Lecture 1: Introduction to Java And the Eclipse Development Environment

Pulling the Arrow Back to Hit the Target

Wholeness of the Lesson

Java is an object-oriented highly portable programming language that arose as an easy alternative to the once dominant, but error-prone, C++ language. Eclipse is one of many open source, powerful but easy-to-use integrated development environments for use with Java and related technologies. Working from deeper levels of intelligence allows one to accomplish more with fewer mistakes and less effort.

History of Java

- The Java language began as a language for programming electronic devices, though the original project was never completed.
- Its creator was James Gosling, of Sun Microsystems. The language was developed privately starting in 1991, and was made publicly available in 1994.
- In 2009, Oracle bought the rights to Java from Sun Microsystems.

First Java Program

- We begin with a simple example - `Hello.java`

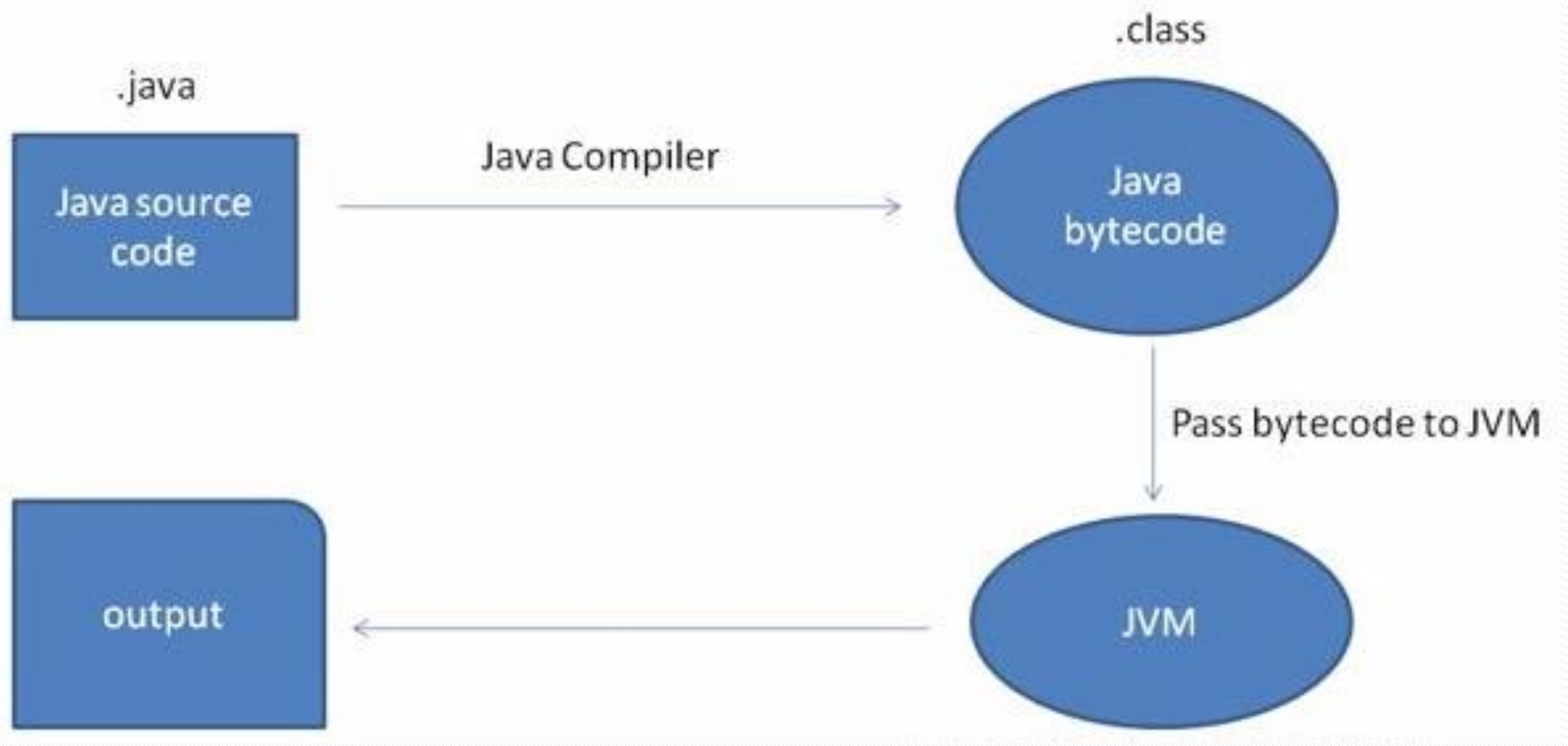
(We will talk more about the syntax in later lecture.)

