Quiz Online System

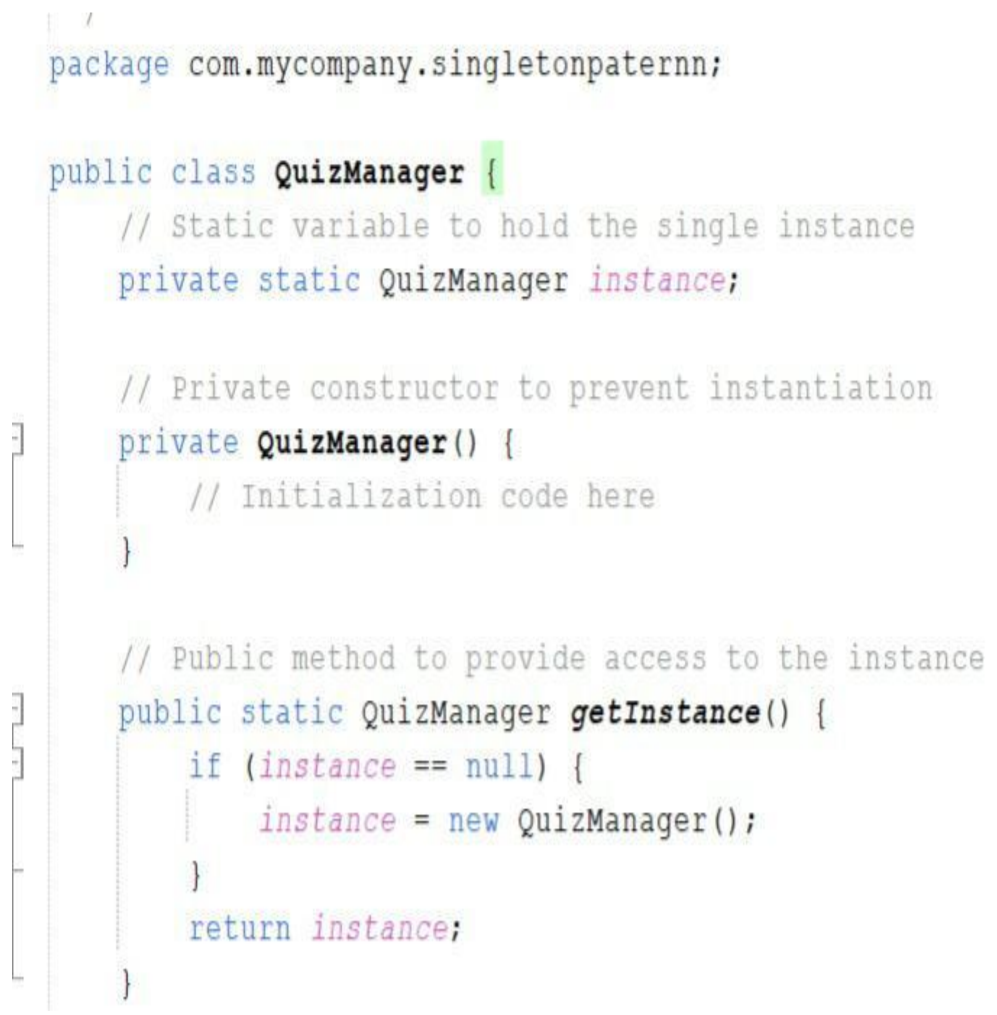
Aya Sayed Thabet	2100540
Esraa hussien abdelhafez	2100647
Ziad Mohamed Abdelhamid	2101786
Athal Tarek Mohammed	2100532
Asmaa hamdallah Soliman	21-00523
Alaa abdelrasol farghly ahmed	2100205

Singleton Pattern

Quiz Manager

This code implements the Singleton Pattern for managing quizzes:

1. *QuizManager Class: Manages quiz operations (creation, starting, and progress tracking).*
2. *. instance Variable: Holds the single instance of QuizManager*
3. *. Private Constructor: Prevents creating multiple instances of the class*
4. *. getInstance Method: Ensures only one Quiz Manager instance is created and provides access to it.*
5. *. Methods:createQuiz(String quizName): Placeholder for creating a quiz.startQuiz(int quizId): Placeholder for starting a quiz.trackProgress(int quizId): Placeholder for tracking quiz progress.*

