Project Of

Selected lab in Software

***Dr. Mahmoud Basiouni***

***Eng. Israa Fikri***

***By Team / Smart Tracking Project idea: Online Ǫuiz System (El-Fayoum)***

***Members Of Team:***

# Mennatallah Soliman Mohamed (ID: 21-01087)

1. ***Safy Hussein Muawad (ID: 21-01347)***
2. ***Mariam Mohamed Rajab (ID: 21-0100C)***
3. ***Hams Khateeb Mousa (ID: 21-007C2)***
4. ***Shahd Ashraf Ramadan (ID: 21-01C03) C.Mohamed Ahmed Osman (ID: 21-01204)***

***Description:*** A desktop application for creating, taking, and scoring quizzes. Patterns Used:

* + Factory
  + Singleton
  + Builder
  + Proxy
  + Adapter

1. ***Factory Pattern:***

* Task: **Factory for Question Types** (e.g., Multiple Choice, True/False, Short Answer) to generate questions based on the quiz setup.
* Task: **Factory for User Roles** (e.g., Admin, Student, Teacher) to create user profiles and assign access levels.

1. **Singleton Pattern:**

* Task: Quiz Manager - A **Singleton class that manages the creation, start**, and progress of quizzes.
* Task: Score Manager - **A Singleton to handle scoring and storing** results for quizzes.

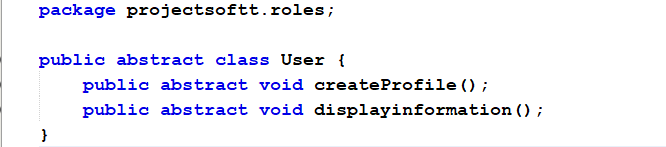
1. ***Factory Pattern:***

* Task: **Factory for User Roles** (e.g., manager, Student, Teacher) to create user profiles and assign access levels.

***Firstly***: We will make Abstract class that called ***User*** class that have two Methods :

## createProfile()

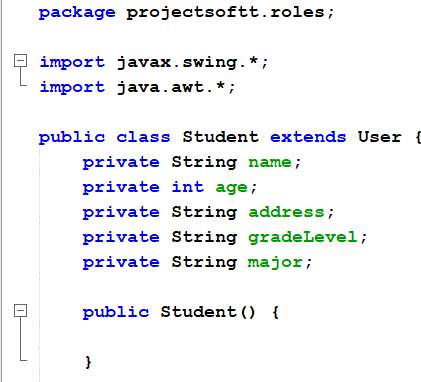
* 1. **displayinformation()**

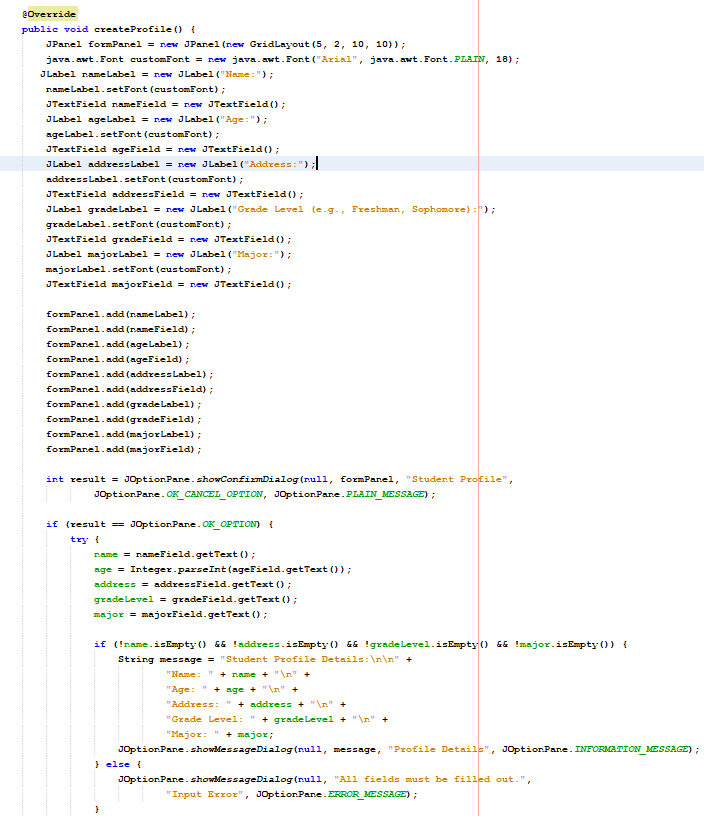
****

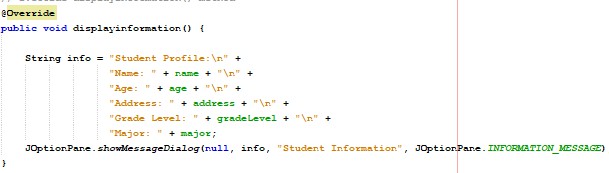
Secondly: we will make 3 **classes for** manager, Student, Teacher that every class will

**Extends *User*** class and override two Methods of User class

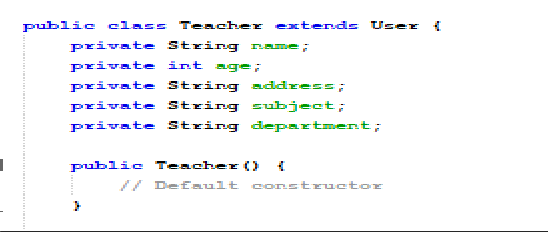
**Student class :**

****

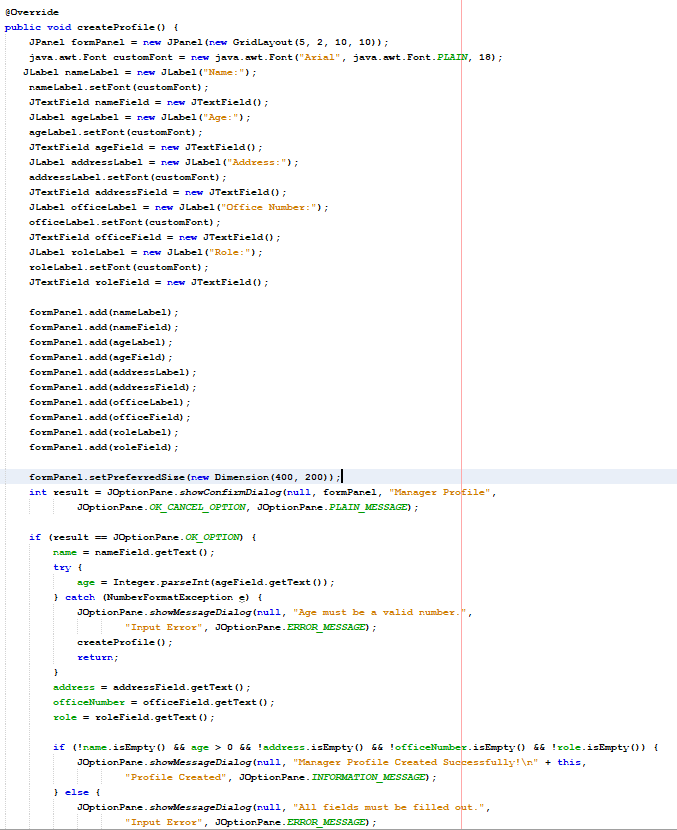
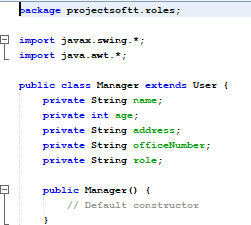


