

# Object-Oriented Programming

## Introduction to Object-Oriented Programming

United International College

# Outline

- Object-Oriented Programming
- Objects vs. Classes
- Class Diagram
- Java Fundamentals

# High-Level Languages

- Historically, the high-level languages can be divided into two groups: **procedural languages** and **object-oriented languages**
- Procedural language
  - specifies a series of well-structured **steps** and **procedures** within its programming context to compose a program
  - Examples include C, Fortran, Perl, HTML, ...

# Object-Oriented Programming

- Object-oriented programming is the dominant language paradigm these days.
- The focus of OOP is on modeling data
- An object-oriented program is made of **objects**, the **class** is the blueprint from which the objects are made.
- Examples of OOP language include C++, Visual Basic.Net, Java, Python.

# Object-Oriented Programming

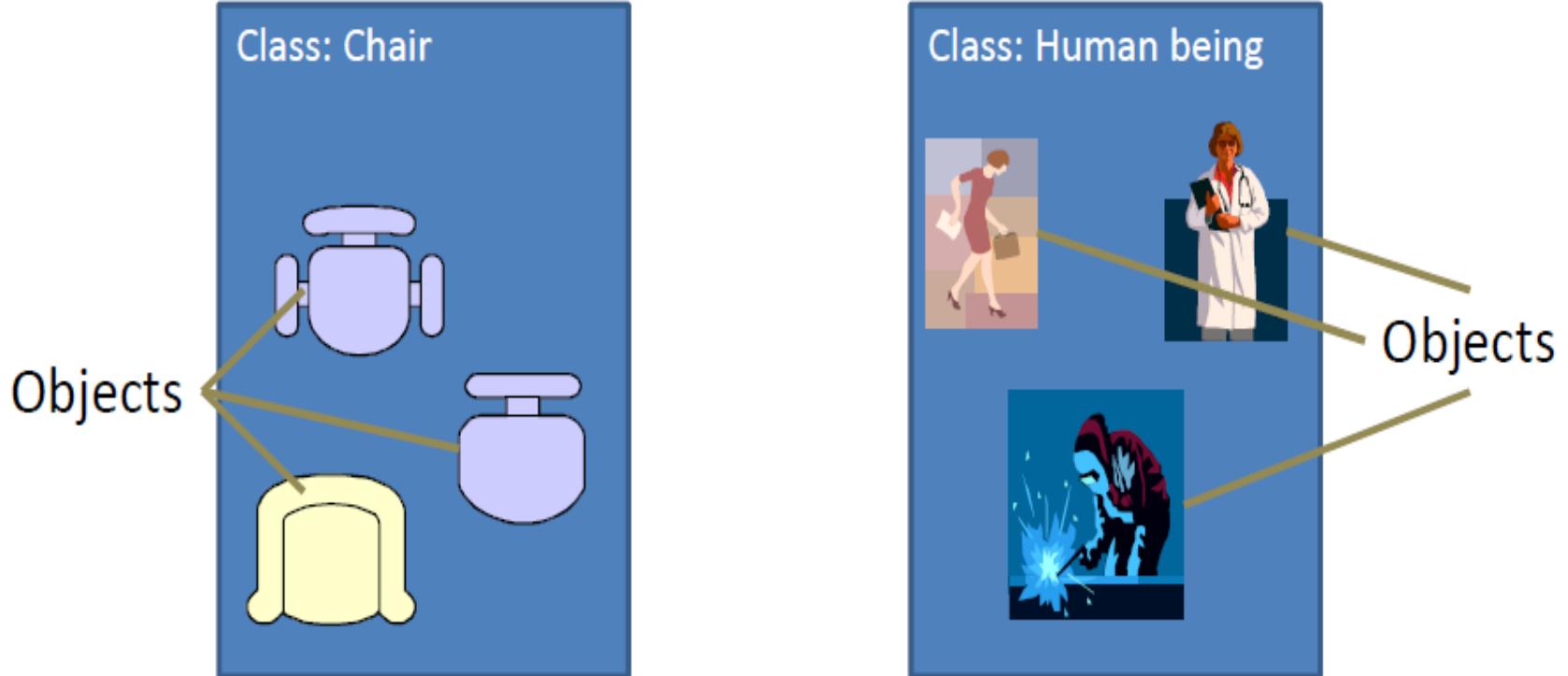
Key idea in OOP:

*The real world can be accurately described as a collection of **objects** that interact.*

# Classes and Objects

- A **class** is a prototype, template and blueprint from which objects are made.
- An **object** is an **instance** (实例) of a class.
- A class is a generalized description that describes a collection of similar objects.
- **Example:** Chair, Human, Students, Teachers, Books, etc.