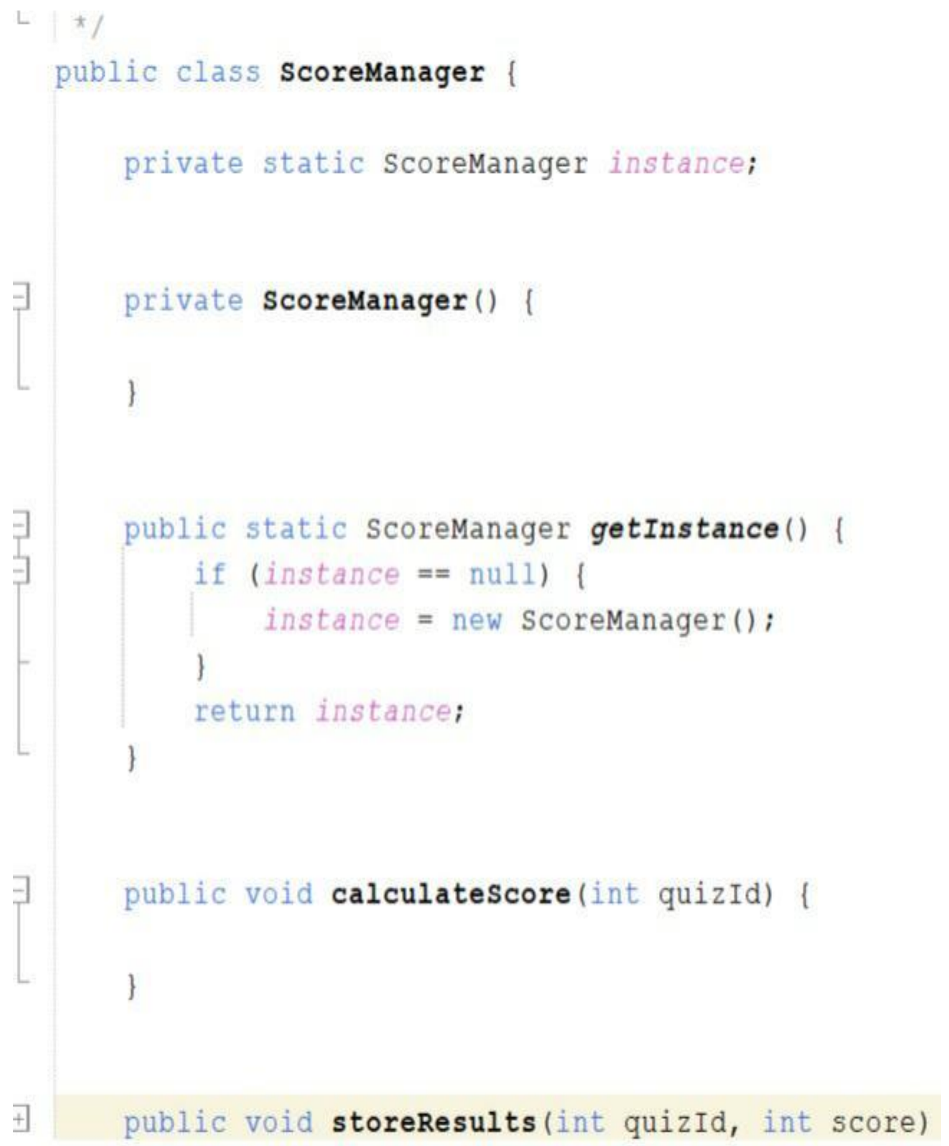


```
package com.mycompany.singletonpaternn;

public class QuizManager {
    // Static variable to hold the single instance
    private static QuizManager instance;

    // Private constructor to prevent instantiation
    private QuizManager() {
        // Initialization code here
    }

    // Public method to provide access to the instance
    public static QuizManager getInstance() {
        if (instance == null) {
            instance = new QuizManager();
        }
        return instance;
    }
```

Score Manager

This code defines a Singleton Pattern for managing scores:

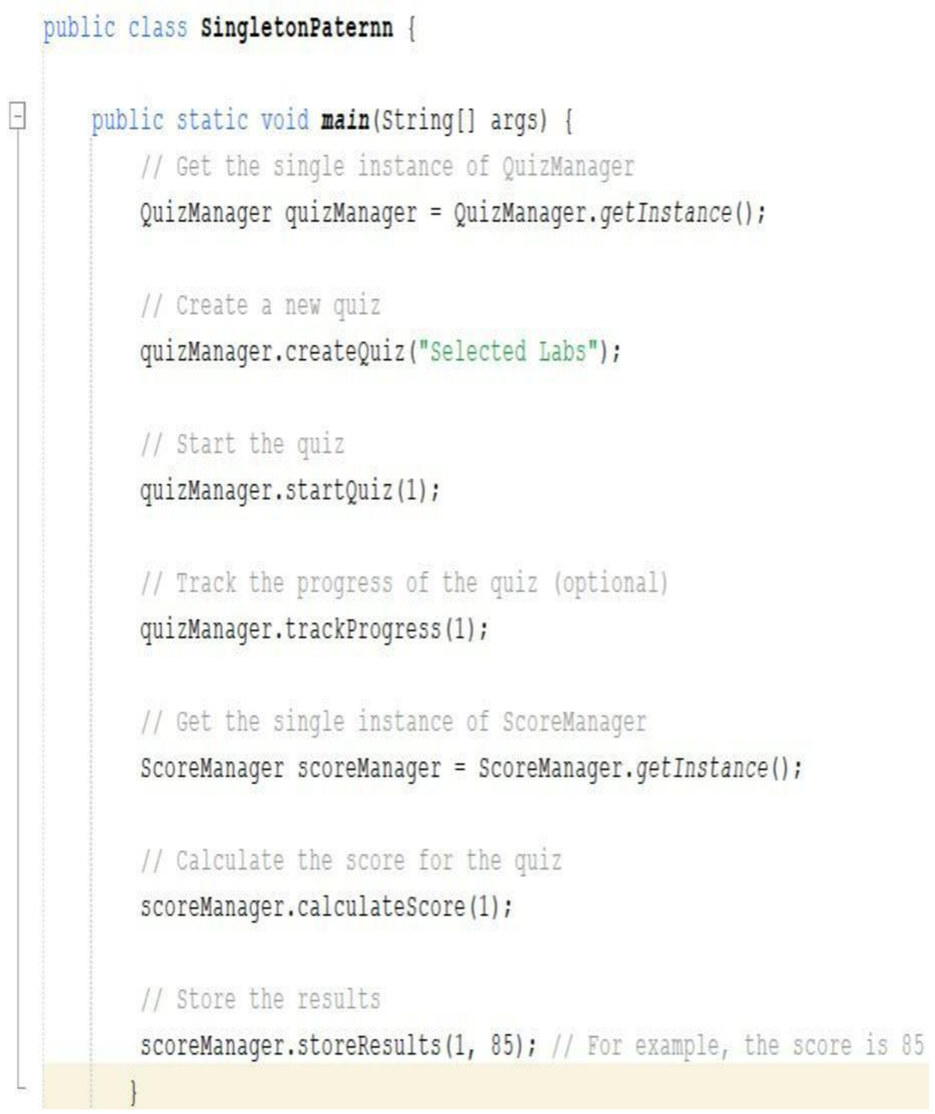
1. ScoreManager Class: Handles score-related operations like calculation and storage.
2. 2. instance Variable: Holds the single instance of ScoreManager.
3. 3. Private Constructor: Prevents creating multiple instances of the class.
4. 4. getInstance Method: Ensures only one instance is created and provides access to it.
5. 5. Methods:calculateScore(int quizId): Calculates scores for a specific quiz.storeResults(int quizId, int score): Stores quiz results in a database or file.



```
 */
public class ScoreManager {

    private static ScoreManager instance;


    private ScoreManager() {

    }


    public static ScoreManager getInstance() {
        if (instance == null) {
            instance = new ScoreManager();
        }
        return instance;
    }


    public void calculateScore(int quizId) {

    }


    public void storeResults(int quizId, int score)
```

