Software design迭代三设计文档

211250074 左皓升

1. 引言

1.1 编制目的

本文档提供software design课程作业迭代三新增部分的软件架构概览

1.2 对象和范围

本文档的读者是作者本人，参考RUP的《软件架构文档模板》，用于指导下一循环的代码开发和测试工作。

1.3 参考资料

1. IBM RUP (2002) 软件架构文档模板
2. 《软件工程与计算（卷三）：团队与软件开发实践》骆斌、刘嘉等著

1.4 名称和术语

2. 具体设计

2.1 项目结构

2.1.1 树状结构

```
├─main
|  └─java
|      └─org
|          └─example
|              ├─exams
|              |  ├─exam
|              |  |  └─question
|              |  └─examReader
|              ├─judge
|              |  ├─judgeOne
|              |  |  ├─codeCalculator
|              |  |  ├─codeCompiler
|              |  |  ├─codeRunner
|              |  |  └─multiStrategy
|              |  └─thread
|              ├─papers
|              |  ├─paper
|              |  |  └─answer
|              |  └─paperReader
|              └─utils
└─test
    ├─java
    └─resources
```

```
└─cases
    ├─answers
    │  └─code-answers
    ├─exams
    └─output
```

2.1.2 类图

2.2 超时代码停止运行

在之前的两次迭代中，timelimit并没有被使用，所以也没有进行读取。在这次的迭代中，Question中增加了timeLimit变量，并且在JsonReader和XmlReader中都进行对timeLimit的读取。

JsonReader中：

```
                JSONArray samples = question.optJSONArray("samples");
                if (samples != null) {
                    for (int j = 0; j < samples.length(); j++) {
                        JSONObject sample = samples.getJSONObject(j);
                        String input = sample.getString("input");
                        String output = sample.getString("output");
                        q.addSample(input, output);
                    }
                    q.setTimeLimit(question.getInt("timeLimit"));
                }

                exam.addQuestion(q);
```

XmlReader中：

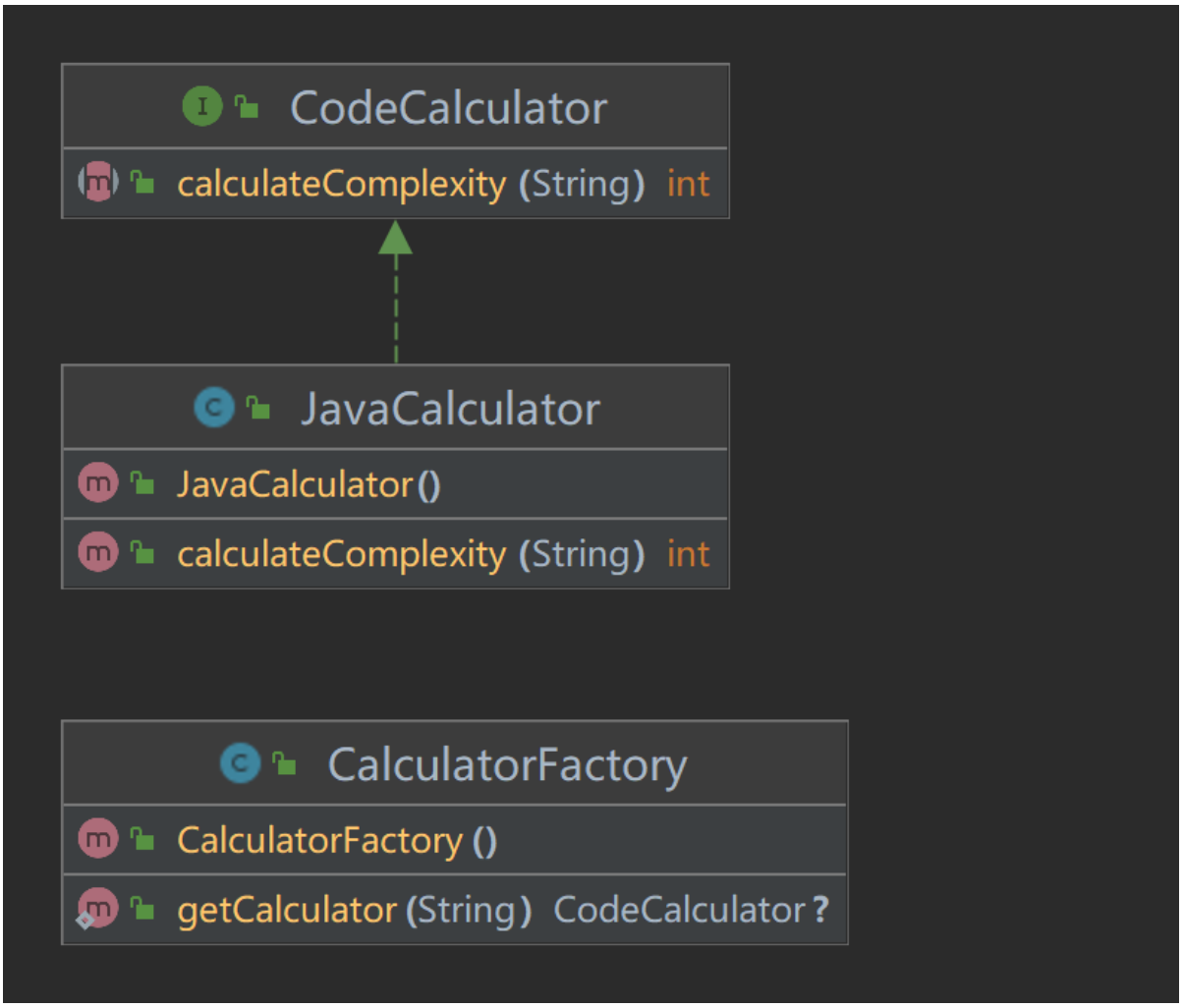
```
                if (q.getType() == 3) {

 q.setTimeLimit(Integer.parseInt(questionElement.getElementsByTagName("timeLimit"
).item(0).getTextContent()));
                }
                exam.addQuestion(q);
```