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

Things to understand:

- public
- class
- void
- main method
- System.out.println
- delimiters: `;`, `}`, `{` ("blocks")
- capitalization conventions

Compiling and Running Java Programs



JVM interprets the bytecode into machine language(binary code) for the specific platform and finally produces output.

About Java

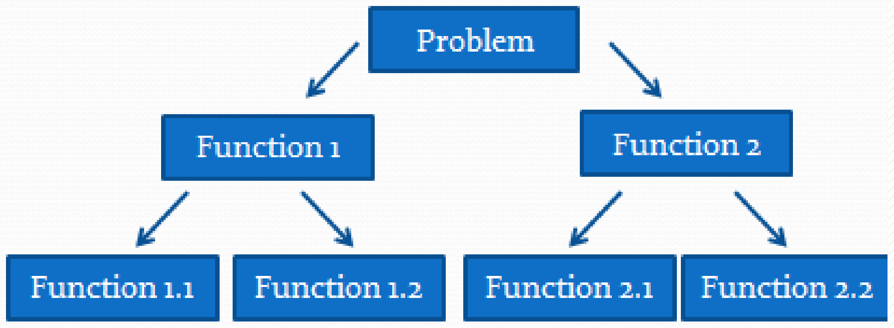
- ***Interpreted Language*.** When you "compile" Java code, the result is not executable binary code, targeted to a particular machine; instead, the result is **bytecode**, having a portable intermediate code format. The bytecode is then executed by running an **interpreter,** called the **Java Virtual Machine** (JVM). This approach makes Java code highly portable; Java will run on any platform for which a JVM has been created.

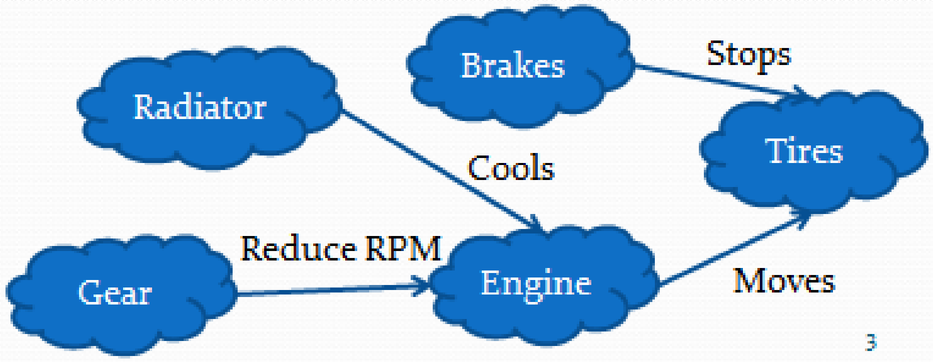
About Java

- ***No Pointers.*** Unlike C and C++, Java does not make use of pointers; developers are not required to manage the memory usage of their applications. Memory management is handled automatically in Java by means of a **garbage collector.** The garbage collector is run by the JVM at different times during program execution to return to memory all unusued object references.
- ***Convenient Libraries.*** Java provides convenient libraries for handling files and streams, networking, http, gui development, and database connection. Compared to languages like C and C++, the increased ease of use is significant.
- ***Good Security Model.*** Java has a relatively good security model to support use over networks and distributed environments. Though security holes are still found sometimes, these are rare and quickly patched.

