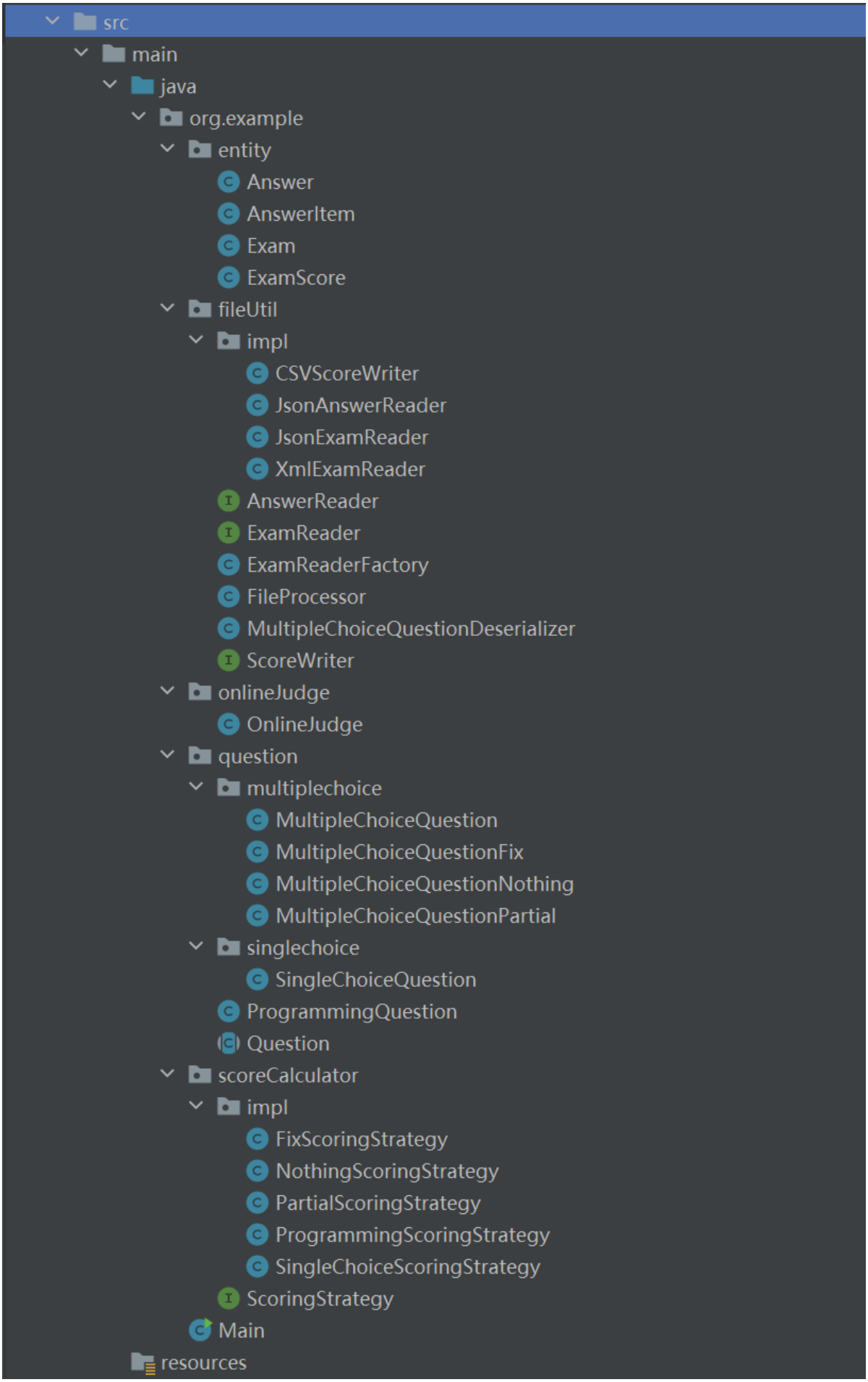
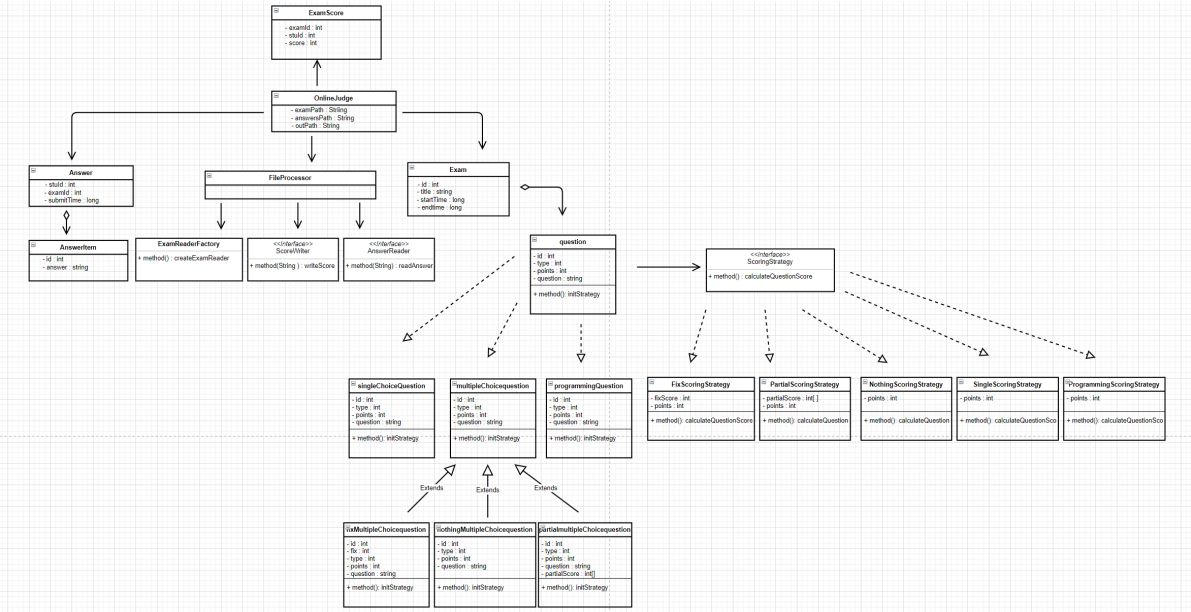
设计文档