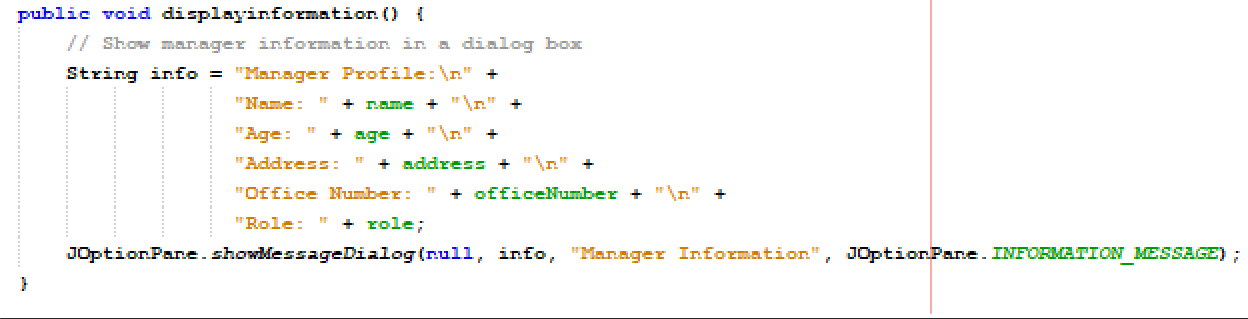


**Teacher Class:**



**Admine class:**



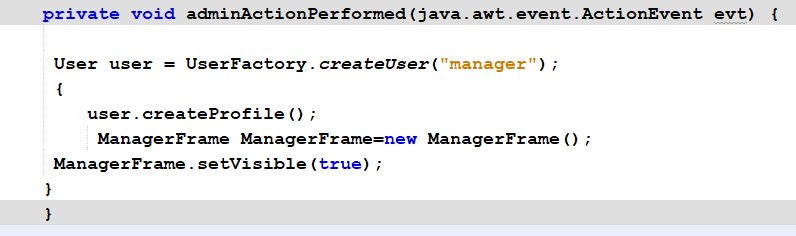
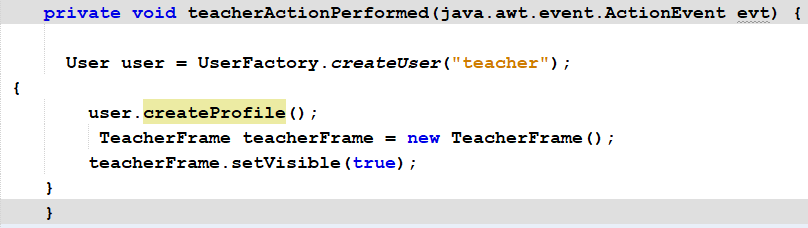


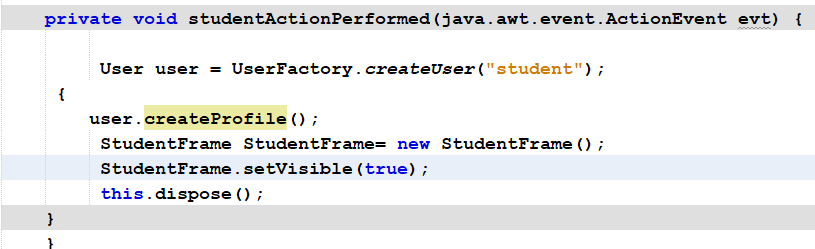
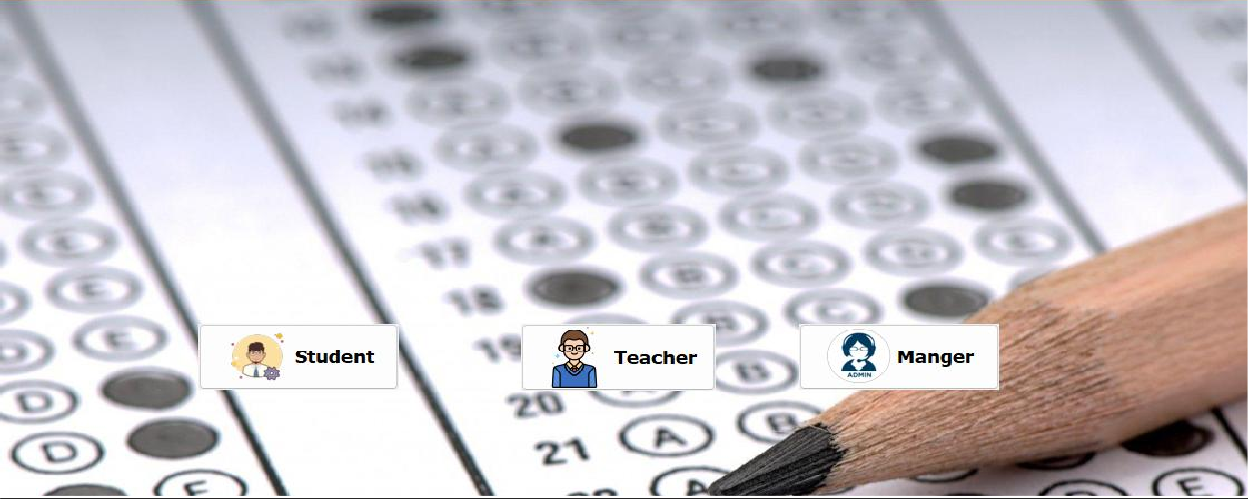
Now We will Make **Factory class** it is responsible for establishing object From 3 class :



