Appendix 3: Setting up your Java Project

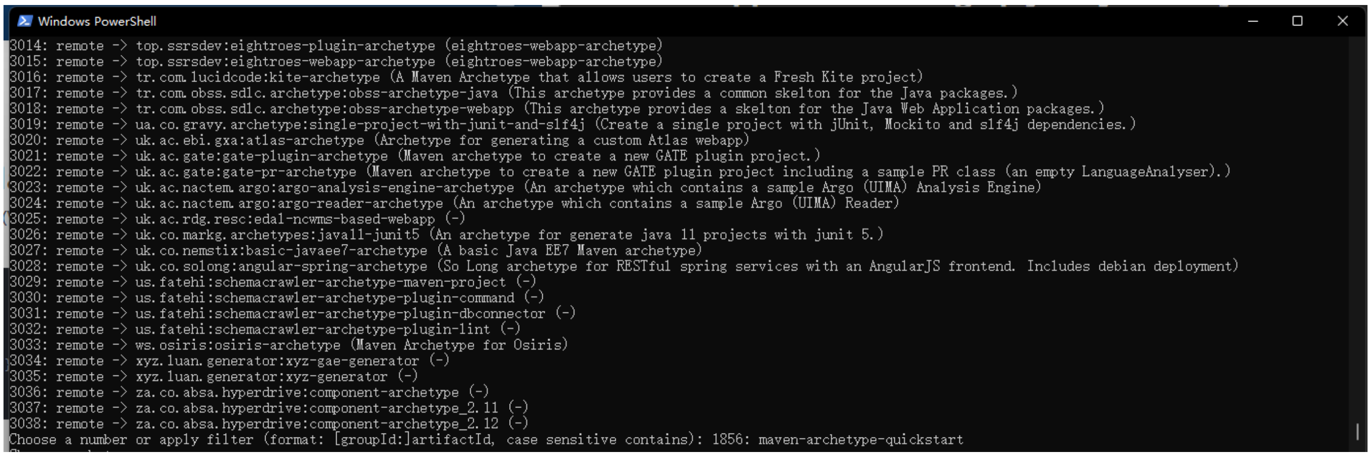
Creating a Project from Maven Template

Using Terminal Commands

In a terminal (linux or Mac) or command prompt (Windows), navigate to the folder you want to create the Java project. Type this command :

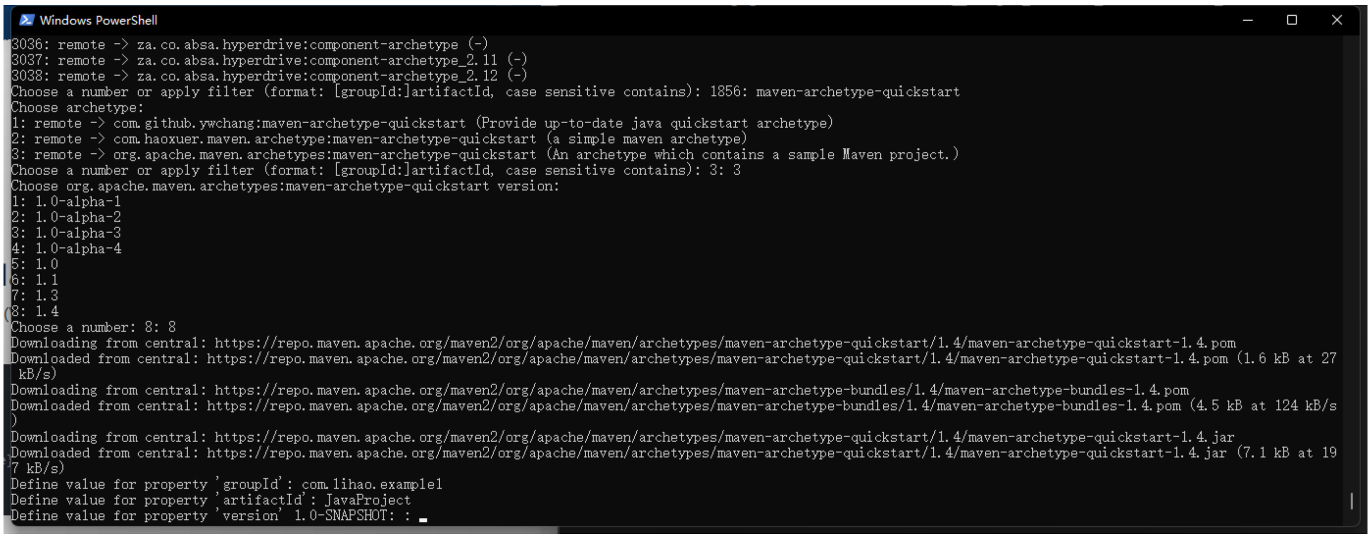
```
mvn archetype:generate
```

Then in the terminal, specify a maven template. For example: `maven-archetype-quickstart`