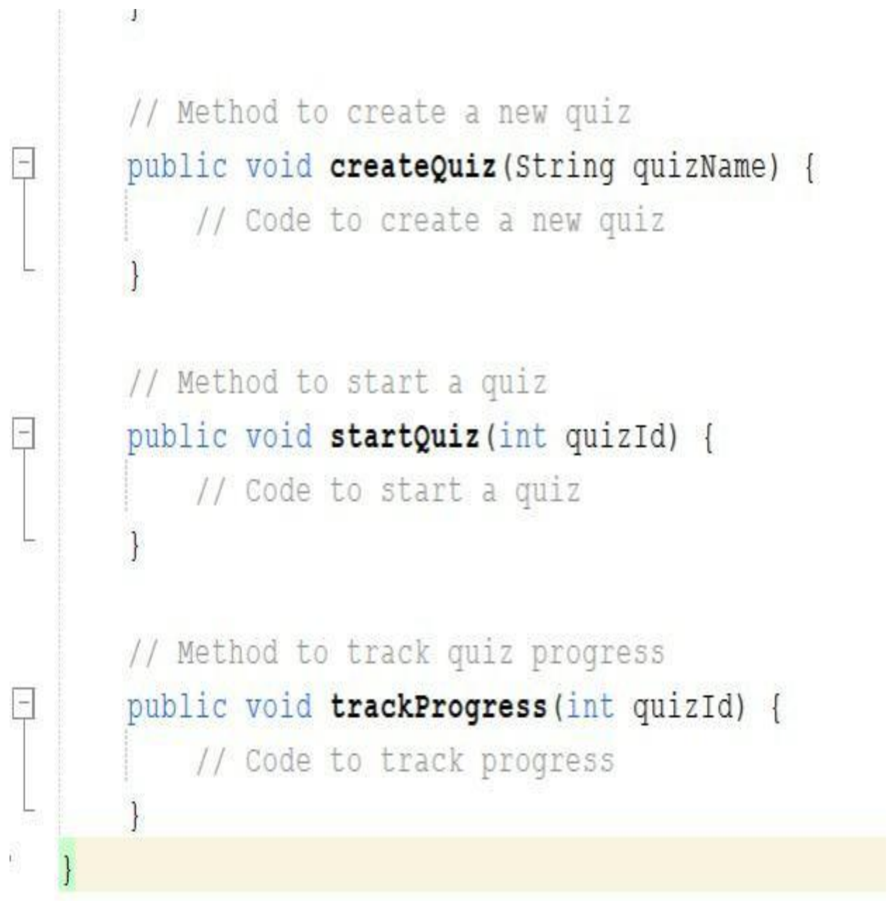
This code defines a Singleton Pattern for managing scores:

1. ScoreManager Class: Handles score-related operations like calculation and storage.
2. 2. instance Variable: Holds the single instance of ScoreManager.
3. 3. Private Constructor: Prevents creating multiple instances of the class.
4. 4. getInstance Method: Ensures only one instance is created and provides access to it.
5. 5. Methods:calculateScore(int quizId): Calculates scores for a specific quiz.storeResults(int quizId, int score): Stores quiz results in a database or file.



```
public class SingletonPaternn {

    public static void main(String[] args) {
        // Get the single instance of QuizManager
        QuizManager quizManager = QuizManager.getInstance();

        // Create a new quiz
        quizManager.createQuiz("Selected Labs");

        // Start the quiz
        quizManager.startQuiz(1);

        // Track the progress of the quiz (optional)
        quizManager.trackProgress(1);

        // Get the single instance of ScoreManager
        ScoreManager scoreManager = ScoreManager.getInstance();

        // Calculate the score for the quiz
        scoreManager.calculateScore(1);

        // Store the results
        scoreManager.storeResults(1, 85); // For example, the score is 85
    }
```

