TURQUIZ

QUIZ GAME

GOALS

Our project focuses on developing an interactive application for teaching basic Turkish, designed to increase engagement and efficiency in the learning process.Problem Statement: Many students find traditional exam preparation methods boring and uninteresting.Solution: Our app uses G to allow you to select the topics and types of quizzes you need to prepare effectively.

ABOUT US

Nargiza (Head) Responsibilities: Project planning and implementation, quality assurance.

Gulayim (Developer) Responsibilities: System architecture and design, Implementation.

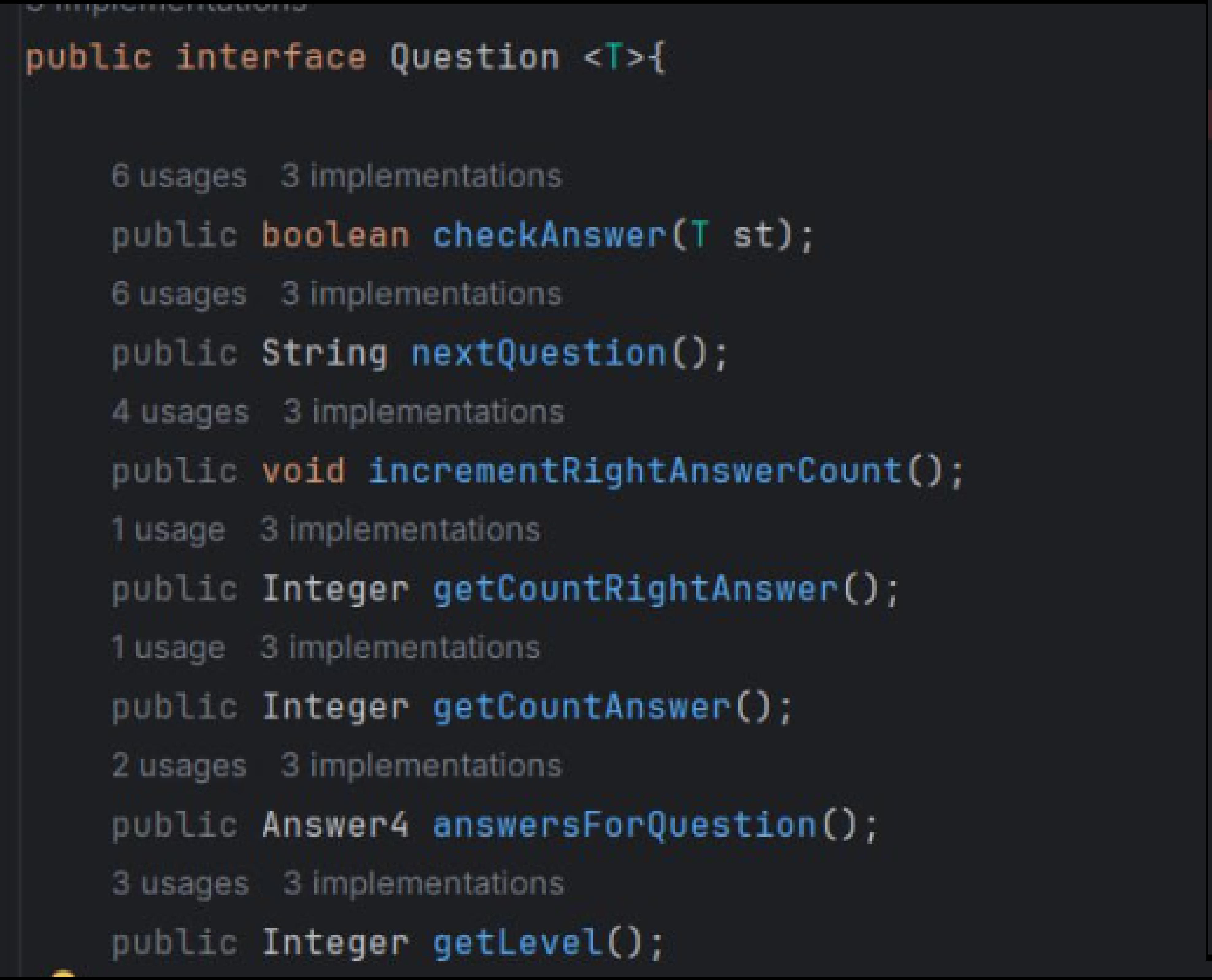
Adina (Designer) Responsibilities: User interface design.