在JavaRunner中，如果进程时间超过了timeLimit，就停止运行

```
            ProcessBuilder pb = new ProcessBuilder("java", "-cp", path, name,
input[0], input[1]);
            pb.redirectErrorStream(true); // 将错误输出重定向到标准输出流
            Process process = pb.start();

            boolean finished = process.waitFor(limit, TimeUnit.MILLISECONDS);
            if (!finished) {
                process.destroy(); // 确保杀死进程
                return false;
            }
```

2.3 计算圈复杂度

圈复杂度的计算使用javaParser来完成，同时采用简单工厂方便之后增加计算其他代码的圈复杂度



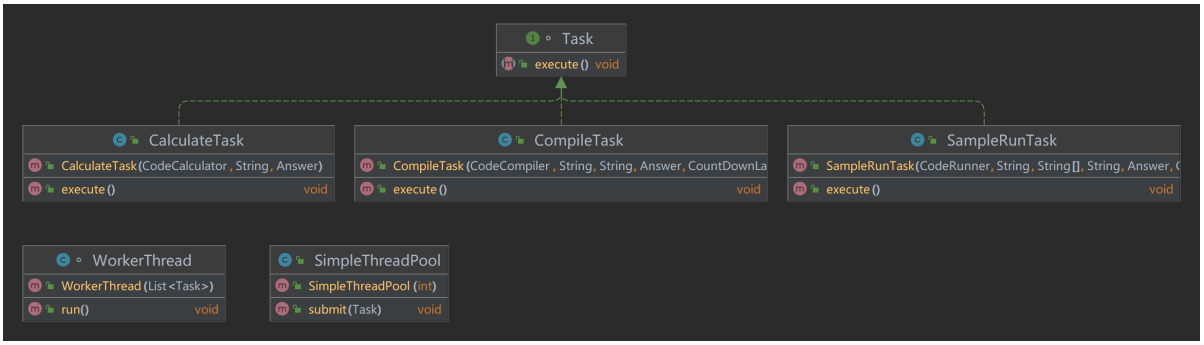
JavaCalculator完整代码如下：

```
package org.example.judge.judgeOne.codeCalculator;

import com.github.javaparser.ast.CompilationUnit;
import com.github.javaparser.ast.body.MethodDeclaration;
import com.github.javaparser.ast.stmt.DoStmt;
import com.github.javaparser.ast.stmt.ForStmt;
import com.github.javaparser.ast.stmt.IfStmt;
import com.github.javaparser.ast.stmt.WhileStmt;
import com.github.javaparser.ast.visitor.VoidVisitorAdapter;
import com.github.javaparser.JavaParser;
import com.github.javaparser.ast.expr.BinaryExpr;
import com.github.javaparser.ast.expr.ConditionalExpr;

import java.io.FileInputStream;
import java.io.FileNotFoundException;
import java.util.concurrent.atomic.AtomicInteger;


public class JavaCalculator implements CodeCalculator{
    @Override
    public int calculateComplexity(String JavaPath) {
        try {
            FileInputStream in = new FileInputStream(JavaPath);
            JavaParser javaParser = new JavaParser();
```

```
            CompilationUnit cu = javaParser.parse(in).getResult().orElseThrow(()
-> new RuntimeException("Parsing failed!"));

            AtomicInteger totalComplexity = new AtomicInteger(0); // Start with 1
for the entire class

            cu.accept(new VoidVisitorAdapter<Void>() {
                @Override
                public void visit(MethodDeclaration n, Void arg) {
                    super.visit(n, arg);
                    AtomicInteger complexity = new AtomicInteger(1); // Start
with 1 for each method
                    n.accept(new VoidVisitorAdapter<Void>() {
                        @Override
                        public void visit(IfStmt n, Void arg) {
                            super.visit(n, arg);
                            complexity.getAndIncrement(); // for "if"
                            n.getElseStmt().ifPresent(elseStmt -> {
                                if (elseStmt instanceof IfStmt) {
                                    complexity.getAndIncrement(); // for "else
if"
                                }
                            });
                        }

                        @Override