# Classes and Objects



# Classes and Objects

- Object - An entity
  - Physical: a chair, a desk, a person
  - Logical: a list, a stack, a rectangle
- Objects have **data** and **behavior**
  - Data in an object is called **instance fields**
  - Behaviors that operate on the data are called **methods**
  - A **specific object** will have a specific values for its instance fields. The set of those values is the current **state** of the object.
- Example: Dog
  - **Data**: Color, Name, Breed
  - **Behaviors**: Fetch stick, Drink water, Wag tail, Bark

# Classes and Objects

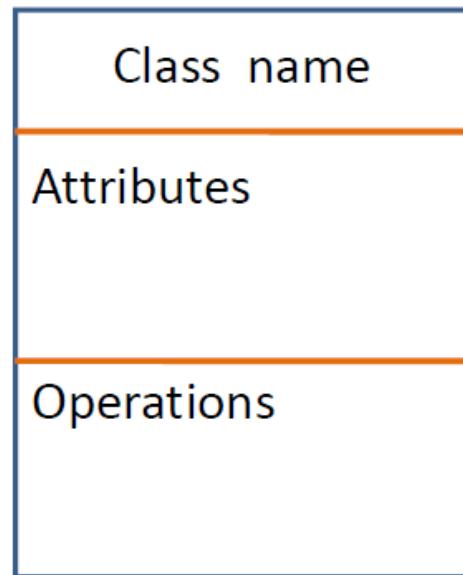
*An **object** is an instance of a **class***

*A **class** is a blueprint from which  
**objects** are made*

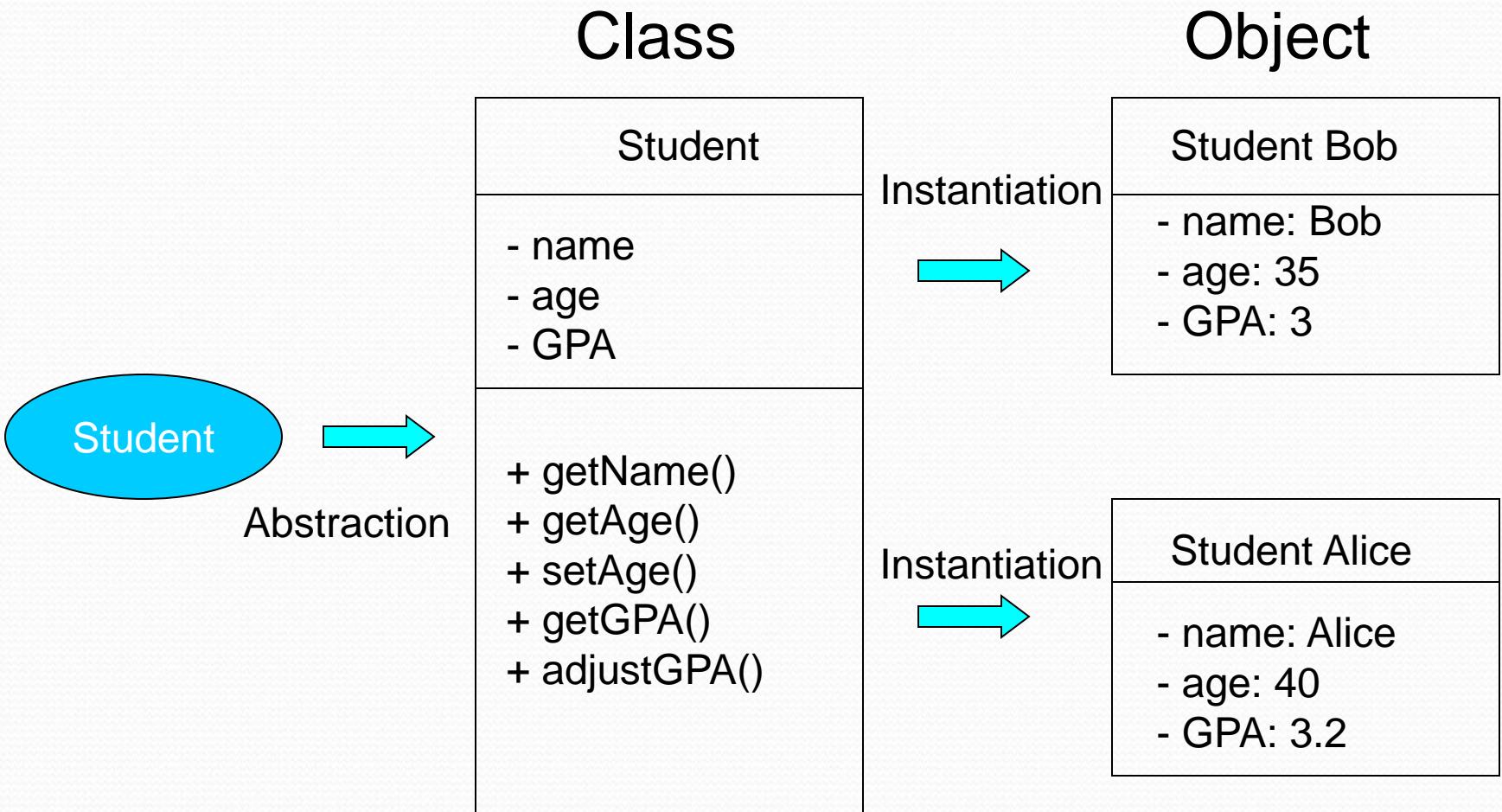
- Animal is a **class**, a cat called “Snowball” is an **object**
- Vehicle is a **class**, my BMW is an **object**
- Student is a **class**, a student called “Tom” is an **object**
- ...