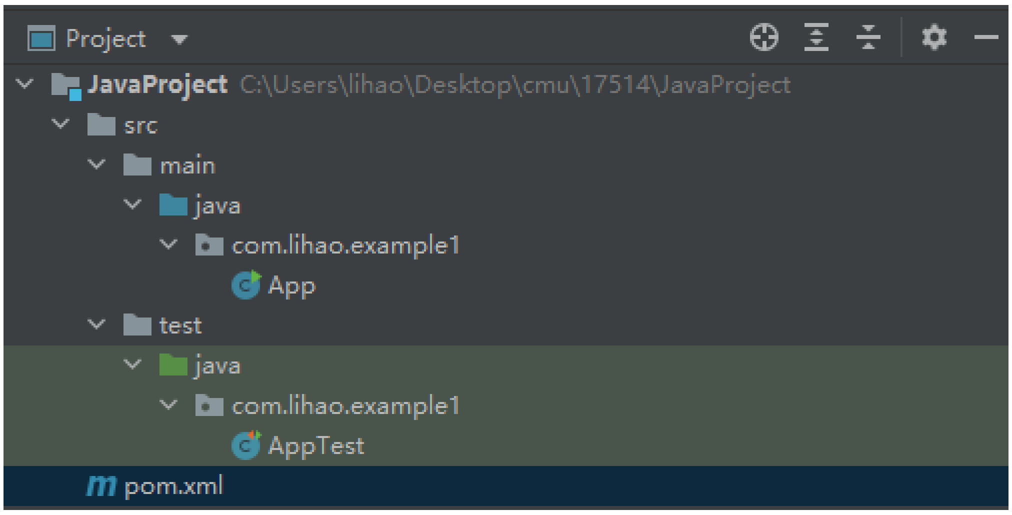
```
3014: remote -> top.ssrsdev:eightroes-plugin-archetype (eightroes-webapp-archetype)
3015: remote -> top.ssrsdev:eightroes-webapp-archetype (eightroes-webapp-archetype)
3016: remote -> tr.com.lucidcode:kite-archetype (A Maven Archetype that allows users to create a Fresh Kite project)
3017: remote -> tr.com.obss.sdlc.archetype:obss-archetype-java (This archetype provides a common skelton for the Java packages.)
3018: remote -> tr.com.obss.sdlc.archetype:obss-archetype-webapp (This archetype provides a skelton for the Java Web Application packages.)
3019: remote -> ua.co.gravy.archetype:single-project-with-junit-and-slf4j (Create a single project with jUnit, Mockito and slf4j dependencies.)
3020: remote -> uk.ac.ebi.gxa:atlas-archetype (Archetype for generating a custom Atlas webapp)
3021: remote -> uk.ac.gate:gate-plugin-archetype (Maven archetype to create a new GATE plugin project.)
3022: remote -> uk.ac.gate:gate-pr-archetype (Maven archetype to create a new GATE plugin project including a sample PR class (an empty LanguageAnalyser).)
3023: remote -> uk.ac.nactem.argo:argo-analysis-engine-archetype (An archetype which contains a sample Argo (UIMA) Analysis Engine)
3024: remote -> uk.ac.nactem.argo:argo-reader-archetype (An archetype which contains a sample Argo (UIMA) Reader)
3025: remote -> uk.ac.rdg.resc:edal-ncwms-based-webapp (-)
3026: remote -> uk.co.markg.archetypes:java11-junit5 (An archetype for generate java 11 projects with junit 5.)
3027: remote -> uk.co.nemstix:basic-javaee7-archetype (A basic Java EE7 Maven archetype)
3028: remote -> uk.co.solong:angular-spring-archetype (So Long archetype for RESTful spring services with an AngularJS frontend. Includes debian deployment)
3029: remote -> us.fatehi:schemacrawler-archetype-maven-project (-)
3030: remote -> us.fatehi:schemacrawler-archetype-plugin-command (-)
3031: remote -> us.fatehi:schemacrawler-archetype-plugin-dbconnector (-)
3032: remote -> us.fatehi:schemacrawler-archetype-plugin-lint (-)
3033: remote -> ws.osiris:osiris-archetype (Maven Archetype for Osiris)
3034: remote -> xyz.luan.generator:xyz-gae-generator (-)
3035: remote -> xyz.luan.generator:xyz-generator (-)
3036: remote -> za.co.absa.hyperdrive:component-archetype (-)
3037: remote -> za.co.absa.hyperdrive:component-archetype_2.11 (-)
3038: remote -> za.co.absa.hyperdrive:component-archetype_2.12 (-)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 1856: maven-archetype-quickstart
```