Here we use singleton with factory when make object

**Now we will go to Gui to make this and use factory class :**

****

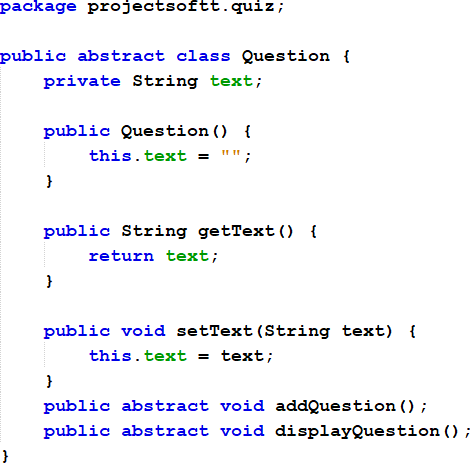


***1-Factory Pattern:***

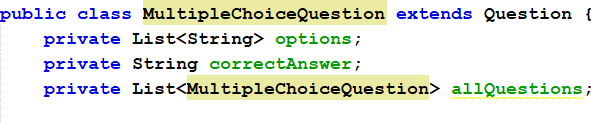
o Task: **Factory for Question Types** (e.g., Multiple Choice, True/False) to generate questions based on the quiz setup.

So we make class for ***Multiple Choice, True/False***

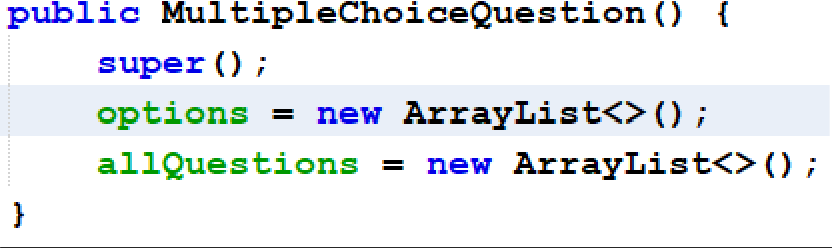
We will make Abstract class called Question class that have 4 methods



Now we make ***MultipleChoiceǪuestion*** extends ***Ǫuestion class***

******

* ***options:*** List to hold the multiple-choice options presented to the user.
* ***correctAnswer:*** A string that stores the correct answer for the question.
* ***allǪuestions:*** A list that stores all instances of MultipleChoiceQuestion created.

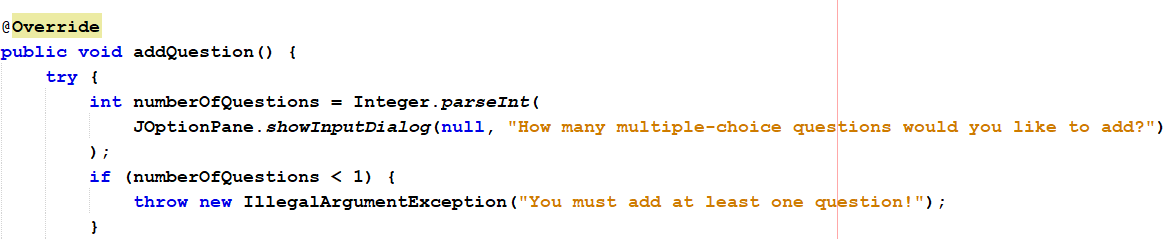


This is constructor :

super():To summon the constructor of ***Ǫuestion class***

***in options and allǪuestions will make a new instance of them when use constructor of MultipleChoiceǪuestio***

*now we make* ***addǪuestion()*** *methods:*

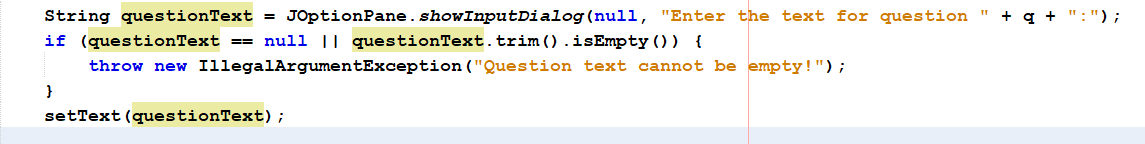
**

The method starts by prompting the user to input the number of questions The input is received as a string and is parsed into an integer

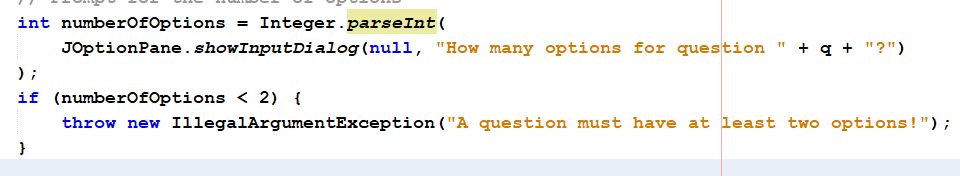
**Loop for Adding Each Question**

****

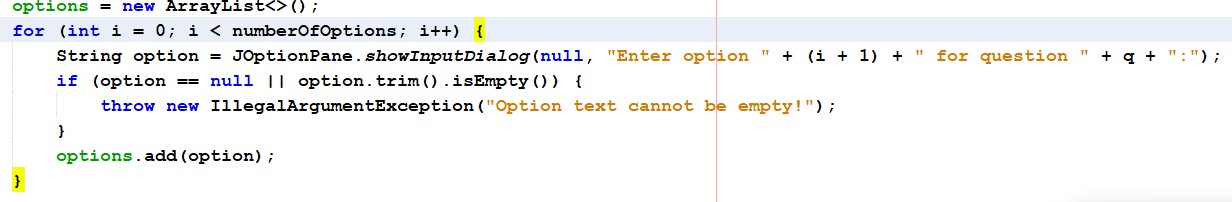
**Question Text Input**:



**Prompt for Number of Options**:

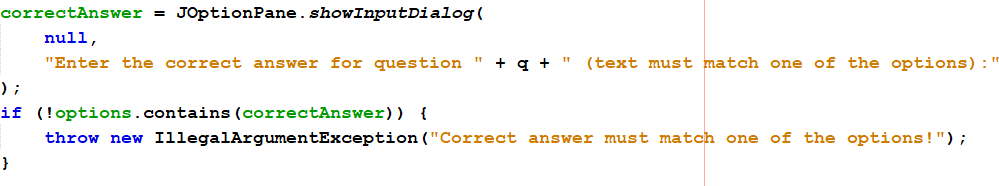


**Gathering Options**:



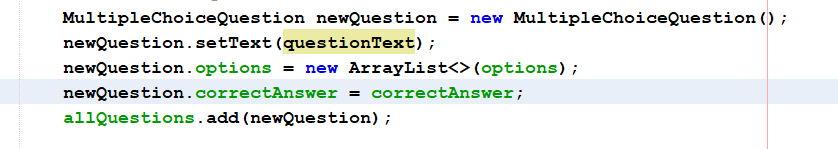
* A new ArrayList is initialized to store the options.
* The user is then prompted to enter each option in a loop that runs for the specified number of options.
* Each option is validated for non-null and non-empty inputs.

