School of Computing

CS2040 Data Structures and Algorithms I

Lecture Note #1

Introduction to Java

Lecture Note #1: **Intro to Java**

- **Objectives:**
  - Able to start writing Java programs
  - Able to translate most C programs learned in CS1010/CS1101 into Java programs

- **Reference:**
  - Chapter 1
    - Section 1.1 (excludes Arrays) to Section 1.3: pages 27 to 45
    - Section 1.7 (excludes Console class): pages 73 to 77

Outline

1. Brief history and background
2. Run cycle
3. Basic program structure
4. Basic Java elements
   - 4.1 Arithmetic Expressions
   - 4.2 Control Flow Statements and Logical Expressions
   - 4.3 Basic Input and Output
   - 4.4 Function

1. Java: Brief History & Background

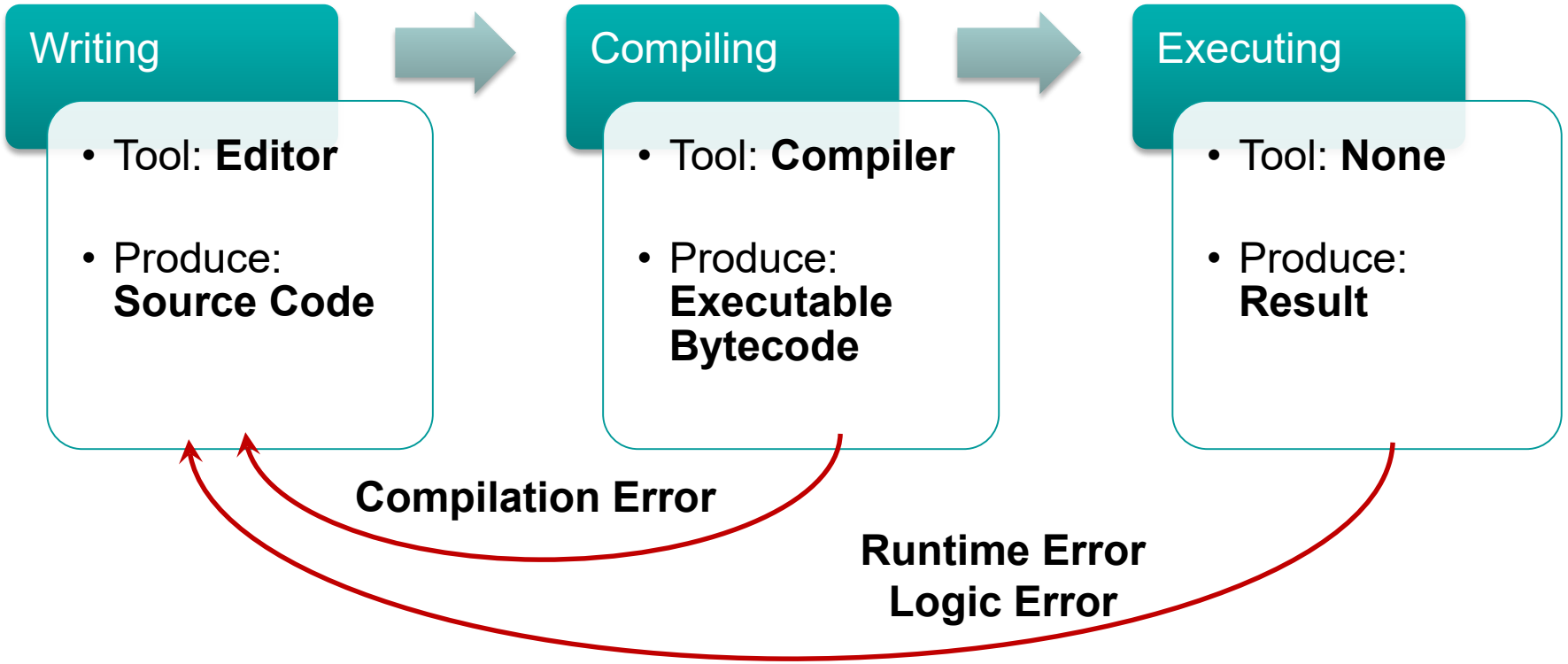
- Developed by **James Gosling:**
  - in **1995**
  - at **Sun Microsystems** (acquired by **Oracle** in 2010)

