OBJECT-ORIENTED PROGRAMMING

Introduction to Java

Acknowledgement

- The contents of these slides have origin from School of Computing, National University of Singapore.

- We greatly appreciate support from Mr. Aaron Tan Tuck Choy, and Dr. Low Kok Lim for kindly sharing these materials.

Policies for students

- These contents are only used for students PERSONALLY.

- Students are NOT allowed to modify or deliver these contents to anywhere or anyone for any purpose.

Outline

1. Java: 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 (Scanner class) and Output
   - 4.4 API
   - 4.5 Math class, Class Attributes
   - 4.6 User-defined Functions

1. Java: Brief History & Background

James Gosling
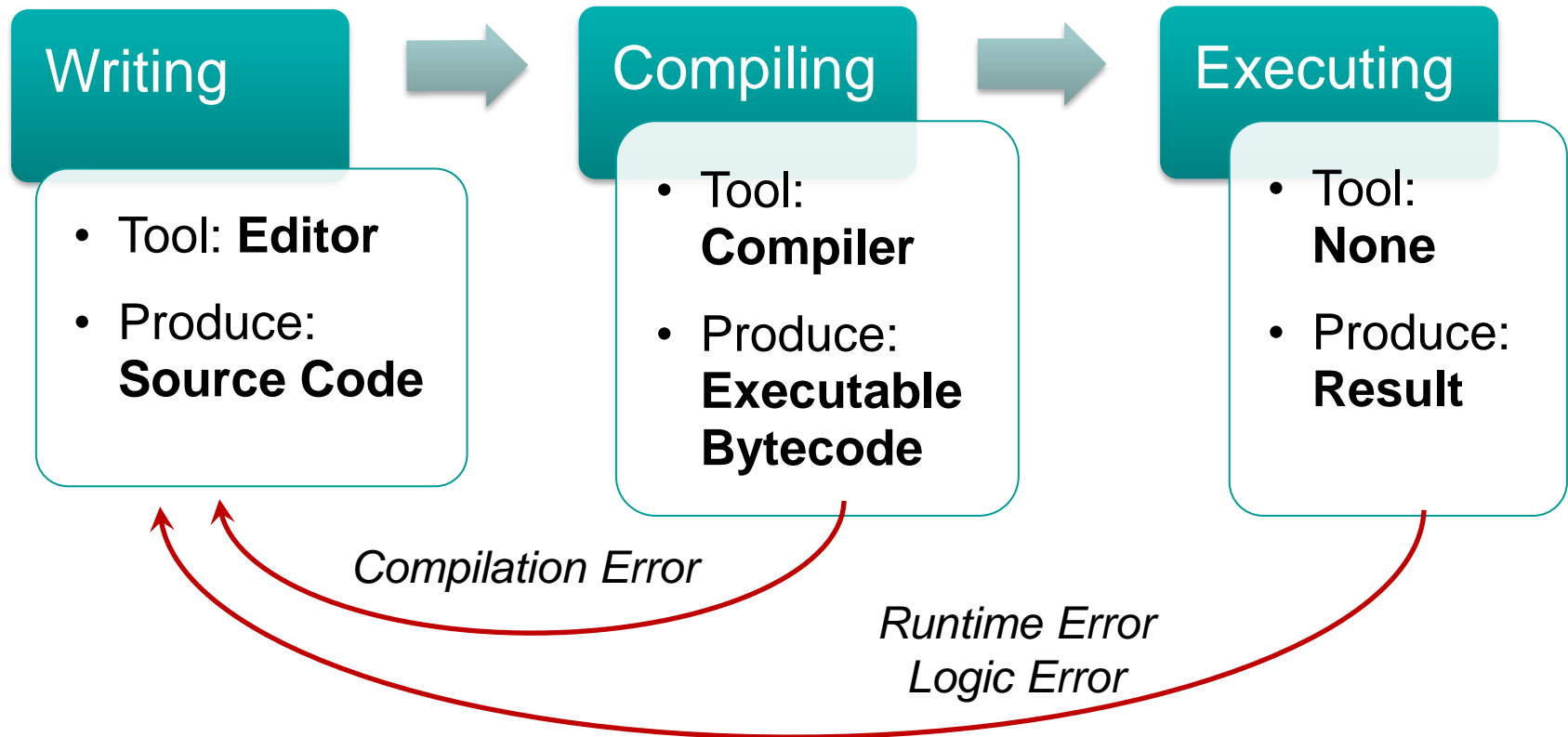1995, Sun Microsystems

Use C/C++ as foundation
- "Cleaner" in syntax
- Less low-level machine interaction