# Class Diagrams

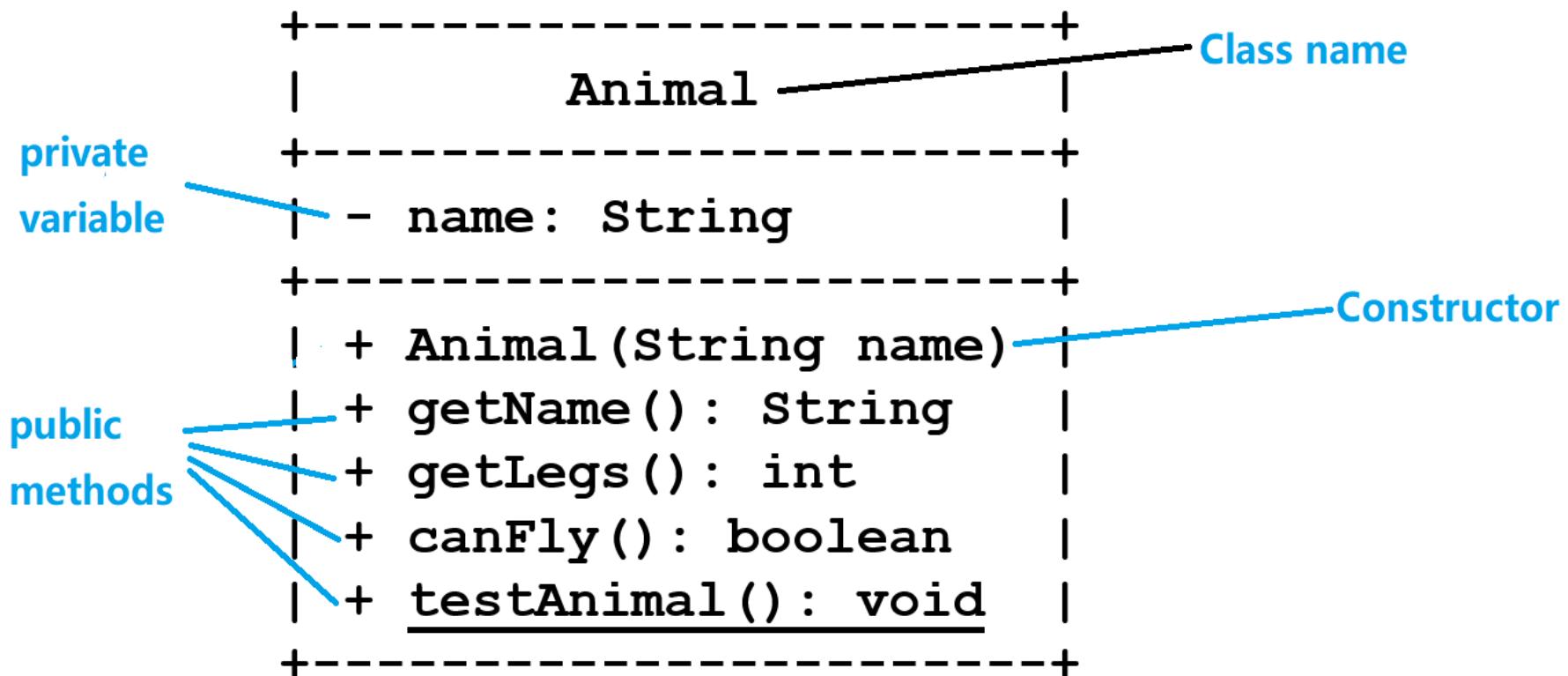
- Describe the system in terms of **classes** and their **relationships**.
- Natural ways of reflecting the real-world entities and their relationships.
- Essential part in OO software Development.



