Software design迭代二设计文档

211250074 左皓升

1. 引言

1.1 编制目的

本文档提供software design课程作业迭代二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

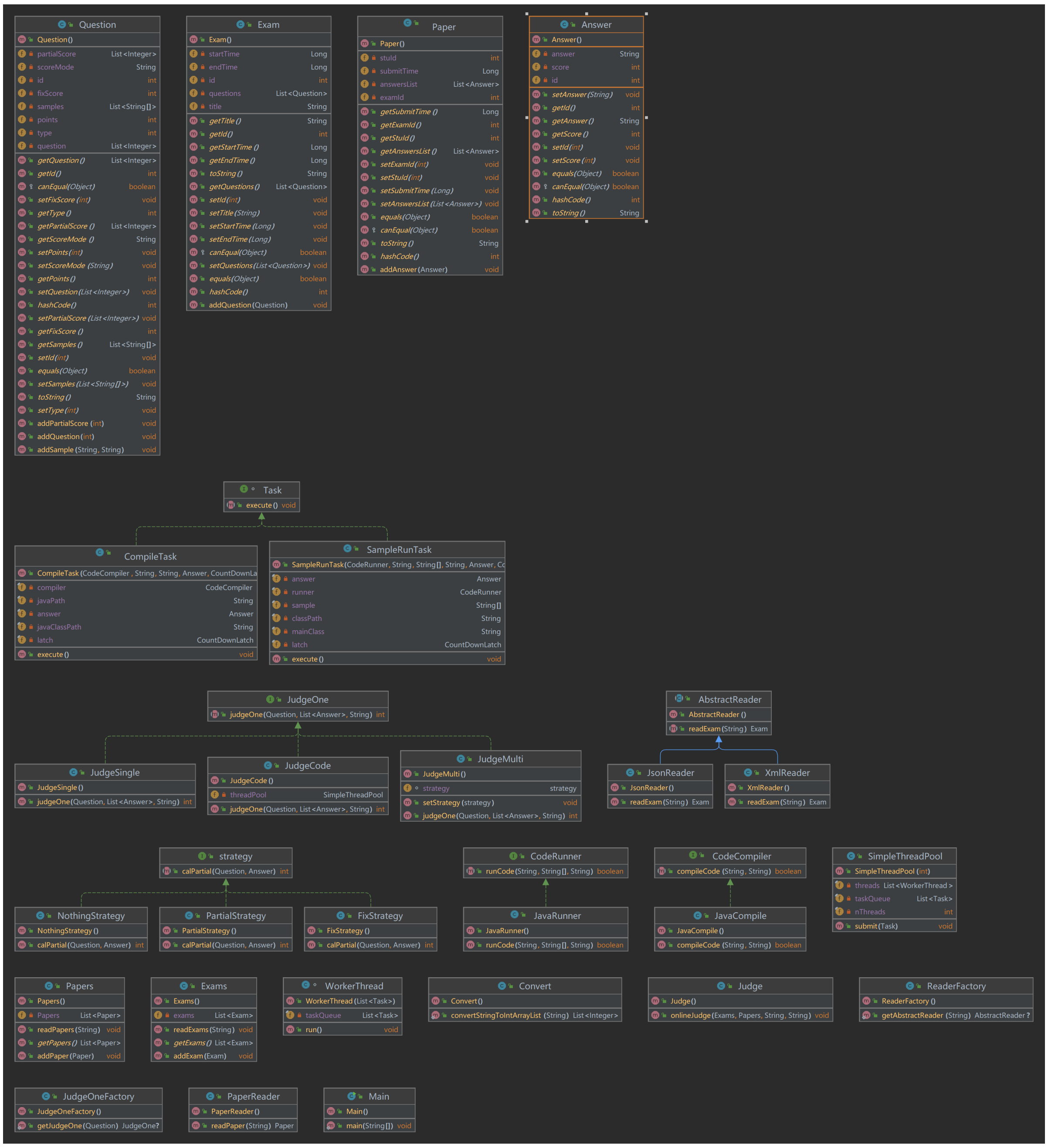
2. 具体设计

2.1 项目结构

2.1.1 树状结构

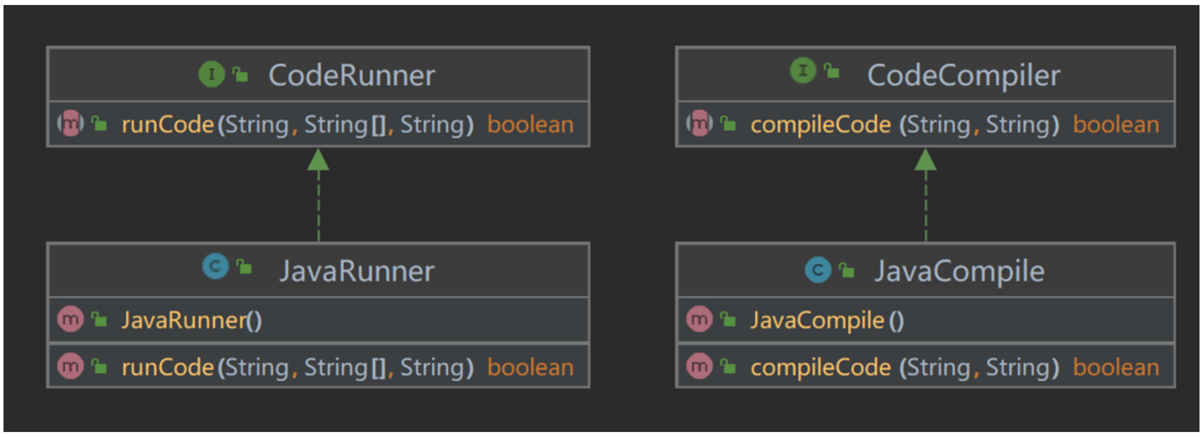
```
├─main
|  └─java
|      └─org
|          └─example
|              ├─exams
|              |   ├─exam
|              |   |   └─question
|              |   └─examReader
|              ├─judge
|              |   ├─judgeOne
|              |   |   ├─codeCompiler
|              |   |   ├─codeRunner
|              |   |   └─multiStrategy
|              |   └─thread
|              ├─papers
|              |   ├─paper
|              |   |   └─answer
|              |   └─paperReader
|              └─utils
└─test
    ├─java
    └─resources
        └─cases
        ├─answers
        │  └─code-answers
        ├─exams
        └─output
```

2.1.2 类图



2.2 编译和运行代码

为了能够在将来兼容不同类型的代码，建立了CodeCompiler和CodeRunner两个接口，再用具体的compiler和runner类去继承他们，方便以后的迭代中在这里应用工厂模式选择不同的compiler和runner。


CodeRunner
runCode(String, String[], String) boolean
CodeCompiler
compileCode (String, String) boolean
JavaRunner
JavaRunner()
runCode(String, String[], String) boolean
JavaCompile
JavaCompile ()
compileCode (String, String) boolean