Specify project packaging and project name.



```
3036: remote -> za.co.absa.hyperdrive:component-archetype (-)
3037: remote -> za.co.absa.hyperdrive:component-archetype_2.11 (-)
3038: remote -> za.co.absa.hyperdrive:component-archetype_2.12 (-)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 1856: maven-archetype-quickstart
Choose archetype:
1: remote -> com.github.ywchang:maven-archetype-quickstart (Provide up-to-date java quickstart archetype)
2: remote -> com.haoxuer.maven.archetype:maven-archetype-quickstart (a simple maven archetype)
3: remote -> org.apache.maven.archetypes:maven-archetype-quickstart (An archetype which contains a sample Maven project.)
Choose a number or apply filter (format: [groupId:]artifactId, case sensitive contains): 3: 3
Choose org.apache.maven.archetypes:maven-archetype-quickstart version:
1: 1.0-alpha-1
2: 1.0-alpha-2
3: 1.0-alpha-3
4: 1.0-alpha-4
5: 1.0
6: 1.1
7: 1.3
8: 1.4
Choose a number: 8: 8
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.pom (1.6 kB at 27
 kB/s)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-bundles/1.4/maven-archetype-bundles-1.4.pom
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-bundles/1.4/maven-archetype-bundles-1.4.pom (4.5 kB at 124 kB/s
)
Downloading from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/archetypes/maven-archetype-quickstart/1.4/maven-archetype-quickstart-1.4.jar (7.1 kB at 19
7 kB/s)
Define value for property 'groupId': com.lihao.example1
Define value for property 'artifactId': JavaProject
Define value for property 'version' 1.0-SNAPSHOT: :
```

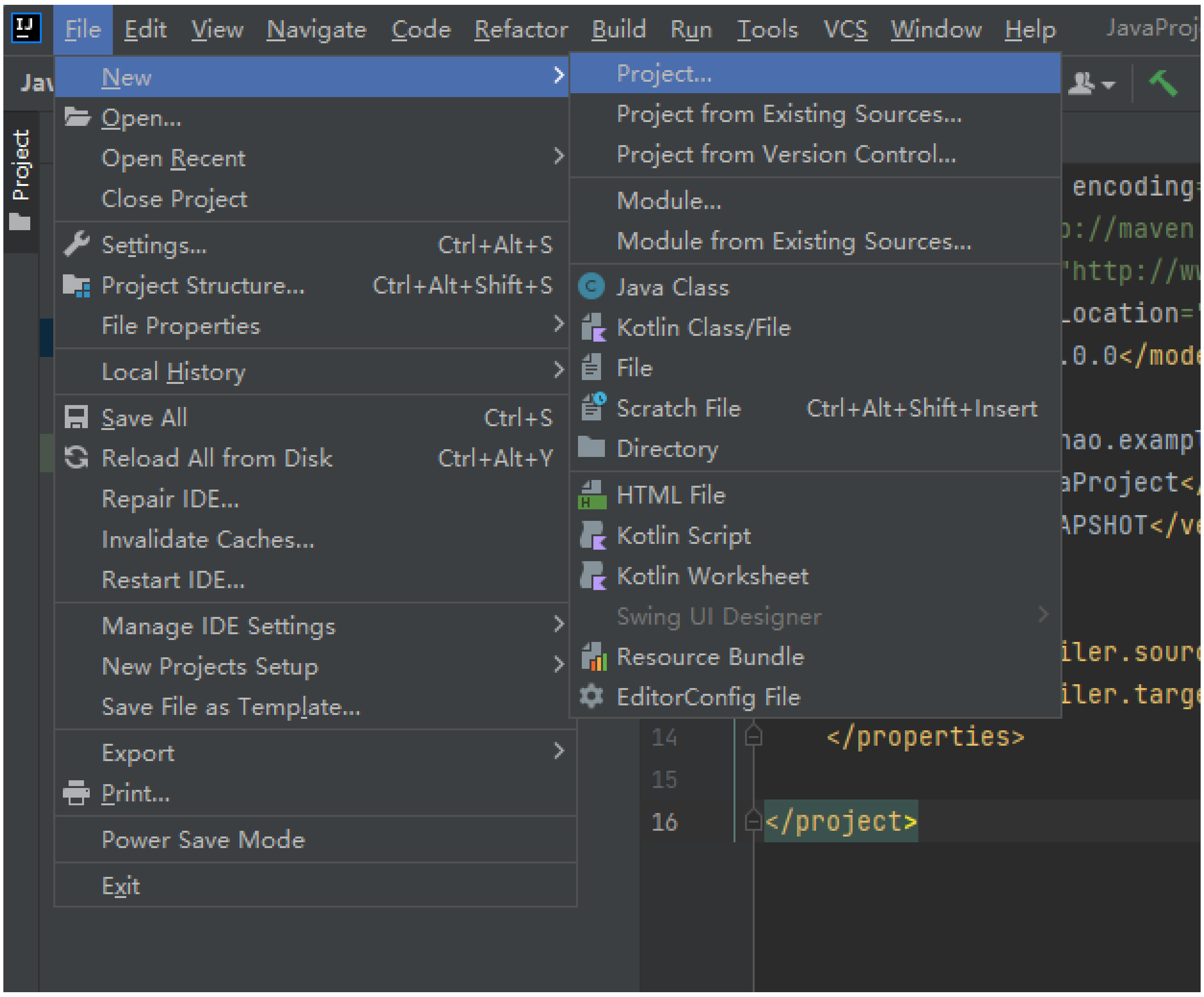
Above command will generate a Java project from `maven-archetype-quickstart` template. It looks like below:



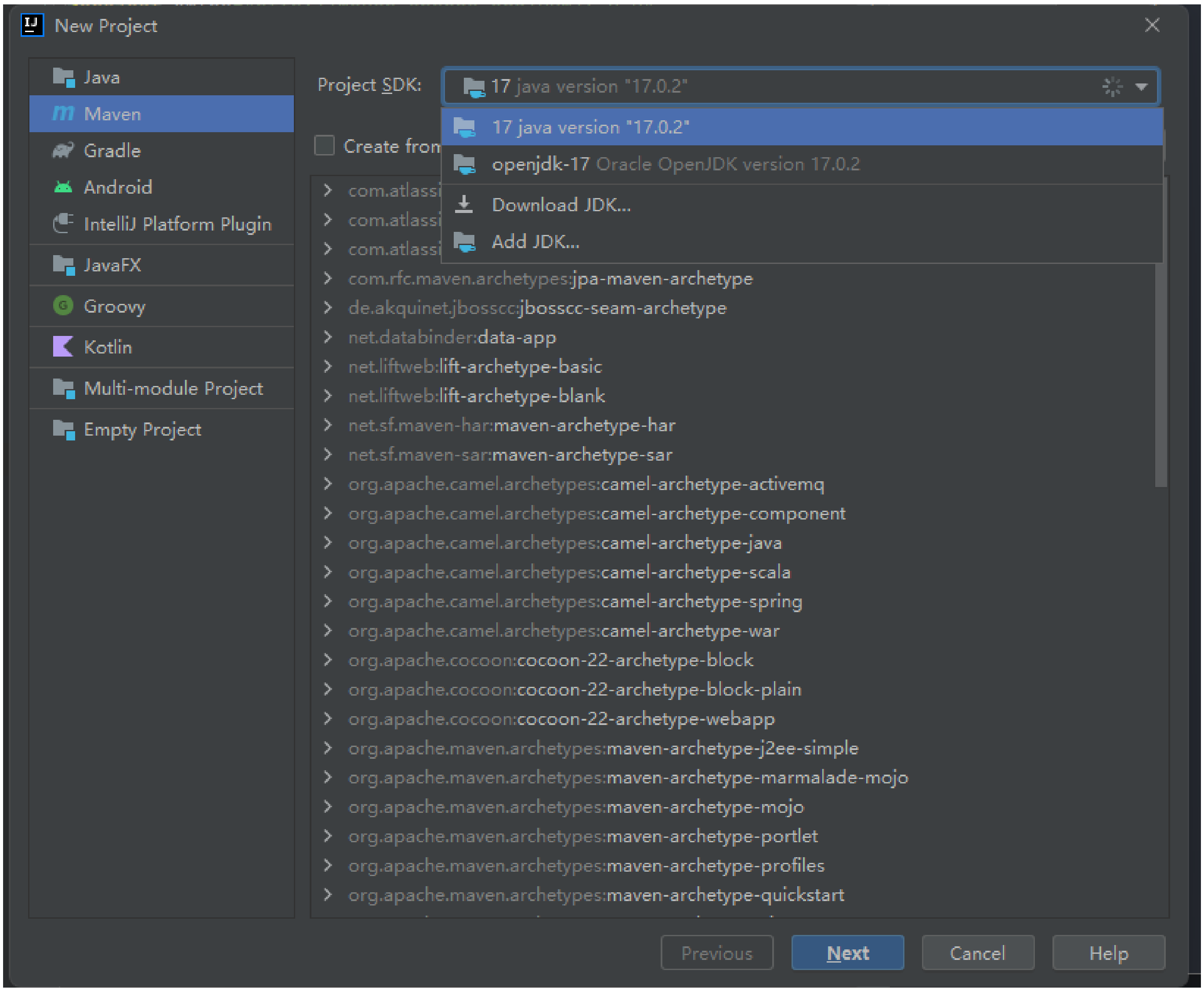
```
Project
JavaProject C:\Users\lihao\Desktop\cmu\17514\JavaProject
  src
    main
      java
        com.lihao.example1
          App
    test
      java
        com.lihao.example1
          AppTest
  pom.xml
```

Using IntelliJ

You can also create a project using IntelliJ. Open your IntelliJ, select File -> New -> Project...



Select Maven from the options on the left, and choose the JDK for the project.