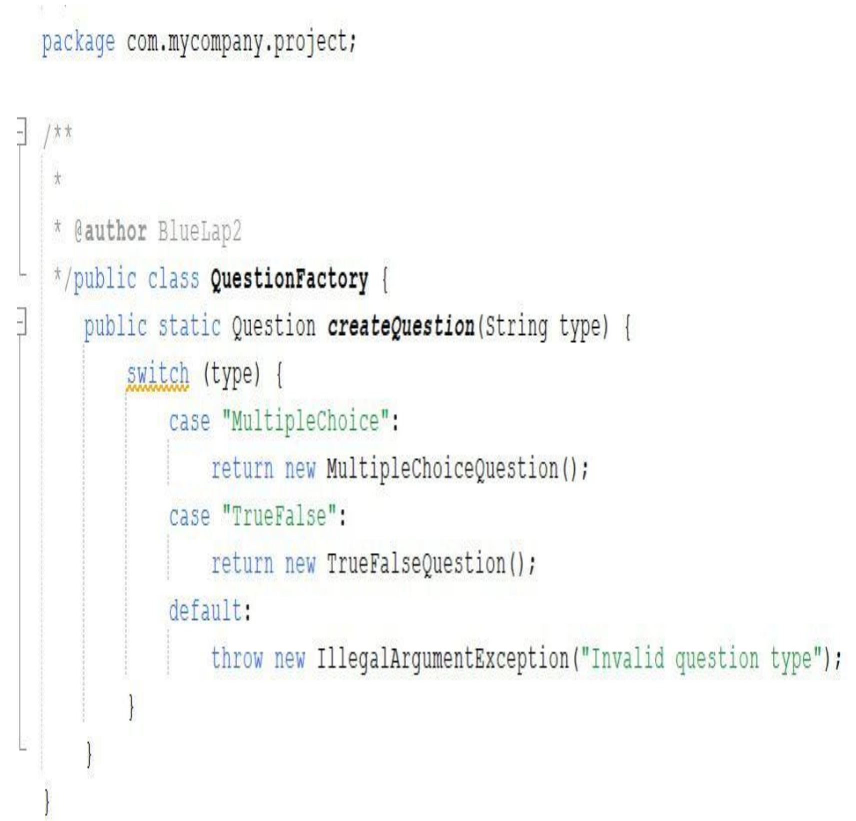
2. Concrete Classes (MultipleChoiceQuestion, TrueFalseQuestion, ShortAnswerQuestion):Extend the Question class and implement the display() method to specify how each type of question is displayed.



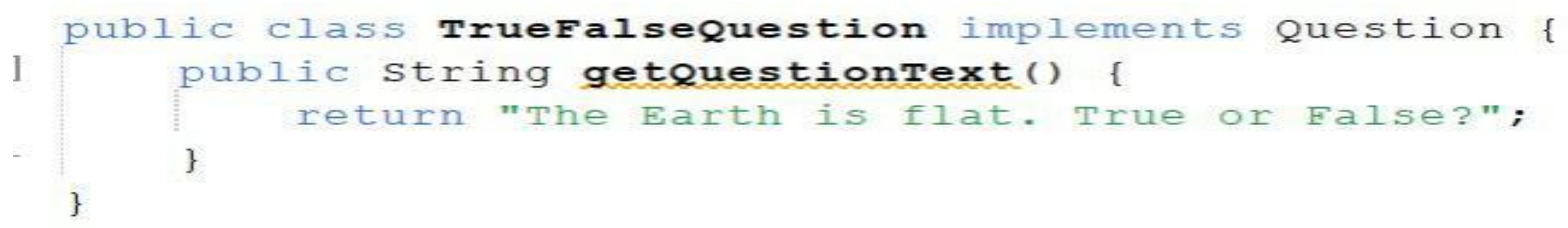
```
    }

    // Method to create a new quiz
    public void createQuiz(String quizName) {
        // Code to create a new quiz
    }

    // Method to start a quiz
    public void startQuiz(int quizId) {
        // Code to start a quiz
    }

    // Method to track quiz progress
    public void trackProgress(int quizId) {
        // Code to track progress
    }
}
```

Factory Pattern

1. Abstract Class Question:Defines a common structure for all question types with an abstract method display().

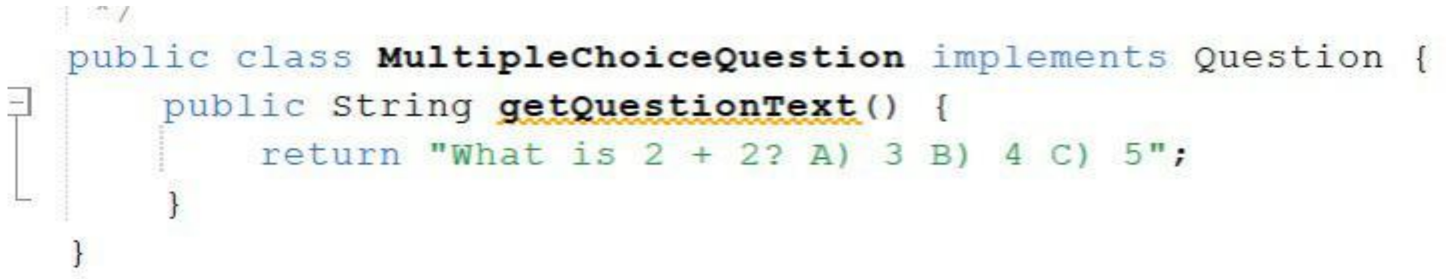


```
package com.mycompany.project;

/**
 *
 * @author BlueLap2
 */public class QuestionFactory {
    public static Question createQuestion(String type) {
        switch (type) {
            case "MultipleChoice":
                return new MultipleChoiceQuestion();
            case "TrueFalse":
                return new TrueFalseQuestion();
            default:
                throw new IllegalArgumentException("Invalid question type");
        }
    }
}
```



```
public class TrueFalseQuestion implements Question {
    public String getQuestionText() {
        return "The Earth is flat. True or False?";
    }
}
```