PROBLEM AND SOLUTION

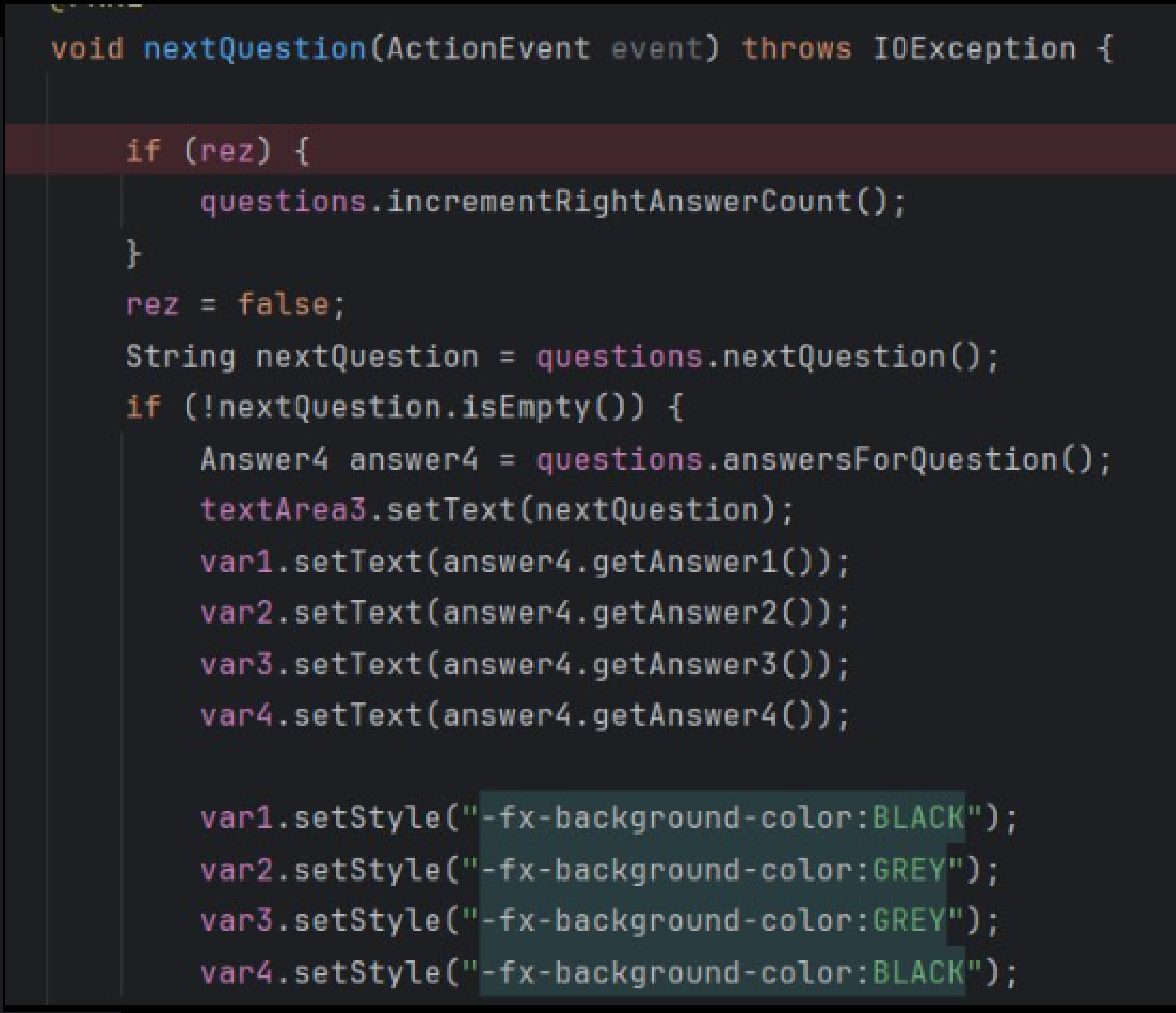
Formulation of the problem:Many students have difficulty with traditional exam preparation methods, finding them boring and uninteresting. There is a need for an interactive platform that enhances learning through game-based quizzesSolution:Our solution is an exam preparation app that uses brainstorming to make exam preparation fun and effective for students.

