Joseph Scherreik
SE 310
Final Exam

1) Bridge Pattern

Classes:

- Document
- Test
- Survey
- Question
- All Inherited Implementations Related to Question
  (All classes in the questions package)

In this implementation we are decoupling implementations of Survey and Test. We have an abstract class Document. This abstraction defines the interface of the object being implemented. It's refined abstractions are Survey and Test. Survey and Test will use the question package to create lists of questions.The implementor in this project is the Question Class. Its concrete implementations are all the other types of questions in the package it's contained in (ValidDate, Essay, etc). In these concrete classes' implementations of all operations take place.

2) Adapter Pattern

Classes:

- Question
- Document
- All Inherited Implementations Related to Question
  (All classes in the questions package)

In this implementation we have lots of preexisting classes with lots of methods with similar implementation. Question is an interface template that the whole design of the package must be designed to. This will be the target. Thus, polymorphism is an effective solution for this issue. All the different inherited concrete implementations are implementing attributes/methods of Question. They also have their own unique class related attributes/methods. The so called "client" using Question to create other types of questions is Document since it needs to store a list of different kinds of questions. This allows Document to not have to understand what type of question it is so it will just interact with the target.

3) Facade

Classes:

- Document
- Survey
- Test
- Question
- All Inherited Implementations Related to Question
- Driver

Facades are used to simplify a complicated system. Here Document is a Facade for Driver to use all functionality with this program. All Driver needs to know is what Document does. This is a much simpler interface for it to use. It also will give it access to all other parts of the underlining subsystem with reduced options because most of the functionality in the subsystem is not required for Driver to use. This is especially true for the Questions package since none of the functionality inside is required for Driver to function. Document/Survey/Test use this package instead.

4) Facade

Classes:

- Question
- All Inherited Implementations Related to Question
- Document

Question also acts as a Facade for Document. It hides all functionality for the inherited implementations of Question. The only thing a Document needs to understand is that it stores questions. It is not responsible for anything else in the subsystem. That is handled by Question.

4) Template

Classes:

- Document
- Survey
- Test
- DocumentTemplate

The abstract class Document is a template for Survey and Test. They have many overlapping attributes and functions. Document will have template methods with basic overlapping functionality that use the interface DocumentTemplate to implement an algorithm. This is decoupled from the actual implementation. Survey and Test are concrete classes that implement the abstract operations inside of Document and are called with these abstract methods are called. Survey and Test also have their own unique implementations for some of these template methods.

5) Decorator

Classes:

- Question
- QuestionDecorator
- ShortAnswer
- ShortAnswerDecorator

Here the object question does not have all the functionality that is needed for ShortAnswer. The QuestionDecorator is an abstract decorator that can be applied to the other concrete implementations of Question. For the sake of time only one has been edited, but this pattern can be applied to all of the inherited concrete implementations of Question. This allows us to expand the functionality of Question. This can easily allow for Question to handle things related to ShortAnswer adding functionality to it using the decorators. It will create an abstract class that represents both the original and the new functions for it. The decorator place new function calls to get the correct order.

6) Observer

Classes:

- Observer
- TestObserver
- Document
- Test
- Driver

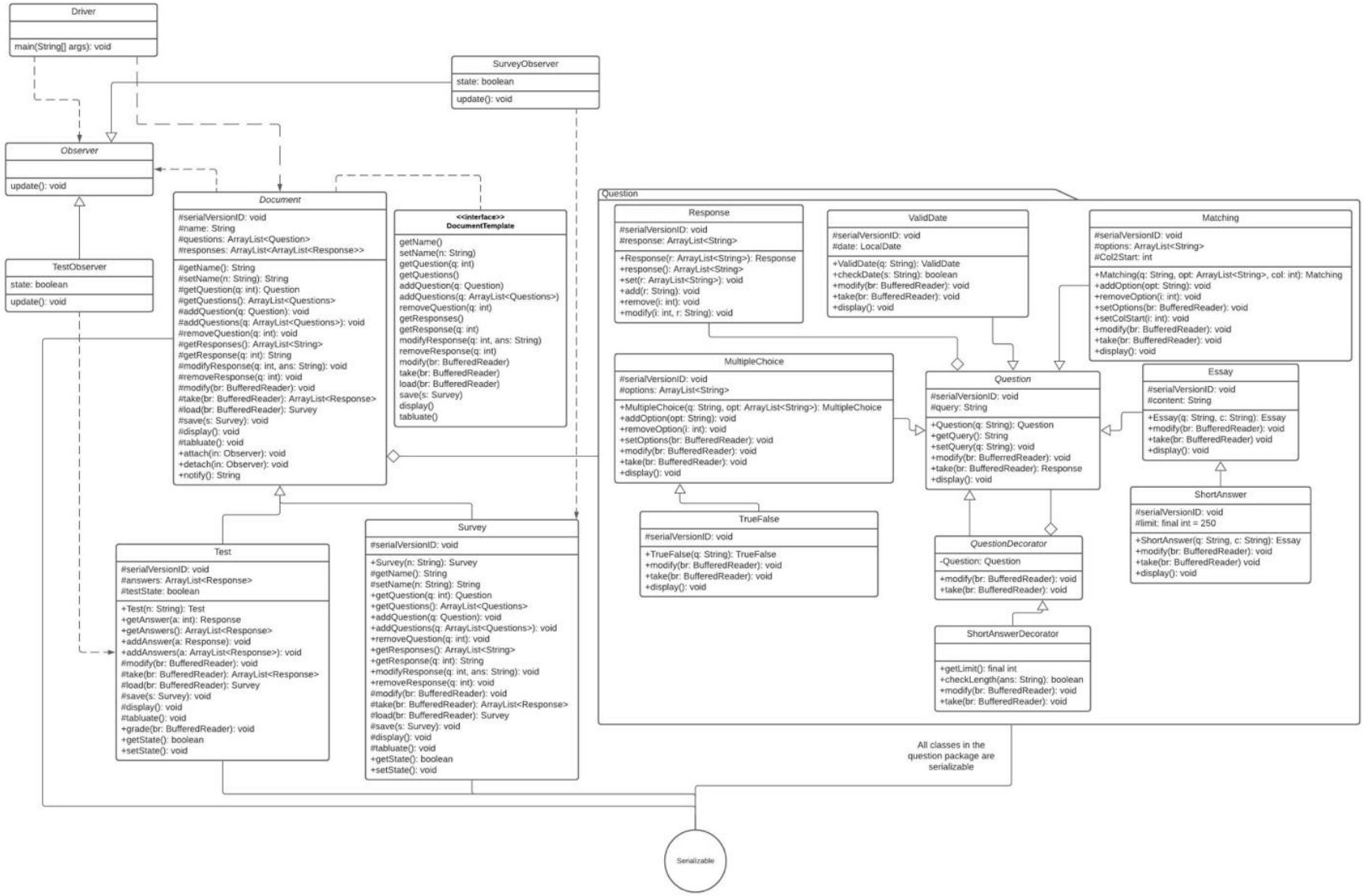
Here we want to observe the save state of a Test as the user takes it to make sure a user doesn't lose their answers. The subject of the Observer is Document since it is the abstract implementation of what we are trying to observe. This Observer class is abstract as well. Attach(), detach(), and notify() will handle adding an observer to the class to watch the state. When a save occurs on the concrete subject Test this will set the testState to true using its state manipulation methods. Then it will call the concrete observer's update() method which will then prompt the client Driver to understand that the Test has been saved to the user in the application.

7) Observer

Classes:

- Observer
- SurveyObserver
- Document
- Survey
- Question
- Driver

Here we want to observe the save state of a Survey as the user deletes a question.The subject of the Observer is Document since it is the abstract implementation of what we are trying to observe. This Observer class is abstract as well. Attach(), detach(), and notify() will handle adding an observer to the class to watch the state. When a save occurs on the concrete subject Survey this will set the testState to true using its state manipulation methods. Then it will call the concrete observer's update() method which will then prompt the client Driver to understand that the Question has been deleted to the user in the application.