2. Concrete Classes (MultipleChoiceQuestion, TrueFalseQuestion, ShortAnswerQuestion):Extend the Question class and implement the display() method to specify how each type of question is displayed.

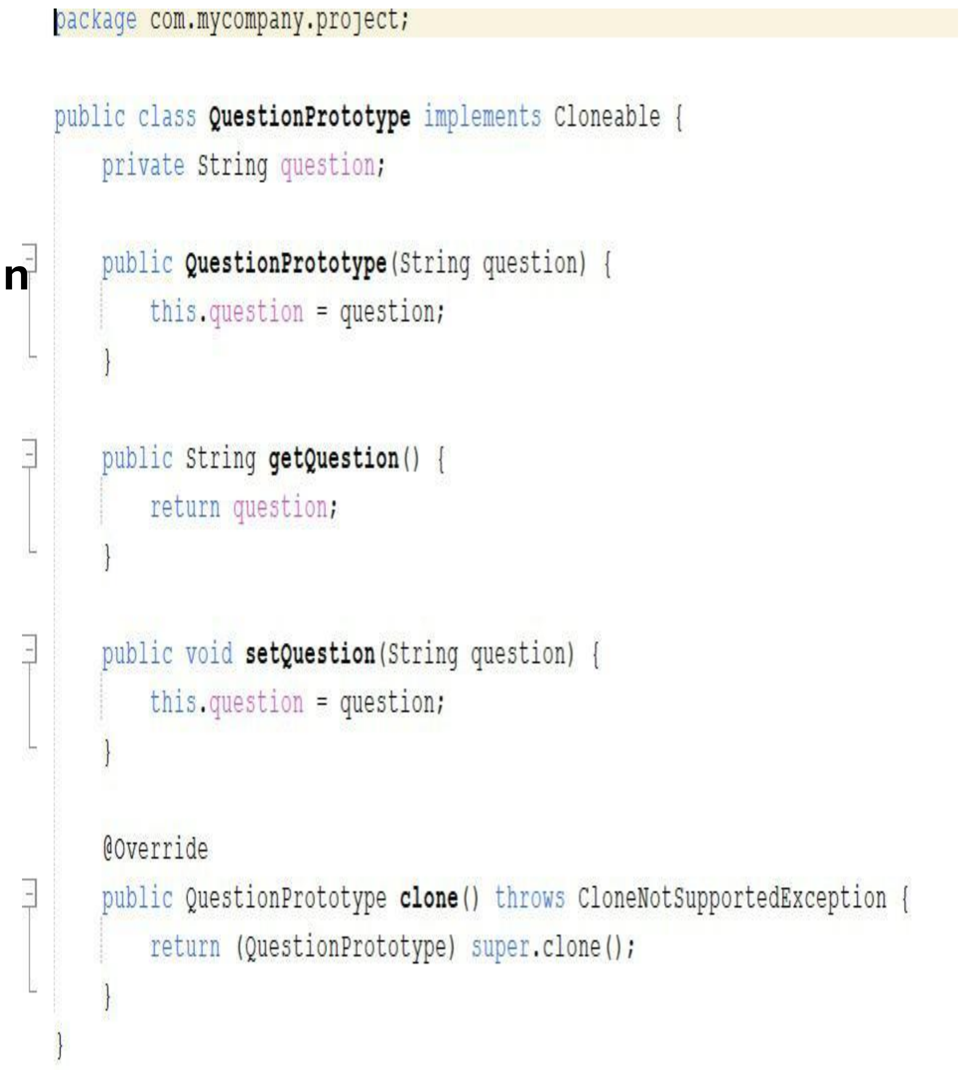


```
public class MultipleChoiceQuestion implements Question {
    public String getQuestionText() {
        return "What is 2 + 2? A) 3 B) 4 C) 5";
    }
}
```

3. QuestionFactory Class:Contains a static method createQuestion() that takes a type as input and returns an instance of the corresponding question type (MultipleChoice, TrueFalse, or ShortAnswer).