# Example



# Examples in the labs



# Object-oriented Design

- Step 1: Given a problem, considering which class / object will exist in the problem domain.
- Step 2: Considering for each class / object, what fields and methods it should have.
- Step 3: Considering the relationships between different classes / objects.

# Object-Oriented

- OO in one sentence:

keep it **DRY**, keep it **Shy** and **Tell the other guy**

- **DRY**: Do not Repeat Yourself
- **Shy**: Should not reveal the information about itself unless really necessary
- **Tell the other guy**: Send a message rather than doing a function call.

- By Andy Hunt and Dave Thomas.

# Benefits of OOP Approach

- **Modularity:**
  - The source code for an object can be written and maintained independently of the source code for other objects.
- **Information-hiding:**
  - By interacting only with an object's methods, the details of its internal implementation are hidden from the outside world.
- **Code re-use:**
  - If an object already exists you can use that object in your program. This allows specialists to implement / test / debug complex, task-specific objects, which you can then trust to run in your own code.

# Object-Oriented Software Development Process

- **OO Analysis:** Requirement specification
- **OO Design:** Architectural design
- **Object Design:** Detailed design
- **Object-Oriented Programming:** Implementation



Our focus this semester!

# History of Java

- Developed by Sun Microsystems.
- Sun was bought by Oracle in 2010.
  - 1996: Java version 1.0;
  - ... Java 1.1, 1.2, 1.3, 1.4;
  - 2004: Java version 1.5, name changed to Java 5;
  - ... Java 6, 7, 8.

One new release every 6 months now:

- September 2017: Java 9;
- March 2018: Java 10;
- September 2018: Java 11 (long term support);
- March 2019: Java 12
- .....



# Features of Java

- Simple
- Object-oriented
- Platform neutral
- Robust, secure, scalable.

E.g.: programmers don't need to care about memory allocation and release.

- Java is everywhere!

# Compiler

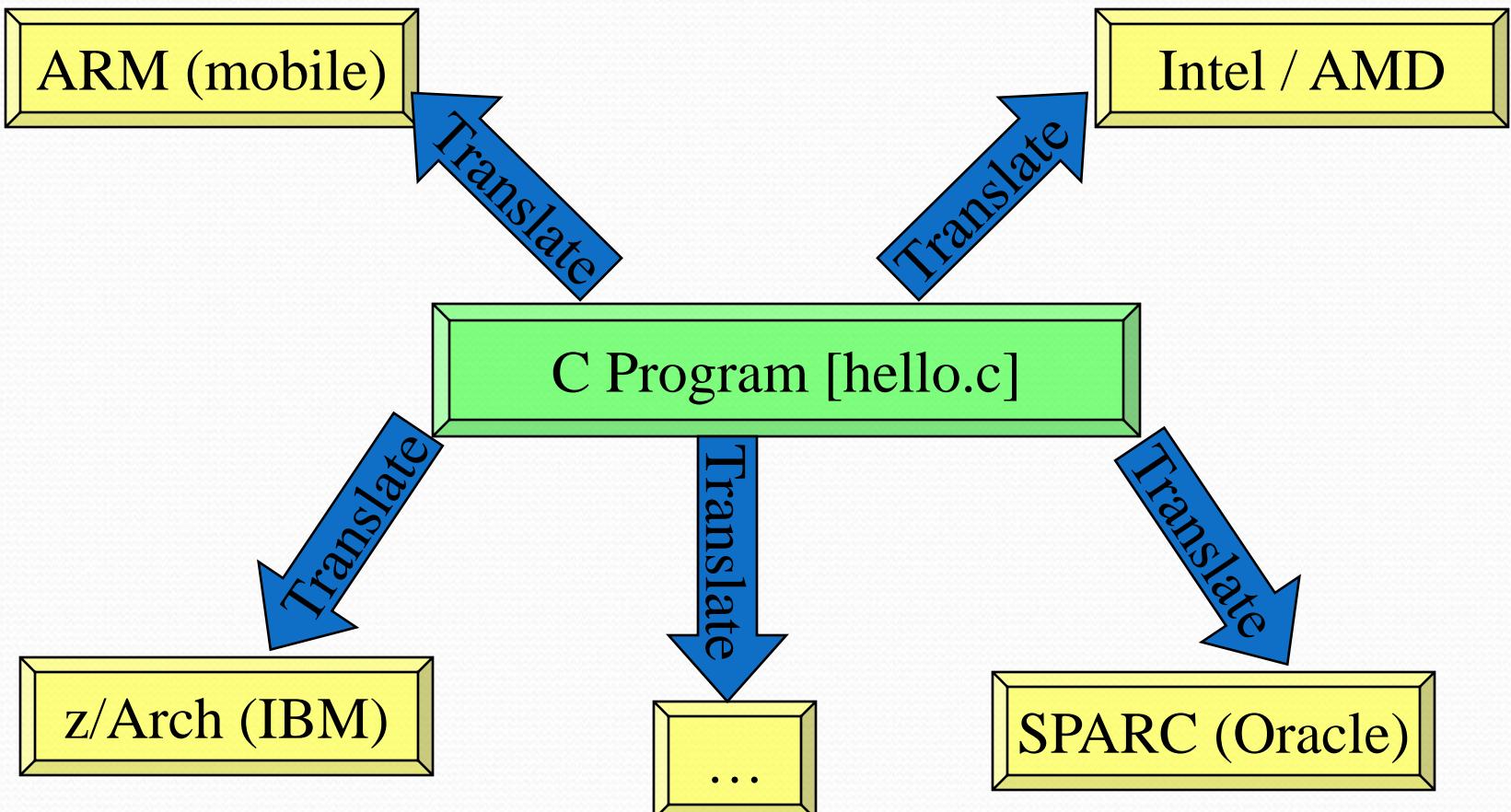
- **Compiler**: a program that translates a high-level language program into an equivalent low-level language program.
- This translation process is called **compiling**.

