Unit 1:

Introduction to Object-Oriented Programming and Java

Object-Oriented Programming (OOP)
CCIT 4023, 2025-2026

U1: Introduction to Object-Oriented Programming & Java

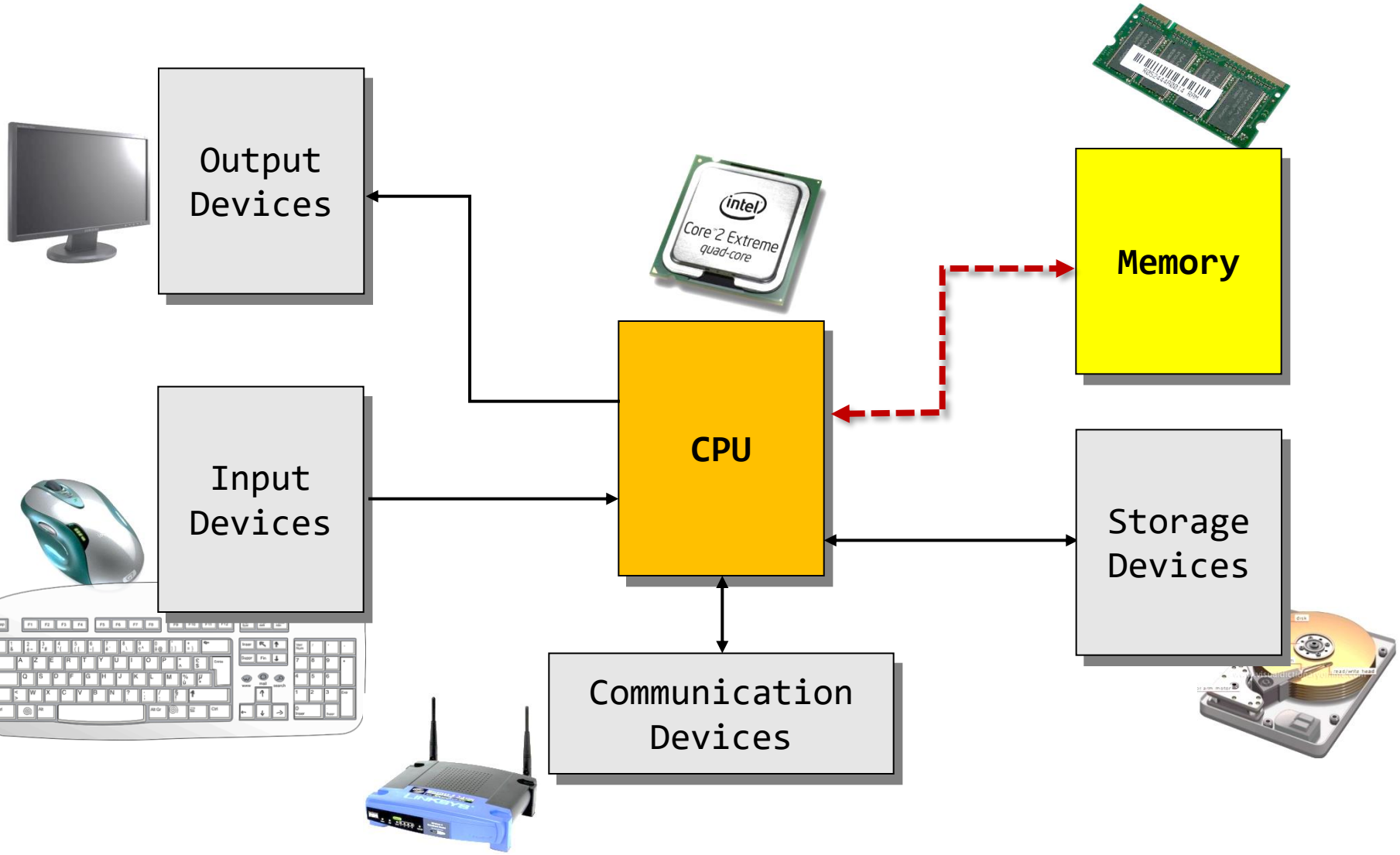
- Computer Programming and Programming Languages
- Introduction to Object-Oriented Programming
- Introduction to Java Programming
- Running and Developing Java Program
- Typical Java Program
  - Getting Start with "Hello World“
- Basic Error Handling in Programming
- Our Course - OOP

** Remark: Details of various topics will be discussed in later units.*

Computer Programming and Programming Languages

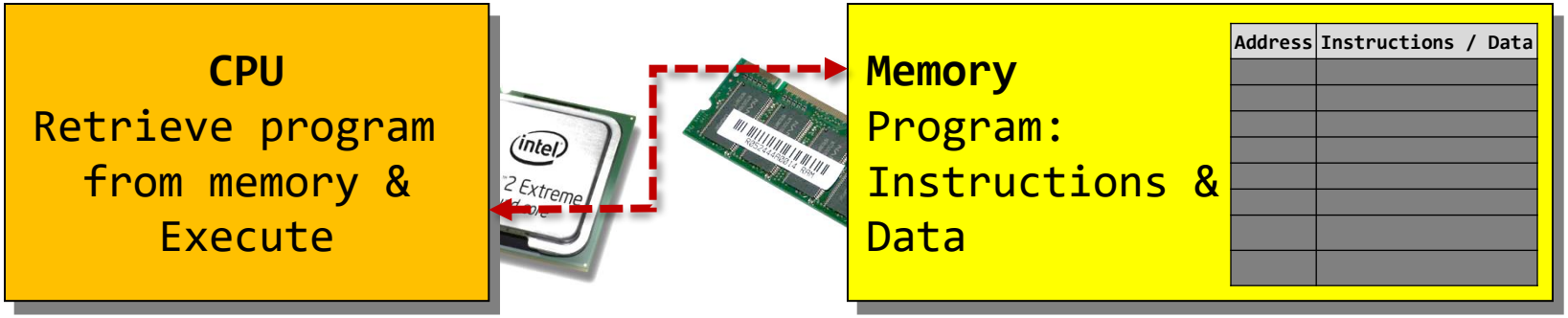
- **Computer** is a general-purpose device that can be programmed (with certain machine instructions) to carry out specified operations
- **Computer program** consists of a collection of instructions/commands that "instruct" computers what to do
- **Computer programming** is about developing and creating computer program (or simply software), in specific programming languages
- Program developers may create programs in different forms of instructions, or different **programming languages**
  - Computer (the hardware, CPU) can only understand its own set of instructions, the "instruction set" (its specific language, machine language)
  - Programming directly in machine language is hard, tedious, and inefficient
  - There are many programming languages available, which are "high-level" enough to develop programs easier and better.