About Java

- ***Java Is an OOP Language.*** Java is an OO programming language, and succeeds in this better than C++, which supports a non-OO programming style (in C++, OO programming is "optional").





- Procedural analysis & design

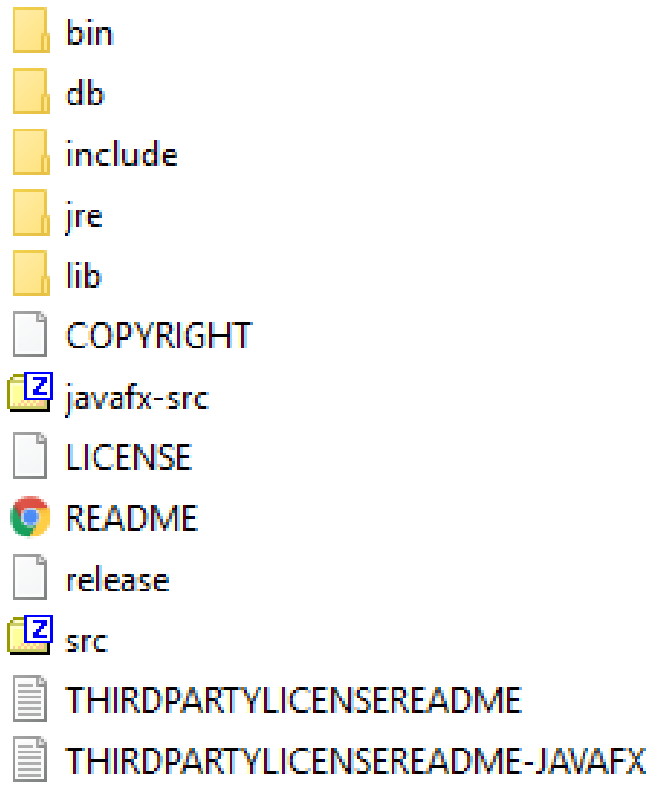
In early days, programmers adapted real-world problems into "computer logic"

- OO analysis & design

Using OO, real-world objects are represented by software objects; real-world behaviors are represented by sending messages between objects.

The JDK Distribution

- bin: Executable files for the development tools contained in the Java Development Kit.
- lib: Files used by the development tools.
- jre: Root directory of the Java runtime environment used by the JDK development tools. The runtime environment is an implementation of the Java platform.
- src.zip: Archive containing source code for the Java platform.
- db: Contains Java DB.
- Oracle provides online documentation of all the Java library classes. For jdk1.8, the link is https://docs.oracle.com/javase/8/docs/api/



```
bin
db
include
jre
lib
COPYRIGHT
javafx-src
LICENSE
README
release
src
THIRDPARTYLICENSEREADME
THIRDPARTYLICENSEREADME-JAVAFX
```

Main Point

Java is an object-oriented programming language that is easier to use, less error-prone, and more portable (and nowadays, more popular) than C++. Java code is compiled to *bytecode*, which can then be converted to native code on a target platform, by way of a JVM interpreter. The inefficiency of an interpreter has largely been eliminated through the use of the *just-in-time compiler*, which compiles frequently occurring bytecode sequences to native code and caches these for further use, at runtime. Any language has the power to reveal or obscure the truth – as Maharishi says in SCI, "it is the power of speech that it can bind the boundless."

Integrated Development Environments

- A good IDE supports compiling, running, and debugging code with tools that are integrated and typically easy to use. For a large Java project, an IDE is indispensable.
- Good choices of IDE are NetBeans, IBM Rational Application Developer (formerly WebSphere Application Developer), Borland's JBuilder, JetBrains' IntelliJ
- Another excellent choice, which has become an industry standard, is the open-source IDE Eclipse, written entirely in Java. Among IDEs for Java, in recent years, Eclipse is the most widely used. We will use Eclipse in this course.