Program Written in  
High-level Language  
e.g. Pascal, C.

Compiler

Executable Program in  
Low-level Language  
e.g. Intel x86 code.

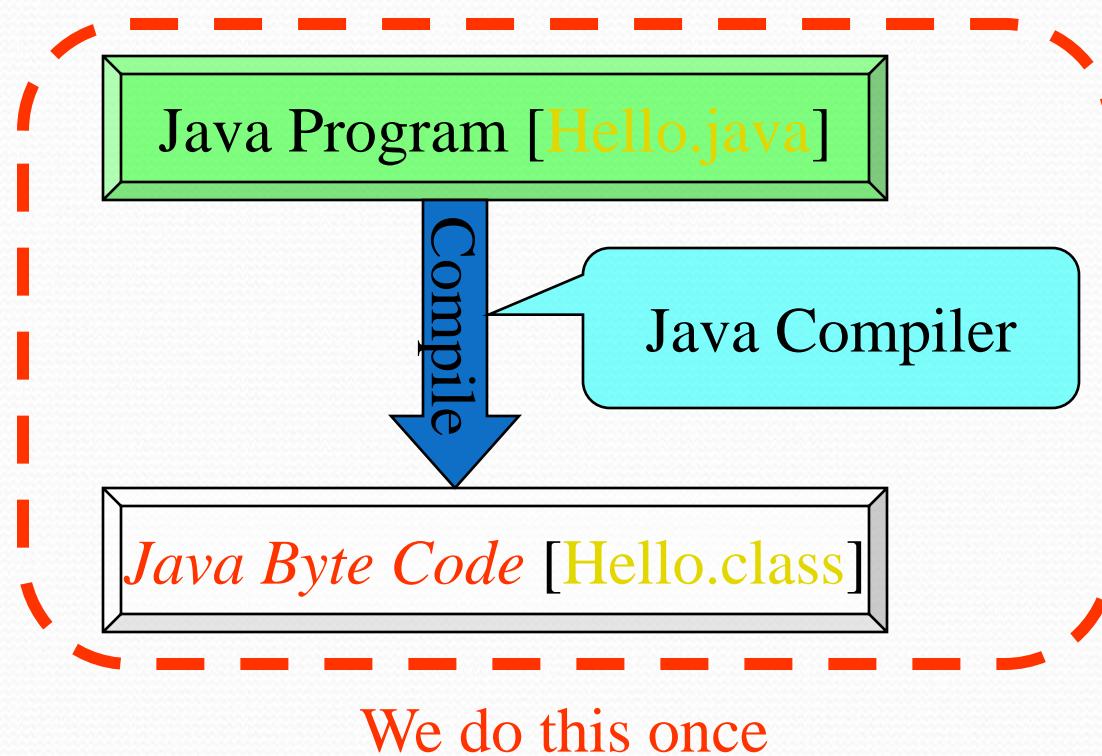
# Really Cross Platform?