A Typical Computer System



Computer Program, CPU and Memory

- A computer program and its associated data are moved into computer's memory (e.g. from the storage), before they can be executed by CPU
  - Essentially, memory is the computer's working area for running a program
- The **central processing unit** *(CPU)* of a computer retrieves instructions (and associated data) from the memory, and executes them
  - CPU understands and executes its own set of instructions (machine-code / machine language)
- Computer **memory** consists of *ordered sequence of bytes* for storing program instructions (opcodes) their associated data (operands) that the program is working with
  - Each memory unit (often in byte) is located with its unique memory address



Programming Languages

- At the bottom, a specific computer (the hardware or its CPU set) can directly understand and follow instructions of its own language only – the machine language (or machine code or instruction set)
- **Machine language** (Machine-dependent)
  - A set of instructions executed by specific CPUs, based on a particular instruction set of those CPUs
  - Difficult for human developers to read and modify
  - As writing programs directly in machine language is very hard, tedious, and inefficient to software developers, there are more convenient ones:
    - Assembly languages
    - High-Level programming languages
  - Machine language and Assembly language are regarded as Low-level programming languages

Programming Languages

- **Assembly language**
  - Use a short descriptive “word” to represent each of the machine-language instructions, e.g.
    - Assembly “word” `add` is typically associated to the machine code to add numbers, and `sub` to subtract numbers
  - Assembler converts assembly code into machine code
- **High-level programming language**
  - Comparatively English-like, easy to learn and program
  - Generally the language is platform independent
  - Written program is called a source program or code
  - Compiler or interpreter is used to translate source codes into its lower-level codes (e.g. machine language program)
    - **Compiler** converts the whole source file to its lower-level form (e.g. C, C++ and Java).
    - **Interpreter** directly executes instructions (one statement a time) written in a programming or scripting language (e.g. Python)

Introduction to Object-Oriented Programming

- There are many different programming languages, supporting different programming approaches (models or paradigms)
- ***Procedural*** approach focuses on procedural abstraction - designing operations or methods (also called procedures or functions) as execution modules
  - Many programming languages support this procedural approach, e.g. C and Python
  - Many programming approaches are often interrelated, and most of them require support from this procedural approach
- **Object-oriented programming** focuses on coupling data fields (attributes) and methods (operations or behaviors) together into objects, e.g. supported by **Java**
- Software design using the object-oriented approach focuses on objects, attributes and operations of objects, and their interactions

Introduction to Object-Oriented Programming
(Procedural vs. Object-Oriented Approaches)

Procedural

- Specify a set of “steps”, in a form of execution units - procedures
- Program is modularized as procedures / functions / methods, associated with certain data
- Example: C, Python

Object-Oriented (OO)

- Specify classes / objects
- Program is essentially conceptualized with a scenario of interacting objects
- Objects are associated with both attributes (fields) and activities (methods)
- 3 major OO features: Inheritance, Encapsulation, Polymorphism
- Example: Java, C++

Object-Oriented Programming
(Classes and Objects)

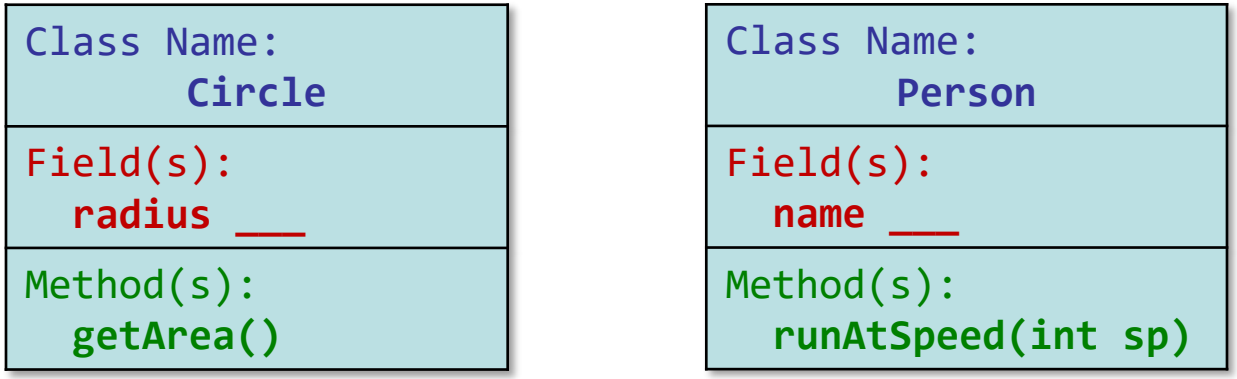
- Object-Oriented (OO) programming involves using of *objects*
- An **object** represents or models an entity of certain system (E.g. a system user, a book of the library, a student of a school, a circle, etc.)
  - An object may consist of data states and related operational behaviors
  - It stores its *data/states* in **fields***,* and defines its *operations/behaviors* with **methods**
    - Fields are also called instance variables, properties, attributes, and data members in Object-Oriented programming
    - Methods are similar to the operational modules as functions, procedures, subroutines in some programming languages
  - Methods may operate on an object's internal states and serve as the primary mechanism for object-to-object communication
- It is common to associate this object concept as a unit of "mini-program" to represent certain entity which contains
  - its *own* set of data (fields, and each field could also be an object)
  - its *own* set of methods operating on these fields

Object-Oriented Programming
(Classes and Objects)

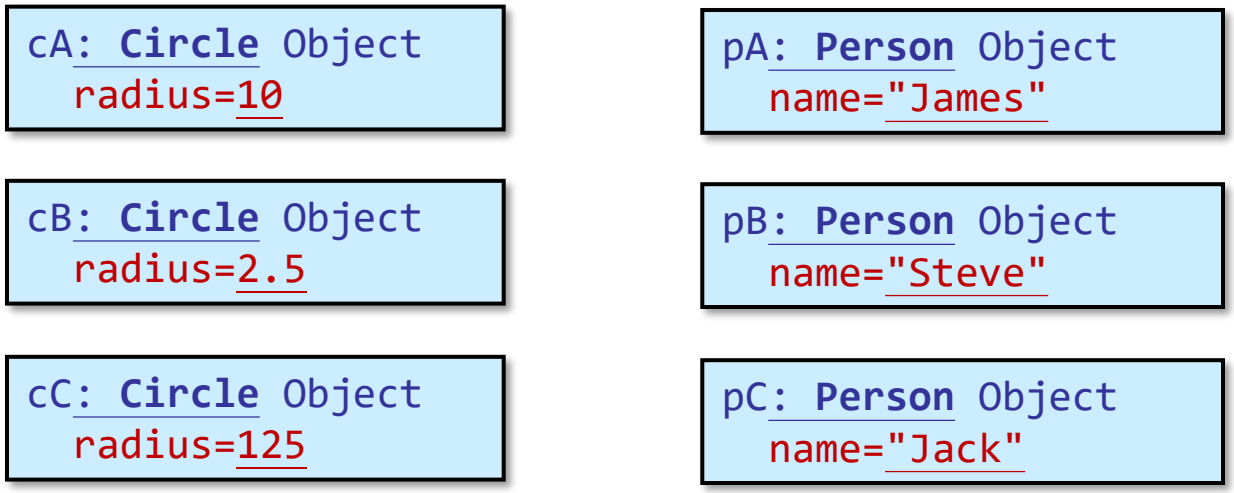
- When modeling our real world, we often find that many individual objects are of the same kind
- **Objects** *of the same kind* are defined using a common ***type class***
- A **class** is a type, template or blueprint that defines what an object will be (including its fields/data members, constructors and methods)
- An object is an **instance** of a class, and many objects / instances can be created from a class
- The terms *object* and *instance* are often interchangeable

Object-Oriented Programming
(Example Classes and Their Objects)