The Eclipse IDE

- Prepare your machine today:
  - Download and install Java (Any version after 9 is fine.)
    - https://www.oracle.com/java/technologies/javase-jdk11-downloads.html
  - Download Eclipse from:
    - http://www.eclipse.org/

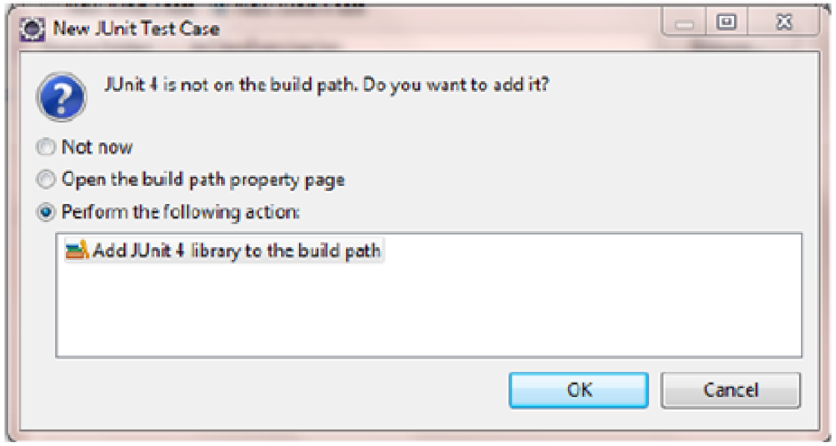
- Features of the IDE [demo]
  - Set up a new empty directory as starting workspace
  - Create Hello world example
  - Show the classes/packages on the file system
  - Show how to compile and run
  - Show how to format Ctrl + Shift + F
  - Attach and View Source Code Ctrl + Shift + T
  - Find some method Ctrl+O
  - Create Unit test

Including Unit Testing in Your Work Environment

- A quick way to test your code as you develop is to run it from the main method as we have been doing
- A better way that is more reusable and useful for larger-scale team development is to have a parallel test project and use JUnit

Steps to set up JUnit

- Right click the default package where your Hello.java class is, and create a JUnit Test Case TestHello.java by clicking New→JUnit Test Case. (After we introduce the package concept, we will put the tests in a separate package.)
- Name it TestHello and click finish. You will see the following popup window.



New JUnit Test Case

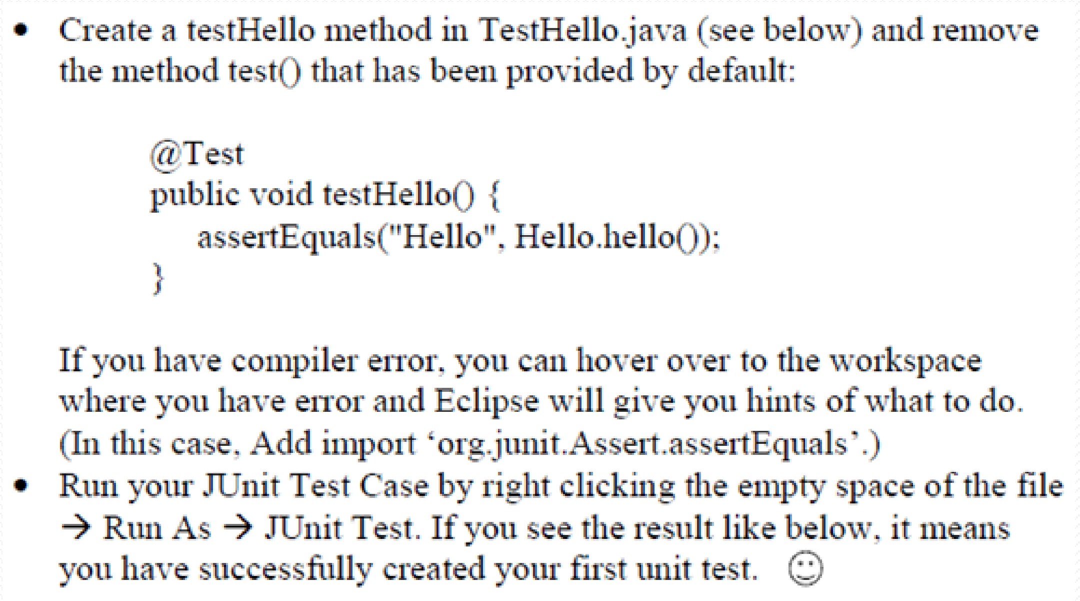
JUnit 4 is not on the build path. Do you want to add it?