# Pitfalls

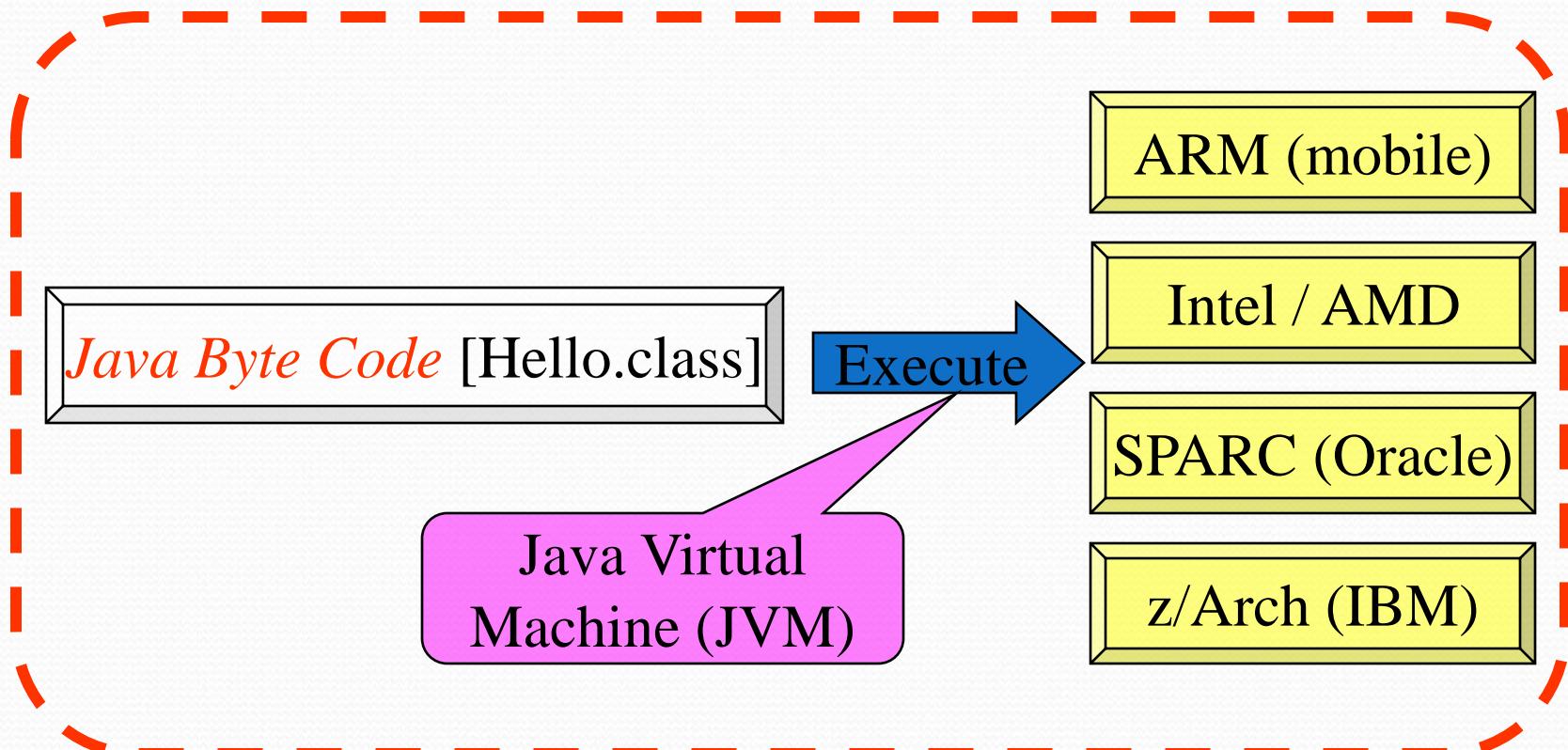
- Store and use different *compiled* versions.
- Availability of compiler(s) is a must.
- Compiler compatibility problems.
- Re-compile all versions after an update.