2 Classes:- **`Circle`** (left) & **`Person`** (right):



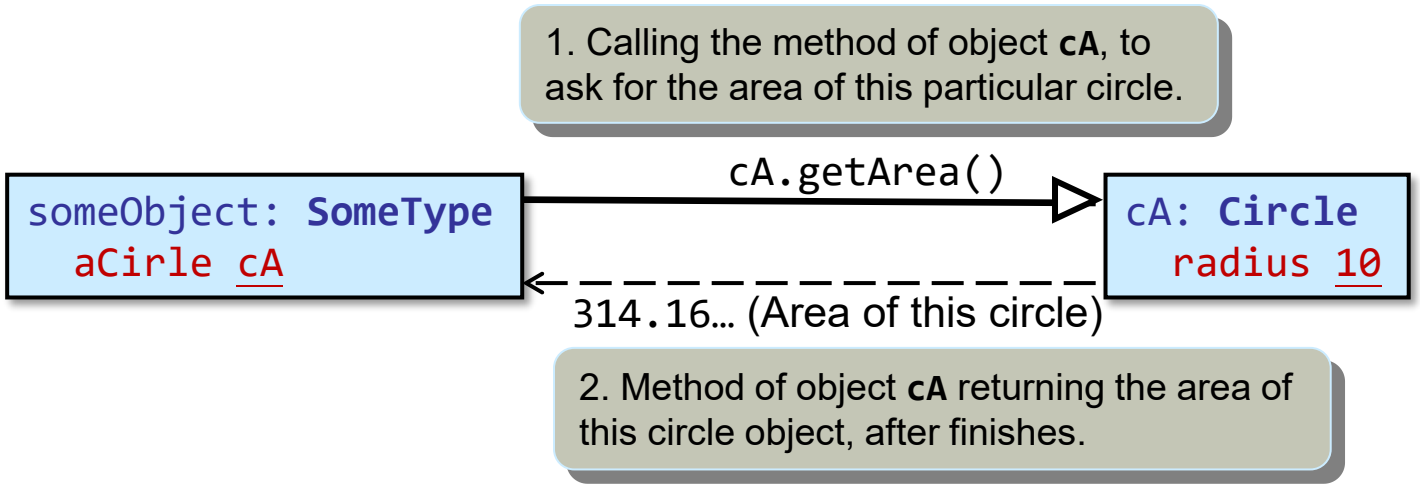
3 Objects/Instances of each class:- **`Circle`** & **`Person`**:



Many objects of the same class created, having different field values.

Object-Oriented Programming
(Class/Object Messaging and Methods)

- To communicate and instruct a class or an object to perform a task, the concept of sending *messages* is applied
- A message can only send to the classes and objects that understand the message you sent
- A class or an object must possess a matching ***method*** to be able to handle the received message
- Below is an example to 1) send a message and 2) return an answer



Object-Oriented Programming
(Some Benefits)

Benefits of bundling program code into individual software objects include:

- *Modularity*:
  - One object can be written and maintained independently of other objects
- *Information-hiding*:
  - Details of its internal implementation remain hidden from the outside world
- *Code re-use*:
  - If object already exists (perhaps written by other developer), we can use that object in our program

Object-Oriented Programming
(Three Major OO Features)

- **Inheritance** relationship
  - A mechanism designed for different entities (subclasses) that share common features (from a superclass)
- **Encapsulation** of data and methods
  - Internal components are encapsulated and hidden from outside (information hiding)
- **Polymorphism**
  - Ability to morph into many (poly) different forms
    - E.g. Same (method / message / name) with different forms of functions / behaviours / results

* *Remark:* Various concepts will be further discussed in later units.

Programming Approaches
(Procedural vs. Object-Oriented)

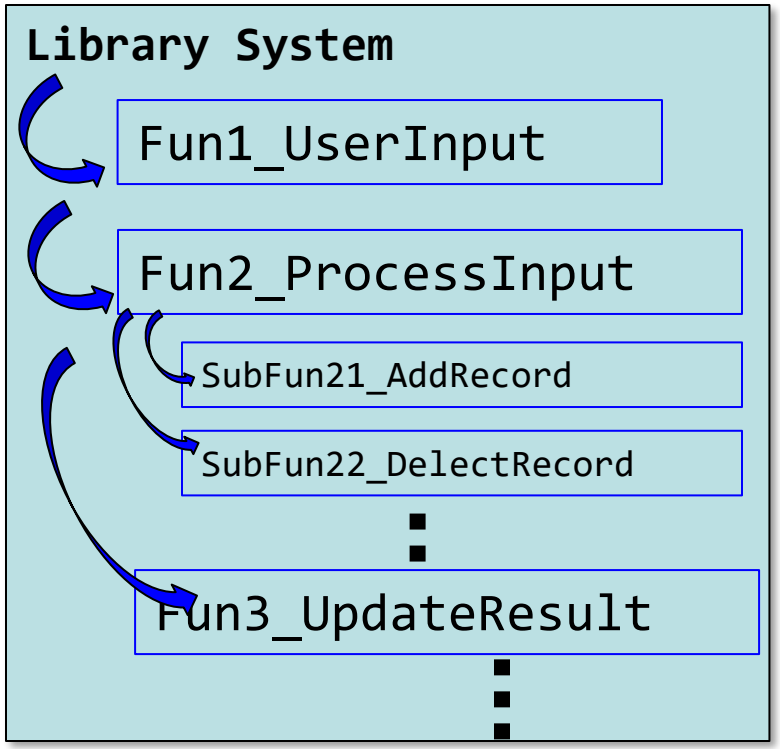
Reference Only

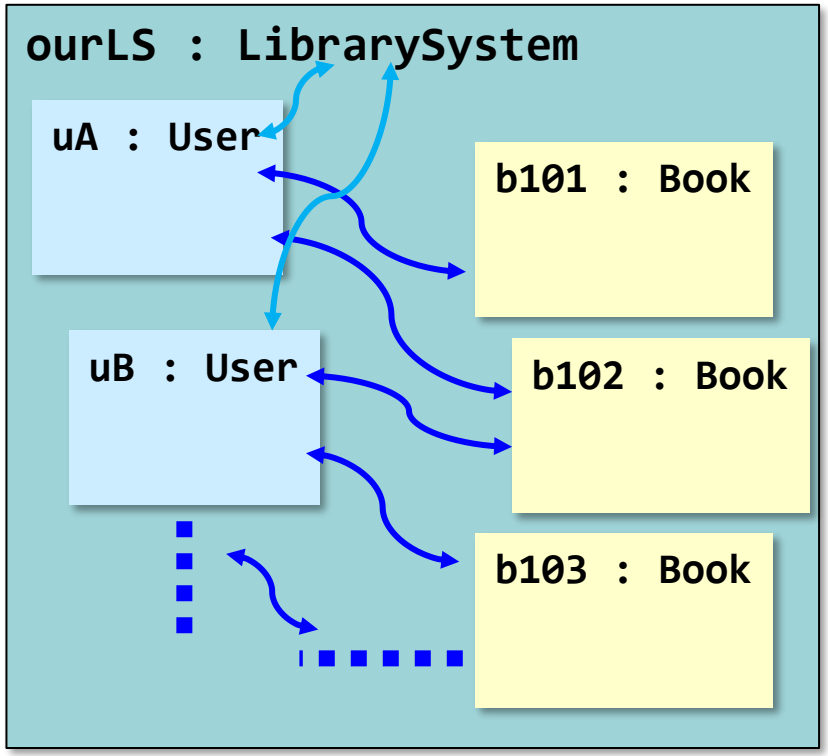
A simplified case: ***Library book record system***:

Procedural **vs.** **Object-Oriented**

* Focus on designing proper procedures/functions/methods as execution modules

* Focus on designing inter-related and interacting "objects"





Introduction to Java programming

- **Java** programming language is a general-purpose, concurrent, **class-based, Object-Oriented** language
- Computer program built with an object-oriented (OO) approach is essentially designed and developed in a way of defining a collection of inter-related *objects* (of their associated classes), including how they interact with each other (internally) and with the outside world (externally)
- Objects of the same “kind” are defined using a common “type” *class*

Introduction to Java programming

- Java is designed with ***similar C/C++ style syntax*** that many programmers would find familiar
  - As such, people with C/C++ background may find learning Java easier than others

```c
/* Helloworld program code in C: File HelloWorld.c  */
#include <stdio.h>
int main() {
    printf("Hello World, in C!\n");
    return 0;
}
```

```
C:\>gcc -o HelloWorld.exe HelloWorld.c
C:\>HelloWorld.exe
Hello World, in C!
C:\>
```

```java
// Helloworld program code in Java: File HelloWorld.java
public class HelloWorld {
     public static void main(String[ ] args) {
           System.out.println("Hello World, in Java!");
     }
} // HelloWorld.java
```

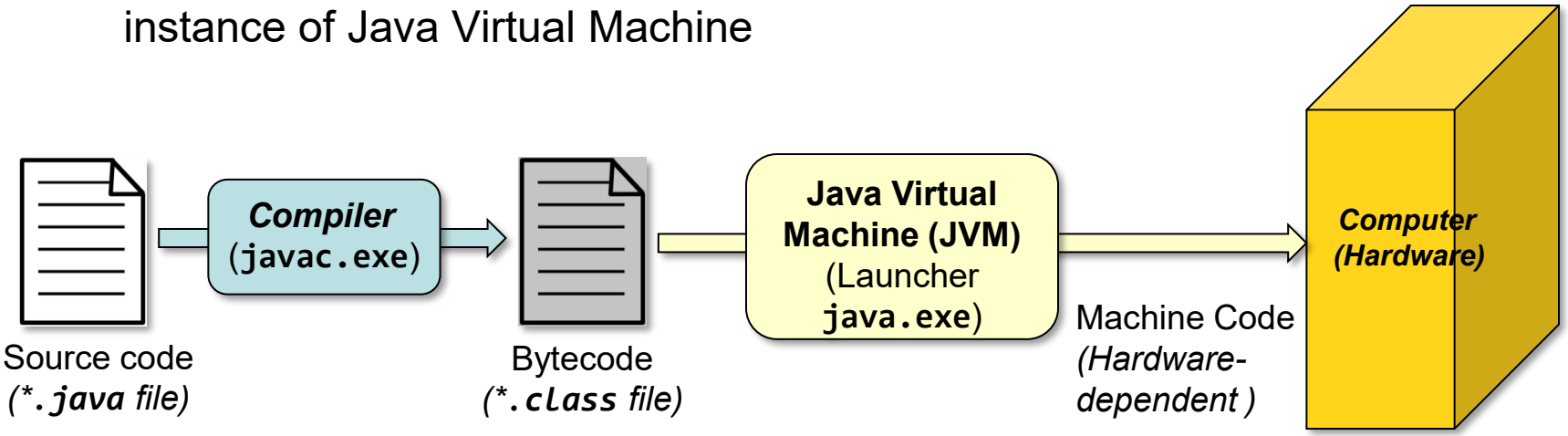
```
C:\>javac HelloWorld.java
C:\>java HelloWorld
Hello World, in Java!
C:\>
```

```python
# Helloworld program code in Python: File HelloWorld.py
print("Hello World, in Python!")
```

```
C:\>python HelloWorld
Hello World, in Python!
C:\>
```

Introduction to Java programming

- **Source code** of Java programs is first written in plain text files ending with the **`.java`** extension
- Those source files are then compiled into **bytecode** files with the **`.class`** extension, by the **`javac.exe` compiler**
- A `.class` file does not contain native machine code
  - it instead contains bytecodes - the language of the **Java Virtual Machine** (**JVM**, Java VM)
  - The **java launcher (`java.exe`)** tool then runs the application with an instance of Java Virtual Machine

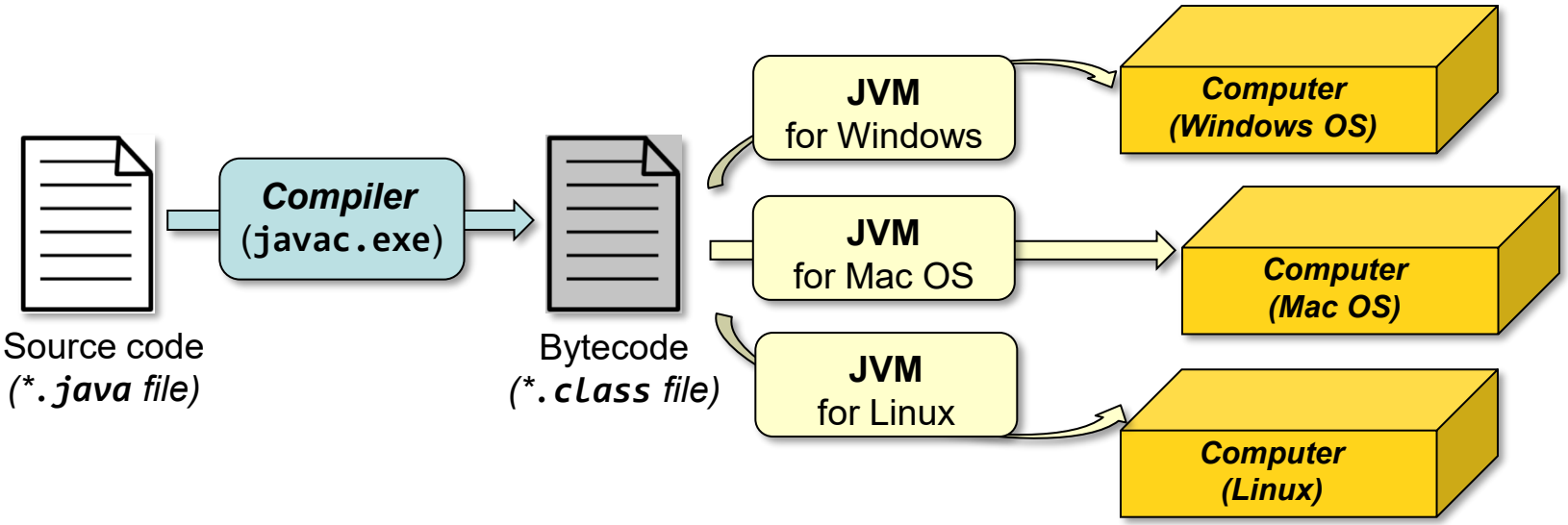


Java as a Platform

- Java is also referred to a **platform** (although many developers may focus more on Java as a programming language)
  - A platform is the ***hardware and/or software environment*** in which a program runs
  - It often includes ***a suite of programs and libraries/frameworks*** that supports developing and executing programs
- Major editions of Java Platform:
  - Java Platform, Enterprise Edition (*Java EE*)
    - Version for developing and running large, multi-tiered, reliable, and secure enterprise applications
  - **Java Platform, Standard Edition (*Java SE*)**
    - Most common version for Java applications developed and run on desktops and servers
    - This is the edition mostly used for learning Java programming
  - Java Platform, Micro Edition (*Java ME*)
    - For developing applications for embedded and mobile devices, such as mobile phones

