Java Project Analyser

Course Project for Java EE, Tongji University

Zhang Yao
zhang@cinea.cc

Tongji University

Table of Contents

Overview 3
- Team Members 3
- Description 3
- Features 3

Usage 3
- Requirement 3
- Compile our project 3
- Simple usage 4
- Open a project 4
- Querying method invocation relationships 4
- Querying method parameter relationships 4
- Automaticly fixing syntax errors 4

Design 5
- System architecture and components 5
- General workflow 5

Implementation 5
- Major data structures 5
- Efficiency considerations 5
- Key techniques 5

Test 5
- Test Cases 5

Conclusion 6

Overview

Team Members

Name	Matr. No.	Contact number	Email address
Zhang Yao	2152955	+86 13518772062	zhang@cinea.cc
Tang Shuhan			
Wang Shuyu	2151894	+86 17709584390	1491456253@qq.com
Tang Kexian			

No specific order in the table.

Description

Features

Usage

Requirement

A Java environment that is **greater than or equal to 17 is required**. We use **Amazon Corretto 17 LTS** in our development and test. But other JDK distribution is also compatiable. Learn more about Amazon Corretto: https://aws.amazon.com/corretto

Compile our project

We use **Maven** to manage our project and its dependencies. A **Maven Wrapper** instance is also included in our project repository. You can just using it to compile our project in most cases. Learn more about Maven Wrapper: https://maven.apache.org/wrapper/

We suggest checking your JDK version before compiling our project:

```
${JAVA_HOME}/bin/java -version    # Never continue if not 17 or higher.
```

To start the compilation process, please execute the following command:

```
./mvnw -B dependency:go-offline
./mvnw -B package
```

The first command will try to download all dependencies our project used from the Maven Central Repository. You may use Aliyun Maven Central Repository in China to accelerate the downloading process. Learn more about Aliyun Maven Central Repository: https://developer.aliyun.com/mvn/guide.

After finishing the compilation, you may find the artifact in `target` directory, which looks like `java-project-analyser-cli-0.0.1-SNAPSHOT.jar`. You may move the output to

a convenient location and change its name as needed. Assuming we have renamed it to `analyser.jar` in convience in the following parts.

Simple usage

Just use `java -jar` command to launch our project and use it:

```
java -jar ./analyser.jar
```

Our project provides a REPL user interface, in which you can easily input commands, obtain results, and reuse the previous state instead of having to restart the program after each command execution. Our project utilizes the **Spring Shell** framework to provide you with an exceptional interactive command-line experience. Learn more about Spring Shell: https://spring.io/projects/spring-shell.

Enter `help` to get a list of functionalities provided by our project.

```
shell:>help
```

Open a project

> **Important**
>
> Our project does not support libraries that generate and process code at compile-time, such as **Lombok**.
>
> If you want to analyse a project which uses Lombok, consider **Delombok** it manually before analysing. Learn more about the Delombok feature: https://projectlombok.org/features/delombok

Use open command to open, load and analyse a Java project:

```
shell:>open /usr/src/sample-project/src/main/java
```

Tips:

- The path should contain the root package, but not the root package self.
- On Windows, please double the backslashes in the path or change them to forward slashes. For example, both `C:/commons-lang/src/main/java` and `C:\\commons-lang\\src\\main\\java` are correct.
- The process of opening the project should be completed within one second, and it should not exceed ten seconds at most. In our test, open a huge project with 200 classes and 3000 methods usually cost about 2 seconds.

Querying method invocation relationships

Querying method parameter relationships

Automaticly fixing syntax errors

Design

System architecture and components

General workflow

Implementation

Major data structures

Efficiency considerations

Key techniques

Test

Test Cases

We have two test cases in different scale:

- Case A: The most basic case

We have chosen the sample code provided in the documentation as our basic functionality test code. Specifically, its content is as follows:

```
package main;
class Test {
    static void sayHello(String name) {
        System.out.println("Hello, " + name + "!");
    }
    static void introduction(String name1) {
        sayHello(name1);
        String name2 = "Garfield";
        sayHello(name2);
    }
    public static void main(String[] args) {
        sayHello("Jon");
        String name3 = "Odie";
        introduction(name3);
    }
}
```

- Case B: Large-scale Java library project.

We have chosen the **commons-lang** library, open-sourced by the **Apache Foundation**, as our test project. Apache Commons is an excellent project within the Java ecosystem, offering high-quality and reliable components for almost every aspect of Java. This library consists of over two hundred classes and more than three thousand methods, totaling 175,000 lines of code. Choosing such a vast project to test our project is a significant challenge. The version

we selected is located at commit `gbe417ff07`. Learn more about Apache Commons: https://commons.apache.org/

Conclusion