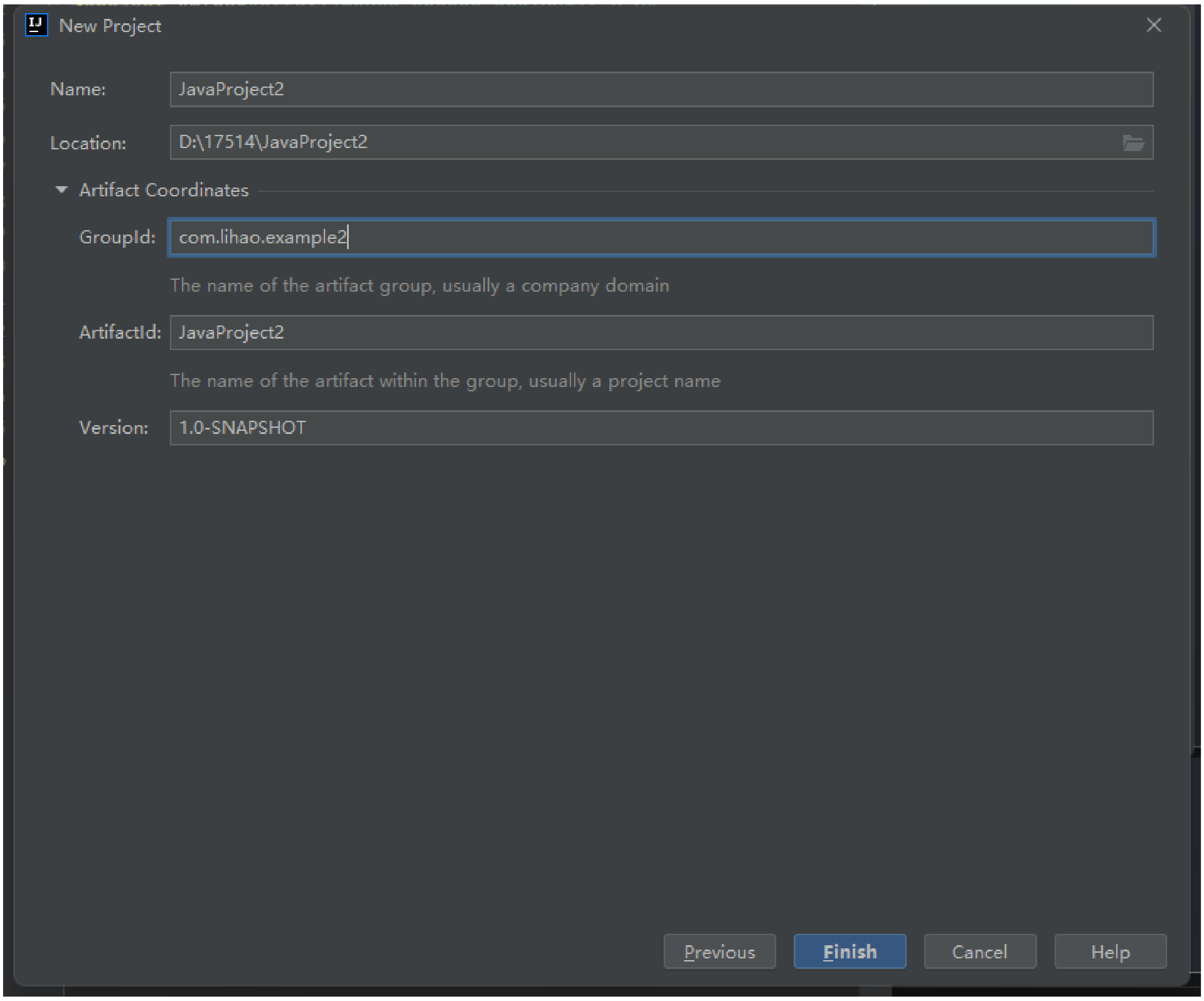
Project SDK: 17 java version "17.0.2"

- 17 java version "17.0.2"
- openjdk-17 Oracle OpenJDK version 17.0.2

```
com.atlassi
com.atlassi
com.atlassi
com.rfc.maven.archetypes:jpa-maven-archetype
de.akquinet.jbosscc:jbosscc-seam-archetype
net.databinder:data-app
net.liftweb:lift-archetype-basic
net.liftweb:lift-archetype-blank
net.sf.maven-har:maven-archetype-har
net.sf.maven-sar:maven-archetype-sar
org.apache.camel.archetypes:camel-archetype-activemq
org.apache.camel.archetypes:camel-archetype-component
org.apache.camel.archetypes:camel-archetype-java
org.apache.camel.archetypes:camel-archetype-scala
org.apache.camel.archetypes:camel-archetype-spring
org.apache.camel.archetypes:camel-archetype-war
org.apache.cocoon:cocoon-22-archetype-block
org.apache.cocoon:cocoon-22-archetype-block-plain
org.apache.cocoon:cocoon-22-archetype-webapp
org.apache.maven.archetypes:maven-archetype-j2ee-simple
org.apache.maven.archetypes:maven-archetype-marmalade-mojo
org.apache.maven.archetypes:maven-archetype-mojo
org.apache.maven.archetypes:maven-archetype-portlet
org.apache.maven.archetypes:maven-archetype-profiles
org.apache.maven.archetypes:maven-archetype-quickstart
```

Press Next.

You need to give the project a name, groupId, artifactId, and you can optionally set the location.