Java as a Platform

- Java platform has two major components:
  - **Java Virtual Machine** (**JVM**, Java VM)
  - **Application Programming Interface (API)**, the Java libraries
- Because Java VM is available on different operating systems (e.g. Windows OS, Linux and Mac OS), the same `.class` files are capable of running on these operating systems
  - Bytecodes are supposed to be (hardware) machine-independent
  - Java's slogan: `"Write Once, Run Anywhere" (WORA)`



Java as a Platform

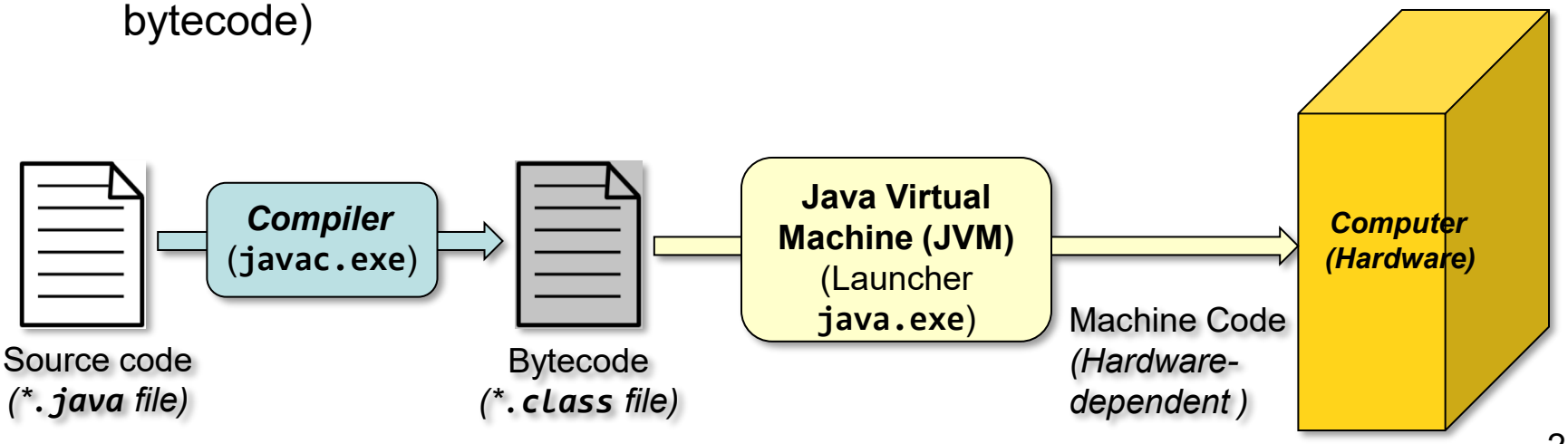
- Java **Application Programming Interface (API)** is a large collection of ready-made software components that provide many useful capabilities
  - It is grouped into libraries of related classes and interfaces; these libraries are known as *packages*
- As such, API and Java Virtual Machine insulate the program from the underlying hardware
  - As a platform-independent environment, the Java platform may be a bit slower than native machine code

Running and Developing Java Program

- Java source code is compiled into **bytecode** (the **.class** file) which is then run by **Java Virtual Machine (JVM)**
  - Unlike some programming languages (e.g. C) where source code is compiled directly to machine code for a specific platform, and run by machines of that platform
- JVM is an abstract computing machine
  - Like a real computing machine, it has an instruction set and manipulates various memory areas at run time.
- JVM knows only the *class* file format. A *class* file contains Java Virtual Machine instructions (or bytecodes) and a symbol table, as well as other supporting information

Running and Developing Java Program

- In order to create programs (application software) in Java, there are different development tools
- Apart from creating and editing Java source code, these development tools also support compiling, running, monitoring, debugging, and documenting Java programs
- E.g. the major tools for beginners include:
  - **`javac` *compiler***, for compiling source code to bytecode,
  - **`java` *launcher***, for running/launching Java program (e.g. the compiled java bytecode)



Running and Developing Java Program with: (JDK, JRE, IDE, API)

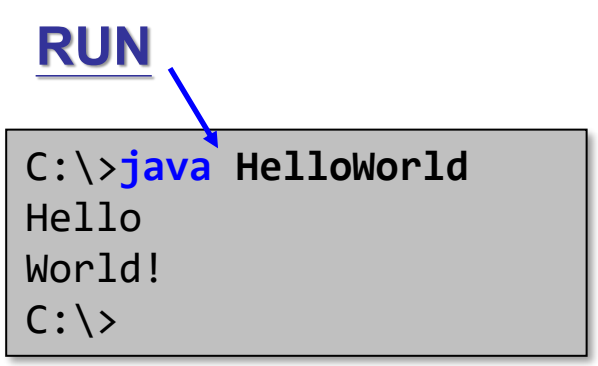
- **Java Development ToolKit** (**JDK**)
  - *For developing, testing and running* Java programs
  - Consists of a set of programs, including **`javac`** compiler and **`java`** launcher
- **Java Runtime Environment** (**JRE**)
  - *For running Java programs, not for developing*
  - Contains the Java virtual machine, runtime class libraries, and Java application launcher that are necessary *to run programs* written in the Java programming language, including **`java`** launcher
  - Does NOT contain development tools such as compilers or debuggers
- **Application Program Interface** (**API**)
  - Part of Java platform, that contains predefined classes and interfaces for developing Java programs (Like Libraries)
- **Integrated Development Environment** (**IDE**)
  - Java development tools for rapidly developing Java programs

How to Run a Given Java Program

- In order to *run* a given Java program (the ***.class** bytecode file, not the ***.java** source file), we only need Java Virtual Machine (JVM) which is included Java Runtime Environment (JRE) already installed in most of common operating systems.

- **RUN** a given Java program (the bytecode *.class file), e.g.
  - To run the program in Windows *console*, type: **`java <ClassName>`**

    * **NOT** typing the file name (`*.class`), although we are running the Java program with this `HelloWorld`**`.class`** file.
  - E.g. To run the class **`HelloWorld`** in the file `HelloWorld.class`, type: **`java HelloWorld`**



How to Create (& Test) a Java Program
(Approach with JDK, Code Editor, and Console)

1. **INSTALL & Setup** JDK:
   - Install JDK, and setup programming environment (e.g. PATH of Java compiler `javac.exe` on Windows OS)
2. **WRITE** java source codes with code editor or IDE
   - E.g. Create a `HelloWorld.java` Java program file in a specific folder
3. **COMPILE** (& Debug), e.g. in same folder of source files
   - To compile in *console*, type: **`javac`** `<*.java, SourceFileName>`
     - Source code file (`*.java`) is compiled to byte-code file (`*.class`)
   - E.g. To compile source code file `HelloWorld.java` to byte-code file `HelloWorld.class`, type: **`javac HelloWorld.java`**
4. **RUN** (**& Test**) the bytecode (*.class file), e.g.
   - To run the program in Windows *console*, type: **`java`** `<ClassName>`

     *** NOT** typing the file name (`*.class`), although we are running the Java program with this `HelloWorld`**`.class`** file.
   - E.g. To run the class **`HelloWorld`** in the file `HelloWorld.class`, type: **`java HelloWorld`**

