ECE4721J 24 SU Lab1

Basic Java

> Author: ECE4721J SU2024 Teaching Group

Target of the lab session:

- Install and setup `java` and `maven` environment
- Have a taste of java programming

Part I: Intro to Java

Java as a programming language

Java is a **compiled (also interpreted)** programming language that is

- High level
- Class based
- Object oriented

The syntax of Java is similar to that of C and C++, but has fewer low-level facilities. Compiled Java applications are run on **Java virtual machine (JVM)** regardless of the underlying computer architecture (cross platform). In particular, **Hadoop** is written in Java, thus knowledge of Java basics is essential to learn Hadoop.

Java development kit

The **Java development kit (JDK)** contains tools we need to use to build a Java application, such as

- Compiler
- **Java runtime environment (JRE)**
- Archiver (jar)
- Other useful tools

Java project

A typical structure of a Java project may look like

```
├── src
    ├── com
        ├── ece472
            ├── Main.java
            └── ...
```

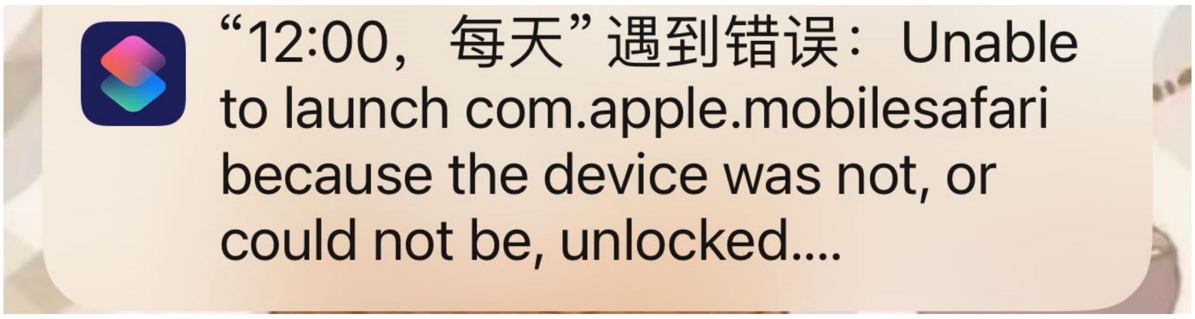
Here, `com.ece472` is the *base package*. In Java, a package is used to group a bunch of related classes.

By convention, the base package is the domain name of your company in reverse

$$\text{company.com} \to \text{com.company}$$

> You don't need to have an actual domain on the internet. It is just a way to create a namespace for classes

Example



"12:00，每天"遇到错误：Unable to launch com.apple.mobilesafari because the device was not, or could not be, unlocked....

The main method

Let's take a look at what's inside `Main.java`.

```java
package com.ve472;

public class Main {
    public static void main(String[] args) {
        // write your code here
    }
}
```

The class name should be the same as the file name (`Main` inside `Main.java`), and the `main` method should be announced as `public static void`.

Naming conventions

- **Classes**: PascalNamingConvention
- **Methods**: camelNamingConvention
- **Packages**: lower.case.letters

Variable Types

- Primitive types (store values):
  - int
  - float
  - Boolean
  - ...
- Reference types are objects. They store references to objects rather than values.

Note: Although `String` is a reference type, Java allows us to declare a string variable *as if* it is primitive. Declare using

```java
  String str = "abcd";
  //rather than
  String str = new String("abcd");
```

Execution

Compile a java source code using `javac`:

```
$ javac Main.java
```

It yields a binary file `Main.class`. Change to the `src` directory and run the program with the full package name plus the class name:

```
$ java com.ece472.Main
```

Exercise I

Build a "Hello world" project in Java. Compile and execute the program with command line.

Notes

To be honest, hamster did not follow the manual to install java, here is where he setup java:

https://bigdl.readthedocs.io/en/latest/doc/Orca/Overview/install.html.

Part II: Maven

What's maven?