211870287 丁旭

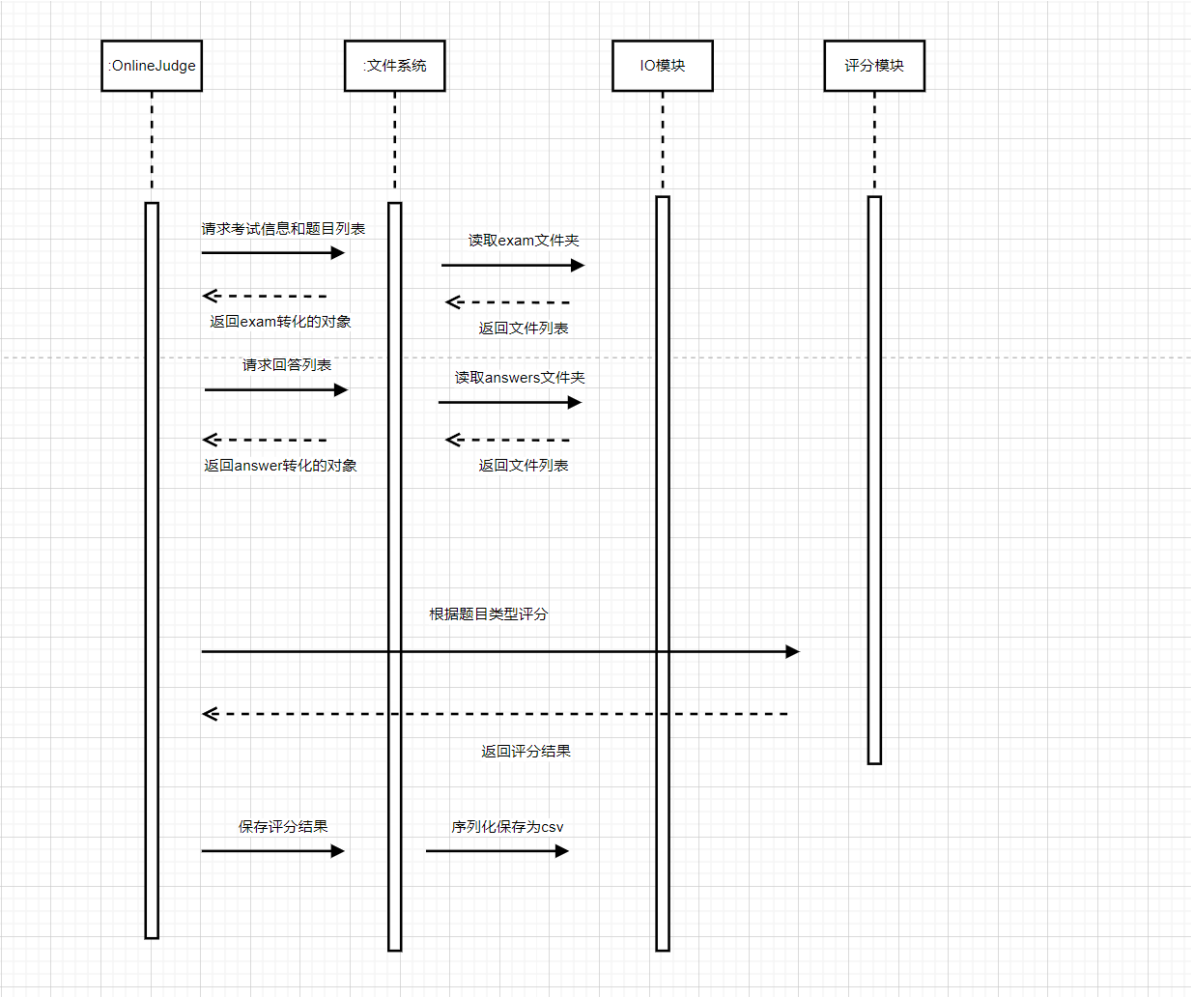
代码目录


src
main
java
org.example
entity
Answer
AnswerItem
Exam
ExamScore
fileUtil
impl
CSVScoreWriter
JsonAnswerReader
JsonExamReader
XmlExamReader
AnswerReader
ExamReader
ExamReaderFactory
FileProcessor
MultipleChoiceQuestionDeserializer
ScoreWriter
onlineJudge
OnlineJudge
question
multiplechoice
MultipleChoiceQuestion
MultipleChoiceQuestionFix
MultipleChoiceQuestionNothing
MultipleChoiceQuestionPartial
singlechoice
SingleChoiceQuestion
ProgrammingQuestion
Question
scoreCalculator
impl
FixScoringStrategy
NothingScoringStrategy
PartialScoringStrategy
ProgrammingScoringStrategy
SingleChoiceScoringStrategy
ScoringStrategy
Main
resources

类图



流程图



设计原则

- [x] 单一职责

  如ExamScore类，仅作为对csv文件的映射，将每一个学生的成绩保存在表中，作为一个实体类即成绩单

```java
@Data
public class ExamScore {
    2 usages
    @JsonProperty(index = 0)
    private int examId;
    2 usages
    @JsonProperty(index = 1)
    private int stuId;
    2 usages
    @JsonProperty(index = 2)
    private int score;

    // 构造函数
    2 usages  211870287
    public ExamScore(int examId, int stuId, int score) {
        this.examId = examId;
        this.stuId = stuId;
        this.score = score;
    }