How to Create (& Test) a Java Program
(Approach with JDK, Code Editor, and Console)

1. INSTALL *JDK and set Java environment*

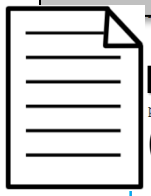
2. WRITE *java source code*

```
public class HelloWorld {
    public static void main(String[ ] args){
        System.out.println("Hello\nWorld!");
    }
}
```

HelloWorld.java
(Source code file)

3. COMPILE

```
C:\> javac HelloWorld.java
C:\>
```



HelloWorld.class
(Bytecode file)

```
Code:
   0: aload_0
   1: invokespecial #1                  // Method java/lang/Object."<init>":()V
   4: return
public static void main(java.lang.String[]);
Code:
   0: new           #2                  // class java/util/Date
   3: dup
   4: invokespecial #3                  // Method java/util/Date."<init>":()V
   7: astore_1
   8: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
  11: aload_1
  12: invokevirtual #5                  // Method java/util/Date.toString:()Ljava/lang/String;
  15: invokevirtual #6                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
  18: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
  21: ldc           #7                  // String Hello\nWorld!
  23: invokevirtual #6                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
  26: getstatic     #4                  // Field java/lang/System.out:Ljava/io/PrintStream;
  29: ldc           #8                  // String Program developed by AU B.C. Dick (10682468)
  31: invokevirtual #6                  // Method java/io/PrintStream.println:(Ljava/lang/String;)V
  34: return
```

bytecode
(*.class file)

4. RUN

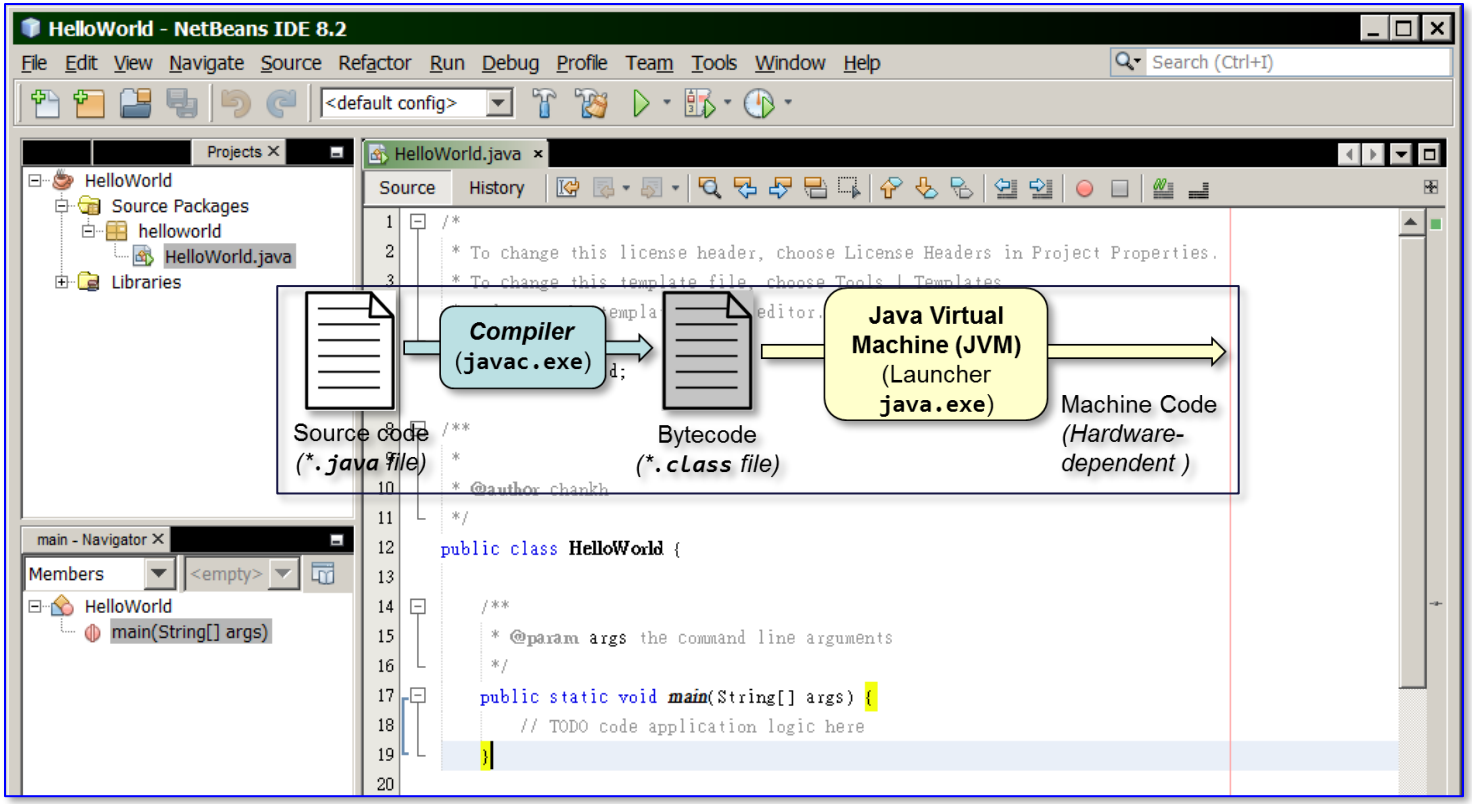
```
C:\>java HelloWorld
```

```
C:\>java HelloWorld
Hello
World!
C:\>
```

***Execution Result**: Message is printed on the console*

How to Create (& Test) a Java Program
(with IDEs, such as NetBeans)

- With *Integrated Development Environment (IDE)* such as *NetBeans*, we can develop (create, test, debug, run, document, etc.) Java program in one single integrated environment:



Install and Setup JDK

Reference Only

- ***Download*** Java SE Development Kit (JDK)
  - https://jdk.java.net/ (Free versions, OpenJDK)
  - http://www.oracle.com/technetwork/java/javase/downloads (Restricted usage)
- ***Install*** JDK to our computer (suppose for version `jdk-24.0.2`)
  - E.g. installed in folder `C:\Program Files\Java\jdk-24\`
- ***Setup*** Java development environment, and path for finding JDK
  - *Reference: http://docs.oracle.com/javase/tutorial/essential/environment/paths.html*
  - E.g. For MS windows, in Command Prompt (cmd):

    ```
    set Path=C:\Program Files\Java\jdk-24\bin;%Path%
    ```

  - We may create a batch file to run cmd, with Java path properly set (below):

```
REM File jCMD.bat, running a cmd command prompt console with Java path set
set Path=C:\Program Files\Java\jdk-24\bin;%Path%
cmd
```

