Tai-e Manual for Assignments

Course “Static Program Analysis” @Nanjing University
Assignments Designed by Tian Tan and Yue Li

1 Introduction

This manual describes how to setup Tai-e to finish the assignments in our course. Tai-e is an easy-to-learn static program analysis framework for Java developed by the two instructors of this course.

Tai-e leverages Soot to parse Java programs and help build Tai-e’s IR. Soot contains two frontends, one for parsing Java source files (.java) and the other one for bytecode files (.class), and the former can preserve variable names (in the source code) in IR, which makes IR closer to the source code and thus is easier to understand than the latter. As a result, **in the assignments**, we provide test cases (i.e., the input programs to be analyzed) in the format of .java files. However, Soot’s frontend for Java source files is outdated (only partially supports Java version up to 7) and not very robust. Meanwhile, Soot’s frontend for bytecode files, which does not keep variable names though, is more robust (it virtually works well for all .class files compiled, at least, by Java 11) than the frontend for Java source files. Thus, **for real-world applications**, Tai-e usually analyzes bytecode.

2 Content of Assignment Package

Tai-e is built by Gradle and follows typical structure of Gradle projects. All assignment packages are organized as the same structure as follows:

- `build.gradle`: The Gradle build script for Tai-e.
- `gradle/`: The folder containing Gradle wrapper files.
- `src/main/java`: The folder containing source code of Tai-e. **You will need to modify the files in this folder to finish the assignments**.
- `src/test/java`: The folder containing the drivers for running test cases.
- `src/test/resources`: The folder containing test cases (i.e., the input Java programs to be analyzed).
- `lib/`: The folder containing classes needed by Tai-e.
- `plan.yml`: The configuration file for Tai-e that specifies the analyses to be executed in the assignments.
- `copyright.txt`: The copyright of Tai-e.

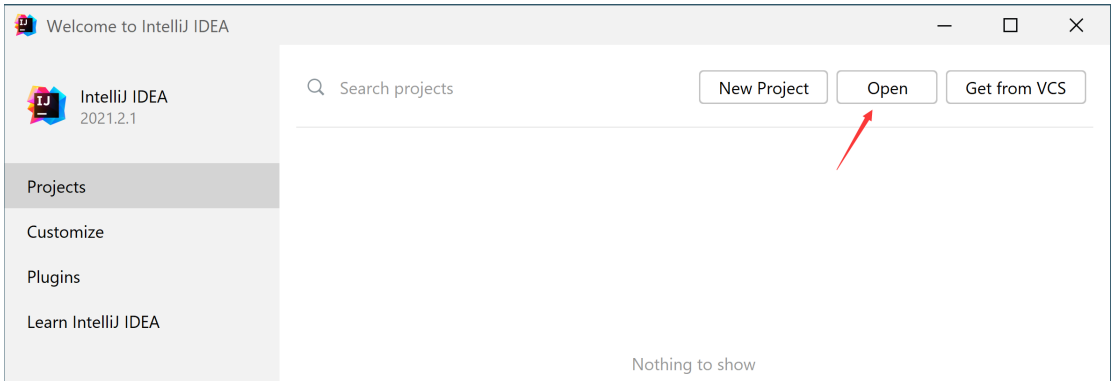
3 Setup Instructions

Tai-e is developed in Java, and it could run on major operating systems including Windows, MacOS, and Linux (Ubuntu). To build and run Tai-e, you need to have Java **11** installed on your system. You could download the Java Development Kit 11 from the following link:
https://www.oracle.com/java/technologies/javase-jdk11-downloads.html

We highly suggest you finish our assignments with IntelliJ IDEA. Given the Gradle build script, it is very easy to import Tai-e to IntelliJ IDEA, as follows.