而具体的实现代码并不复杂，compiler只需要知道路径，runner知道路径、测试用例、名称便可运行。这两个函数都返回boolean类型来决定是否正确。

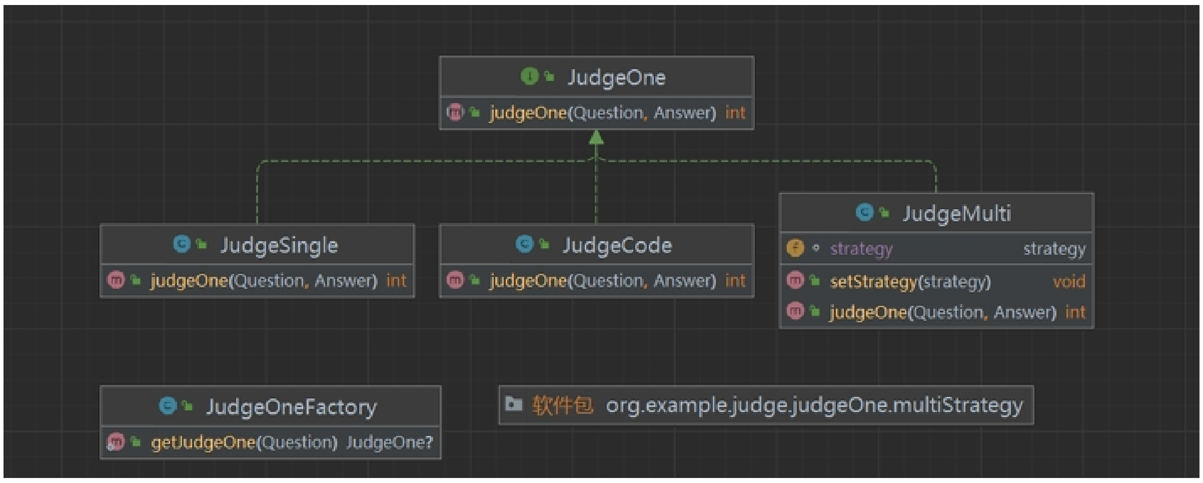
```
package org.example.judge.judgeOne.codeCompiler;

import org.example.papers.paper.answer.Answer;

import javax.tools.*;
import java.util.Arrays;

public class JavaCompile implements CodeCompiler {
    @Override
    public boolean compileCode(String pathJava, String pathClass) {
        JavaCompiler compiler = ToolProvider.getSystemJavaCompiler();

        DiagnosticCollector<JavaFileObject> diagnostics = new
DiagnosticCollector<>();

        try (StandardJavaFileManager fileManager =
compiler.getStandardFileManager(diagnostics, null, null)) {
            Iterable<String> compilationOptions = Arrays.asList("-d", pathClass);

            Iterable<? extends JavaFileObject> compilationUnits =
fileManager.getJavaFileObjects(pathJava);

            JavaCompiler.CompilationTask task = compiler.getTask(null,
fileManager, diagnostics, compilationOptions, null, compilationUnits);

            boolean success = task.call();

            if (!success) {
                return false;
            }
        } catch (Exception e) {
            return false;
        }

        return true;
    }
}
```