1) FACTORY METHOD:

We used the Factory method to take questions for our quiz from files. The most important thing here is that we have a class in which we initialize, depending on external conditions, the class we need for the interface.



```
public interface Question <T>{

    6 usages   3 implementations
    public boolean checkAnswer(T st);
    6 usages   3 implementations
    public String nextQuestion();
    4 usages   3 implementations
    public void incrementRightAnswerCount();
    1 usage   3 implementations
    public Integer getCountRightAnswer();
    1 usage   3 implementations
    public Integer getCountAnswer();
    2 usages   3 implementations
    public Answer4 answersForQuestion();
    3 usages   3 implementations
    public Integer getLevel();
```

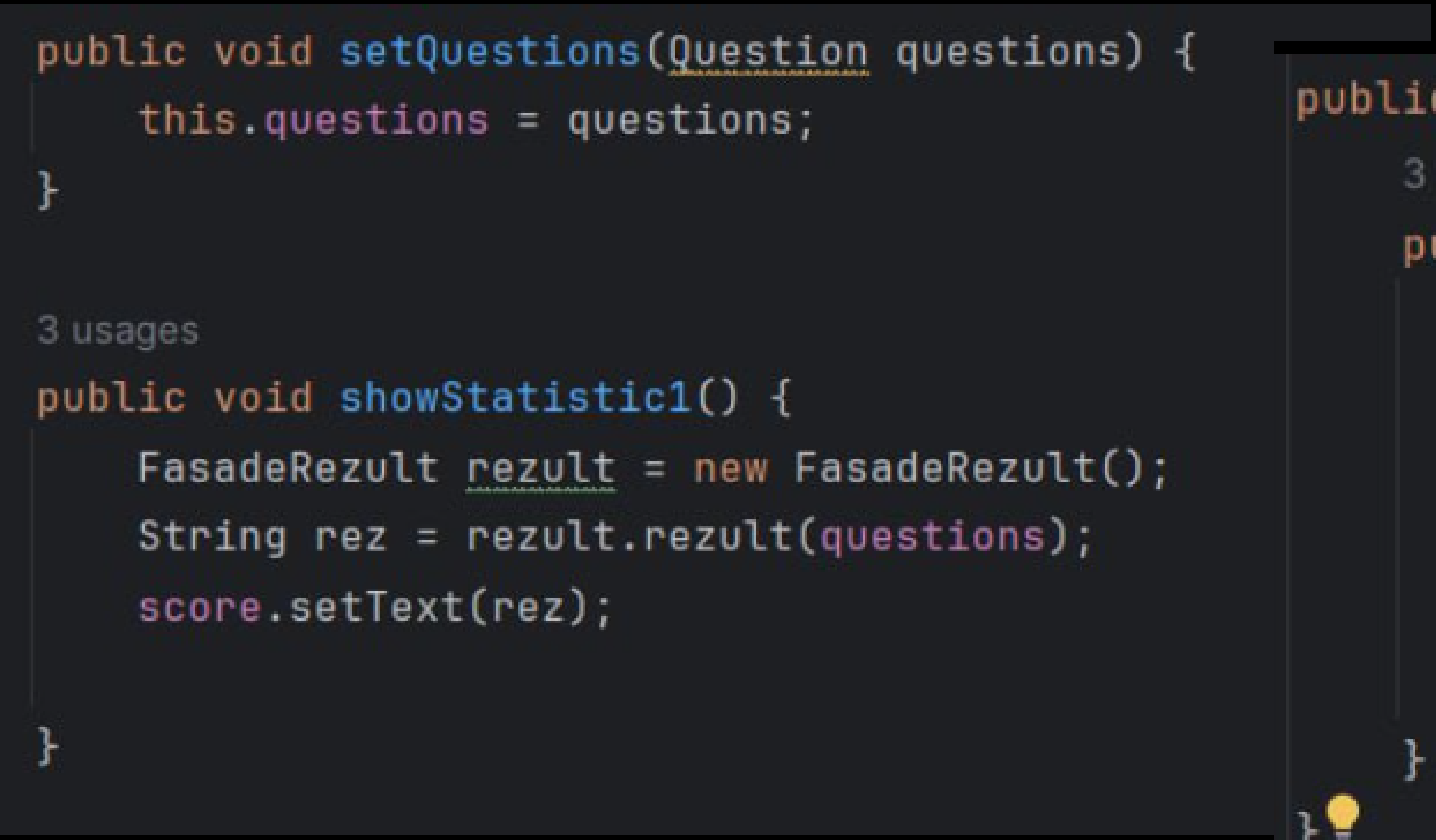


```
void nextQuestion(ActionEvent event) throws IOException {

    if (rez) {
        questions.incrementRightAnswerCount();
    }
    rez = false;
    String nextQuestion = questions.nextQuestion();
    if (!nextQuestion.isEmpty()) {
        Answer4 answer4 = questions.answersForQuestion();
        textArea3.setText(nextQuestion);
        var1.setText(answer4.getAnswer1());
        var2.setText(answer4.getAnswer2());
        var3.setText(answer4.getAnswer3());
        var4.setText(answer4.getAnswer4());

        var1.setStyle("-fx-background-color:BLACK");
        var2.setStyle("-fx-background-color:GREY");
        var3.setStyle("-fx-background-color:GREY");
        var4.setStyle("-fx-background-color:BLACK");
```