    211870287
    @Override
    public String toString() {
        return "ExamScore{" +
                "examId='" + examId + '\'' +
                ", stuId='" + stuId + '\'' +
                ", score=" + score +
                '}';
    }
}
```

- [x] 开闭原则

使用抽象类question作为所有子类的父类，包含最基本的属性id，type，question，points，scoringStrategy（对问题的计算策略，这里的scoringStrategy是一个接口，根据不同question的计分策略按需实现）

```java
@Data
public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage  5 overrides  211870287
    public void initStrategy(){}

    6 overrides  211870287
    @Override
    public String toString() {...}
}
```

- [x] 里氏代换原则

这里使用到Question父类进行计分策略的初始化，可以用其任意子类来进行各自的计分策略的初始化。

```java
1 usage  211870287
private  void initQuestionScoreStrategy(){
    for (Exam exam:idToExamMap.values()){
        exam.getQuestions().forEach(Question::initStrategy);
    }
}
```

- [x] 依赖倒转原则

Exam依赖于抽象类的列表List,而非具体的 多选题、 单选题、编程题

```java
@Data
public class Exam {
    1 usage
    private int id;
    1 usage
    private String title;
    1 usage
    private long startTime;
    1 usage
    private long endTime;

    1 usage
    private List<Question> questions;

    211870287
    @Override
    public String toString() {...}
}
```