- Write Once, Run Everywhere^{TM}
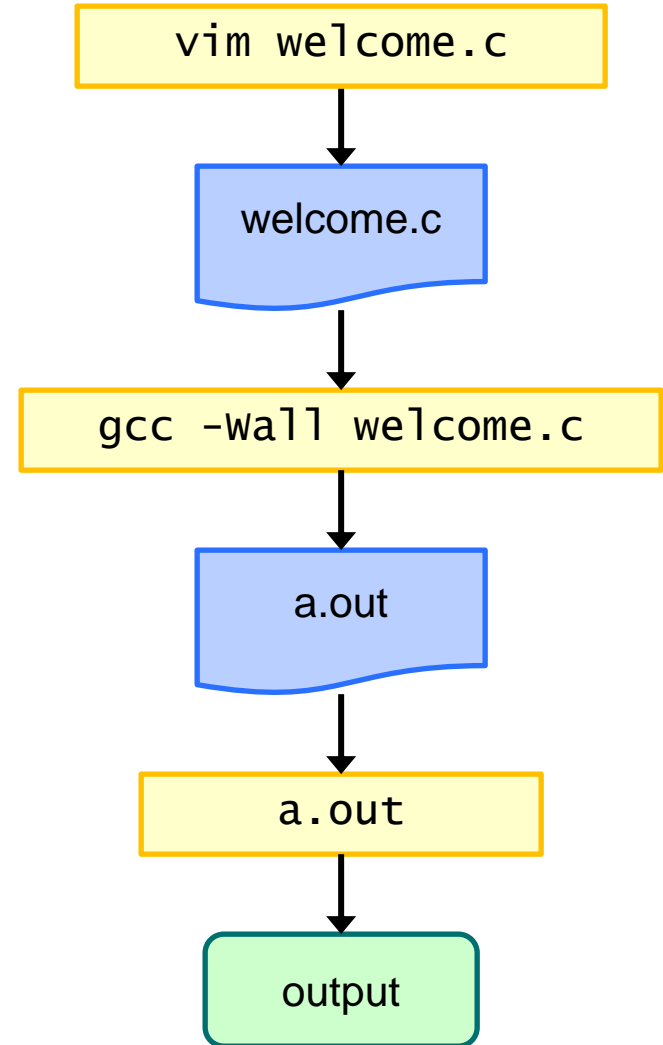- Extensive and well documented standard library

- Less efficient

Recap: Process

Writing	→	Compiling	→	Executing
• Tool: **Editor** • Produce: **Source Code**		• Tool: **Compiler** • Produce: **Executable Bytecode**		• Tool: **None** • Produce: **Result**

Compilation Error

Runtime Error
Logic Error

Recap: Run Cycle for C Programs

- ## Writing/Editing Program
  - ❑ Use an editor, e.g.: vim
  - ❑ Source code must have a `.c` extension

- ## Compiling Program
  - ❑ Use a C compiler, eg: `gcc`
  - ❑ Default executable file: `a.out`

- ## Executing Binary
  - ❑ Type name of executable file

```
vim welcome.c
```
↓
```
welcome.c
```
↓
```
gcc –Wall welcome.c
```
↓
```
a.out
```
↓
```
a.out
```
↓
```
output
```

Java: Compile Once, Run Anywhere?

- Normal executable files are directly dependent on the OS/Hardware
    - Hence, an executable file is usually <u>not</u> executable on different platforms
    - E.g: The a.out file compiled on sunfire is not executable on your Windows computer

- Java overcomes this by running the executable on an uniform hardware environment <u>simulated by software</u>
    - The hardware environment is know as the Java Virtual Machine (JVM)
    - So, we only need a specific JVM for a particular platform to execute all Java bytecodes <u>without</u> recompilation

Run Cycle for Java Programs

- ## Writing/Editing Program

  - ❑ Use an text editor, e.g: vim
  - ❑ Source code must have **`.java`** extension

- ## Compiling Program

  - ❑ Use a Java compiler, e.g.: **`javac`**
  - ❑ Compiled binary has **`.class`** extension
  - ❑ The binary is also known as **Java Executable Bytecode**

- ## Executing Binary

  - ❑ Run on a ***Java Virtual Machine (JVM)***
    - ■ e.g.: **`java HelloWorld`**
      (leave out the **.class** extension)
  - ❑ Note the difference here compared to C executable

```
vim HelloWorld.java
```