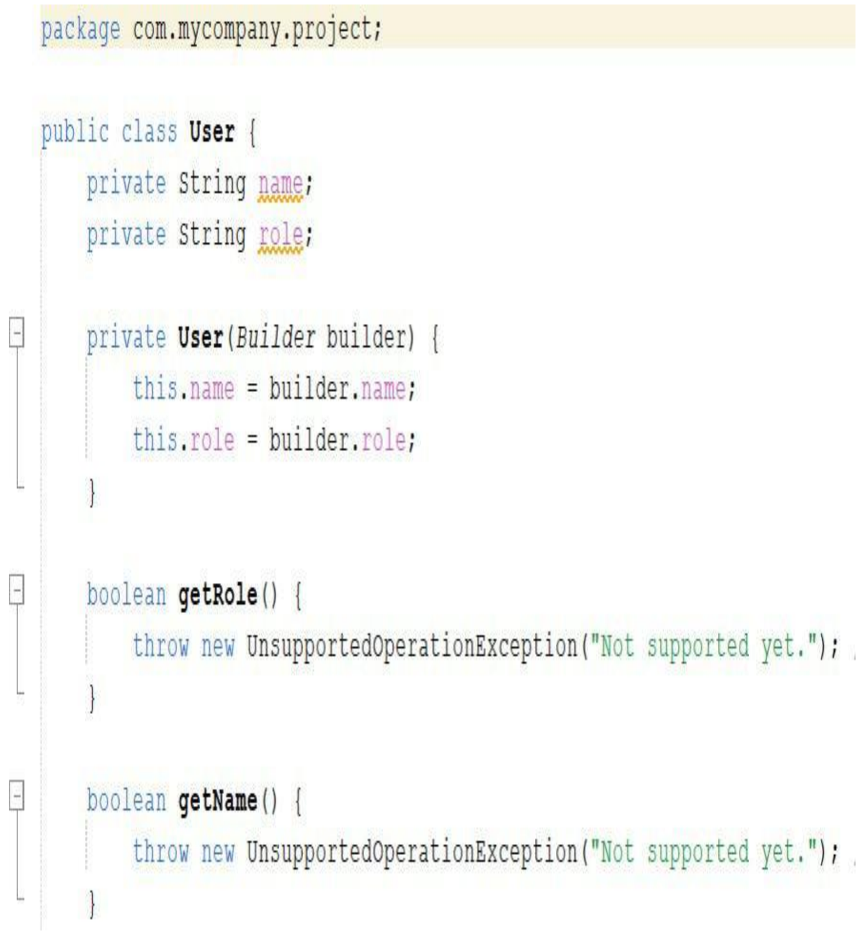
Prototype pattern

1. **Class QuestionPrototype:Represents a prototype for a question object.Contains a private field question to store the question text.Provides getter and setter methods (getQuestion and setQuestion) to access and modify the question field.**
2. **2. Constructor:QuestionPrototype(String question) initializes the question field with the given text.**
3. **3. Cloning with Cloneable:The class implements the Cloneable interface, which allows creating a copy of an object.The clone() method is overridden to perform a shallow copy using super.clone().**
4. **4. Usage:This pattern is useful for creating duplicate objects with similar properties without reinitializing them.By calling clone(), a new instance of QuestionPrototype is created with the same question value.**



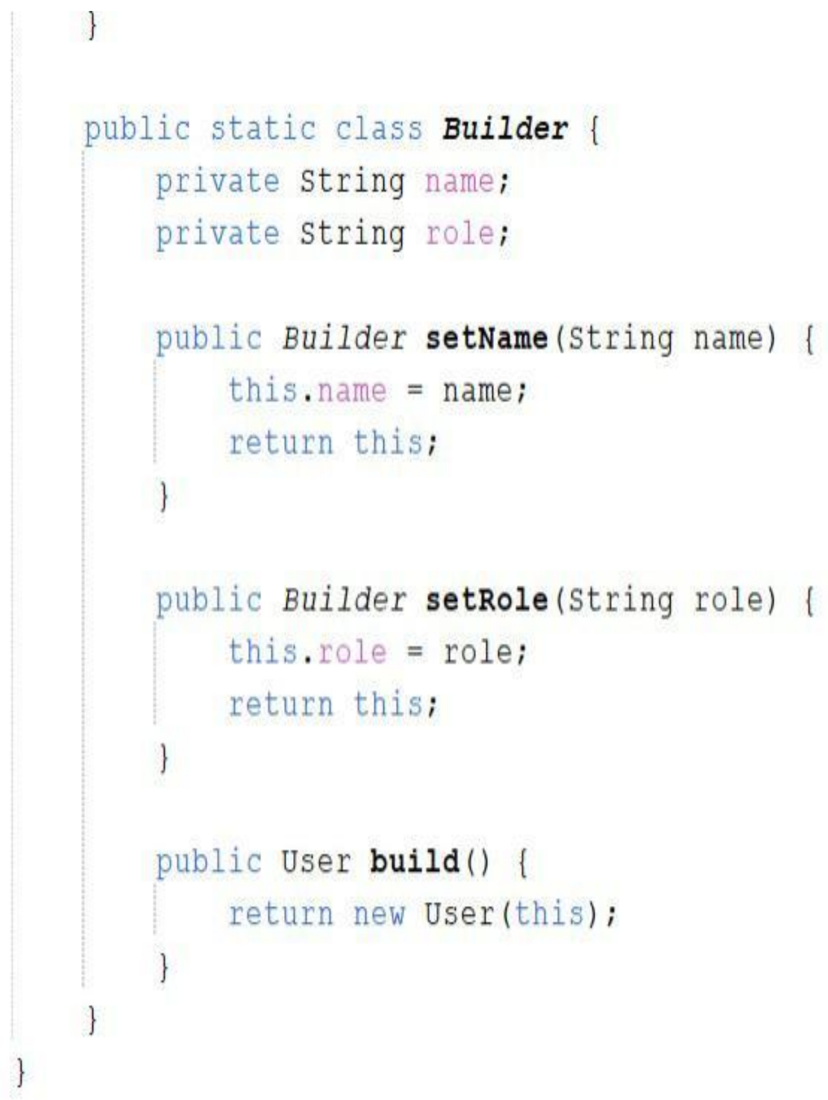
```java
package com.mycompany.project;

public class QuestionPrototype implements Cloneable {
    private String question;

    public QuestionPrototype(String question) {
        this.question = question;
    }

    public String getQuestion() {
        return question;
    }

    public void setQuestion(String question) {
        this.question = question;
    }

    @Override
    public QuestionPrototype clone() throws CloneNotSupportedException {
        return (QuestionPrototype) super.clone();
    }
}
```