- Use **C** and **C++** as the foundation
  - "Cleaner" in syntax
  - Less low-level functionality
    - Shield the user from low-level machine interaction
  - More uniform object model

- Some selling points:
  - **Write Once, Run Everywhere™**
    - Compiled binary can be executed **across different platforms**
  - Extensive and well documented standard library

2. Run Cycle: Quick Recap

- **Run Cycle:**
  - The process of ***writing, compiling*** and ***executing*** a program



2. Run Cycle for Java Program

- **Writing / Editing Program**
  - Any text editor
  - Source code must have a `.java` extension
    - e.g. `Hello.java`
- **Compiling Program**
  - Use Java compiler `javac`
    - e.g. "`javac Hello.java`"
  - Compiled binary has `.class` extension:
    - e.g. `XXXX.class`
  - The binary is also known as **Java Executable Bytecode**
- **Executing Binary**
  - Run on a ***Java Virtual Machine (JVM)***
    - e.g. "`java XXXX`" (leave out the **.class** extension)
  - Note the difference here compared to normal C executable

3. Java: Basic Language Introduction

- Cover the elementary language components:
  - Basic Program Structure
  - Primitive data types and simple variables
  - Control flow and repetition statements

- Purpose: ease you into the language
  - You can attempt to "translate" a few simple C programs into Java syntax to familiarize yourself

- Note:
  - Many important concepts will be introduced later
  - Do not take this section as the full language overview!

3. Hello World, Again!

```c
#include <stdio.h>

int main( )
{
    printf("Hello World!\n");

    return 0;
}
```

HelloWorld.c

C program to print out the message "Hello World!"

```java
import java.lang.*;     //optional

class HelloWorld {

  public static void main(String[] args) {

     System.out.println("Hello World!");

  }

}
```

HelloWorld.java

Java program to print out the message "Hello World!"

3. Key Observations (1/2)

- Library in Java is known as **package**
  - Packages are organized into hierarchical grouping
  - E.g., the "**`System.out.println()`**" is defined in the package "**`java.lang.system`**"
    - i.e. "**`lang`**" (language) is a group of packages under "**`java`**" (the main category) and "**`system`**" is a package under "**`lang`**"

- To use a predefined library, the appropriate package should be **imported**:
  - Using the "**`import XXXXXX`**;" statement
  - All packages under a group can be imported with a "**`*`**" wildcard

- Packages under "**`java.lang`**" are imported **by default**:
  - i.e. the import statement in this example is optional

3. Key Observations (2/2)

- The main method (function) is now enclosed in a "**class**"
  - There should be only one **main method** in a program, which serves as the execution starting point
  - Each source code file contains **one or more classes**
  - Each class will be compiled into a separate **`XXXX.class`** bytecode
    - The "**`XXXX`**" is taken from the class name ("**`HelloWorld`**" in this example)

4.1 Arithmetic Expressions

4.1 Identifier and Variable

- **Identifier** is a **name** that we associate with program entity
- Java Identifier Rule:
  - Can consists of letters, digits, underscore (_) and **dollar sign ($)**
  - Cannot begin with a digit

- **Variable** is used to store data in a program
  - A variable must be declared with a specific data type

4.1 Numeric Data Types

- Summary of numeric data types in Java:

	Type Name	Range
Integer Data Types	`byte`	-2^7 to 2^7-1
	`short`	-2^{15} to $2^{15}-1$
	`int`	**-2^{31} to $2^{31}-1$**
	`long`	-2^{63} to $2^{63}-1$
Floating Point Data Types	`float`	Negative: -3.4028235E+38 to -1.4E-45 Positive: 1.4E-45 to 3.4028235E+38
	`double`	**Negative: -1.7976931348623157E+308 to -4.9E-324** **Positive: 4.9E-324 to 1.7976931348623157E+308**