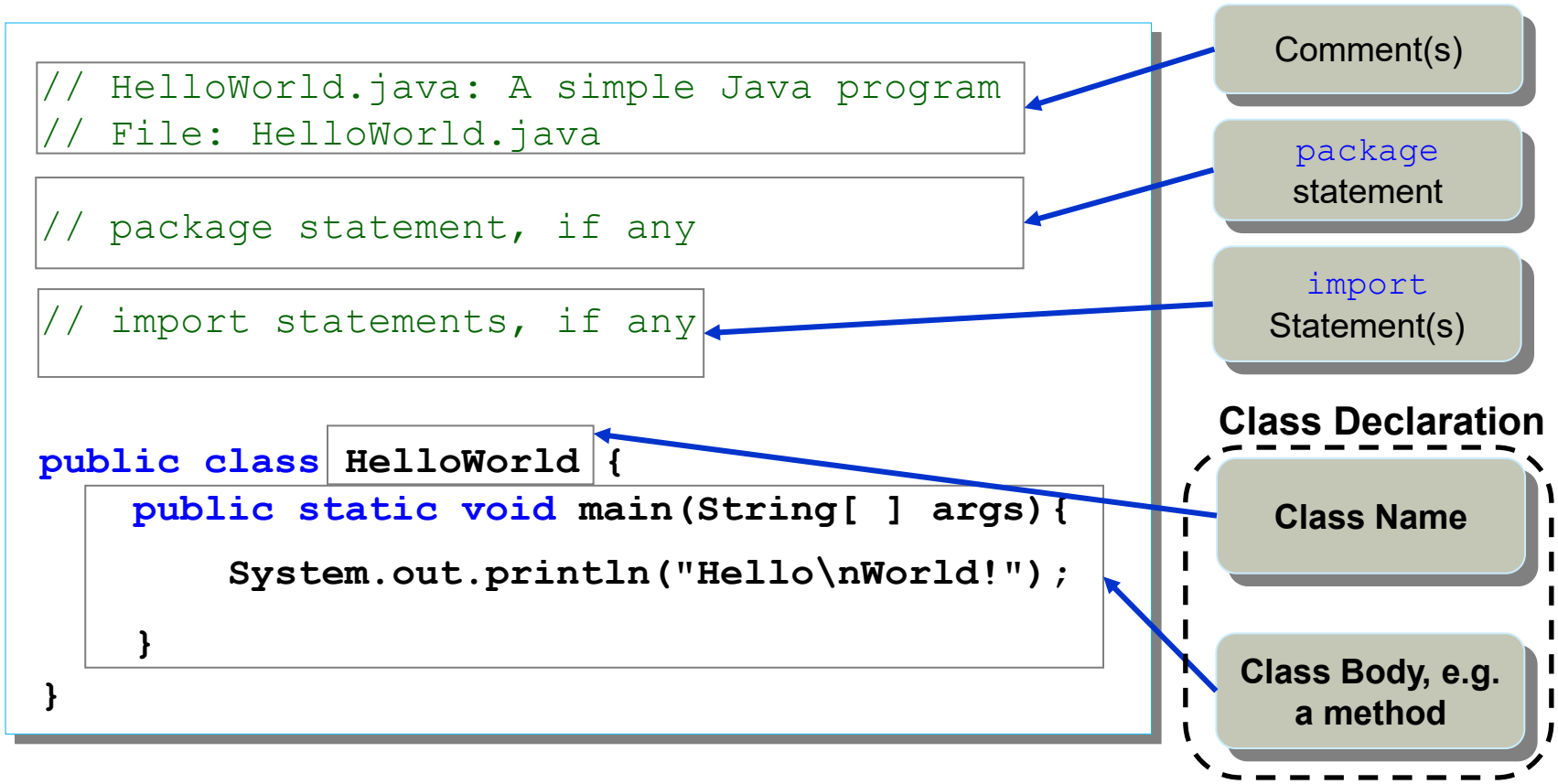
- ***Check*** if JDK (`javac.exe`) is properly installed and set
  - Run **`cmd`** (or our own batch file, such as the sample `jCMD.bat` above) to check, type: **`javac -version`**

Typical Java Program

- **Typical Java program** source file is composed of
  - `Comments`: At the top of each Java source file
  - `package` statement: To indicate which package (like a Java library) this program is attached to
  - `import` statements: To use components in a specific package (package is like a Java library)
  - `class` declaration
    - To run a Java program (of a class), a specific `main()` method of the class acts as the *execution entry point* of an application program.
- Proper programming practice follows common conventions, and includes
  - Proper ***commenting***
  - Proper ***indentation***
  - Proper ***naming***

Getting Start with "Hello World"
(Java Source File: `HelloWorld.java`)

- To start learning Java programming, let us have a look at the Java codes of a very simple **"Hello World" program**:
  - This simple Java program code displays the string message on the standard output (console) when it is executed/run.



Java Programs and Source Codes
("Hello World" Sample Program)

- A brief line-by-line explanation of "Hello World" Sample Program
  - Line 1: a comment line, that shows information to readers, but is ignored by compiler
  - Line 2: class declaration, that declares (defines) our own class, named `HelloWorld`
  - Line 3: the entry-point, where the program starts, a method named `main()`
  - Line 4: body of the method `main()`, which will be executed if the method is called
    - When we run this program (the class) → call `main()` → execute this body
      - This specific method body calls `System` class in a standard package/library to print / display `"Hello\nWorld!"` on console (the standard output),
      - and `'\n'` is a special "new line" character that jumps to next line
  - Line 5: the closing brace } symbol, for pairing with opening brace { in line 3
  - Line 6: the closing brace } symbol, for pairing with opening brace { in line 2

Line	Java Code
1	`// A simple program source code, Hello World`
2	`public class HelloWorld {`
3	`    public static void main(String[ ] args){`
4	`        System.out.println("Hello\nWorld!");`
5	`    }`
6	`}`

COMPILE, on console

```
C:\>javac HelloWorld.java
C:\>
```

RUN, on console

```
C:\>java HelloWorld
Hello
World!
C:\>
```

Sample Java Program – 1
(Sample1.java)

Reference Only

```
/* Sample Program: Display a Window; File: Sample1.java
   The program named Sample1 displays a window on the screen
   Size of the window is set to:
      300 pixels wide x 200 pixels high
*/

import javax.swing.*;

public class Sample1 {
   public static void main(String[ ] args) {
      JFrame myWindow;
      myWindow = new JFrame( );
      myWindow.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
      myWindow.setSize(300, 200);
      myWindow.setVisible(true);
   }
}
```

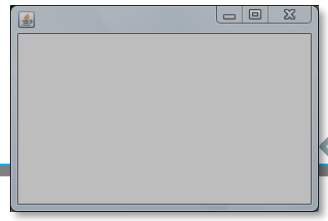
Comment(s)

Import Statement(s)

Class Name

Class Body

Class Declaration



Sample1.java

Compile and Run
Sample1

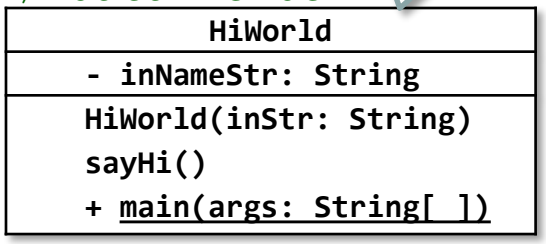
* *Swing application (e.g. with `JFrame`) has its own single thread, called Event Dispatch Thread.*

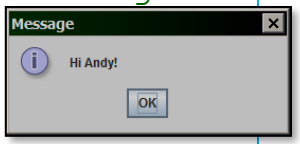
Sample Java Program – 2
(HiWorld.java)

Reference Only

```
// HiWorld.java: A bit complicated Java program
import javax.swing.JOptionPane; // import statement here
public class HiWorld {  // class (named HiWorld) declaration

  private String inNameStr;   // declare a field / data member

  HiWorld(String inStr) {  // a constructor
        inNameStr = inStr;
  }

  void sayHi() {  // a method
        JOptionPane.showMessageDialog(null, "Hi " + inNameStr + "!");
  }

// This special method, main(), is the program entry point as in C
  public static void main(String[ ] args) {
        String nameStr = JOptionPane.showInputDialog(null,
                         "What is your name?");
        HiWorld myHW = new HiWorld(nameStr); //declare & create object
        myHW.sayHi(); //call the method (of object)
  }
}
```

UML Class Diagram





* *Remark:* Various concepts related to this program will be discussed in later units.