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

```java
import org.example.papers.paper.answer.Answer;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.List;


public class JavaRunner implements CodeRunner{
    @Override
    public boolean runCode(String path, String[] sample, String name) {
        String [] input = sample[0].split(" ");
        int expect = Integer.parseInt(sample[1]);

        try {
            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();

            BufferedReader reader = new BufferedReader(new
InputStreamReader(process.getInputStream()));
            String line;
            while ((line = reader.readLine()) != null) {
                if (Integer.parseInt(line) != expect) {
                    return false;
                }
            }
        } catch (IOException  e) {
            e.printStackTrace();
        }
        return true;
    }
}
```

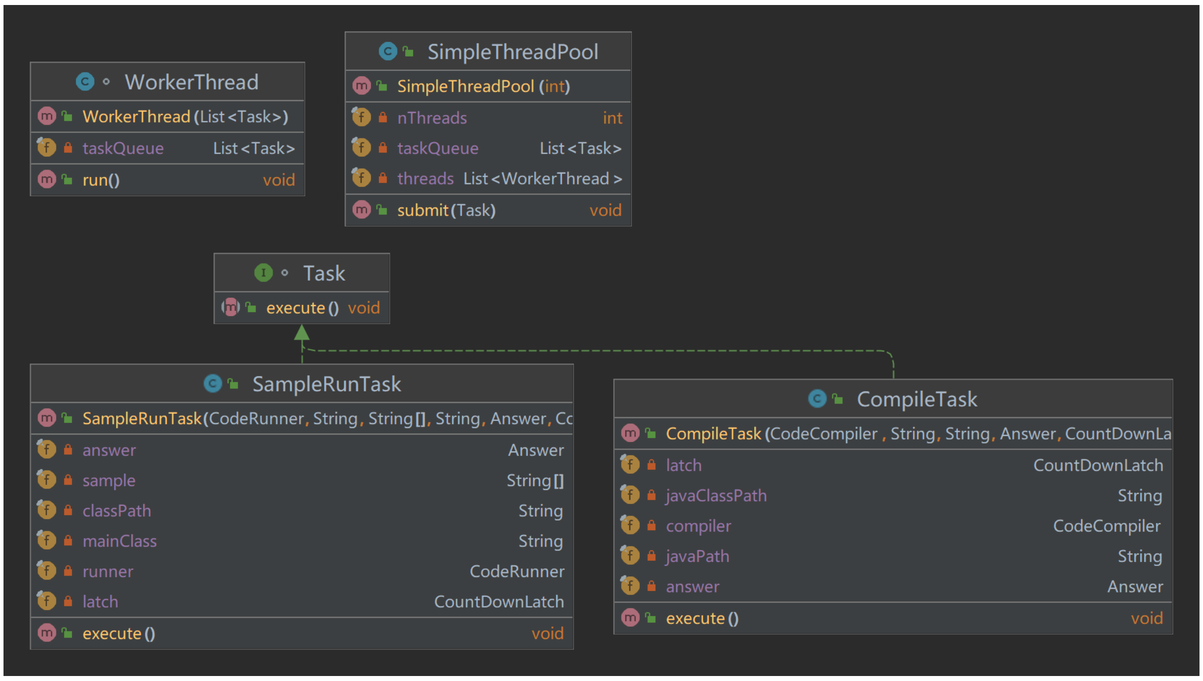
2.3 更改judgeOne的职责

在判断题目的过程中，上一次迭代采用了工厂模式来判别不同的题目。


JudgeOne
judgeOne(Question, Answer) int
JudgeSingle
judgeOne(Question, Answer) int
JudgeCode
judgeOne(Question, Answer) int
JudgeMulti
strategy strategy
setStrategy(strategy) void
judgeOne(Question, Answer) int
JudgeOneFactory
getJudgeOne(Question) JudgeOne?
软件包 org.example.judge.judgeOne.multiStrategy