- [x] 接口隔离原则

将io处理拆分成三个细分的专业接口

1. 满足**单一职责原则**，将一组相关的操作定义在一个接口中，且在满足高内聚的前提下，接口中的方法越少越好。
2. 可以在进行系统设计时采用**定制服务**的方式，即**为不同的客户端提供宽窄不同的接口**

```java
2 usages  1 implementation  211870287
public interface AnswerReader {
    1 implementation  211870287
    Answer readAnswer(String filePath) throws IOException;
}
```

```java
8 usages  2 implementations  211870287
public interface ExamReader {
    3 usages  2 implementations  211870287
    Exam readExam(String filePath) throws IOException;
}
```


```java
8 usages  2 implementations  211870287
public interface ExamReader {
    3 usages  2 implementations  211870287
    Exam readExam(String filePath) throws IOException;
}
```


- [x] 合成复用原则

Question类组合ScoringStrategy，组合复用以使其可以黑箱复用

```java
@Data
public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage  5 overrides  211870287
    public void initStrategy(){}

    6 overrides  211870287
    @Override
    public String toString() {...}
}
```

- [x] 迪米特法则

在设计的过程中，尽可能的减少了类之间的不必要的通信，使用抽象工厂和接口来减少不同类之间的通信

```java
public class FileProcessor {
    2 usages
    private ExamReaderFactory examReaderFactory = new ExamReaderFactory();
    2 usages
    private ScoreWriter scoreWriter;
    2 usages
    private AnswerReader answerReader;

    1 usage  211870287
    public Map<Integer, Exam> readExam(String examsPath) throws IOException {...}

    1 usage  211870287
    public void saveExamScoreList(String outputPath, List<ExamScore> scoreList) throws IOException {...}

    1 usage  211870287
    public List<Answer> readAnswer(String answersPath) throws IOException {...}
}
```

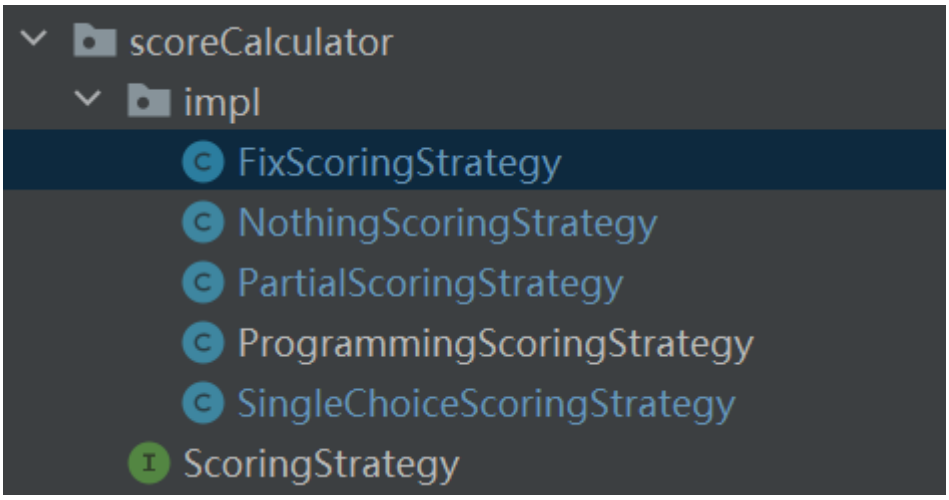
设计模式：

策略模式

定义接口

```
public interface ScoringStrategy {
    1 usage   5 implementations   211870287
    public int calculateQuestionScore(Object self_writtenAnswer);
}
```

实现不同的**计分策略**



再由对应的Question**组合复用**

```java
public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage   5 overrides   211870287
    public void initStrategy(){}

    6 overrides   211870287
    @Override
    public String toString() {...}
}
```

工厂模式

设计读取exam的简单工厂

```java
public class ExamReaderFactory {
    2 usages   211870287
    public static ExamReader createExamReader(String fileExtension) {
        switch (fileExtension) {
            case "json":
                return new JsonExamReader();
            case "xml":
                return new XmlExamReader();
            default:
                throw new IllegalArgumentException("Unsupported file type: " + fileExtension);
        }
    }
}
```