**Correct Answer Input**:



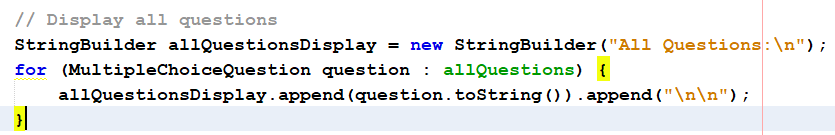
* After collecting the options, the user is prompted to enter the correct answer.
* The method checks to ensure that the entered correct answer matches one of the previously entered options.

**Storing the Question**:

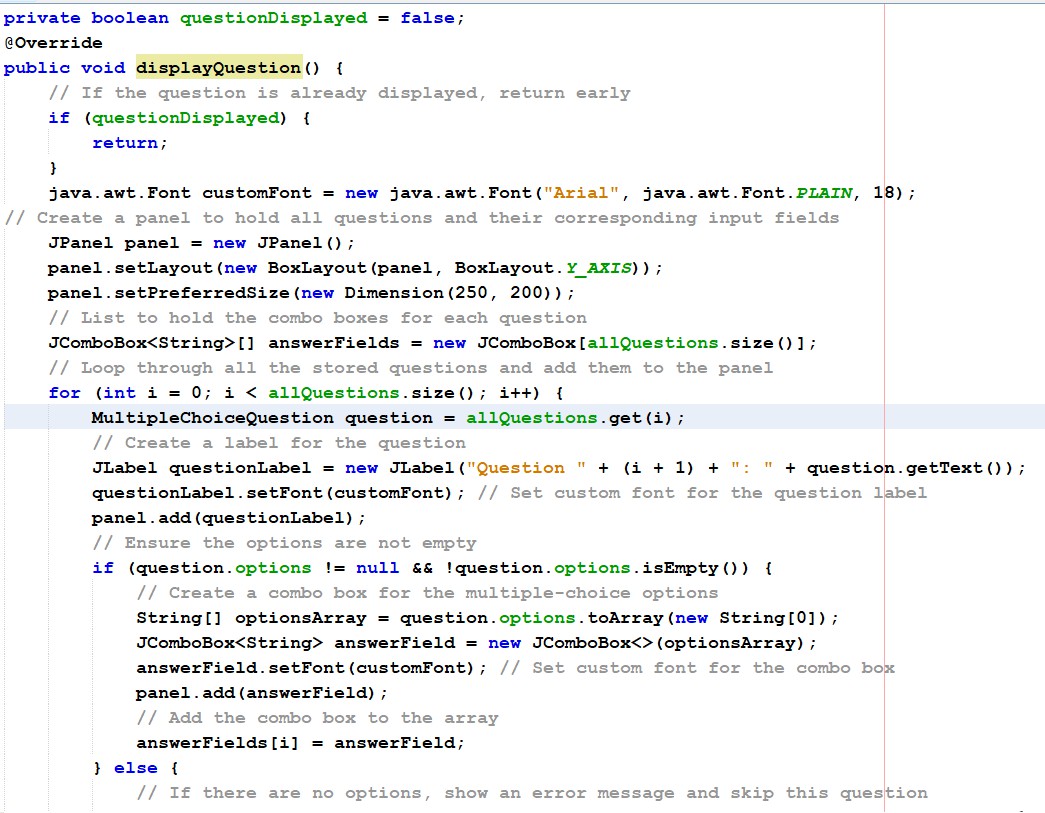


* A new instance of MultipleChoiceQuestion is created.
* The question text, options, and correct answer are set for this instance.
* This new question is added to the allQuestions list.