                        public void visit(WhileStmt n, Void arg) {
                            super.visit(n, arg);
                            complexity.getAndIncrement(); // for "while"
                        }

                        @Override
                        public void visit(DoStmt n, Void arg) {
                            super.visit(n, arg);
                            complexity.getAndIncrement(); // for "do-while"
                        }

                        @Override
                        public void visit(ForStmt n, Void arg) {
                            super.visit(n, arg);
                            complexity.getAndIncrement(); // for "for"
                        }

                        @Override
                        public void visit(BinaryExpr n, Void arg) {
                            super.visit(n, arg);
                            if (n.getOperator() == BinaryExpr.Operator.AND ||
                                    n.getOperator() == BinaryExpr.Operator.OR) {
                                complexity.getAndIncrement(); // for "&&" or "||"
                            }
                        }

                        @Override
                        public void visit(ConditionalExpr n, Void arg) {
                            super.visit(n, arg);
                            complexity.getAndIncrement(); // for ternary "? :"
```

```
                        }
                    }, null);
                    totalComplexity.addAndGet(complexity.get());
                }
            }, null);

            return totalComplexity.get();
        } catch (FileNotFoundException e) {
            return -1;
        }
    }
}
```

从逻辑上来说，只有代码题需要计算圈复杂度，所以圈复杂度的计算工作理应和compiler、runner在一起。既然compiler和runner都使用线程池来加速，并且之前设计的线程池和任务是解耦的，所以这里很容易的把calculater也加入线程池。



Calculate Task代码如下：

```
package org.example.judge.thread;

import org.example.judge.judgeOne.codeCalculator.CodeCalculator;
import org.example.papers.paper.answer.Answer;

public class CalculateTask implements Task {
    private final CodeCalculator calculator;
    private final String javaPath;
    private final Answer answer;

    public CalculateTask(CodeCalculator calculator, String javaPath, Answer
answer) {
        this.calculator = calculator;
        this.javaPath = javaPath;
        this.answer = answer;
    }

    @Override
    public void execute() {
        try {
            int success = calculator.calculateComplexity(javaPath);
            synchronized (answer) {
                answer.setComplexity(success);
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
```

```
    }
}
```