↓

HelloWorld.java

↓

```
javac HelloWorld.java
```

↓

HelloWorld.class

↓

```
java HelloWorld
```

↓

output
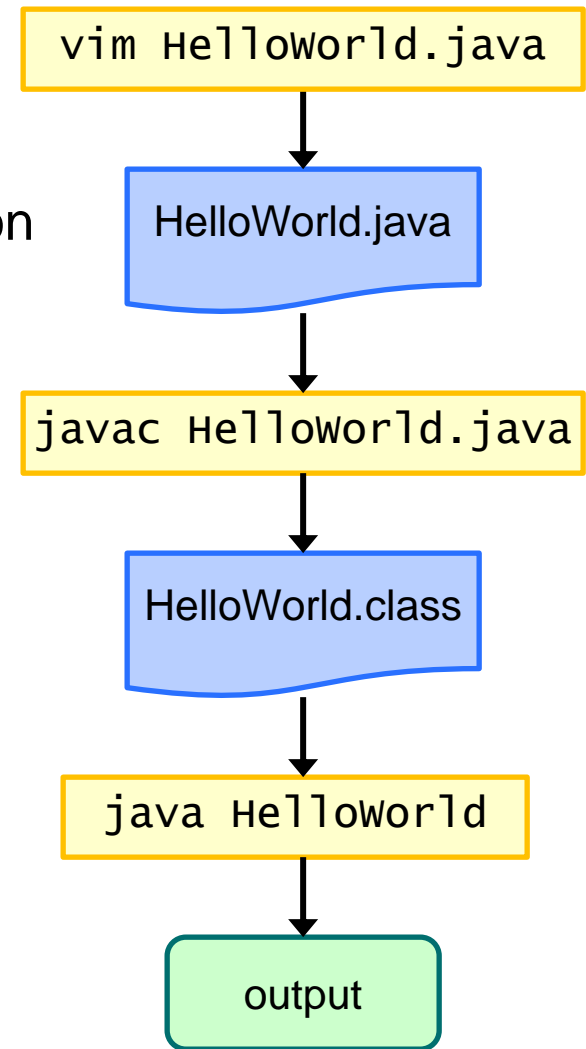
Java Execution Illustration

a.out
Windows 7 on Core 2

Normal executable (e.g.: C programs) are tied to a specific platform (OS + Hardware) This a.out cannot work in a machine of different architecture.

HelloWorld.class
Java Virtual Machine
Windows 7 on Core 2

JVM provides a uniform environment for Java bytecode execution.

HelloWorld.class
Java Virtual Machine
MacOS on PowerPC

They are the same portable file.

3. Basic Java Program Structure

- Today: just the basic language components:
  - Basic Program Structure
  - Primitive data types and simple variables
  - Control flow (selection and repetition statements)
  - Input/output statements

- Purpose: ease you into the language
  - You can attempt to "translate" some simple C programs done in 501042 into Java

- We will gradually cover many other Java features over the next few weeks
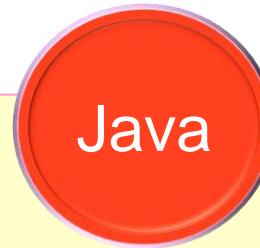
Hello World!

C

```c
#include <stdio.h>

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

    return 0;
}
```