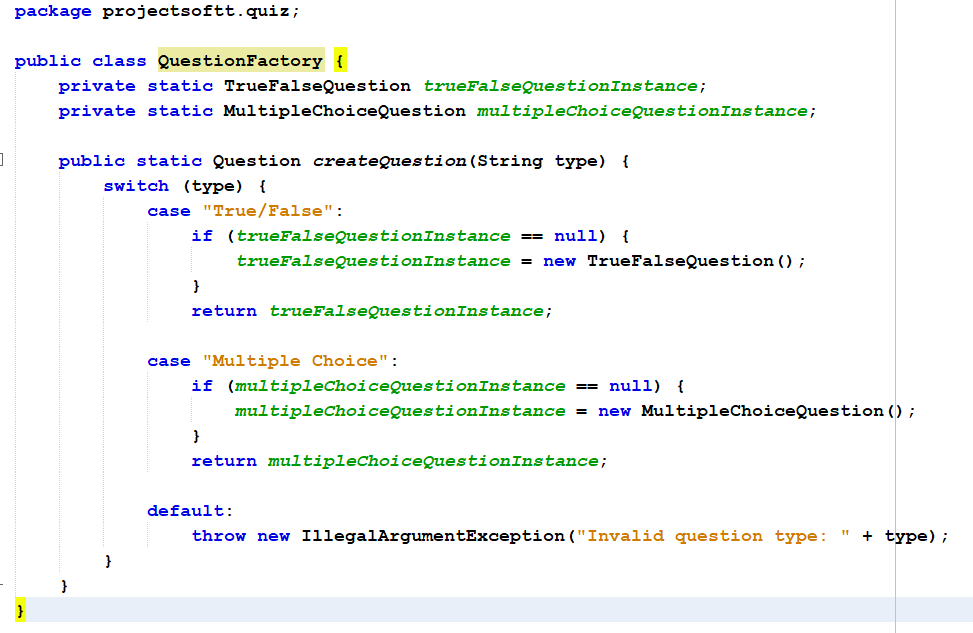
**Display all questions:**

****

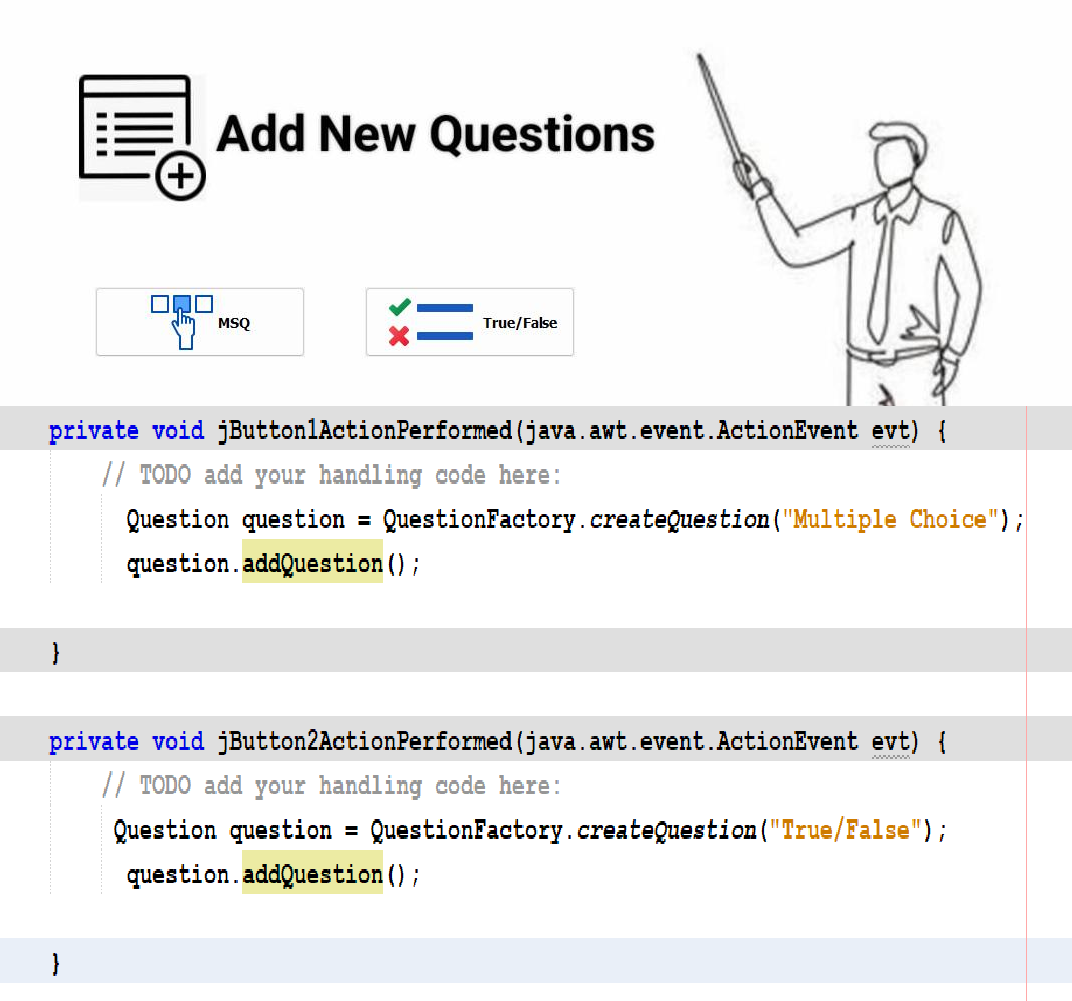
Now we will make displayQuestion():



We make the same for TrueFalseQuestion() class We make the QuestionFactory():



Now we will go to **GUI:**

****

1. **Singleton Pattern:**

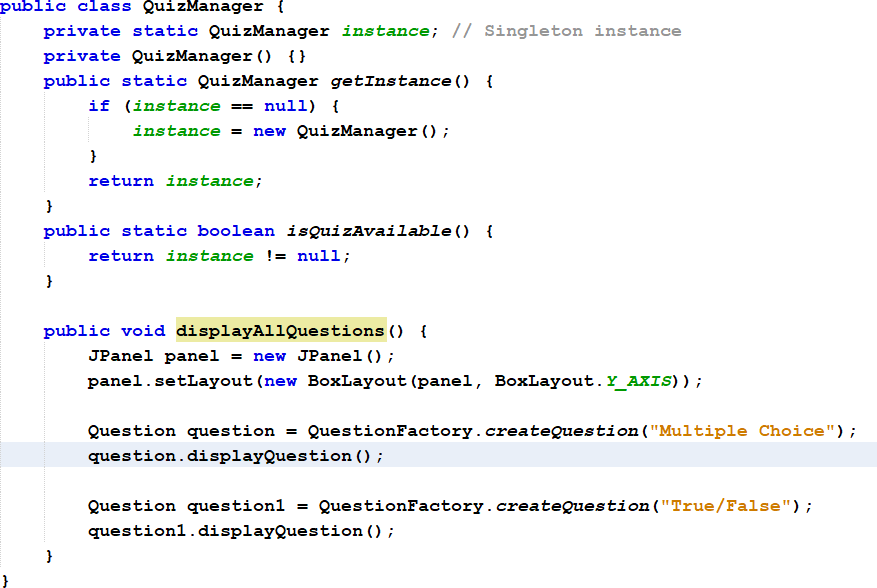
* Task: Quiz Manager - A **Singleton class that manages the creation, start**,of quizzes. Now how we make Quiz Manager **creation and start quizzes?**

**It make that when make instance from** Quiz Manager class and from student will use

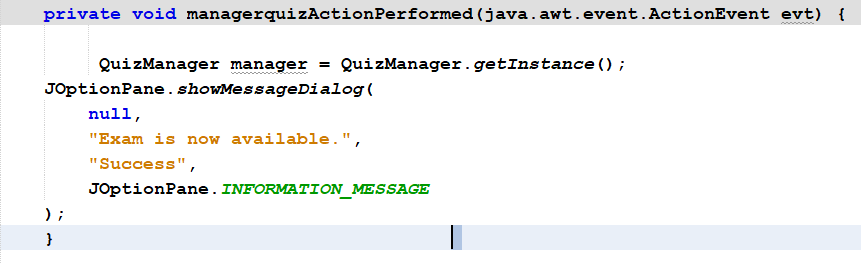
**instance from** Quiz Manager to use displayAllQuestions() that in Quiz Manager class

So student will use isQuizAvailable() to know if get **make instance from** Quiz Manager class or not