最后，在JudgeCode中加入计算圈复杂度的代码，完成了圈复杂度的计算

```
package org.example.judge.judgeOne;

import org.example.exams.exam.question.Question;
import org.example.judge.judgeOne.codeCalculator.CodeCalculator;
import org.example.judge.judgeOne.codeCompiler.CodeCompiler;
import org.example.judge.judgeOne.codeRunner.CodeRunner;
import org.example.judge.thread.CalculateTask;
import org.example.judge.thread.CompileTask;
import org.example.judge.thread.SampleRunTask;
import org.example.judge.thread.SimpleThreadPool;
import org.example.papers.paper.answer.Answer;
import java.nio.file.Paths;
import java.util.List;
import java.util.concurrent.CountDownLatch;

import static
org.example.judge.judgeOne.codeCalculator.CalculatorFactory.getCalculator;
import static
org.example.judge.judgeOne.codeCompiler.CompilerFactory.getCompiler;
import static org.example.judge.judgeOne.codeRunner.RunnerFactory.getRunner;


public class JudgeCode implements JudgeOne {
    private SimpleThreadPool threadPool;

    public JudgeCode() {
        this.threadPool = new SimpleThreadPool(5);
    }

    @Override
    public int judgeOne(Question q, List<Answer> an, String path) {
        for (Answer answer : an) {
            String temp = answer.getAnswer().replace("/",
System.getProperty("file.separator"));
            String JavaPath = path + System.getProperty("file.separator") + temp;
            String JavaClassPath = Paths.get(JavaPath).getParent().toString();

            CodeCalculator codeCalculator = getCalculator(JavaPath);
            CalculateTask calculateTask = new CalculateTask(codeCalculator,
JavaPath, answer);
            threadPool.submit(calculateTask);

            if (!answer.isValid()) {
                answer.setScore(0);
                continue;
            }
            CountDownLatch compileLatch = new CountDownLatch(1);

            CodeCompiler codeCompiler = getCompiler(JavaPath);
```

```
            CompileTask compileTask = new CompileTask(codeCompiler, JavaPath,
JavaClassPath, answer, compileLatch);
            threadPool.submit(compileTask);

            try {
                compileLatch.await(); // Wait for compilation to complete before
continuing
                if (answer.getScore() == 0) {
                    continue; // Skip running samples if compilation failed
                }

                answer.setScore(q.getPoints());
                String mainClass = answer.getAnswer().substring(13,
answer.getAnswer().length() - 5);
                CountDownLatch runLatch = new
CountDownLatch(q.getSamples().size());
                for (String[] sample : q.getSamples()) {
                    CodeRunner codeRunner = getRunner(JavaPath);
                    SampleRunTask runTask = new SampleRunTask(codeRunner,
JavaClassPath, sample, mainClass, answer, runLatch, q.getTimeLimit());
                    threadPool.submit(runTask);
                }
                runLatch.await(); // Wait for all run tasks to complete

            } catch (InterruptedException e) {
                e.printStackTrace();
            }
        }
        return 0;
    }
}
```

2.4 无效答卷的处理

对于作答时间超出规定时间的试卷，不需要进行判分，但是需要进行代码圈复杂度的计算。在之前的迭代中都进行了评分，输出的时候才把无效的标记为0分。为了提高效率，使得评分过程更加合理，本次迭代中仅对有效的答卷进行评分，无效的直接判为零分。同时为了不影响圈复杂度的计算，所以在Answer中引入了一个valid变量，如果为false则只进行圈复杂度的计算，如果为true那么正常进行判分同时进行圈复杂度计算。

在JudgeCode中，判断逻辑如下：

```
            CodeCalculator codeCalculator = getCalculator(JavaPath);
            CalculateTask calculateTask = new CalculateTask(codeCalculator,
JavaPath, answer);
            threadPool.submit(calculateTask);

            if (!answer.isValid()) {
                answer.setScore(0);
                continue;
            }
```