HelloWorld.c

Java

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

public class HelloWorld {

  public static void main(String[] args) {

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

  }

}
```

Beginners' common mistake: Public class name not identical to program's file name.

HelloWorld.java

When you see this icon at the top right corner of the slide, it means that in the interest of time the slide might be skipped over in lecture and hence is intended for your own reading.

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 "`java.lang.System`"
    - i.e. "`lang`" (language) is a package under "`java`" (the main category) and "`System`" is a class 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 "`*`" (the wildcard character)

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

Key Observations (2/2)

- The main() method (function) is now enclosed in a "**class**"

  - More about class will be explained in lecture 2

  - There should be only <u>one</u> **main()** method in a program, which serves as the execution starting point

  - A source code file may contain **one or more classes**

    - There are restrictions which will be explained later – this is a bit too advanced at this point

    - For the moment, we will restrict ourselves to one class per source code

  - 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, Variable, Constant (1/2)

- **Identifier** is a **name** that we associate with some program entity (class name, variable name, parameter name, etc.)

- Java Identifier Rule:
  - May consist of letters ('a' – 'z', 'A' – 'Z'), digit characters ('0' – '9'), underscore (_) and dollar sign ($)
  - Cannot begin with a digit character

- **Variable** is used to store data in a program
  - A variable must be declared with a specific data type
  - Eg:
    ```
    int countDays;
    double priceOfItem;
    ```

4.1 Identifier, Variable, Constant (2/2)

- **Constant** is used to represent a fixed value
  - Eg: `public static final int PASSING_MARK = 65;`
  - Keyword `final` indicates that the value cannot change

- Guidelines on how to name classes, variables, and constants: see 501043 website → Resources → Online:
  - http://sakai.it.tdt.edu.vn
  - Class name: UpperCamelCase
    - Eg: `Math`, `HelloWorld`, `ConvexGeometricShape`
  - Variable name: LowerCamelCase
    - Eg: `countDays`, `innerDiameter`, `numOfCoins`
  - Constant: All uppercase with underscore
    - Eg: `PI`, `CONVERSION_RATE`, `CM_PER_INCH`

4.1 Numeric Data Types

- ## Summary of numeric data types in Java:

Type Name	Size (#bytes)	Range
byte	1	-2^7 to 2^7-1
short	2	-2^{15} to $2^{15}-1$
int	4	**-2^{31} to $2^{31}-1$**
long	8	-2^{63} to $2^{63}-1$
float	4	Negative: -3.4028235E+38 to -1.4E-45 Positive: 1.4E-45 to 3.4028235E+38
double	8	**Negative: -1.7976931348623157E+308 to -4.9E-324** **Positive: 4.9E-324 to 1.7976931348623157E+308**

Integer Data Types

Floating-Point Data Types
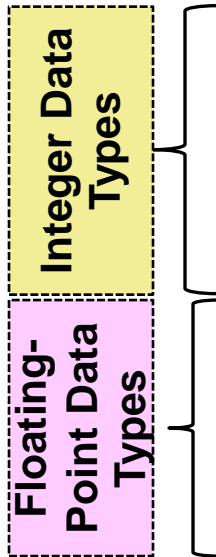
- ## Unless otherwise stated, you are to use:
  - **int** for integers
  - **double** for floating-point numbers

4.1 Numeric Operators

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

Higher Precedence ↑

■ 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;
```

Q: What is assigned to **d**?

■ ## Type casting:

```
double d;
int i;

d = 3.14159;
i = (int) d; // i is assigned 3
```

Q: What is assigned to **i** if **d** contains 3.987 instead?

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

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

Syntax:

(**datatype**) **value**

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

4.1 Problem: Fahrenheit to Celsius

- Write a simple Java program Temperature.Java:
  - To convert a temperature reading in Fahrenheit, a real number, to Celsius degree using the following formula:

$$celsius = \frac{5}{9} \times (fahrenheit - 32)$$

  - Print out the result

- For the time being, you can hard code a value for the temperature in Fahrenheit instead of reading it from user

4.1 Solution: Fahrenheit to Celsius

```java
public class Temperature {

  public static void main(String[] args) {

    double fahrenheit, celsius;

    fahrenheit = 123.5;
    celsius = (5.0/9) * (fahrenheit – 32);
    System.out.println("Celsius: " + celsius);

  }
}
```

Temperature.java

Output:
Celsius: 50.833333333333336

Compare with C: `printf("Celsius: %f\n", celsius);`

- Notes:
  - **5.0/9** is necessary to get the correct result (what will 5/9 give?)
  - "**+**" in the printing statement
    - Concatenate operator, to combine strings into a single string
    - Variable values will be converted to string automatically
  - There is another printing statement, **System.out.print()**, which does not include newline at the end of line (more in section 4.3)

4.2 **Control Statements**

Program Execution Flow

4.2 Boolean Data Type [new in Java]

- 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;`

```
boolean isEven;
int input;
// code to read input from user omitted
if (input % 2 == 0)
    isEven = true;
else
    isEven = false;
if (isEven)
    System.out.println("Input is even!");
```

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

4.2 Boolean Operators

	Operators	Description
Relational Operators	**<**	less than
	>	larger than
	<=	less than or equal
	>=	larger than 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 variables/expressions.