Basic Error Handling in Programming

- There are 3 common types of programming errors (or called bugs).
  - **Compilation / Syntax Errors** occur during compilation (detected by the compiler), often caused by invalid syntax (not following the rules)
    - Syntax refers to rules governing proper programming based on specific language (similar to grammar rules in natural languages)
      - E.g. `inT 3hundred == 300;`
  - **Run-Time Errors** are detected by machine during program execution
    - A program may pass compilation but contains codes to ask computer to do works which could not be *properly* finished.
    - This causes problems during run-time operation (detected by machine, the JVM in Java program), e.g. a number divided by value zero.
  - **Semantic / Logic Errors** are not detected by machine.
    - A program may pass compilation and execute without run-time errors, but it wrongly finishes the original expected works.
      - Example 1: assign a wrong value to a variable: `int ten = 100;`
      - Example 2: get a multiplication result for an addition task: `float sum = a*b;`
    - It heavily relies on developers to handle this type of errors.

Basic Error Handling in Programming

- **Compilation / Syntax Errors** are often much easier to be handled:
  - Error messages given by the compiler are good reference to support solving programming errors (debugging)
  - With IDE supports and better programming skills of developers, compilation errors could normally be solved in a very short time.
- **Run-Time Errors** may be a bit harder to be handled:
  - Error messages given by the run-time machine (JVM in Java) are good reference to support error-handling
  - Debugging Run-Time errors often replies on proper testing cases planned to check the program, before releasing for actual operation.
- **Semantic / Logic Errors** are often the hardest to be solved
  - These errors rely much on programmers (or users, etc.) to discover, who may need specific knowledge in the application areas.

** Other tools may also help, including debugger, tools in IDE*

Basic Error Handling in Programming

- A general approach of handling / solving programming errors (**debugging**) includes:
  - Identify ***what*** the error is
  - Isolate ***where*** the error occurs
  - find ways ***how*** to correct the error
- Compiler (and run-time) error messages are often very helpful

Common Problems in Java Programming

Common compilation problems (E.g. with javac.exe on Windows OS, type `C:\javac MyTest.java` on console)

- **Compiler not found** (on Windows, `javac.exe`), in compilation

```
'javac' is not recognized as an internal or external command, operable
program or batch file
```

  * The right path (of the JDK) should be set before compilation with **`javac.exe`**

- **Syntax Errors**, in compilation, e.g

```
MyTest.java:14: ';' expected.
System.out.println("Input has " + count + " chars.")
                                                   ^
```

- **Some "non-syntax" compilation errors** , in compilation, e.g.

```
MyTest.java:13: Variable count may not have been initialized.
    count++
    ^
```

Common Problems in Java Programming

Common Runtime problems (E.g. with java.exe on Windows OS, type `C:\java MyTest` on console)

- **Bytecode `.class` file not found**, in running

```
Exception in thread "main" java.lang.NoClassDefFoundError: MyTest
```

  * The right CLASSPATH (of the JDK) should be set

  * Often common error to beginners if running program with `java MyTest.class` instead of `java MyTest`

- **The "proper" main method cannot be found**, in running

```
Exception in thread "main" java.lang.NoSuchMethodError: main
```

  - Java VM requires the class we execute has a proper `main` method to begin execution
  - The form should be `public static void main (String [] args)`

Our Course - OOP

- Our course, “Object-Oriented Programming”:
  - Is a course introducing Object-Oriented Programming
    - We expect that all our students have already obtained enough skills and knowledge background of basic computer programming
      - Finished our ICP course without much problem
  - Focuses on writing programs to solve not very complicated programming tasks in OO approach, with a specific programming language: Java
    - Tasks and example programs given may be simple, but they help to consolidate our understanding
    - Some tasks look simple to be finished in Python, but not in Java.
      - Java is another popular programming language, with more conventional / traditional features not in Python.
      - Students who can do programming in both Java and Python are really in an advantaged position.
    - Some tasks look simple in functions (procedural), but not in OOP.
      - OO Programming is important and popular in programming
      - Python also supports OOP

Our Course - OOP

- Schedule and Duration:
  - Total 12 lessons, 3 hours per lesson
- Course Content and Intended Learning Outcomes:
  - Check our given schedules and the official course document
  - Students are STRONGLY recommended to download the official course document of our course from the learner portal, for reference
- Platform for Materials and Submissions: College SOUL System
  - https://soul2.hkuspace.hku.hk/
  - Course materials are available on SOUL, according to our schedule.
  - Release and Submission of assessment works via SOUL: include
    - Online (SOUL-Quiz feature) assignment
    - Coding and Programming assignment
    - Written assignment

Our Course - OOP
(2025-2026 Sem. 1)

- Continuous Assessment: 50% of the whole course
  - There will be 4 assignments. Each of the first 3 assignments carries 15% of the whole course. The 4th assignment carries 5% of the whole course. Details of the assignments will be given accordingly
  - Assignments help you understand the course materials, and feedback how much and deep you have understood
- Written Examination: 50% of the whole course
  - It is the “final” assessment on our learning outcomes
- Attendance is taken, but is NOT counted towards the final grade
  - However, students should take care of their own learning progress properly, particularly our course schedule and the assignment deadlines
- Follow given instructions and guidelines:
  - *Also follow specific instructions given by individual class teacher*

Our Course – OOP

- For our course, students should:
  - Review our “Introduction to Computer Programming” (ICP) course materials if necessary, especially the concepts of basic programming and multiple functions in procedural programming approach, as we expect our students have properly finished the ICP course.
  - Have Java Development Kit (JDK) installed and setup properly in your own computer at home, before lesson 1 if you can. Follow installation instructions in official website, our lecture or lab manual for information.
  - Preview the course materials of each coming lesson if the materials are released in advance. Write down any questions you want to discuss in the coming lesson.
  - Review the course materials of each lesson after finished. Write down any questions and raise them in the next lesson for discussion.
  - Keep checking your college email account and the SOUL platform regularly, in case of any updated announcement / notice of our course.

References

- This set of slides is only for educational purpose.

- Part of this slide set is referenced, extracted, and/or modified from the followings:
  - Deitel, P. and Deitel H. (2017) “Java How To Program, Early Objects”, 11ed, Pearson.
  - Liang, Y.D. (2017) “Introduction to Java Programming and Data Structures”, Comprehensive Version, 11ed, Prentice Hall.
  - Wu, C.T. (2010) “An Introduction to Object-Oriented Programming with Java”, 5ed, McGraw Hill.
  - Oracle Corporation, “Java Language and Virtual Machine Specifications” https://docs.oracle.com/javase/specs/
  - Oracle Corporation, “The Java Tutorials” https://docs.oracle.com/javase/tutorial/
  - Wikipedia, Website: https://en.wikipedia.org/