在这个设计中，judgeOne只负责对一道题目的一个作答进行判别。在本次迭代中，改为对一道题目的所有作答进行判别。这样可以提高使用线程池之后的效率，同时减少了函数的调用次数和参数重复传递。

2.4 设计线程池



设计一个简单的线程池，包括两个简单的任务继承自Task。

提供创建线程池的接口和提交任务接口

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

import java.util.LinkedList;
import java.util.List;

public class SimpleThreadPool {
    private final List<Task> taskQueue = new LinkedList<>();
    private final List<WorkerThread> threads = new LinkedList<>();
    private final int nThreads;

    public SimpleThreadPool(int nThreads) {
        this.nThreads = nThreads;
        for (int i = 0; i < nThreads; i++) {
            WorkerThread worker = new WorkerThread(taskQueue);
            worker.start();
            threads.add(worker);
        }
    }

    public void submit(Task task) {
        synchronized (taskQueue) {
            taskQueue.add(task);
            taskQueue.notify();
        }
    }
}
```

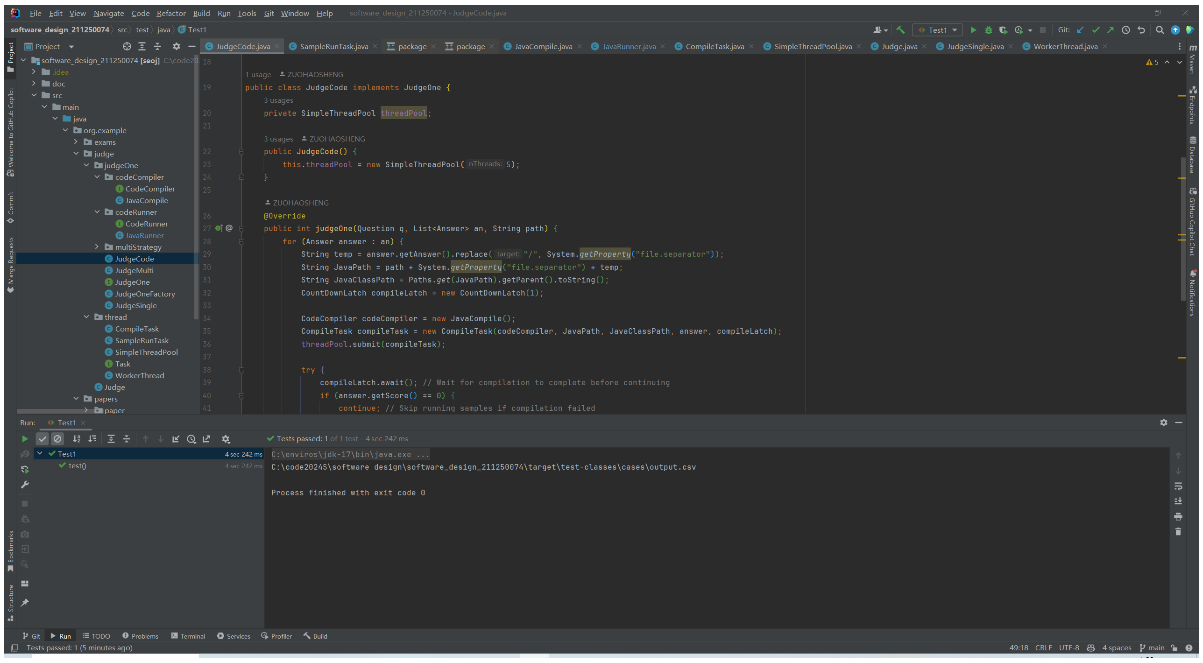
WorkerThread负责进行调度

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

import java.util.List;

class WorkerThread extends Thread {
    private final List<Task> taskQueue;

    public WorkerThread(List<Task> taskQueue) {
        this.taskQueue = taskQueue;
    }

    @Override
    public void run() {
        while (true) {
            Task task = null;
            synchronized (taskQueue) {
                while (taskQueue.isEmpty()) {
                    try {
                        taskQueue.wait();
                    } catch (InterruptedException e) {
                        e.printStackTrace();
                    }
                }
                task = taskQueue.remove(0);
            }
            try {
                task.execute();
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
}
```