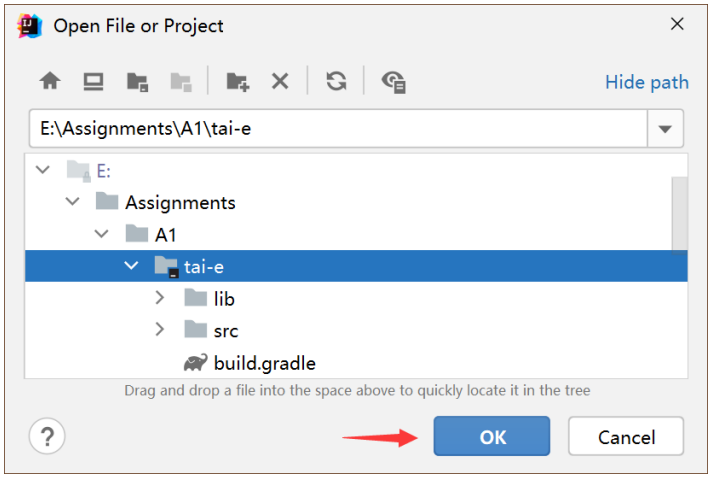
Step 1
Download IntelliJ IDEA from JetBrains (http://www.jetbrains.com/idea/download/) and install it.

Step 2
Start to open a project



(Note: if you have already used IntelliJ IDEA, and opened some projects, then you could choose File > Open… to open the same dialog for the next step.)

Step 3
Select the `tai-e/` directory, then click "OK".



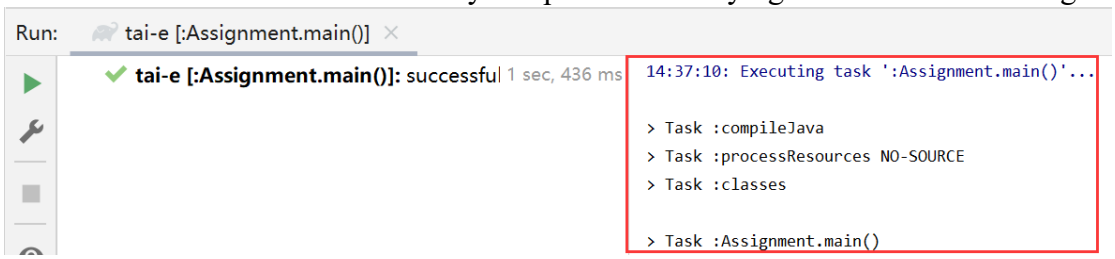
Step 4

IntelliJ IDEA may pop up a dialog asking if you trust the Gradle project. Just click "Trust Project" (Don't worry. Tai-e is benign :-)).

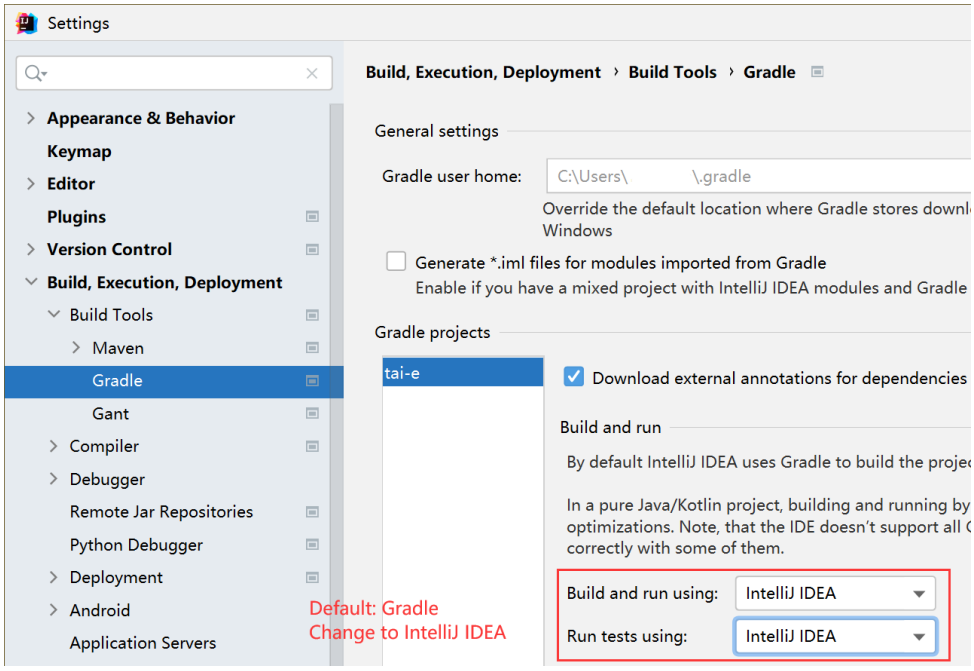
That's it! You may wait a moment for importing Tai-e. After that, some Gradle-related files/folders will be generated in `tai-e/` directory, and you can ignore them.

