs

Method	Description
nextInt()	Scans the next token of the input as an int.
nextFloat()	Scans the next token of the input as a float.
nextDouble()	Scans the next token of the input as a double.
nextBoolean()	Scans the next token of the input into a boolean value and returns that value.

Fore more information, visit
http://docs.oracle.com/javase/7/docs/api/java/util/Scanner.html

Operators

Operators

- Unary Operators
- Binary Operators
- Ternary Operators

Operators

- Arithmetic Operators
- Relational Operators
- Logical/Conditional Operators
- Assignment Operators
- Bitwise Operators

Arithmetic Operators - Binary

Operator	Description	Example
+	Additive Operator Also used for String concatenation	x + y
-	Subtraction Operator	x – y
*	Multiplication Operator	x * y
/	Division Operator	x / y
%	Remainder Operator	x % y

Arithmetic Operators - Unary

Unary Operator	Description	Example
+	Unary plus operator (indicates positive value)	result = +1;

-	Unary minus operator
	(negates an expression
		result = -15;
		x = 10;
		x = -result;
		x = -x;

++	Increment operator Increase the value by 1	++i ; i++ ;
--	Decrement operator Decrease the value by 1	--i ; i-- ;

Equality and Relational Operators

Operator	Description	Example
==	Equal to	2 == 4 x == 15
!=	Not equal to	3 != 5 y != 10
>	Greater than	25 > 12 x > 6
>=	Greater than or equal	-1 >= 4 y >= 10
<	Less than	6 < 3 6 < x
<=	Less than or equal	10 <= y x <= y

Logical Operators

Operator	Description	Example
&&	Conditional AND	x > 3 && x <= 10
||	Conditional OR	x == 2 || y < 5
!	Logical complement (Unary operator)	! (x < 3)

op1	op2	op1 AND op2
true	true	true
true	false	false
false	true	false
false	false	false

op1	op2	op1 OR op2
true	true	true
true	false	true
false	true	true
false	false	false

Conditional Operator – Ternary Operator

Ternary Operator	Description	Example
? : Usage: *condition* **?** *value1* **:** *value2*	If the condition is true, the expression gets the value of *value1*. If not, the expression has *value2*.	See the source code ConditionalDemo2.java

Examples

- x > 5 && x < 10
- !x > 5
- !(x > 5)
- x > 5 && x < 10 || y > 200
- x < 10 || y > 200 && x > 5

Short-Circuit Evaluation

- Java evaluates a logical expression from left to right.
- The evaluation stops as soon as it knows the answer.
- Guess what happens if x equals to 0.
  - `x != 0 && 1.0 / x > 100`
  - `x == 0 || very_complex_function(x)`

Operator Precedence in Java

- Unary Operators
- Arithmetic Operators
- Relational Operators
- Logical Operators
- Assignment Operators

Assignment Operators

- =
- +=
- -=
- *=
- /=
- %=
- Bitwise assignment

Exercise

```
int a = 10;
int b = 5;
int c = 2;

c %= a *= b += a /= c;
```

=

```
a = 50
b =  10
c = 2
```

```
a = a / c;
```

a = 5

```
b = b + a;
```

b = 10

```
a = a * b;
```

a = 50

```
c = c % a;
```

c = 2

Assignment Operators

- Right associative
- Conversion on assignment
- No problem if you are assigning a value to a type with a greater range.
  - Assign a short value to a long variable
- Assigning a large value to a smaller variable type results in precision loss.

Potential Problems

- Bigger floating-point type → Smaller floating-point type
  - Loss of precision (significant figures)
  - double → float
- Floating-point type → Integer type
  - Loss of fractional part
- Bigger integer type → Smaller integer type
  - Just low-order bytes are copied.

Block

- A block of code in Java is normally enclosed with curly braces i.e., { }
  - { indicates the beginning of the block
  - } indicates the end of the block

Reference

- Java Tutorials https://docs.oracle.com/javase/tutorial/java/nutsandbolts/index.html