2.5 线程池使用

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

import org.example.exams.exam.question.Question;
import org.example.judge.judgeOne.codeCompiler.CodeCompiler;
import org.example.judge.judgeOne.codeCompiler.JavaCompile;
import org.example.judge.judgeOne.codeRunner.CodeRunner;
import org.example.judge.judgeOne.codeRunner.JavaRunner;
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;




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();
            CountDownLatch compileLatch = new CountDownLatch(1);

            CodeCompiler codeCompiler = new JavaCompile();
            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()) {
                    SampleRunTask runTask = new SampleRunTask(new JavaRunner(),
JavaClassPath, sample, mainClass, answer, runLatch);
                    threadPool.submit(runTask);
                }
                runLatch.await(); // Wait for all run tasks to complete

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

3. 功能测试



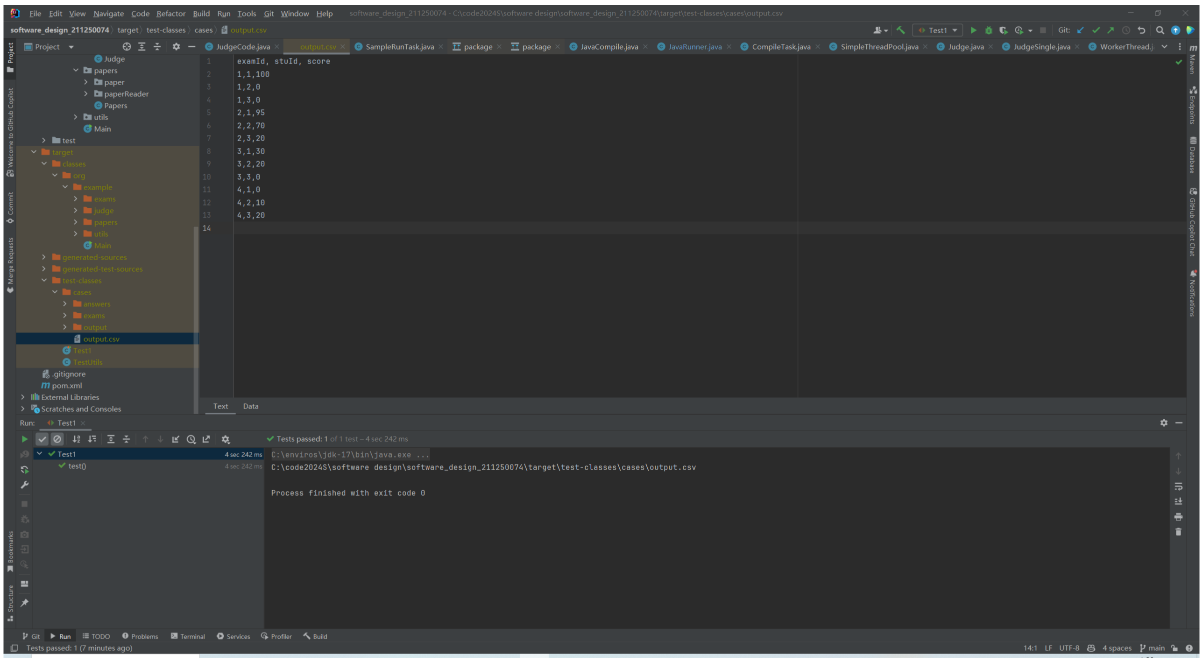
```java
public class JudgeCode implements JudgeOne {
    private SimpleThreadPool threadPool;

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

    @Override
    public int judgeOne(Question q, List<Answer> an, String path) {
        for (Answer answer : an) {
            String temp = answer.getAnswer().replace( target: "/", System.getProperty("file.separator"));
            String JavaPath = path + System.getProperty("file.separator") + temp;
            String JavaClassPath = Paths.get(JavaPath).getParent().toString();
            CountDownLatch compileLatch = new CountDownLatch(1);

            CodeCompiler codeCompiler = new JavaCompile();
            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
```

Tests passed: 1 of 1 test – 4 sec 242 ms

```
C:\enviros\jdk-17\bin\java.exe ...
C:\code2024S\software design\software_design_211250074\target\test-classes\cases\output.csv

Process finished with exit code 0
```

未使用线程池的运行时间在10s左右，使用后变成4s左右，成功提高了效率

输出结果展示



```
examId, stuId, score
1,1,100
1,2,0
1,3,0
2,1,95
2,2,70
2,3,20
3,1,30
3,2,20
3,3,0
4,1,0
4,2,10
4,3,20
```

Tests passed: 1 of 1 test – 4 sec 242 ms

```
C:\enviros\jdk-17\bin\java.exe ...
C:\code2024S\software design\software_design_211250074\target\test-classes\cases\output.csv

Process finished with exit code 0
```