根据不同的参数来获取不同的“产品”——对应的reader类

接口设计

```
public interface AnswerReader {
    public Answer readAnswer(String filePath) throws IOException;
    public Answer readAnswer(File jsonFile) throws IOException;
}
```

```
public interface ExamReader {
    Exam readExam(String filePath) throws IOException;
}
```

```
public interface ScoreWriter {
    void writeScore(String filePath) throws IOException;
}
```

```
public interface ScoringStrategy {
    public int calculateQuestionScore(Object self_writtenAnswer);
}
```

抽象类设计

```
public abstract class Question<T> {

    private int id;
    private int type;
    private String question;
    private int points;

    private ScoringStrategy scoringStrategy;

    public void initStrategy(){}

    @Override
    public String toString() {
        return "Question{" +
                "id=" + id +
                ", type=" + type +
                ", question='" + question + '\'' +
                ", points=" + points +
                '}';
    }
}
```

实体设计

```
public class Answer {
    @JsonProperty("examId")
    private int examId;

    @JsonProperty("stuId")
    private int studentId;
```

```
    @JsonProperty("submitTime")
    private long submitTime;

    @JsonProperty("answers")
    private List<AnswerItem> answers;

    @Override
    public String toString() {
        return "Answer{" +
                "\nexamId=" + examId +
                "\nstuId=" + studentId +
                "\nsubmitTime=" + submitTime +
                "\nanswers=" + answers +
                "\n}";
    }

}
```

```
public class AnswerItem {
    private int id;
    private String answer;
}
```

```
public class Exam {
    private int id;
    private String title;
    private long startTime;
    private long endTime;

    private List<Question> questions;

    @Override
    public String toString() {
        return "Exam{\n" +
                "id=" + id +
                "\ntitle='" + title + '\'' +
                "\nstartTime=" + startTime +
                "\nendTime=" + endTime +
                "\nquestions=\n" + questions +
                '}';
    }
}
```

```
public class ExamScore {
    @JsonProperty(index = 0)
    private int examId;
    @JsonProperty(index = 1)
```

```
    private int stuId;
    @JsonProperty(index = 2)
    private int score;

    // 构造函数
    public ExamScore(int examId, int stuId, int score) {
        this.examId = examId;
        this.stuId = stuId;
        this.score = score;
    }

    @Override
    public String toString() {
        return "ExamScore{" +
                "examId='" + examId + '\'' +
                ", stuId='" + stuId + '\'' +
                ", score=" + score +
                '}';
    }
}
```

IO

使用jackon库读取xml文件或者json文件自动转化对其对应的子类实体

1.通过type属性来实例化不同子类

```java
@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        property = "type")
@JsonSubTypes({
        @JsonSubTypes.Type(value = SingleChoiceQuestion.class, name = "1"),
        @JsonSubTypes.Type(value = MultipleChoiceQuestion.class, name = "2"),
        @JsonSubTypes.Type(value = ProgrammingQuestion.class, name = "3")
})
@Data
public abstract class Question<T> {

    1 usage
    private int id;
    1 usage
    private int type;
    1 usage
    private String question;
    1 usage
    private int points;

    private ScoringStrategy scoringStrategy;

    1 usage  5 overrides  211870287
    public void initStrategy(){}

    6 overrides  211870287
    @Override
    public String toString() {...}
}
```

2.通过scoreMode属性来实例化不同多选题的类

```java
@JsonTypeInfo(
        use = JsonTypeInfo.Id.NAME,
        property = "scoreMode")
@JsonSubTypes({
        @JsonSubTypes.Type(value = MultipleChoiceQuestionFix.class, name = "fix"),
        @JsonSubTypes.Type(value = MultipleChoiceQuestionPartial.class, name = "partial"),
        @JsonSubTypes.Type(value = MultipleChoiceQuestionNothing.class, name = "nothing"),
})

@Data
public class MultipleChoiceQuestion extends Question {
    1 usage
    private String scoreMode;
//    @JacksonXmlProperty(localName = "option")
//    @JacksonXmlElementWrapper(useWrapping = false)
    1 usage
    private List<String> options;
    1 usage
    @JsonProperty("answer")
    @JsonAlias("answers")
    private List<Integer> answer;

    3 usages  211870287
    public MultipleChoiceQuestion() { this.setType(2); }
```