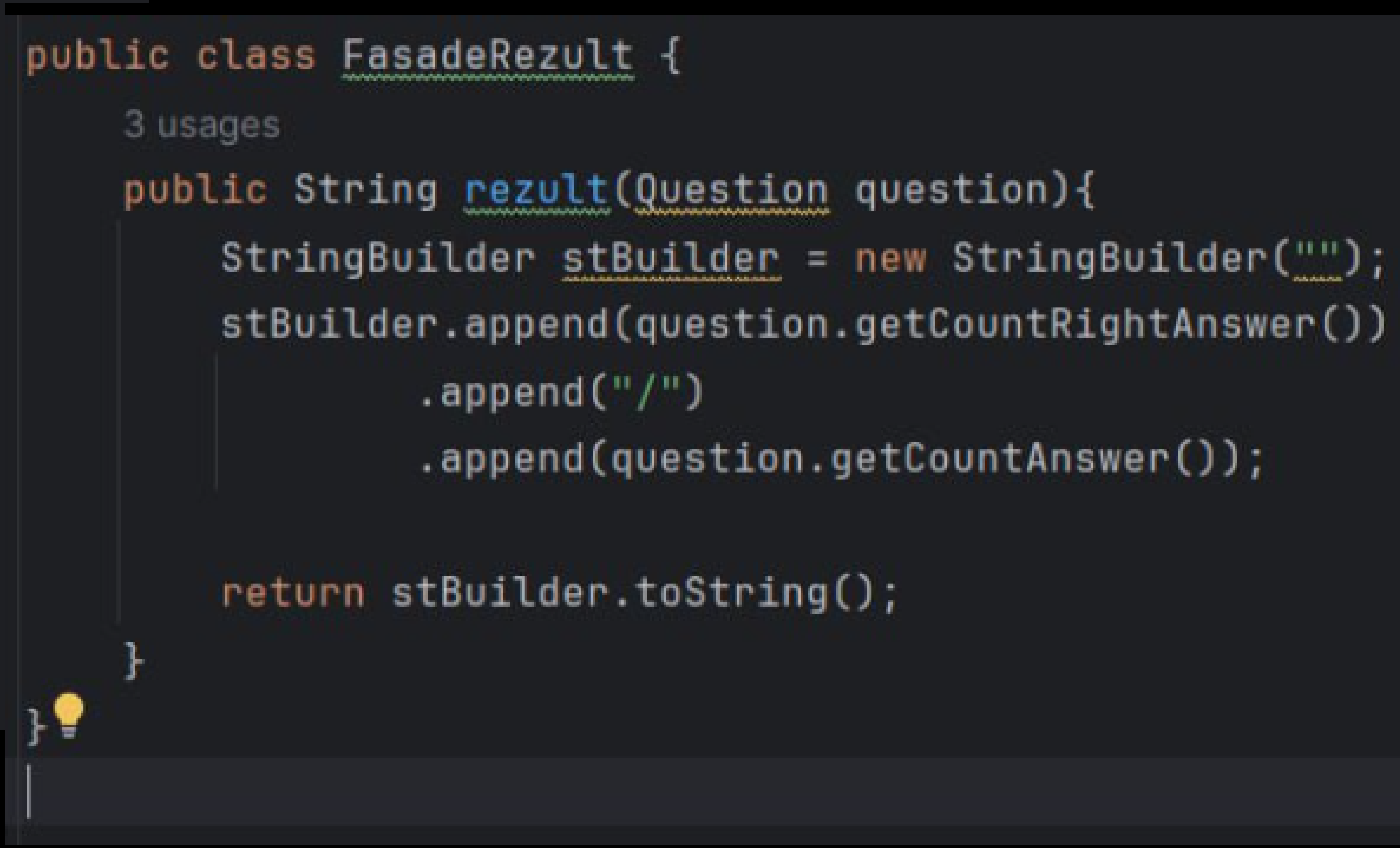
1) FACTORY METHOD: CHECK ANSWER AND NEXTQUESTION METHODS WHICH GIVES US QUESTIONS FOR OUR QUIZ AND CHECK THE RIGHT ANSWERS

2) FACADE:

It was used to "count and generate a result line.
We pass an instance of the question class into it, and it itself pulls out the necessary information.