# Java Compilation Model

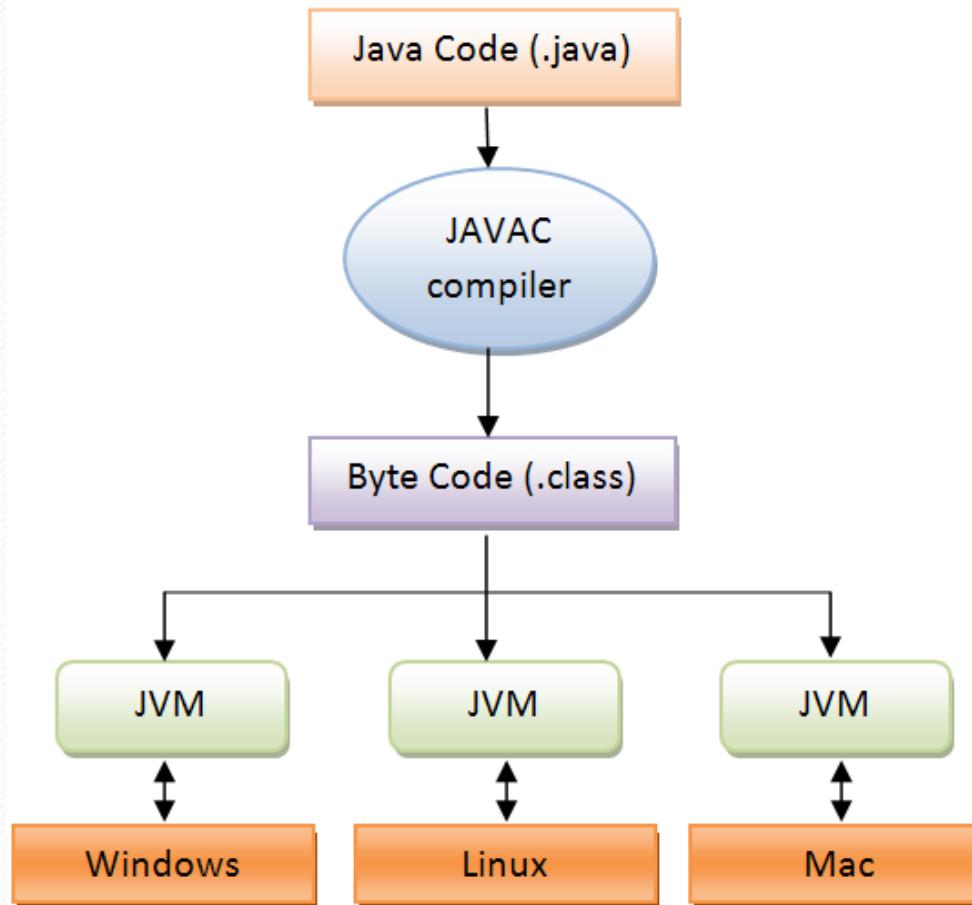


# Java Compilation Model

Someone else creates this for us.



# Java Compilation Model



# Byte-Code and the Java Virtual Machine

- Java compiler translates Java programs into ***byte-code***, a language for the ***Java Virtual Machine***.
- Once compiled to ***byte-code***, a Java program can be used on ***any*** computer that has a JVM, making it very portable.

**Write-Once → Run-Anywhere**

# The heart of Java

- **Java Virtual Machine**
  - Reads “byte code” and executes it.
- **Garbage collection**
  - Manages memory automatically;
  - Included in the JVM.

# Two Java Environments

- JRE
  - Java Runtime Environment = JVM + Java libraries.
  - For users to run byte-code programs on their computer.
- JDK
  - Java Development Kit = JRE + Java compiler + other development tools.
  - For developers to write Java software, create byte-code, and test it.

# Three Java Editions

- Java Micro Edition (**JavaME**)
  - Very small Java edition for smart cards, pages, phones, and set-top boxes.
  - Subset of the standard Java libraries aimed at limited size and processing power.
- Java Standard Edition (**JavaSE**)
  - The basic platform, which this course will cover.
- Java Enterprise Edition (**JavaEE**)
  - For business applications, web services, mission-critical systems.
  - JSP, Servlets, JDBC, EJB3.0, Struts – Spring – Hibernate.

# Example: the Java SE 8u161 JDK

- Java programming language;
- Standard Edition: includes all libraries for PCs;
- Version 8;
- Update 161: the Java environment is updated regularly to fix bugs;
- Java Development Kit: includes tools for programmers, such as the Java compiler.

# The First Program

- Install the Java SE JDK on your computer:  
<https://www.oracle.com/java/technologies/downloads/>
- Install Eclipse: <https://www.eclipse.org/downloads/>  
(select “Eclipse IDE for Java Developers” when installing);
- See the link “*How To Install Eclipse on Your Own Computer*” on iSpace for more information.

# Running a Java Program

- Start Eclipse and create a workspace: a folder that will contain all your Java projects.
- Create a Java project.
- Add a new class to your project, name it HelloWorld.
- Type the code of the HelloWorld class: see next slide.
- Building (compiling Java into byte code) is automatic.
- Run the program, which executes the byte code on a JVM.

# Hello World Program

```
public class HelloWorld {  
  
    /* The HelloWorld Program  
    -----  
    Illustrates a simple program displaying  
    a message.  
 */  
  
    public static void main (String[] args) {  
        System.out.println("HelloWorld!");  
    }  
}
```

Type all carefully and save it to a file named **HelloWorld.java**.