在JudgeMulti和JudgeSingle中也有类似的逻辑。

2.5 Output路径

在测试中发现，output路径有时为分数的输出文件，有时为圈复杂度的输出文件，所以对output进行了处理，分别得到分数和圈复杂度输出文件。

在main函数中：

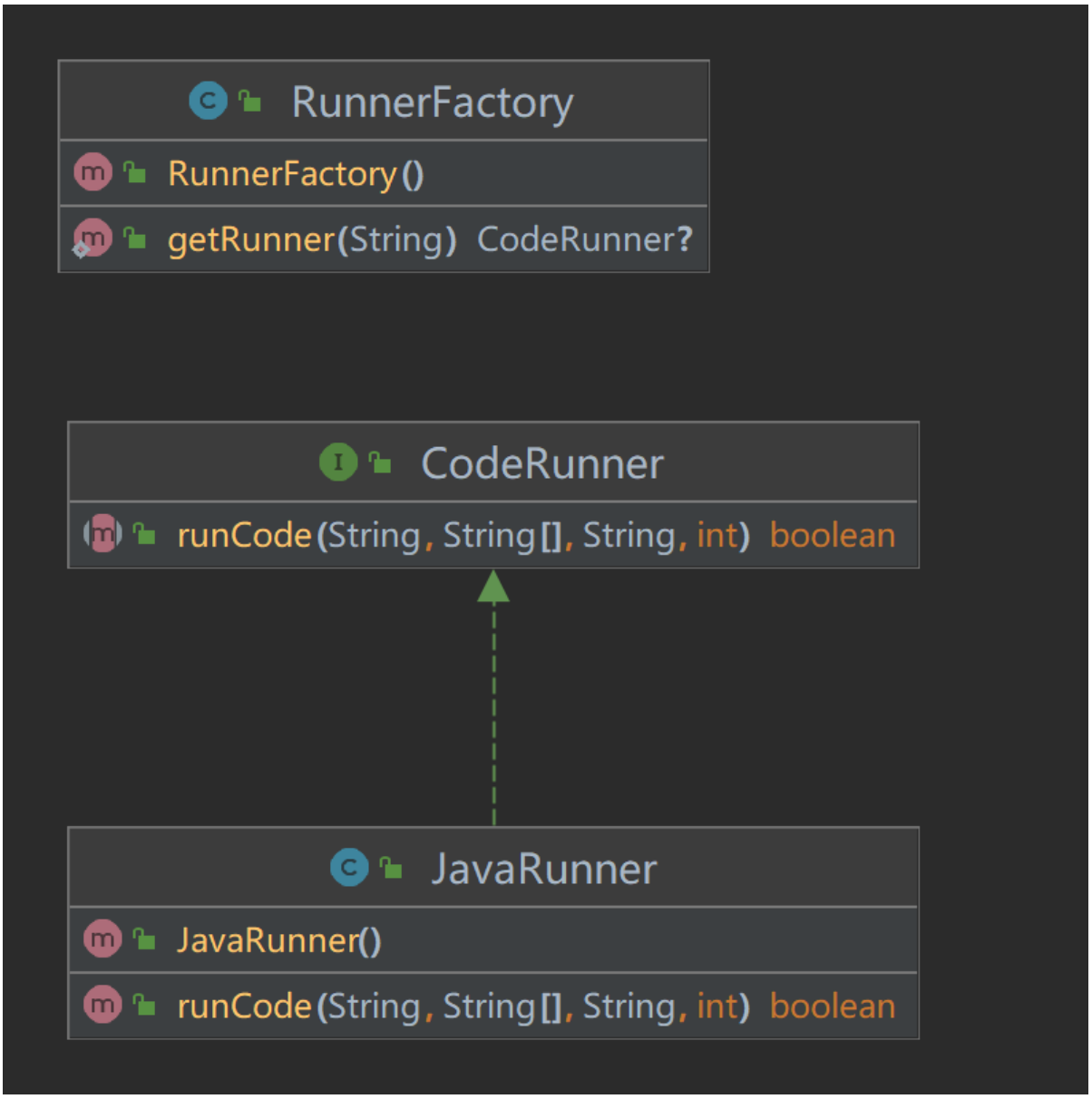
```
        if (output.endsWith("output.csv")) {
            output = output.substring(0, output.length() - 10);
        } else if (output.endsWith("output_complexity.csv")) {
            output = output.substring(0, output.length() - 21);
        } else {
            System.out.println("Invalid output file name");
            return;
        }
```