Maven is a tool that is used to build and manage any Java-based project. Areas of concerns Maven deals with include

- Making the build process easy
- Providing a uniform build system
- Providing quality project information
- Encouraging better development practices

Standard project structure

```
├── my-app
    ├── pom.xml
    ├── src
        ├── main
        |   ├── java
        |       ├── com
        |           ├── mycompany
        |               ├── app
        |                   ├── App.Java
        ├── test
            ├── java
                ├── com
                    ├── mycompany
                        ├── app
                            ├── AppTest.Java
```

A `target` folder will be generated automatically under `my-app` after the Maven project is built. It contains compiled classes and test-classes.

What's POM?

The **Project Object Model (POM)** is the core of a Maven project.

- Held in a XML file `pom.xml`
- Contains all necessary information about a project and the build process

POM basics

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
        https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>

    <groupId>ece472</groupId>
    <artifactId>ece472l1</artifactId>
    <version>1.0</version>
</project>
```

- **groupId**: Unique amongst an organisation or a project
- **artifactId**: Name of the project
- **version**: Version of the project

POM relationships

```
<project ...>
    ...
    <dependencies>
        <dependency>
            <groupId>commons-cli</groupId>
            <artifactId>commons-cli</artifactId>
            <version>1.4</version>
        </dependency>
    ...
    </dependencies>
</project>
```

A **dependency list** is the cornerstone of POM, since most projects depend on others to build and run correctly.

POM build

```
<project ...>
    ...
    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>2.6</version>
            </plugin>
        </plugins>
    </build>
</project>
```

Build handles things like declaring your project’s directory structure and managing plugins.

Maven repository

By default, the packages are installed on local under the path `${HOME}/.m2/repository/`. You can modify `settings.xml` of Maven to customize configurations. For example, you can setup source mirror and use proxy to download packages:

```
  <mirrors>
    <mirror>
      <id>nexus-aliyun</id>
      <name>Nexus aliyun</name>
      <url>http://maven.aliyun.com/nexus/content/groups/public</url>
      <mirrorOf>central</mirrorOf>
    </mirror>
  </mirrors>

  <proxies>
    <proxy>
      <id>optional</id>
      <active>true</active>
      <protocol>http</protocol>
      <username>proxyuser</username>
      <password>proxypass</password>
      <host>proxy.host.net</host>
      <port>80</port>
      <nonProxyHosts>local.net|some.host.com</nonProxyHosts>
    </proxy>
  </proxies>
```

Execution

Compile a Maven project using

```
$ mvn compile
```

If your project compiled successfully, you may see from the terminal:

```
[INFO] Scanning for projects...
[INFO]
[INFO] ---------------------------< ve472:ve472l1 >----------------------------
[INFO] Building ve472l1 1.0-SNAPSHOT
[INFO] --------------------------------[ jar ]---------------------------------
.......
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  0.900 s
[INFO] Finished at: 2024-05-17T14:48:10+08:00
[INFO] ------------------------------------------------------------------------
```

The output (class binary executables), by default, will be dumped into `${basedir}/target`.

After compilation, run a program using

```
$ mvn exec:java -D exec.mainClass=<your main class> -D exec.args="arg1 arg2"
```

Exercise II

Use the `pom.xml` given in the lab manual to build a Maven project. The main method takes a command line argument followed by `-i`, and prints the argument to stdout directly.

E.g.

```
$ mvn exec:java -D exec.mainClass=com.ve472.l1.Main -D exec.args="-i hello"
```

Output:

```
hello
```

An IDE (e.g, IDEA, VSCode) simplifies the build process, and it could be very helpful.

Notes

To be honest, hamster did not follow the manual to install maven, here is where he downloaded maven:

https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/. Then he updated `.bashrc`:

```
# not quite sure whether this version still works :/
export PATH=/home/<user>/apache-maven-3.6.3/bin:$PATH
```

Lab Submissions

Please implement the `Cinema` as described in the lab manual and submit the code on JOJ. You do not need to hand in a report for this lab. **Lab 1 is an individual lab**.

Join JOJ: http://joj.sjtu.edu.cn/d/ece472_24su/join?code=ece472su24

Reference

[1] Yuxuan Zheng. Lab 1: Basic Java, ECE472 Summer 2023.