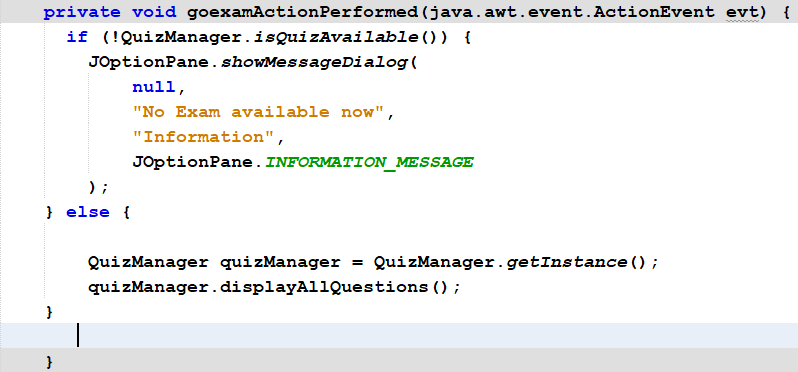
We will make Quiz Manager class that have **3 methods**

****

In GUI :



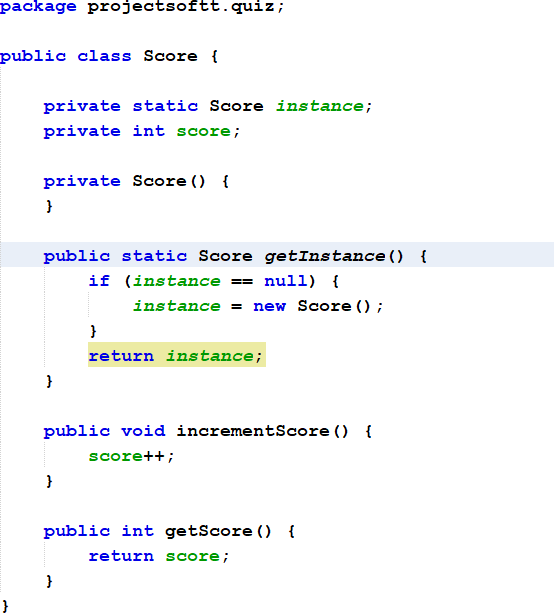
So in Student class we will see :



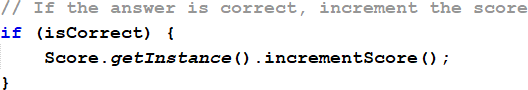
1. **Singleton Pattern:**

* Task: Score Manager - **A Singleton to handle scoring and storing** results for quizzes.

We will make one instance of score class( ***lazzy singleton*** ) to increment when answers is correct



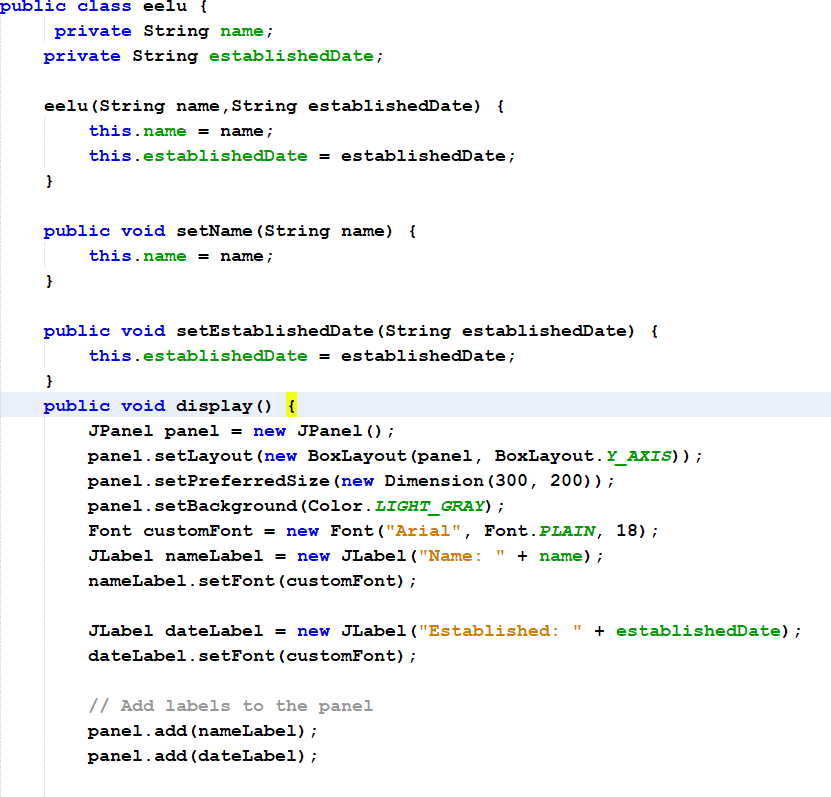
It will used in displayQuestion() in both MultipleChoiceQuestion and TrueFalseQuestion:



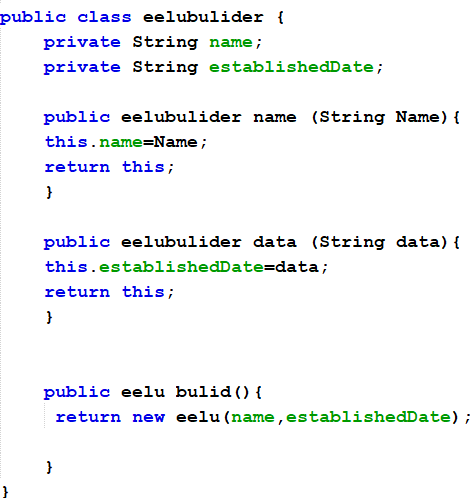
1. **Builder:**

sed to construct complex objects step by step. It allows for more controlled and flexible construction of an object by separating the construction process from the representation.

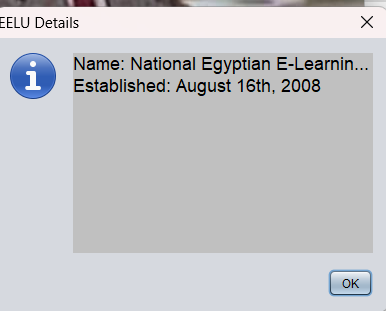
This pattern is particularly useful when an object requires multiple attributes, some of which may be optional or have default values.

