ECE 448/528
Application Software Design

Lecture 3. Java Overview
Spring 2025

Won-Jae Yi, Ph.D.

Department of Electrical and Computer Engineering
Illinois Institute of Technology

Hello World

Java Virtual Machine (JVM)

- Java bytecode: machine code of Java
  - Unlike C/C++ programs which compile codes to x86, ARM, etc.
- Java Virtual Machine (JVM): a program that executes Java bytecode
  - Virtual in a sense that the bytecode needs to be further compiled to x86_64, ARM, etc. before executing it on actual processors.
  - There were attempts to build processors that natively support Java bytecode in order to speed-up Java programs.
  - Innovations within JVM like just-in-time compilation improve performance a lot on commodity processors.
- JVM can be started by the command `java`.
  - Use java –version to check the JVM version and vendor
  - JVM version 1.**8**.xxx supports Java **8** (and earlier)
  - Popular vendors are Oracle and OpenJDK

JRE and JDK

- Java Runtime Environment (JRE): a software package that allows users to execute Java programs.
  - Includes a JVM and the standard Java libraries
- Java Development Kit (JDK): a software package that allows developers to develop Java programs.
  - Includes a JRE and the Java compiler.
- Java compiler can be accessed by the command `javac`, and you can check the version by `javac -version`.

Hello World

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

- Naming conventions
  - Class names should match file names, **C**amel**C**ase
  - Others use camel**C**ase
- Compile: `javac HelloWorld.java`
  - This results in creating the class file, `HelloWorld.class`, containing the Java bytecode.
- Create a jar: `jar -cf first.jar HelloWorld.class`
- Run a jar: `java -cp first.jar HelloWorld`
  - The classpath option -cp allows to include additional jar files
  - e.g., from a third-party to support your program.

What is wrong with HelloWorld?

- Recall Lecture 2: Software is designed in a way so it can be improved and reused!
- It's hard to extend or reuse HelloWorld.
  - Manage and reuse 3rd party libraries.
  - Avoid naming conflicts among source files.
  - Manage tests.
  - Release and deploy Java programs.
- Need better tool and language supports.

A Better Hello World

Gradle

- A build tool to streamline the development process.
- Need to provide a `build.gradle` file so that Gradle knows what to do with our projects.
- Content of `build.gradle`
  - Plugins: language and build process, e.g., Java
  - IDE supports: e.g., idea and eclipse; additional tools, etc.
  - 3rd party libraries
    - Repository locations, e.g., `jcenter, mavenCentral`
    - Library names and versions, e.g., `junit:junit:4.13`
  - Source file locations
  - Other configurations, e.g., for specific plugins.

Standard Directory Layout

- A convention of where source files should be located.
- `src/main/java`: Java source files for application/library
- `src/main/resources`: resources, e.g. default configurations
- `test/main/java`: Java source files for testing
- `test/main/resources`: resources for testing
- When building a jar to release to end users, those under `src`
- will be included and those under `test` will be excluded.

Packages

- A Java feature to organize source files into a meaningful hierarchy
  - Avoiding naming conflicts
  - Controlling access
  - Facilitating search
- Package naming convention
  - Dot-separated names consisting of lowercase letters or numbers, e.g., `ece448.lec03`
  - From the least specific to the most specific, starting with the reverse of your company domain name
  - Use snake_case if necessary (e.g., `ece448_lec03`)
- Requirements for Java source files
  - Java source files belong to a package should be found under the directory where dot is replaced by /. e.g., `ece448/lec03` for the package `ece448.lec03`.
  - Each Java source file should indicate which package it belongs to

HelloWorld in a Package

```
package ece448.lec03;

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

- As `src/main/java/ece448/lec03/HelloWorld.java`
  - Build and include everything into a jar: `gradle shadowJar`
    - This generates `ece448-1.0.0-fat.jar`.
    - The shadowJar task is specified in `build.gradle`.
- Run: `java -cp ece448-1.0.0-fat.jar ece448.lec03.HelloWorld`

Troubleshooting Your Program

- The earlier the better
- Build errors: READ THE ERROR MESSAGE! Always indicates what problem you are facing with your code.
- Testing errors: you may debug your program with a help of a working test case.
  - Set a breakpoint in the IDE and start debugging
  - Wait for the program to hit the breakpoint
  - Use “step into”, “step over”, and “step out” to navigate between lines and method calls
- After deployment: very difficult, mostly rely on logs.
  - Logging is very helpful for testing as well.
  - For a complex program, logs are very useful.
  - e.g., networking, multi-threading, and GUI in our course project

Java Logging

- Logging: print information to indicate progress and compare with your expectations.
  - To determine what to print is tricky and requires a good understanding of your program to be effective.
- Use a logging library instead of `System.out.print`.
  - Control what is being logged via log levels.
  - Add additional information like time and line number.
  - Save logs to files that can be searched.
- The Simple Logging Façade for Java (SLF4J)
  - A 3rd party library to support many kinds of logging in Java
  - A popular choice despite Java has its own logging features.
  - 6 logging levels: **fatal, error, warn, info, debug, trace**

Java Logging

Log Level	Importance
Fatal	One or more key business functionalities are not working and the whole system doesn't fulfill the business functionalities.
Error	One or more functionalities are not working, preventing some functionalities from working correctly.
Warn	Unexpected behavior happened inside the application, but it is continuing its work and the key business features are operating as expected.
Info	An event happened, the event is purely informative and can be ignored during normal operations.
Debug	A log level used for events considered to be useful during software debugging when more granular information is needed.
Trace	A log level describing events showing step by step execution of your code that can be ignored during the standard operation, but may be useful during extended debugging sessions.

Hello World with Logging

```
package ece448.lec03;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;

public class HelloWorld {
  public static void main(String[] args) {
    logger.info("Hello World");
  }

  private static final Logger logger
    = LoggerFactory.getLogger(HelloWorld.class);
}
```

- Use import to refer to classes from other packages.
  - Use the IDE to generate those import statements.
  - Make sure to pick the correct one if there are multiple choices. You may choose again after deleting the old one.
- The log will also be stored to `debug.log` as configured in `src/main/resources/logback.xml`.