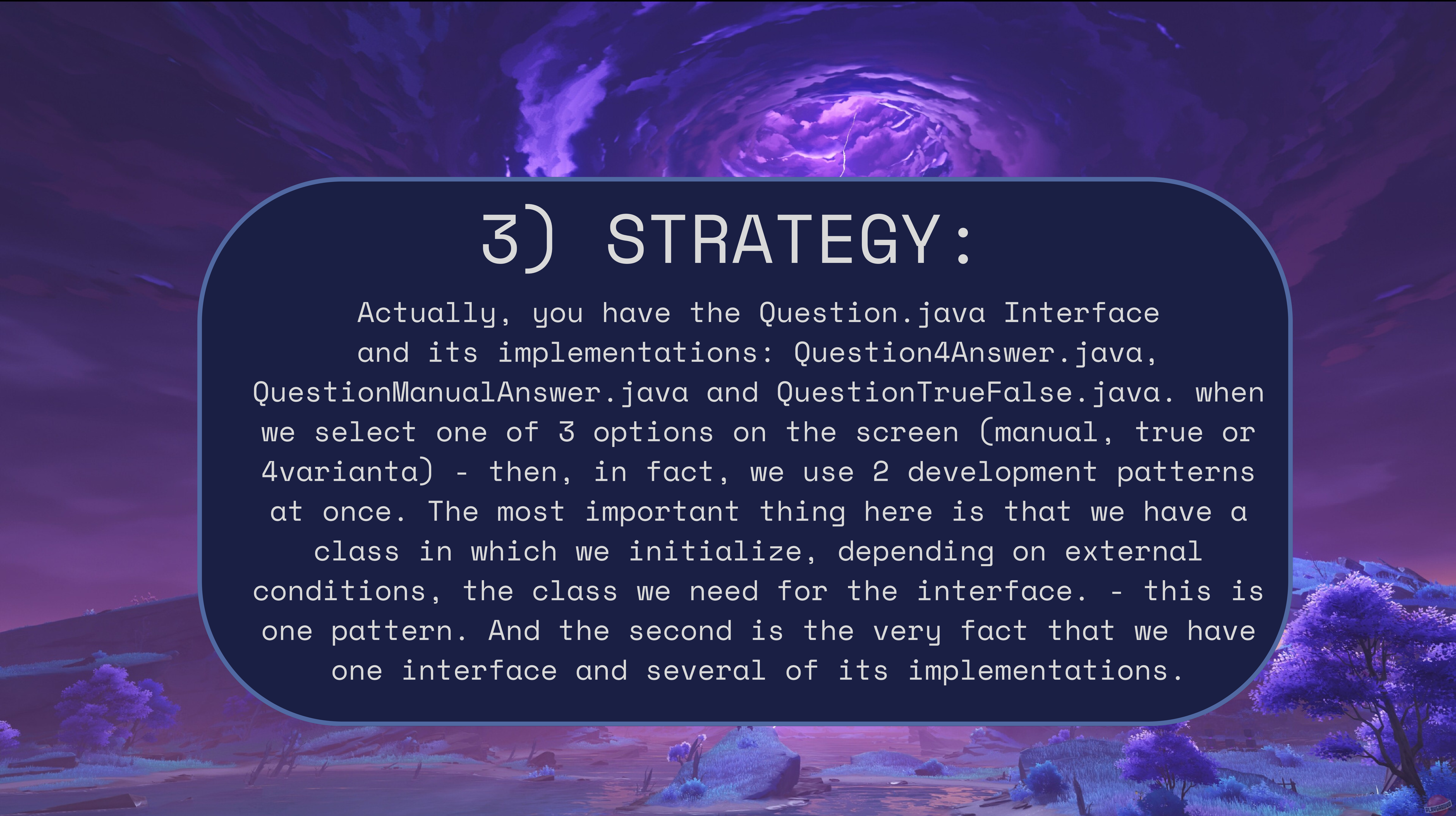


```java
public void setQuestions(Question questions) {
    this.questions = questions;
}

3 usages
public void showStatistic1() {
    FasadeRezult rezult = new FasadeRezult();
    String rez = rezult.rezult(questions);
    score.setText(rez);

}
```