# The First Java Program

Java program source files (.java)  
contain definitions of **classes**.

```
public class HelloWorld {  
  
    /* The HelloWorld Program  
    -----  
    Illustrates a simple program displaying  
    a message.  
 */  
  
    public static void main (String[] args) {  
        System.out.println("HelloWorld!");  
    }  
}
```

# The First Java Program

Curly braces pair enclose a block of code,  
class **HelloWorld** here.

```
public class HelloWorld {  
  
    /* The HelloWorld Program  
    -----  
    Illustrates a simple program displaying  
    a message.  
 */  
  
    public static void main (String[] args) {  
        System.out.println("HelloWorld!");  
    }  
}
```

Don't miss me!

# The First Java Program

Curly braces pair enclose a block

```
public class HelloWorld { of code, method main( ) here.
```

```
/* The HelloWorld Program
-----
Illustrates a simple program displaying
a message.
*/
```

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

Don't miss me!

# The First Java Program

```
public class HelloWorld {
```

This is a block of **comments**,

```
for human, not for computer.
```

```
/* The HelloWorld Program
```

```
-----
```

```
Illustrates a simple program displaying  
a message.
```

```
*/
```

It explains to you what happens.

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

# The First Java Program

/\* and \*/ pair encloses a comment block.

```
public class HelloWorld{
```

```
    /* The HelloWorld Program
```

```
    -----  
    Illustrates a simple program displaying  
    a message.
```

```
*/
```

Don't miss me!

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

# Comments

- `// Comment single line`
- `/* Comment  
multiple  
lines  
*/`
- `/**  
*  
**/`

# The First Java Program

```
public class HelloWorld { HelloWorld, named main( )  
  
/* The HelloWorld Program  
-----  
Illustrates a simple program displaying  
a message.  
*/  
  
public static void main (String[] args) {  
    System.out.println("HelloWorld!");  
}  
}
```

This is a method of the class

HelloWorld, named main( )

# The First Java Program

Standard properties of the `main()` method.

```
public class HelloWorld {  
  
    /* The HelloWorld Program  
    -----  
    Illustrates a simple program displaying  
    a message.  
 */  
  
    public static void main (String[] args) {  
        System.out.println("HelloWorld!");  
    }  
}
```

# The First Java Program

A **statement** (instruction) to display

```
public class HelloWorld {
```

a message.

```
/* The HelloWorld Program
```

```
-----  
Illustrates a simple program displaying  
a message.
```

```
*/
```

```
public static void main (String[] args) {
```

```
    System.out.println("HelloWorld!");
```

```
}
```

```
}
```

# The First Java Program

```
public class HelloWorld
```

After every statement, there must  
be a semi-colon!

```
/* The HelloWorld Program
-----
Illustrates a simple program displaying
a message.
```

```
 */
public static void main (String[] args) {
    System.out.println("HelloWorld!") ;
}
```

What does this  
String[] args do?

# Java Programming

- A Java program consists of *objects* that interact with one another by means of actions (called *methods*).
- Other high-level languages (C, Fortran) have procedures, functions, and subprograms.
  - These are called *methods* in Java.
  - All programming constructs in Java, including *methods*, appear in an *object* (which is an *instance* of a *class*).

# Java Programming

- A Java ***application program*** is a class with a method named **main**.
  - When a Java application program is run, the JVM automatically executes the byte-code for the method named **main**.
  - All Java application programs start with the **main** method.
  - Your Java software is going to contain many classes, but only one class has a **main** method.

# Java Programming

- Application programs may use:
  - a windowing interface (GUI): Java Swing;
  - or a console (i.e., text) I/O.

# Program terminology

- **Code**: a program or a part of a program.
- **Source code** (or *source program*): a program written in a high-level language such as Java.
  - The input to the compiler program, e.g., Java code.
- **Object code**: the translated low-level program.
  - The output from the compiler program, e.g., Java byte-code.
  - Java byte-code the input to the Java Virtual Machine.

# Syntax and Semantics

- **Syntax:** the arrangement of words and punctuations that are legal in a language, the *grammar rules* of a language.
- **Semantics:** the **meaning** of things written while following the syntax rules of a language.

# Tip: Error Checking

- **Bug**: a mistake in a program.
  - The process of eliminating bugs is called *debugging*.
- **Syntax error**: a grammatical mistake in a program.
  - The compiler (or Eclipse) can detect these errors, and will output an error message saying what it thinks the error is, and where it thinks the error is.

# Tip: Error Checking

- ***Run-time error***: an error that is not detected until a program is run.
  - The compiler cannot detect these errors: an error message is not generated during compilation, but during execution (usually the program then dies!)
- ***Logic error***: a mistake in the underlying algorithm for a program.
  - *The compiler cannot detect these errors, and no error message is generated during compilation or during execution, but the program does not do what it is supposed to do.*

# Summary

- Introduction to Programming
- Object-Oriented Programming
- Objects vs. Classes
- Class Diagram
- Java Fundamentals