- You are strongly encouraged to use:
  - `int` for integers
  - `double` for floating point numbers

4.1 Numeric Operators

Higher Precedence

`()`	Parentheses Grouping	Left-to-right
`++, --`	Postfix incrementor/decrementor	Right-to-left
`++, --` `+, -`	Prefix incrementor/decrementor Unary +, -	Right-to-left
`%`	Remainder of integer division	Left-to-right
`*, /`	Multiplication, Division	Left-to-right
`+, -`	Addition, Subtraction	Left-to-right
`=` `+= -= *= /= %=`	Assignment Operator Shorthand Operators	Right-to-left

- Evaluation of numeric expression:
  - Determine grouping using precedence
  - Use associativity to differentiate operators of same precedence
  - Data type conversion is performed for operands with different data type

4.1 Numeric Data Type Conversion

- When operands of an operation have differing types:
  1. If one of the operands is `double`, convert the other to `double`
  2. Otherwise, if one of them is `float`, convert the other to `float`
  3. Otherwise, if one of them is `long`, convert the other to `long`
  4. Otherwise, convert both into `int`

- When value is assigned to a variable of differing types:
  - **Widening (Promotion):**
    - Value has a smaller range compared to the variable
    - Converted automatically
  - **Narrowing (Demotion):**
    - Value has a **larger range** compared to the variable
    - **Explicit type casting is needed**

4.1 Data Type Conversion

- Conversion mistake:

```
double d;
int i;

i = 31415;
d = i / 10000; //attempt to get 3.14
```

What's the mistake? How do you correct it?

- Type casting:

```
double d;
int i;

d = 3.14159;
i = (int) d;    //attempt to get 3
```

The "`(int) d`" expression is known as **type casting**

Syntax:

`(datatype) value`

Effect:
The `value` is converted explicitly to the data type stated if possible.

4.2 Control Statements

Program Execution Flow

4.2 Selection Statements

```
if (a > b) {
    ...
} else {
    ...
}
```

- `if-else` statement
  - else-part is optional
- Valid condition:
  - **Must be a *boolean* expression**
  - **Unlike C, integer values are NOT valid**

```
switch (a) {
    case 1:
        ...
        break;
    case 2:
    case 3:
        ...
    default:
}
```

- `switch-case` statement
- Expression in `switch()` must evaluate to a value of `char`, `byte`, `short` or `int` type
- `break`: stop the fall-through execution
- `default`: catch all unmatched cases
  - Optional

4.2 Repetition Statements

```
while (a > b) {
    ... //body
}
```

- Valid conditions:
  - **Must be a boolean expression**
- **`while`** : check condition before executing body

```
do {
    ... //body
} while (a > b);
```

- **`do-while`**: execute body before condition checking

```
for (A; B; C) {
    ... //body
}
```

- **`A`** : initialization (e.g. `i = 0`)
- **`B`** : condition (e.g. `i < 10`)
- **`C`** : update (e.g. `i++`)
- Any of the above can be empty
- Execution order:
  - `A, B, body, C, B, body, C …`

4.2 Boolean Data Type

- Java provides an actual **boolean** data type
  - Store boolean value ***true*** or ***false***, which are keywords in Java
  - Boolean expression evaluates to either ***true*** or ***false***

SYNTAX

```
boolean variable;
```

Example

```
boolean isEven = false;
int input;

// code to read input from user omitted
if (input % 2 == 0)
      isEven = true;

if (isEven)
      System.out.println( "Input is even!" );
```

Equivalent:

```
isEven = ( input % 2 == 0 );
```

4.2 Boolean Operators

	Operators	Description
Comparison Operators	<	lesser than
	>	larger than
	<=	lesser or equal
	>=	larger or equal
	==	equal
	!=	not equal
Logical Operators	&&	AND
	||	OR
	!	NOT
	^	EXCLUSIVE-OR

Operands are variables / values that can be compared directly.