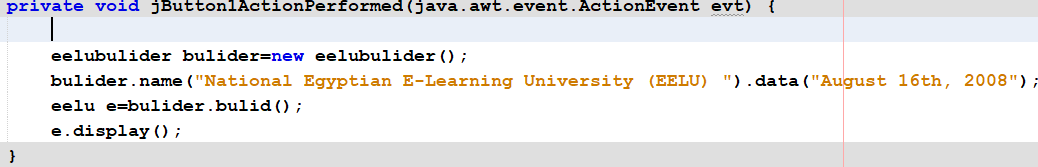
So we male contractor by any value that I want to add We will make two class :

We will make builder class :



In GUI :



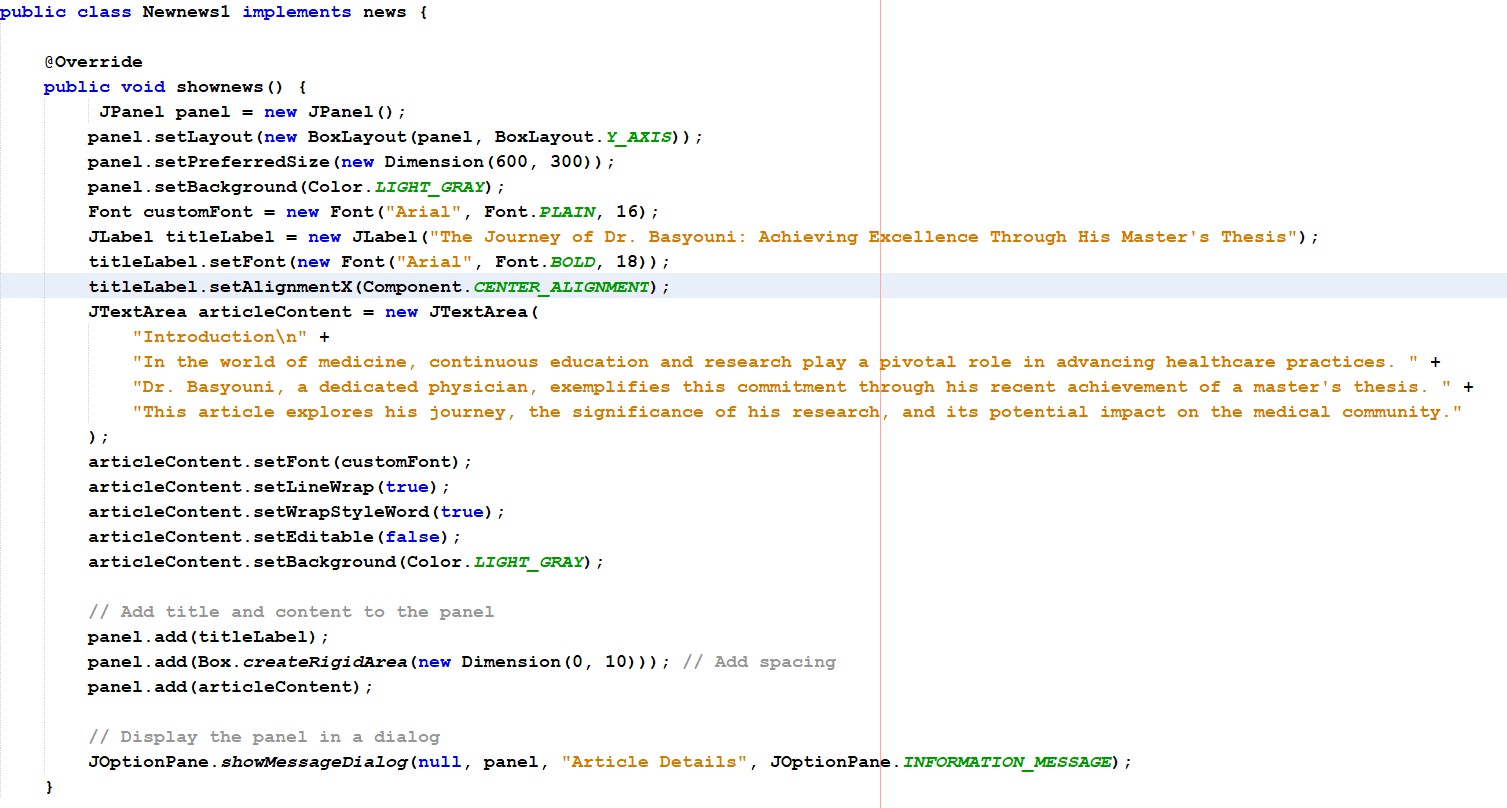


# Adapter

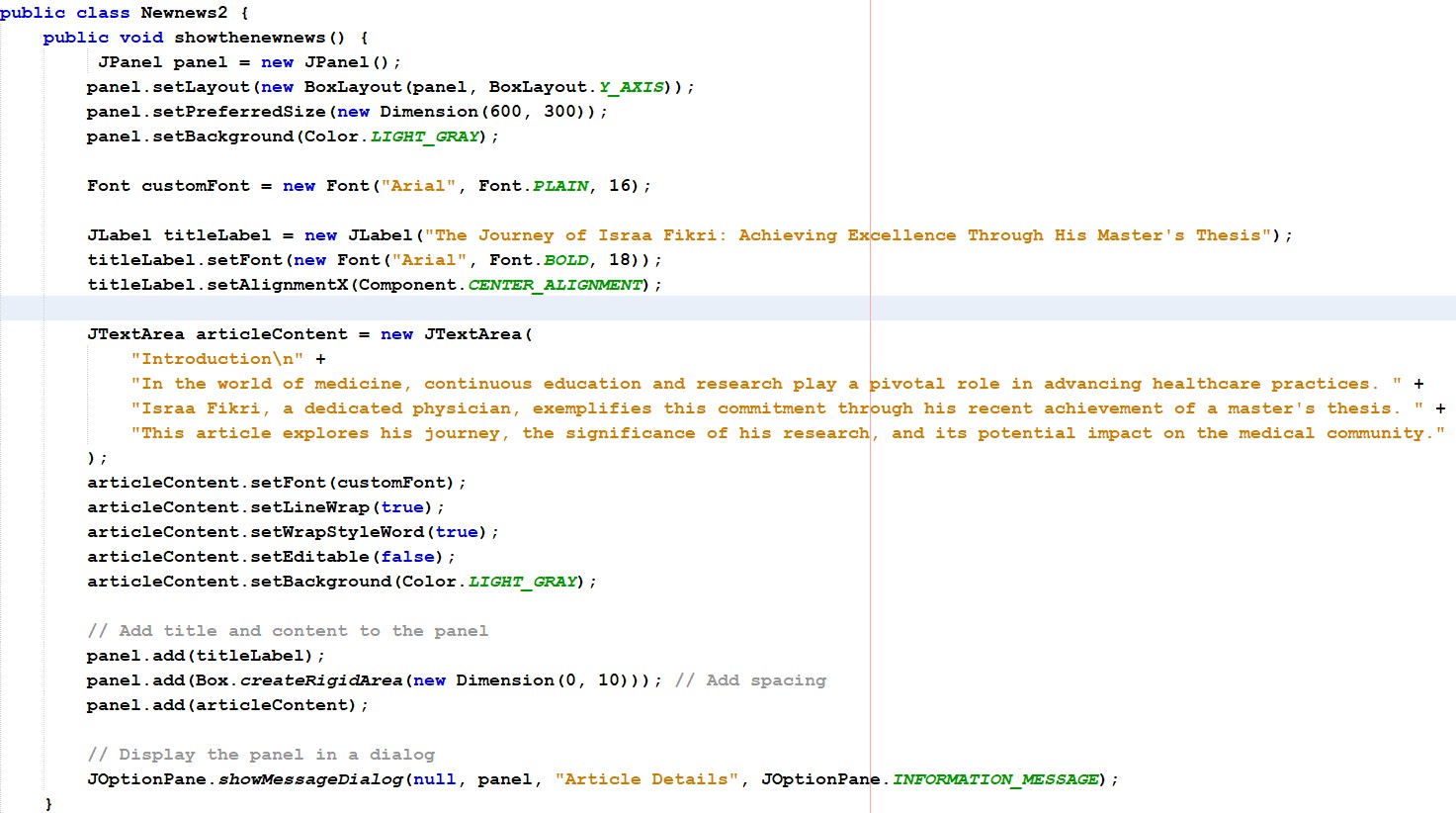
*is a structural design pattern that allows objects with incompatible interfaces to work*