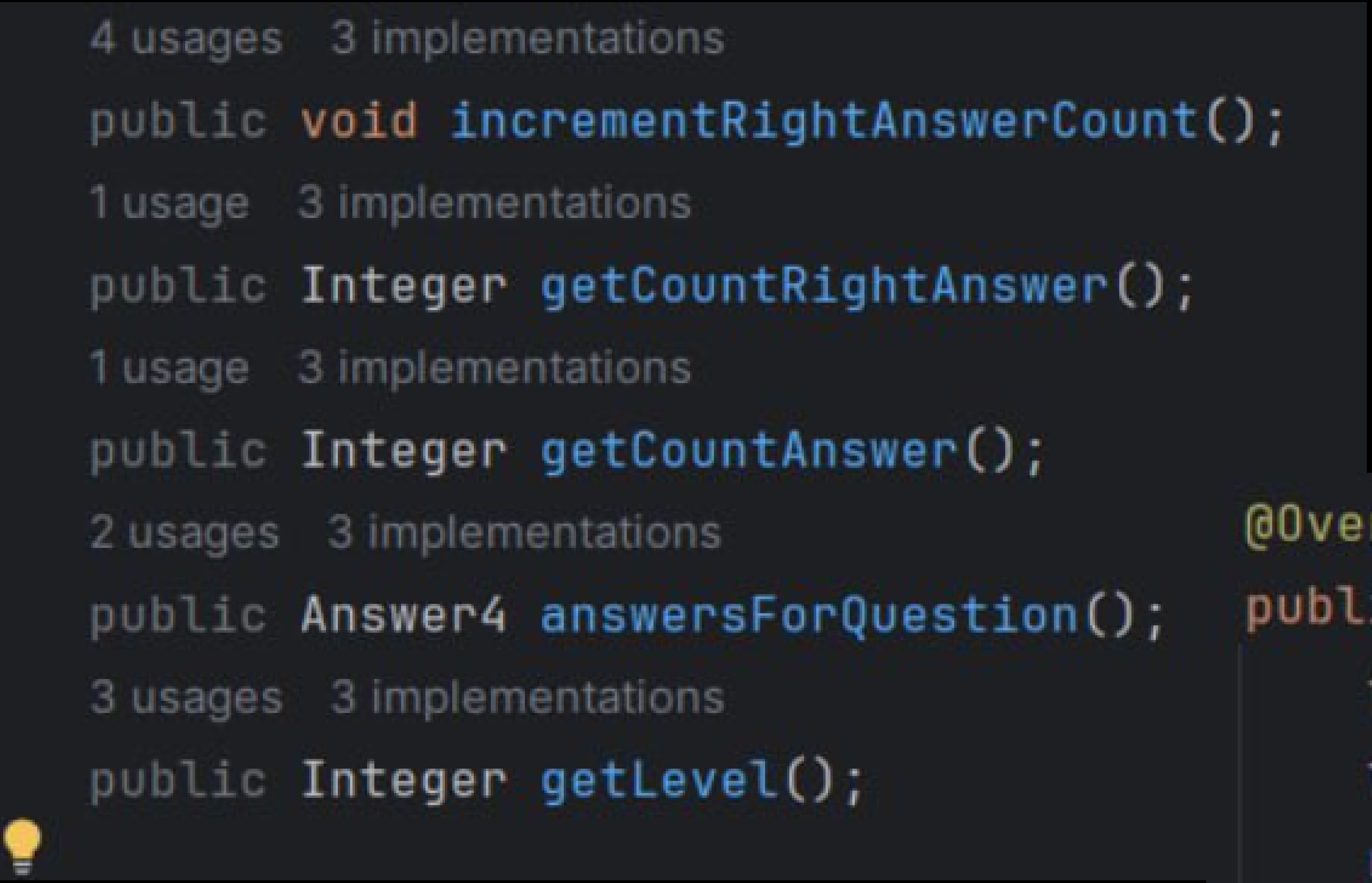
```java
public class FasadeRezult {
    3 usages
    public String rezult(Question question){
        StringBuilder stBuilder = new StringBuilder("");
        stBuilder.append(question.getCountRightAnswer())
                .append("/")
                .append(question.getCountAnswer());

        return stBuilder.toString();
    }
}
```

2) IN OUR CASE, WE USED FACADE TO SHOW HOW MANY POINTS OR CORRECT ANSWERS WE GET. WE USED FACADE IN THE LAST PAGE IN OUR QUIZ - RESULT PAGE