4. Usage:This pattern is useful for creating duplicate objects with similar properties without reinitializing them.By calling clone(), a new instance of QuestionPrototype is created with the same question value.



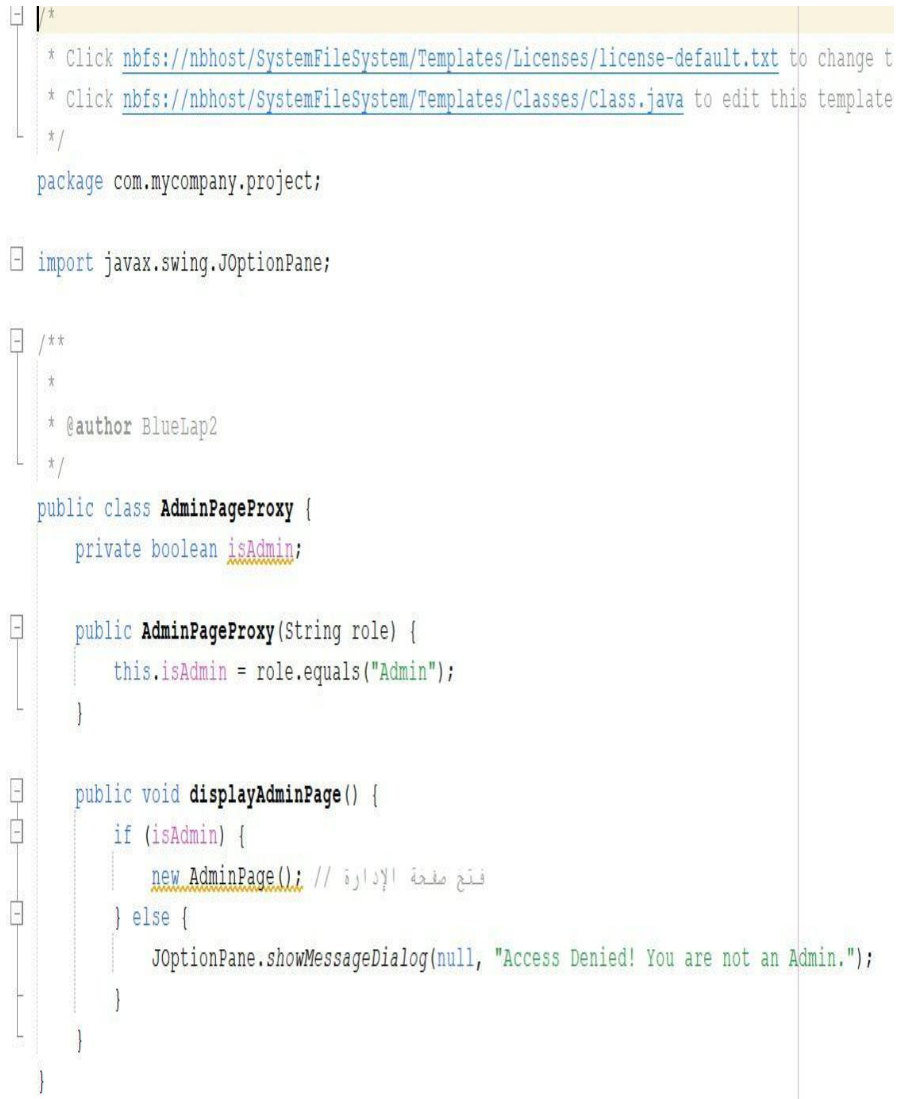
```
package com.mycompany.project;

public class User {
    private String name;
    private String role;

    private User(Builder builder) {
        this.name = builder.name;
        this.role = builder.role;
    }

    boolean getRole() {
        throw new UnsupportedOperationException("Not supported yet.");
    }

    boolean getName() {
        throw new UnsupportedOperationException("Not supported yet.");
    }
```

1. ***User Class:The class has private fields, name and role, to store the user's data.The constructor of the User class is private, meaning that you cannot directly create an object of User using the new keyword. Instead, you must use the Builder class***
2. ***.2. Getter Methods:The getName() and getRole() methods are placeholders (not implemented), typically used to retrieve the User object properties. These would normally be implemented to return the values of name and role.***



```
    }

    public static class Builder {
        private String name;
        private String role;

        public Builder setName(String name) {
            this.name = name;
            return this;
        }

        public Builder setRole(String role) {
            this.role = role;
            return this;
        }

        public User build() {
            return new User(this);
        }
    }
}
```