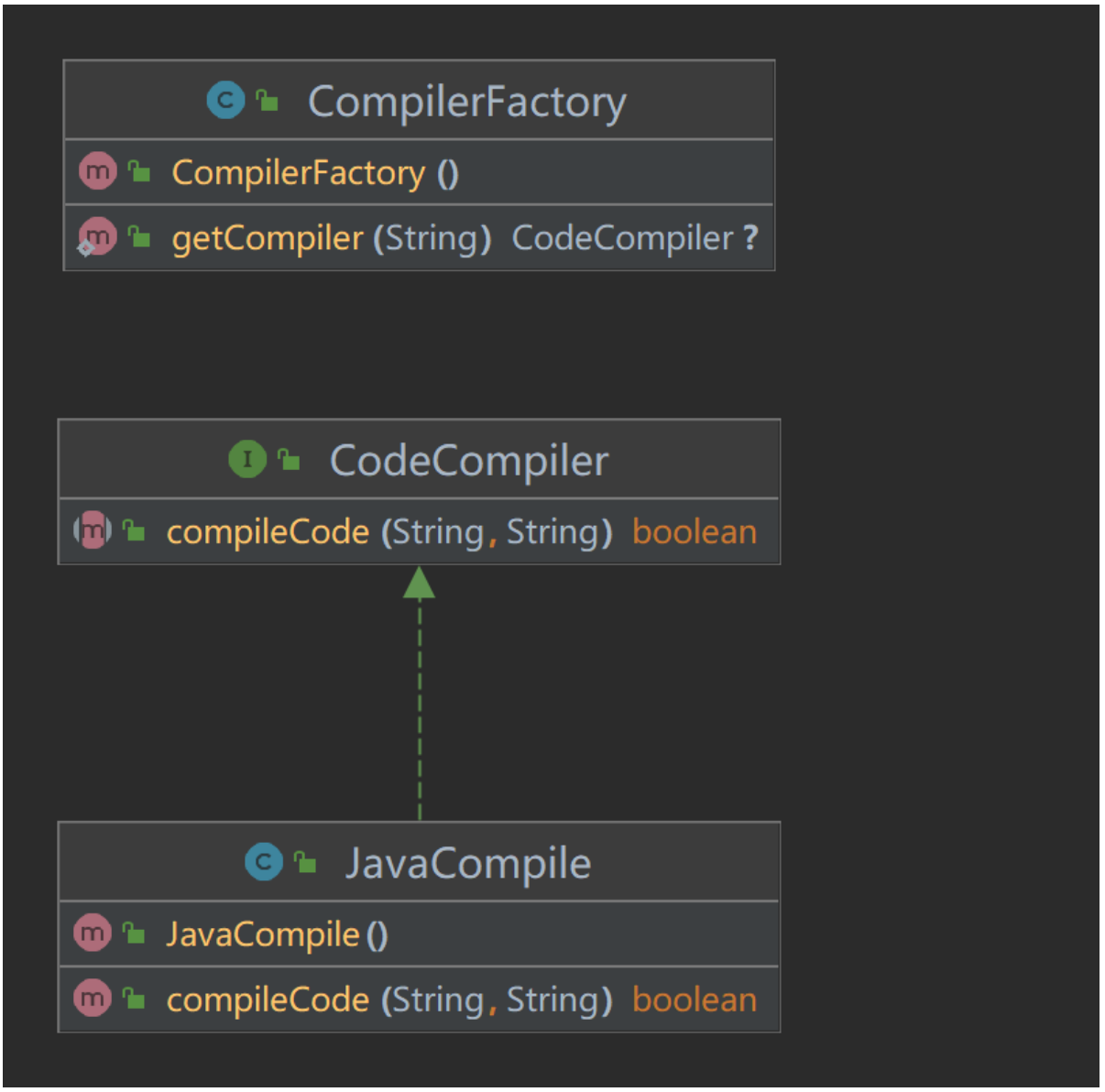
Judge中：

```
        String output1 = output + "output.csv";
        String output2 = output + "output_complexity.csv";
```