Step 5 (optional)

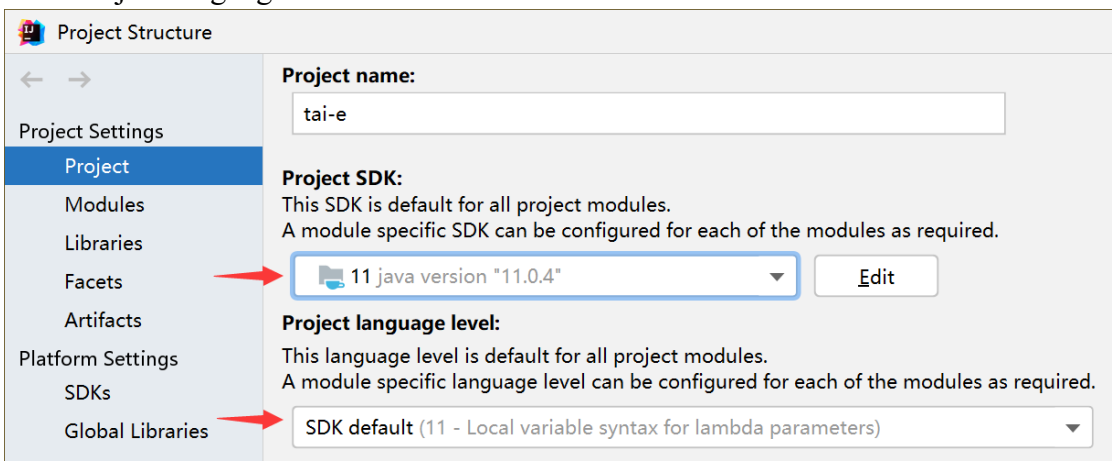
As Tai-e is a Gradle project, IntelliJ IDEA always build and run it with Gradle by default, which makes it a bit slower and always output some annoying Gradle-related messages:



To get rid of these problems, you could use IntelliJ IDEA instead of Gradle to build and run Tai-e. Just go to File > Settings, and change the *build and run* tool from Gradle to IntelliJ IDEA as shown:



Notice: If your system has multiple JDKs, make sure that IntelliJ IDEA uses Java 11. To configure this, go to File > Project Structure…, and select **11** for "Project SDK" and "Project language level":



Alternatively, if you (really :-)) want to build Tai-e from command line, you could change working directory to `tai-e/` folder, and build it with Gradle:

```
$ gradle compileJava
```

4 Run Tai-e as An Application

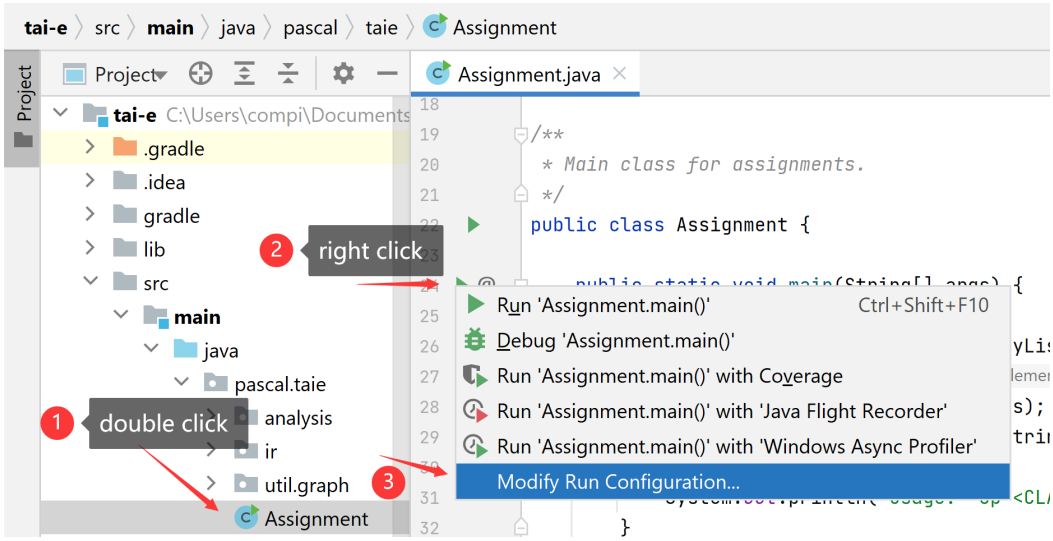
We provide a special main class of Tai-e for our assignments:

```
pascal.taie.Assignment
```