( ) Not now
( ) Open the build path property page
[✓ Perform the following action:]

Add JUnit 4 library to the build path

- Select "Perform the following action" → Add JUnit 4 library to the build path → OK
- Then you will see that JUnit 4 has been added to the Java Build Path by right clicking your project name → properties → Java Build Path.

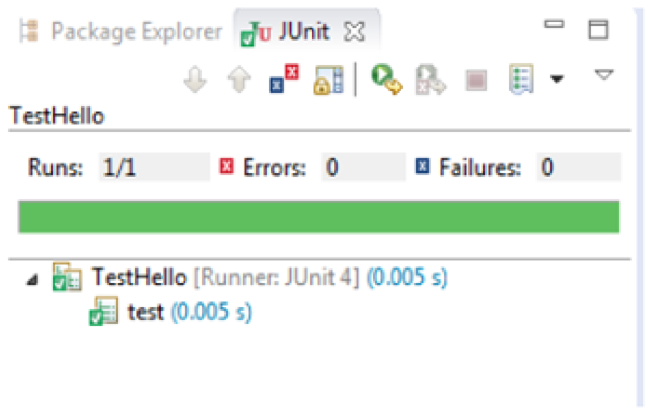
Including Unit Testing in Your Work Environment



- Create a testHello method in TestHello.java (see below) and remove the method test() that has been provided by default:

```
@Test
public void testHello() {
    assertEquals("Hello", Hello.hello());
}
```

  If you have compiler error, you can hover over to the workspace where you have error and Eclipse will give you hints of what to do. (In this case, Add import 'org.junit.Assert.assertEquals'.)
- Run your JUnit Test Case by right clicking the empty space of the file → Run As → JUnit Test. If you see the result like below, it means you have successfully created your first unit test. ☺



```
TestHello
Runs: 1/1    Errors: 0    Failures: 0

TestHello [Runner: JUnit 4] (0.005 s)
    test (0.005 s)
```

Reference Example

- Java applications are *object-oriented*. This means that, unlike C, a Java program works by invoking multiple objects that then interact to produce results.
- We provide a sample to give a feeling for how a Java program works. The details about it will be explained in Lesson 3. For now, it is possible to learn quite a bit by just playing with the syntax to create other programs.
- See the code in the package

  `lesson1.basics.typicalprogram`

  This code will be referred to in future lessons as the

  reference example

Comments In Java

- commenting out a line with //
- commenting out a block with /* … */
- commenting using javadoc format /** … */
- some javadoc keywords @author, @since, @param @return
- javadoc in Eclipse
- running javadoc (demo) - Project > Generate javadoc
- ***Style:*** Every significant method you write should be documented with comments, javadoc style. (This is also true of every Java class you create.)
- See how this is done in the ***reference example.***

Main Point

Eclipse is a leading, open-source, 100% Java, integrated development environment, which provides excellent support for editing, compiling, running, and de-bugging Java applications. By analogy, to create a good life, we need to handle inner life and at the same time, structure a life-supporting environment – the goal is to live 200% of life.

Connecting the Parts of Knowledge With the Wholeness of Knowledge

1. Using Java, highly functional applications can be built more quickly and with fewer mistakes than is typically possible using C or C++.
2. To optimize the use of Java's features, IDE's such as Eclipse ease the work of the developer by handling in the background many routine tasks.

---

3. **Transcendental Consciousness:** To be successful, action must be based on the field of pure intelligence, which is located at the source of thought.

4. **Wholeness moving within itself**: In Unity Consciousness, the pure intelligence located in TC is found pervading all of creation, from gross to subtle.