Examples:

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

4.2 Comparison with C

- In ANSI C, there is no boolean type.
  - Zero means 'false' and any other value means 'true'

```c
int x;
... // assume x is assigned a non-negative value
if (x%3)
    printf("%d is not divisible by 3.\n", x);
else
    printf("%d is divisible by 3.\n", x);
```

- In Java, the above is <u>invalid</u>
- Java code:

```java
int x;
... // assume x is assigned a non-negative value
if (x%3 != 0)
    System.out.println(x + " is not divisible by 3.");
else
    System.out.println(x + " is divisible by 3.");
```

4.2 Selection Statements

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

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

```java
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; may be optional

4.2 Repetition Statements (1/2)

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

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

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

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

- `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 Repetition Statements (2/2)

- In ANSI C, the loop variable must be declared before it is used in a 'for' loop

```
int i;
for (i=0; i<10; i++) {
    ...
}
```

- In Java, the loop variable may be declared in the initialisation part of the 'for' loop

- In example below, the scope of variable `i` is within the 'for' loop only

```
for (int i=0; i<10; i++) {
    ...
}
```

4.3 Basic Input/Output

Interacting with the outside world

4.3 Reading input: The Scanner Class

PACKAGE

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

SYNTAX

```java
//Declaration of Scanner "variable"
Scanner scVar = new Scanner(System.in);

//Functionality provided
scVar.nextInt();
```
Read an integer value from source System.in

```java
scVar.nextDouble();
```
Read a double value from source System.in

```java
......
```
Other data types, to be covered later

4.3 Reading Input: Fahrenheit Ver 2

```java
import java.util.Scanner;    // or import java.util.*;

public class TemperatureInteractive {

  public static void main(String[] args) {

      double fahrenheit, celsius;
      Scanner sc = new Scanner(System.in);

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

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

  }

}
```

TemperatureInteractive.java

4.3 Reading Input: Key Points (1/3)

- ## The statement

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

  - Declares a variable "`sc`" of `Scanner` type

  - The initialization "`new Scanner(System.in)`"

    - Constructs a `Scanner` object

      - We will discuss more about object later

    - Attaches it to the standard input "`System.in`" (which is the keyboard)

      - This Scanner object **sc** will receive input from this source

    - Scanner can attach to a variety of input sources; this is just a typical usage

4.3 Reading Input: Key Points (2/3)

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

- The statement

  `fahrenheit = sc.nextDouble();`

  - ❏ **`nextDouble()`** works like a function (called method in Java) that returns a double value read interactively

  - ❏ The Scanner object **sc** converts the input into the appropriate data type and returns it

    - in this case, user input from the keyboard is converted into a double value

4.3 Reading Input: Key Points (3/3)

- Typically, only one Scanner object is needed, even if many input values are to be read.

  - The same Scanner object can be used to call the relevant methods to read input values

- Note: In CodeCrunch, your program will NOT work if you use more than one Scanner object in your program.

4.3 Writing Output: The Standard Output

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

<table>
<tr>
<td>SYNTAX</td>
<td>

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


System.out.println( output_string );


System.out.printf( format_string, [items] );
```
</td>
</tr>
</table>

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


System.out.printf("Very C-like %.3f\n", 3.14159);
```

Output:

```
ABCDEF
GHI
Very C-like 3.142
```

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 specifiers
  - 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

4.3 Problem: Approximating PI

- One way to calculate the PI (π) constant:

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \ldots\ldots$$

- Write ApproximatePI.java to:

  1. Ask the user for the number of terms to use for approximation

  2. Calculate π with the given number of terms

  3. Output the approximation in 6 decimal places

4.3 Solution: Approximating PI

```java
import java.util.*; // using * in import statement

public class ApproximatePI {

  public static void main(String[] args) {

      int nTerms, sign = 1, denom = 1;
      double pi = 0.0;

      Scanner sc = new Scanner(System.in);

      System.out.print("Enter number of terms: ");
      nTerms = sc.nextInt();