New Project

Name: JavaProject2

Location: D:\17514\JavaProject2

Artifact Coordinates

GroupId: com.lihao.example2

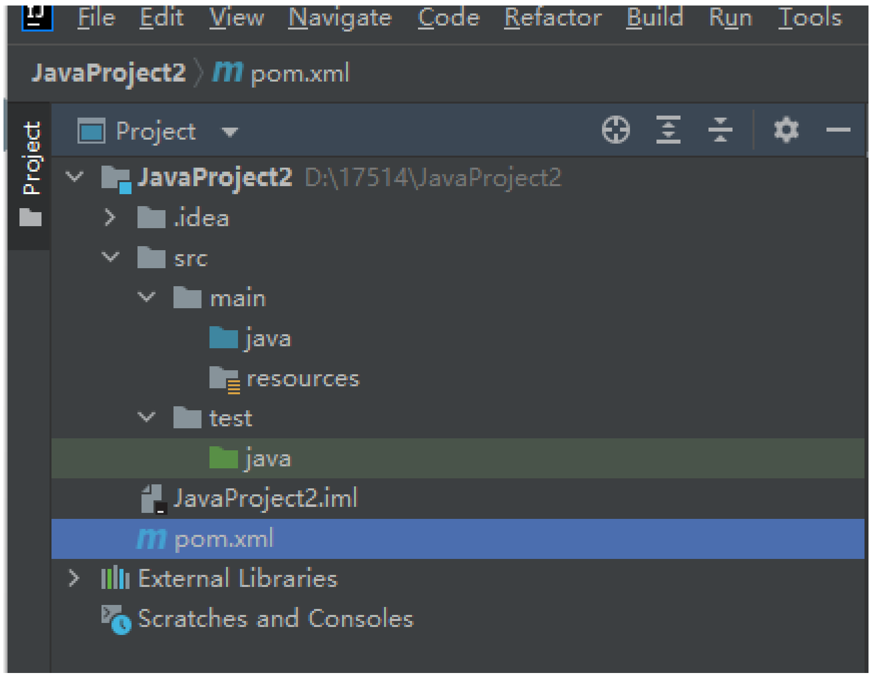
The name of the artifact group, usually a company domain

ArtifactId: JavaProject2

The name of the artifact within the group, usually a project name

Version: 1.0-SNAPSHOT

Press Finish, and IntelliJ IDEA will generate the Maven pom.xml for you.



```
JavaProject2  D:\17514\JavaProject2
  .idea
  src
    main
      java
      resources
    test
      java
  JavaProject2.iml
  pom.xml
External Libraries
Scratches and Consoles
```

Maven Command Cheat Sheet

- **validate**: validate the project is correct and all necessary information is available
- **compile**: compile the source code of the project
- **test**: test the compiled source code using a suitable unit testing framework. These tests should not require the code be packaged or deployed
- **package**: take the compiled code and package it in its distributable format, such as a JAR.
- **integration-test**: process and deploy the package if necessary into an environment where integration tests can be run
- **verify**: run any checks to verify the package is valid and meets quality criteria
- **install**: install the package into the local repository, for use as a dependency in other projects locally
- **deploy**: done in an integration or release environment, copies the final package to the remote repository for sharing with other developers and projects.
- **clean**: cleans up artifacts created by prior builds
- **site**: generates site documentation for this project

pom.xml

> A Project Object Model or POM is the fundamental unit of work in Maven. It is an XML file that contains information about the project and configuration details used by Maven to build the project.
>
> When executing a task or goal, Maven looks for the POM in the current directory. It reads the POM, gets the needed configuration information, then executes the goal.
>
> Some of the configuration that can be specified in the POM are the project dependencies, the plugins or goals that can be executed, the build profiles, and so on. Other information such as the project version, description, developers, mailing lists and such can also be specified.

The minimum requirement for a POM are the following:

- `project` root
- `modelVersion` - should be set to 4.0.0

- `groupId` - the id of the project's group.
- `artifactId` - the id of the project
- `version` - the version of the artifact under the specified group

```
<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
http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>

  <groupId>com.lihao.example1</groupId>
  <artifactId>JavaProject</artifactId>
  <version>1.0-SNAPSHOT</version>
</project>
```

Based on the minimum POM, you can expand your configuration file by adding new elements, such as `<build>`, `<dependencies>` and `<reporting>`.

Check the documentation for their usages: Maven – Introduction to the POM

checkstyle.xml

The Checkstyle Plugin generates a report regarding the code style used by the developers. It needs a xml file containing predefined rulesets. Predefined rulesets available for use are: sun_checks.xml and google_checks.xml. You can also copy the `checkstyle.xml` file from previous homework.

The plugin can be configured in the project's POM. Example configuration looks like below:

```xml
<build>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-site-plugin</artifactId>
            <version>3.7.1</version>
        </plugin>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-project-info-reports-plugin</artifactId>
            <version>3.1.2</version>
        </plugin>
    </plugins>
    <pluginManagement>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-checkstyle-plugin</artifactId>
                <version>3.1.2</version>
                <configuration>
                    <failsOnError>true</failsOnError>
                    <consoleOutput>true</consoleOutput>
                </configuration>
                <executions>
                    <execution>
                        <id>validate</id>
                        <phase>validate</phase>
                        <goals>
                            <goal>check</goal>
                        </goals>
                    </execution>
                </executions>
                <dependencies>
                    <dependency>
                        <groupId>com.puppycrawl.tools</groupId>
                        <artifactId>checkstyle</artifactId>
                        <version>8.45</version>
                    </dependency>
                </dependencies>
            </plugin>
        </plugins>
    </pluginManagement>
</build>

<reporting>
    <plugins>
        <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-checkstyle-plugin</artifactId>
            <version>3.1.2</version>
```

```
                <configuration>
                    <configLocation>"path-to-your-checkstyle-file"/"checkstyle-
name".xml</configLocation>
                </configuration>
            </plugin>
            <!-- a plugin to include test results in the `mvn site` report -->
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-surefire-report-plugin</artifactId>
                <version>2.22.2</version>
            </plugin>
        </plugins>
    </reporting>
```

.gitignore

Git sees every file in your working copy as one of three things:

- tracked - a file which has been previously staged or committed;
- untracked - a file which *has not* been staged or committed; or
- ignored - a file which Git has been explicitly told to ignore.