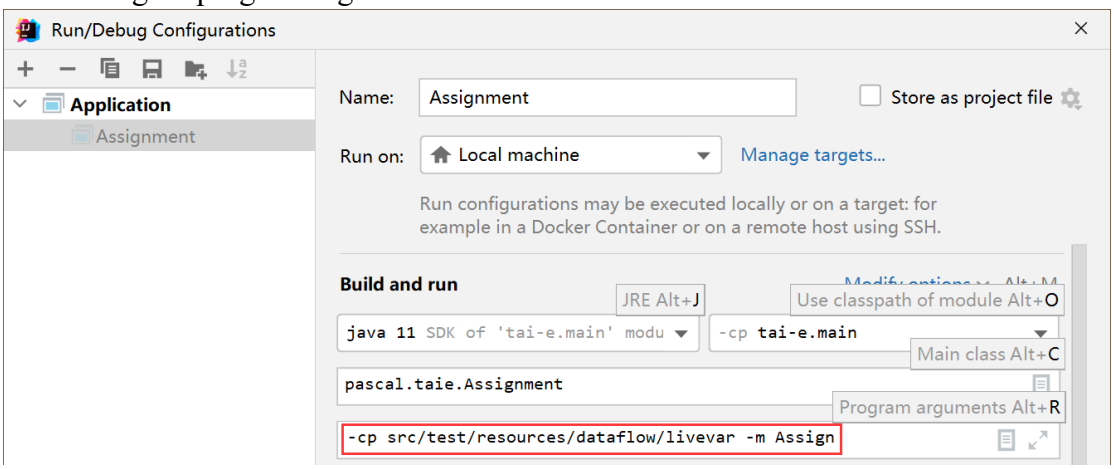
which offers a simple usage as follows:

```
-cp <CLASS_PATH> -m <CLASS_NAME>
```

where `<CLASS_PATH>` is the class path, and `<CLASS_NAME>` is the name of the input main class to be analyzed. Tai-e locates classes from given class path. For example, to analyze `Assign.java` in directory `src/test/resources/dataflow/livevar`, first open "Run Configuration" for `Assignment` in IntelliJ IDEA as follows:



then configure program arguments as follows:



Tai-e performs the analysis for the input program and outputs the analysis results. The analyses and their outputs vary for different assignments, and we will explain the details in the document of each assignment.

Of course, you could also run the analysis using Gradle, with the following command:

```
$ gradle run --args="-cp <CLASS_PATH> -m <CLASS_NAME>"
```

5 Test Your Assignments with JUnit

To make testing convenient, we have prepared some Java classes as test inputs in folder `src/test/resources/`. Every class has an associated file named `*-expected.txt`, which contains the expected results of the analysis. You could analyze these test inputs by running test class (powered by JUnit) in `src/test/java/`. Different assignments contain different test cases and test case drivers, and we will explain their details in each assignment document.

The test case driver analyzes all provided test cases in `src/test/resources/`, and compares the given analysis results to the expected results. If your implementation is correct, the tests will pass; otherwise it may fail and output the differences between expected and your results.

Again, you could run tests with Gradle, just type:

```
$ gradle clean test
```

This command will clean the build directory, rebuild Tai-e, and run tests.