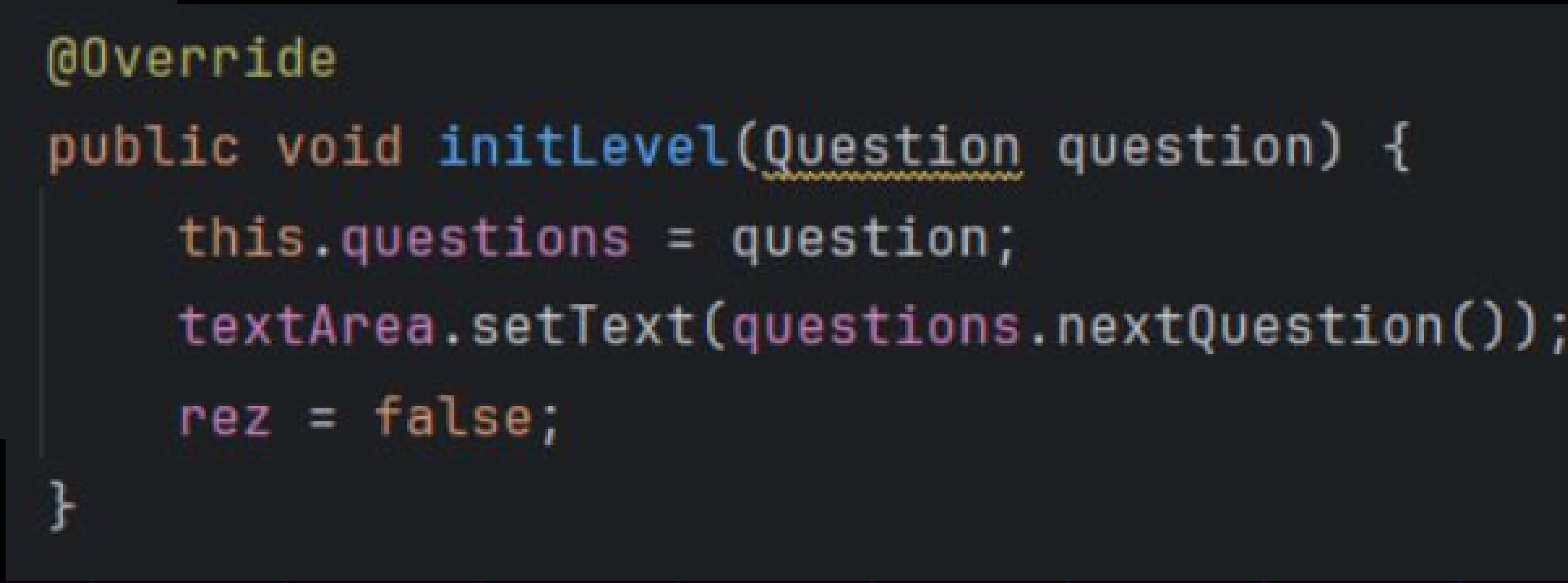


3) STRATEGY:

Actually, you have the Question.java Interface and its implementations: Question4Answer.java, QuestionManualAnswer.java and QuestionTrueFalse.java. when we select one of 3 options on the screen (manual, true or 4varianta) - then, in fact, we use 2 development patterns at once. The most important thing here is that we have a class in which we initialize, depending on external conditions, the class we need for the interface. - this is one pattern. And the second is the very fact that we have one interface and several of its implementations.



```
4 usages   3 implementations
public void incrementRightAnswerCount();
1 usage   3 implementations
public Integer getCountRightAnswer();
1 usage   3 implementations
public Integer getCountAnswer();
2 usages   3 implementations
public Answer4 answersForQuestion();
3 usages   3 implementations
public Integer getLevel();
```



```
@Override
public void initLevel(Question question) {
    this.questions = question;
    textArea.setText(questions.nextQuestion());
    rez = false;
}
```

3) STRATEGY OF COUNTING ANSWERS AND GETTING LEVEL

THANKS FOR ATTENTION!

THE END