Ignored files are usually build artifacts and machine generated files that can be derived from your repository source or should otherwise not be committed. Some common examples are:

- dependency caches, such as the contents of `/node_modules` or `/packages`
- compiled code, such as `.o`, `.pyc`, and `.class` files
- build output directories, such as `/bin`, `/out`, or `/target`
- files generated at runtime, such as `.log`, `.lock`, or `.tmp`
- hidden system files, such as `.DS_Store` or `Thumbs.db`
- personal IDE config files, such as `.idea/workspace.xml`

In order to avoid committing unnecessary files, you should create a `.gitignore` file in the root directory of your repo. Example `.gitignore` file looks like below:

```
# Compiled class file
*.class

# Log file
*.log

# Package Files #
*.jar
*.war
*.nar
*.ear
*.zip
*.tar.gz
*.rar

# virtual machine crash logs, see
http://www.java.com/en/download/help/error_hotspot.xml
hs_err_pid*

target/
.idea/

*.DS_Store
```

Quickstart for GitHub Actions

Creating Your workflow

- Create a `.github/workflows` directory in your repository on GitHub if this directory does not already exist.
- In the `.github/workflows` directory, create a file named `demo.yml`.
- Copy the following YAML contents into the `demo.yml` file:

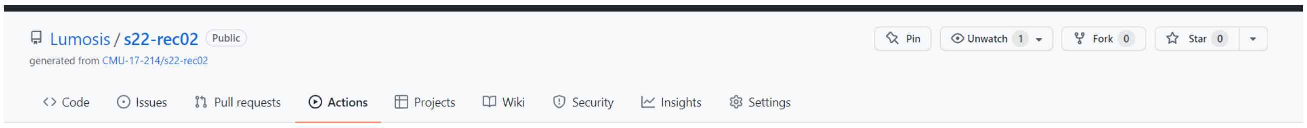
```yaml
name: Demo
on: [push]
jobs:
  Explore-GitHub-Actions:
    runs-on: ubuntu-latest
    steps:
      - run: echo "🎉 The job was automatically triggered by a ${{ github.event_name }} event."
      - run: echo "🐧 This job is now running on a ${{ runner.os }} server hosted by GitHub!"
      - run: echo "🔎 The name of your branch is ${{ github.ref }} and your repository is ${{ github.repository }}."
      - name: Check out repository code
        uses: actions/checkout@v2
      - run: echo "💡 The ${{ github.repository }} repository has been cloned to the runner."
      - run: echo "🖥️ The workflow is now ready to test your code on the runner."
      - name: List files in the repository
        run: |
          ls ${{ github.workspace }}
      - run: echo "🍏 This job's status is ${{ job.status }}."
```

For your project, please follow the documentation to learn and customize your own actions. You can also start with reviewing and modifying the `.github/workflows/main.yml` file in your previous homework repo.

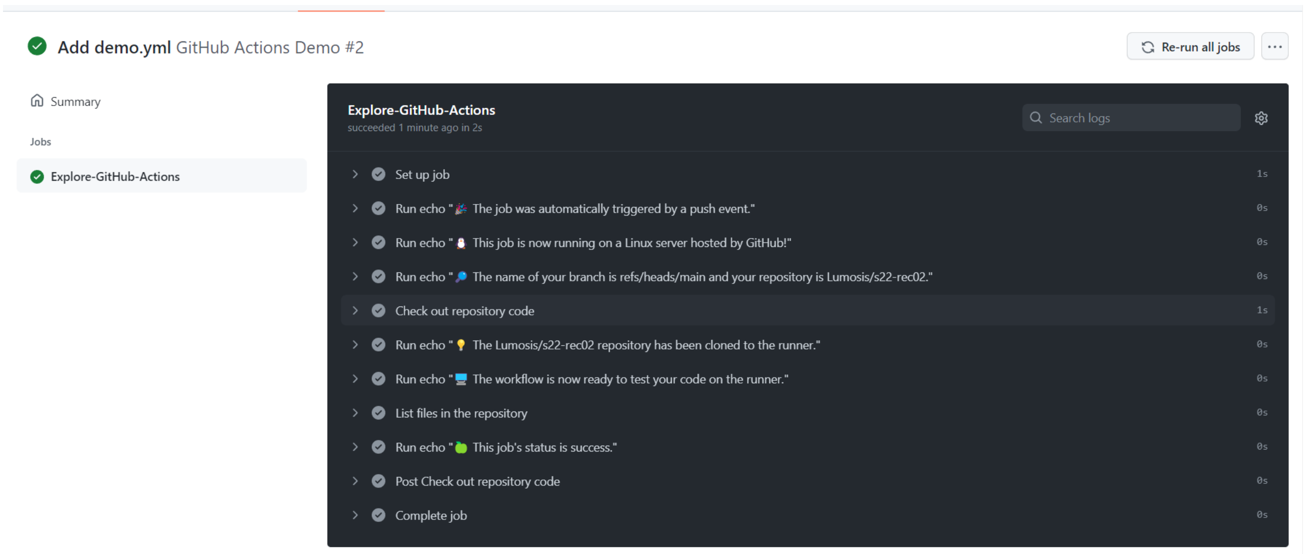
- Commit and push this file.