Driver
main(String[] args): void
SurveyObserver
state: boolean
update(): void
Observer
update(): void
TestObserver
state: boolean
update(): void
Document
#serialVersionID: void
#name: String
#questions: ArrayList<Question>
#responses: ArrayList<ArrayList<Response>>
#getName(): String
#setName(n: String): String
#getQuestion(q: int): Question
#getQuestions(): ArrayList<Questions>
#addQuestion(q: Question): void
#addQuestions(q: ArrayList<Questions>): void
#removeQuestion(q: int): void
#getResponses(): ArrayList<String>
#getResponse(q: int): String
#modifyResponse(q: int, ans: String): void
#removeResponse(q: int): void
#modify(br: BufferedReader): void
#take(br: BufferedReader): ArrayList<Response>
#load(br: BufferedReader): Survey
#save(s: Survey): void
#display(): void
#tabluate(): void
+attach(in: Observer): void
+detach(in: Observer): void
+notify(): String
<<interface>>
DocumentTemplate
getName()
setName(n: String)
getQuestion(q: int)
getQuestions()
addQuestion(q: Question)
addQuestions(q: ArrayList<Questions>)
removeQuestion(q: int)
getResponses()
getResponse(q: int)
modifyResponse(q: int, ans: String)
removeResponse(q: int)
modify(br: BufferedReader)
take(br: BufferedReader)
load(br: BufferedReader)
save(s: Survey)
display()
tabluate()
Test
#serialVersionID: void
#answers: ArrayList<Response>
#testState: boolean
+Test(n: String): Test
+getAnswer(a: int): Response
+getAnswers(): ArrayList<Response>
+addAnswer(a: Response): void
+addAnswers(a: ArrayList<Response>): void
#modify(br: BufferedReader): void
#take(br: BufferedReader): ArrayList<Response>
#load(br: BufferedReader): Survey
#save(s: Survey): void
#display(): void
#tabluate(): void
+grade(br: BufferedReader): void
+getState(): boolean
+setState(): void
Survey
#serialVersionID: void
+Survey(n: String): Survey
#getName(): String
#setName(n: String): String
+getQuestion(q: int): Question
+getQuestions(): ArrayList<Questions>
+addQuestion(q: Question): void
+addQuestions(q: ArrayList<Questions>): void
+removeQuestion(q: int): void
+getResponses(): ArrayList<String>
+getResponse(q: int): String
+modifyResponse(q: int, ans: String): void
+removeResponse(q: int): void
#modify(br: BufferedReader): void
#take(br: BufferedReader): ArrayList<Response>
#load(br: BufferedReader): Survey
#save(s: Survey): void
#display(): void
#tabluate(): void
+getState(): boolean
+setState(): void
Question
Response
#serialVersionID: void
#response: ArrayList<String>
+Response(r: ArrayList<String>): Response
+response(): ArrayList<String>
+set(r: ArrayList<String>): void
+add(r: String): void
+remove(i: int): void
+modify(i: int, r: String): void
ValidDate
#serialVersionID: void
#date: LocalDate
+ValidDate(q: String): ValidDate
+checkDate(s: String): boolean
+modify(br: BufferedReader): void
+take(br: BufferedReader): void
+display(): void
Matching
#serialVersionID: void
#options: ArrayList<String>
#Col2Start: int
+Matching(q: String, opt: ArrayList<String>, col: int): Matching
+addOption(opt: String): void
+removeOption(i: int): void
+setOptions(br: BufferedReader): void
+setColStart(i: int): void
+modify(br: BufferedReader): void
+take(br: BufferedReader): void
+display(): void
MultipleChoice
#serialVersionID: void
#options: ArrayList<String>
+MultipleChoice(q: String, opt: ArrayList<String>): MultipleChoice
+addOption(opt: String): void
+removeOption(i: int): void
+setOptions(br: BufferedReader): void
+modify(br: BufferedReader): void
+take(br: BufferedReader): void
+display(): void
Question
#serialVersionID: void
#query: String
+Question(q: String): Question
+getQuery(): String
+setQuery(q: String): void
+modify(br: BufferredReader): void
+take(br: BufferedReader): Response
+display(): void
Essay
#serialVersionID: void
#content: String
+Essay(q: String, c: String): Essay
+modify(br: BufferedReader): void
+take(br: BufferedReader) void
+display(): void
TrueFalse
#serialVersionID: void
+TrueFalse(q: String): TrueFalse
+modify(br: BufferedReader): void
+take(br: BufferedReader): void
+display(): void
QuestionDecorator
-Question: Question
+modify(br: BufferedReader): void
+take(br: BufferedReader): void
ShortAnswer
#serialVersionID: void
#limit: final int = 250
+ShortAnswer(q: String, c: String): Essay
+modify(br: BufferedReader): void
+take(br: BufferedReader) void
+display(): void
ShortAnswerDecorator
+getLimit(): final int
+checkLength(ans: String): boolean
+modify(br: BufferedReader): void
+take(br: BufferedReader): void
All classes in the question package are serializable
Serializable