2.6 简单工厂

在上一次迭代中，compiler和runner使用继承方便之后可能的多态，这次在这两个地方加入了简单工厂，类图如下：


RunnerFactory
RunnerFactory()
getRunner(String) CodeRunner?
CodeRunner
runCode(String, String[], String, int) boolean
JavaRunner
JavaRunner()
runCode(String, String[], String, int) boolean

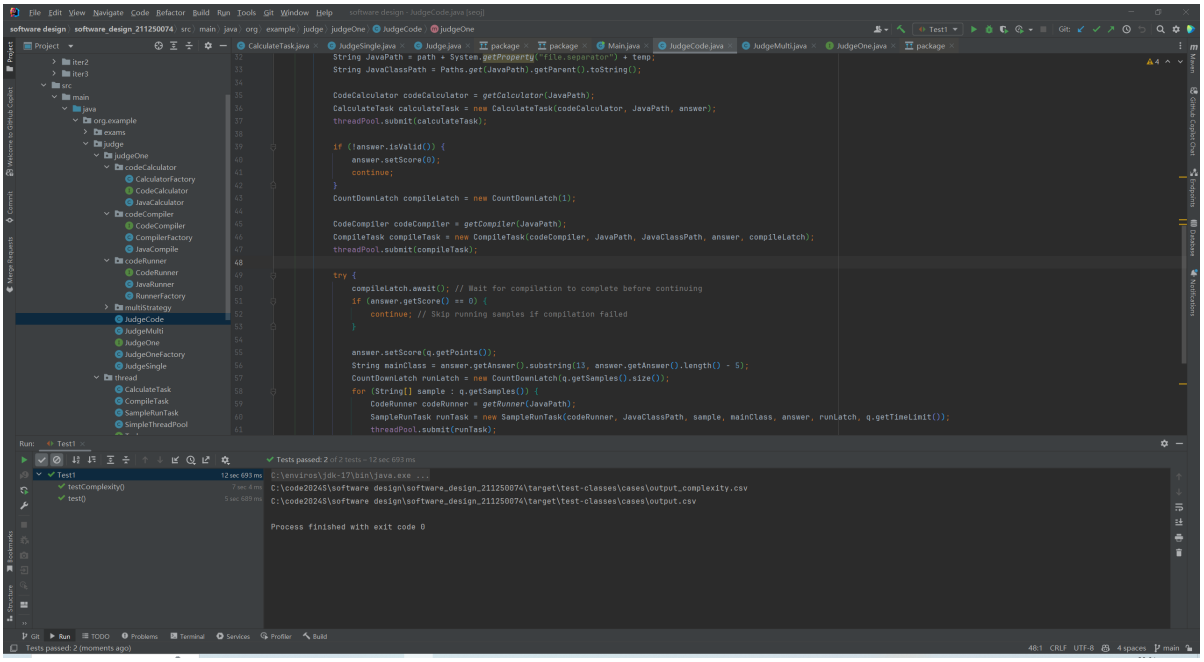


在JudgeCode中，使用工厂来获得compiler和runner

```
            CodeCompiler codeCompiler = getCompiler(JavaPath);
            CompileTask compileTask = new CompileTask(codeCompiler, JavaPath,
JavaClassPath, answer, compileLatch);
```

```
                    CodeRunner codeRunner = getRunner(JavaPath);
                    SampleRunTask runTask = new SampleRunTask(codeRunner,
JavaClassPath, sample, mainClass, answer, runLatch, q.getTimeLimit());
```

3. 功能测试



圈复杂度和评分，两个测试都顺利通过。