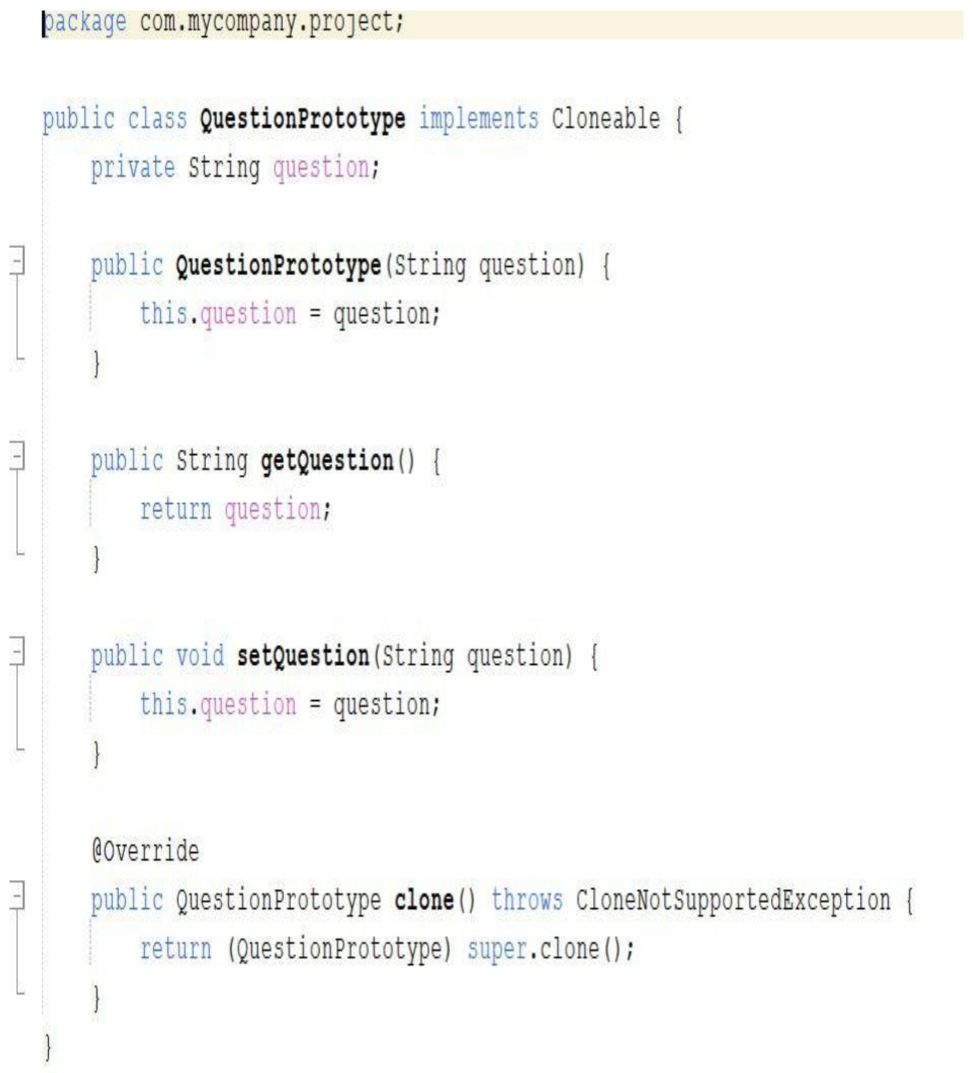
3. Static Inner Builder Class:The Builder class is nested inside the User class and acts as a helper for constructing User objects.It has fields corresponding to the User class (name and role).Methods like setName() and setRole() allow setting these fields. These methods return the Builder object itself, enabling method chaining (calling multiple methods in a single statement).The build() method takes the current values of Builder fields and creates a new User object with those values.



```
/*
 * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change t
 * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Class.java to edit this template
 */
package com.mycompany.project;

import javax.swing.JOptionPane;

/**
 *
 * @author BlueLap2
 */
public class AdminPageProxy {
    private boolean isAdmin;

    public AdminPageProxy(String role) {
        this.isAdmin = role.equals("Admin");
    }

    public void displayAdminPage() {
        if (isAdmin) {
            new AdminPage(); // فتح صفحة الإدارة
        } else {
            JOptionPane.showMessageDialog(null, "Access Denied! You are not an Admin.");
        }
    }
}
```

Proxy Pattern

1. ***Quiz Interface:Defines the startQuiz() method, which both the RealQuiz and QuizProxy implement.***
2. ***2. RealQuiz Class:Represents the actual quiz logic. It has a quizName and starts the quiz when startQuiz() is called***
3. ***.3. QuizProxy Class:Acts as an intermediary to control access to RealQuiz.Checks if the user is authenticated before allowing the quiz to start.Uses lazy initialization to create the RealQuiz object only when needed.***
4. ***4. Main Class (OnlineQuizProxyDemo):Demonstrates two scenarios:A user who is not authenticated is denied access.An authenticated user is allowed to start the quiz.***



```java
package com.mycompany.project;

public class QuestionPrototype implements Cloneable {
    private String question;

    public QuestionPrototype(String question) {
        this.question = question;
    }

    public String getQuestion() {
        return question;
    }

    public void setQuestion(String question) {
        this.question = question;
    }

    @Override
    public QuestionPrototype clone() throws CloneNotSupportedException {
        return (QuestionPrototype) super.clone();
    }
}
```

Thank you