Examples:

```
X < Y
1 >= 4
```

Operands are boolean variable / expression.

Examples:

```
(X < Y) && (Y < Z)
(!isEven)
```

4.3 Basic Input / Output

Interacting with the outside world

4.3 Reading input: The **Scanner** Class

PACKAGE

```
import java.util.Scanner;
```

SYNTAX

```
//Declaration of Scanner "variable"
Scanner scVar = _initialization_;

//Functionality provided
scVar.nextInt();
```

Read an integer value from source

```
scVar.nextDouble();
```

Read a double value from source

```
......
```

Other data type, to be covered later

4.3 Reading Input: **Fahrenheit Ver 2.0**

```
import java.util.Scanner;

class TemperatureInteractive {

  public static void main(String[] args) {

     double fahrenheit, celcius;
     Scanner myScanner = new Scanner(System.in);

     System.out.print("Enter temperature in Fahrenheit: ");
     fahrenheit = myScanner.nextDouble();

     celcius = (5.0 / 9) * (fahrenheit - 32);
     System.out.println("Celcius: " + celcius);

  }

}
```

TemperatureInteractive.java

4.3 Reading Input: **Key Points (1/2)**

- The statement

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

  - Declare a variable "**`myScanner`**" of **`Scanner`** type
    - "**`myScanner`**" is just a variable name, i.e. you are free to rename it
  - The initialization "**`new Scanner(System.in)`**"
    - Construct a **`Scanner`** object
      - We will discuss more later
    - Attach it to the standard input "**`System.in`**" (which is the keyboard)
      - This scanner variable will receive input from this source
    - Scanner can attach to a variety of input source, this is just a typical usage

4.3 Reading Input: **Key Points (2/2)**

- After proper initialization, a Scanner object provides functionality to read value of various type from the input source

- The statement

```
fahrenheit = myScanner.nextDouble();
```

  - ***nextDouble()*** works like a function that returns a double value
  - The scanner object converts the input into the appropriate data type and returns it
    - in this case, user input from the keyboard is converted into double value

4.3 Writing Output: The Standard Output

- The **`System.out`** is the predefined output device
  - Refers to the monitor / screen of your computer

SYNTAX

```
//Functionality provided
System.out.print( output_string );

System.out.println( output_string );

System.out.printf( format_string, [items] );
```

```
System.out.print("ABC");
System.out.println("DEF");


System.out.printf("Very C-like %.2f\n", 3.14159);
```

4.3 Writing Output: `printf()`

- Java introduces `printf()` in Java 1.5
  - Very similar to the C version
- The format string contains normal characters and a number of specifier
  - Specifier starts with a percent sign (%)
  - Value of the appropriate type must be supplied for each specifier
- Common specifiers and modifiers:

`%d`	for integer value
`%f`	for double floating point value
`%s`	for string
`%b`	for boolean value
`%c`	for character value

SYNTAX

`%[-][W].[P]type`

`-`: For left alignment
`W`: For width
`P`: For precision

4.4 Function

Reusable and independent code unit

4.4 Function with a new name

- In Java, C-like function is known as **static/class method**
  - Denoted by the "**static**" keyword before return data type
  - Another type of method, known as **instance method** will be covered later

```
class Factorial {

  public static int factorial (int n) {
       if (n == 0) return 1;
       return n * factorial(n-1);
  }

  public static void main(String[] args) {

     int n = 5;   //You can change it to interactive input

     System.out.printf("Factorial(%d) = %d\n", n, factorial(n));
  }
}
```

Factorial.java

4.4 Method Parameter Passing

- **All** parameters in Java are **passed by value**:
  - A copy of the actual argument is created upon method invocation
  - The method parameter and its corresponding actual parameter are two independent variable

- In order to let a method modify the actual argument:
  - An **object reference data type** is needed (similar to pointer in C)