Viewing Your Workflow Results

- On GitHub.com, navigate to the main page of the repository.
- Under your repository name, click **Actions**.



- In the left sidebar, click the workflow you want to see.



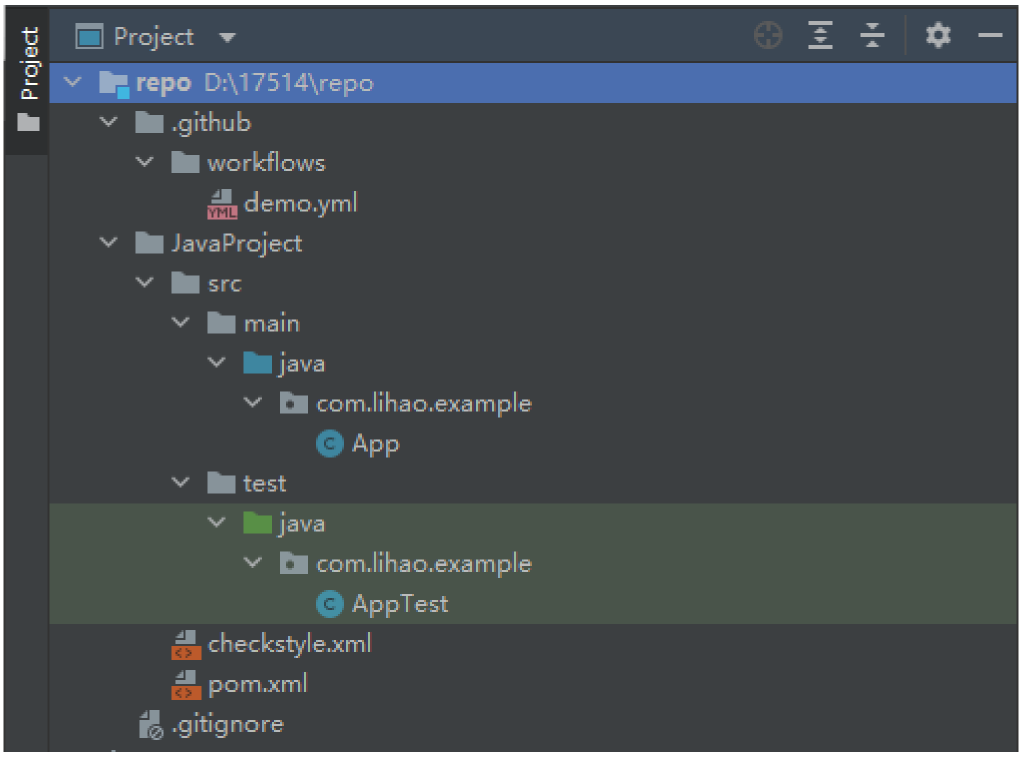
```
Add demo.yml GitHub Actions Demo #2

Explore-GitHub-Actions
succeeded 1 minute ago in 2s

> Set up job                                                                                          1s
> Run echo "🎉 The job was automatically triggered by a push event."                                  0s
> Run echo "🐧 This job is now running on a Linux server hosted by GitHub!"                           0s
> Run echo "🔎 The name of your branch is refs/heads/main and your repository is Lumosis/s22-rec02."  0s
> Check out repository code                                                                           1s
> Run echo "💡 The Lumosis/s22-rec02 repository has been cloned to the runner."                       0s
> Run echo "🖥️ The workflow is now ready to test your code on the runner."                            0s
> List files in the repository                                                                        0s
> Run echo "🍏 This job's status is success."                                                         0s
> Post Check out repository code                                                                      0s
> Complete job                                                                                        0s
```

Summary

After you set up everything above, you will end up with a repo like this:



```
repo D:\17514\repo
  .github
    workflows
      demo.yml
  JavaProject
    src
      main
        java
          com.lihao.example
            App
      test
        java
          com.lihao.example
            AppTest
    checkstyle.xml
    pom.xml
  .gitignore
```

You can begin your adventure from here. Good luck, have fun!

Reference

- Maven - How to create a Java project - Mkyong.com
- Maven Archetype Plugin – archetype:generate

- Maven – Introduction to the POM
- Maven – Maven in 5 Minutes
- .gitignore file - ignoring files in Git | Atlassian Git Tutorial
- Quickstart for GitHub Actions - GitHub Docs