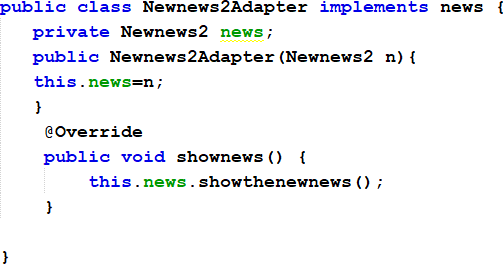
*together. It acts as a bridge between two incompatible interfaces, enabling clients to use those interfaces as if they were compatible. This is particularly useful when you have a class that requires a specific interface, but you want to use classes that don’t match that interface****.***

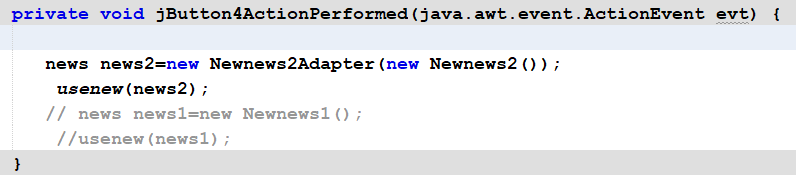
***A close-up of a white background  Description automatically generated***

******

Now when add A new news we ***will not implements interface*** we will make adapter for it that will ***implements interface.***

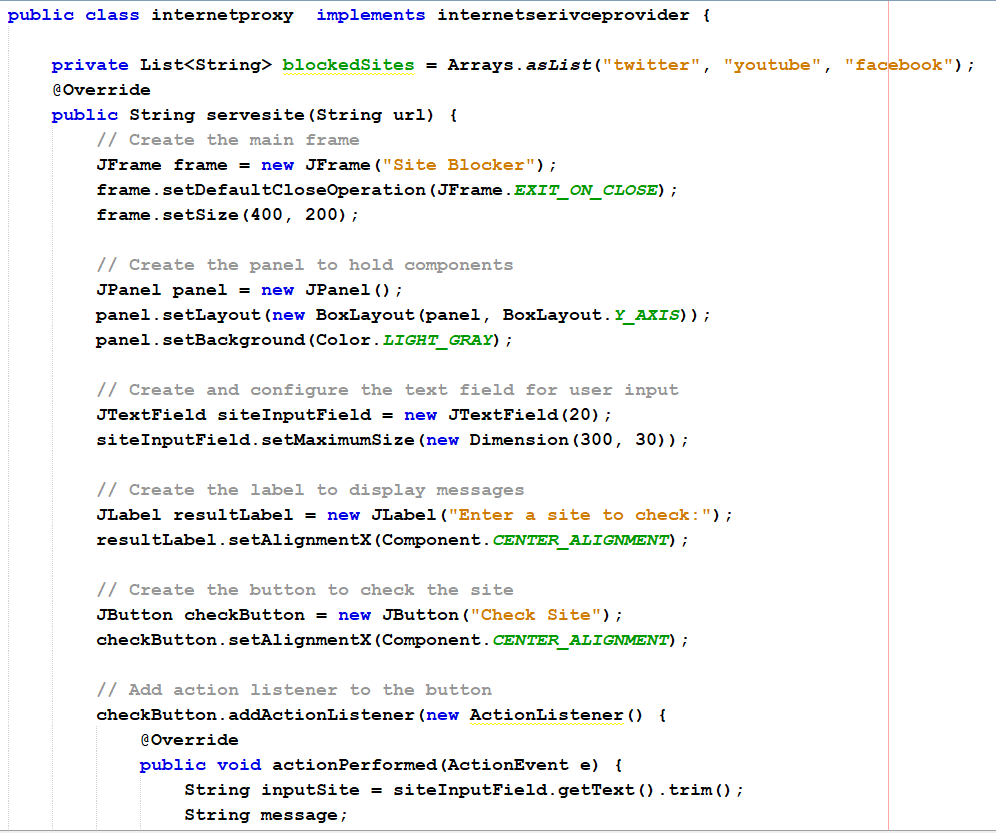
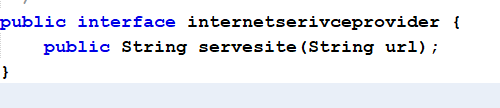
******

******

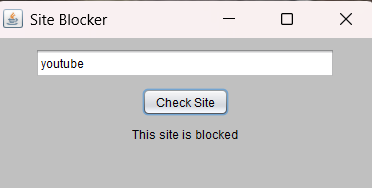
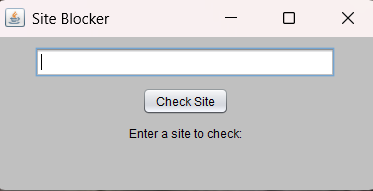
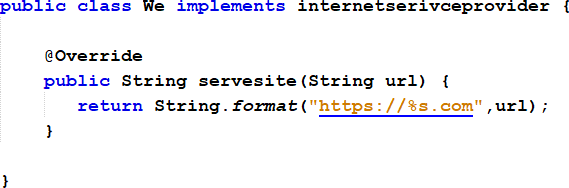
******

1. **Proxy:**

## is a structural design pattern that provides a surrogate or placeholder for another object to control access to it. It involves creating a proxy class that represents the real object, allowing for control over operations being performed, such as lazy loading, access control

****



****

Githup of project : https://github.com/MohamedOsman30/projectsoftware