      for (int i = 0; i < nTerms; i++) {
          pi += 4.0 / denom * sign;
          sign *= -1;
          denom += 2;
      }
      System.out.printf("PI = %.6f\n", pi);
  }
}
```

$$\pi = \frac{4}{1} - \frac{4}{3} + \frac{4}{5} - \frac{4}{7} + \frac{4}{9} - \ldots$$

ApproximatePI.java

4.4 API

Application Programming Interface

4.4 API (1/2)

- The Scanner class you have seen is part of the Java API
  - API: an interface for other programs to interact with a program without having direct access to the internal data of the program
  - Documentation, SE7: http://docs.oracle.com/javase/7/docs/api/
  - You may also access the above link through 501043 website → Resources → Online (http://sakai.it.tdt.edu.vn)
  - For Java programmers, it is very important to refer to the API documentation regularly!

- The API consists of many classes
  - You do not need to know all the classes (there are easily a few thousand classes altogether!)
  - You will learn some more classes in this course

- This week reading assignment
  - Read up Scanner class in the API documentation

4.4 API (2/2)

4.5 Math class, Class Attributes

Using the Math class

4.5 The Math class (1/2)

- From the API documentation

Modifier and Type	Field and Description
static double	**E** The `double` value that is closer than any other to *e*, the base of the natural logarithms.
static double	**PI** The `double` value that is closer than any other to *pi*, the ratio of the circumference of a cir

Method Summary

Methods

Modifier and Type	Method and Description
static double	**abs**(double a) Returns the absolute value of a `double` value.
static float	**abs**(float a) Returns the absolute value of a `float` value.
static int	**abs**(int a) Returns the absolute value of an `int` value.
static long	**abs**(long a) Returns the absolute value of a `long` value.
static double	**acos**(double a) Returns the arc cosine of a value; the returned angle is in the range 0.0 through *pi*.
static double	**asin**(double a) Returns the arc sine of a value; the returned angle is in the range -*pi*/2 through *pi*/2.
static double	**atan**(double a) Returns the arc tangent of a value; the returned angle is in the range -*pi*/2 through *pi*/2.

Java™ Platform Standard Ed. 7

All Classes

Packages

java.applet
java.awt
java.awt.color
java.awt.datatransfer
java.awt.dnd
java.awt.event

Marshaller
Marshaller.Listener
MaskFormatter
Matcher
MatchResult
Math
MathContext
MatteBorder
MBeanAttributeInfo
MBeanConstructorInfo
MBeanException
MBeanFeatureInfo
MBeanInfo
MBeanNotificationInfo
MBeanOperationInfo
MBeanParameterInfo
MBeanPermission
MBeanRegistration
MBeanRegistrationException
MBeanServer
MBeanServerBuilder
MBeanServerConnection
MBeanServerDelegate
MBeanServerDelegateMBean

4.5 The Math class (2/2)

- Package: java.lang (default)
- Some useful **Math** methods:
  - **abs()**
  - **ceil()**
  - **floor()**
  - **max()**
  - **min()**
  - **pow()**
  - **random()**
  - **sqrt()**

4.5 Class Attributes

- The `Math` class has two class attributes

static double	E	
	The `double` value that is closer than any other to *e*, the base of the natural logarithms.	
static double	PI	
	The `double` value that is closer than any other to *pi*, the ratio of the circumference of a circle to its diameter.	

- A class attribute (or class member) is associated with the class, not the individual instances (objects). Every instance of a class shares the class attribute.
  - We will explain about "objects" later.

- How to use it?
  - Example:
    ```
    double area = Math.PI * Math.pow(radius,2);
    ```
  - Here, `Math.PI` is used as the constant π

4.5 The Math class: Demo

```java
// To find the area of the largest circle inscribed
// inside a square, given the area of the square.
import java.util.*;

public class TestMath {

  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);

    System.out.print("Enter area of a square: ");
    double areaSquare = sc.nextDouble();

    double radius = Math.sqrt(areaSquare)/2;
    double areaCircle = Math.PI * Math.pow(radius, 2);

    System.out.printf("Area of circle = %.4f\n",
                      areaCircle);
  }
}
```
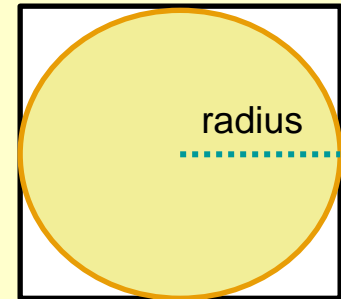
4.6 User-defined Functions

Reusable and independent code units

4.6 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

```java
public class Factorial {

    // Returns n!
    // Pre-cond: n >= 0
    public static int factorial (int n) {
        if (n == 0) return 1;
        else 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));
    }
}
```

If *n* is too big, say 40, what will happen? Why?

4.6 Method Parameter Passing

- All parameters in Java are passed by value (as in C):
  - ❑ A copy of the actual argument is created upon method invocation
  - ❑ The method parameter and its corresponding actual parameter are two independent variables

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

Summary

Java Elements

Data Types:
- *Numeric Data Types:*
    byte, short, int, float, double
- *Boolean Data Type:*
    boolean

Expressions:
- *Arithmetic Expression*
- *Boolean Expression*

Control Flow Statements:
- *Selection Statements: if-else, switch-case*
- *Repetition Statements: while, do-while, for*

Classes:
- *Scanner*
- *Math*

Thank you