Chun-Kit Chung, Isaiah Linares, Leong Li, Yuming Yan
217125329, 217287509, 217270380, 215210990
EECS 3311, Section B, Second Software Project, TA: Naeiji Alireza

ConverterProject Application Report

Isaiah Linares, Leong Li, Chun-Kit Chung, Yumming Yan

11/24/2021

Table of Contents

Introduction

The goal of this project is to create an application using the MVC (Model, View, and Controller) software design pattern that converts a value entered in centimeters into feet and meters.

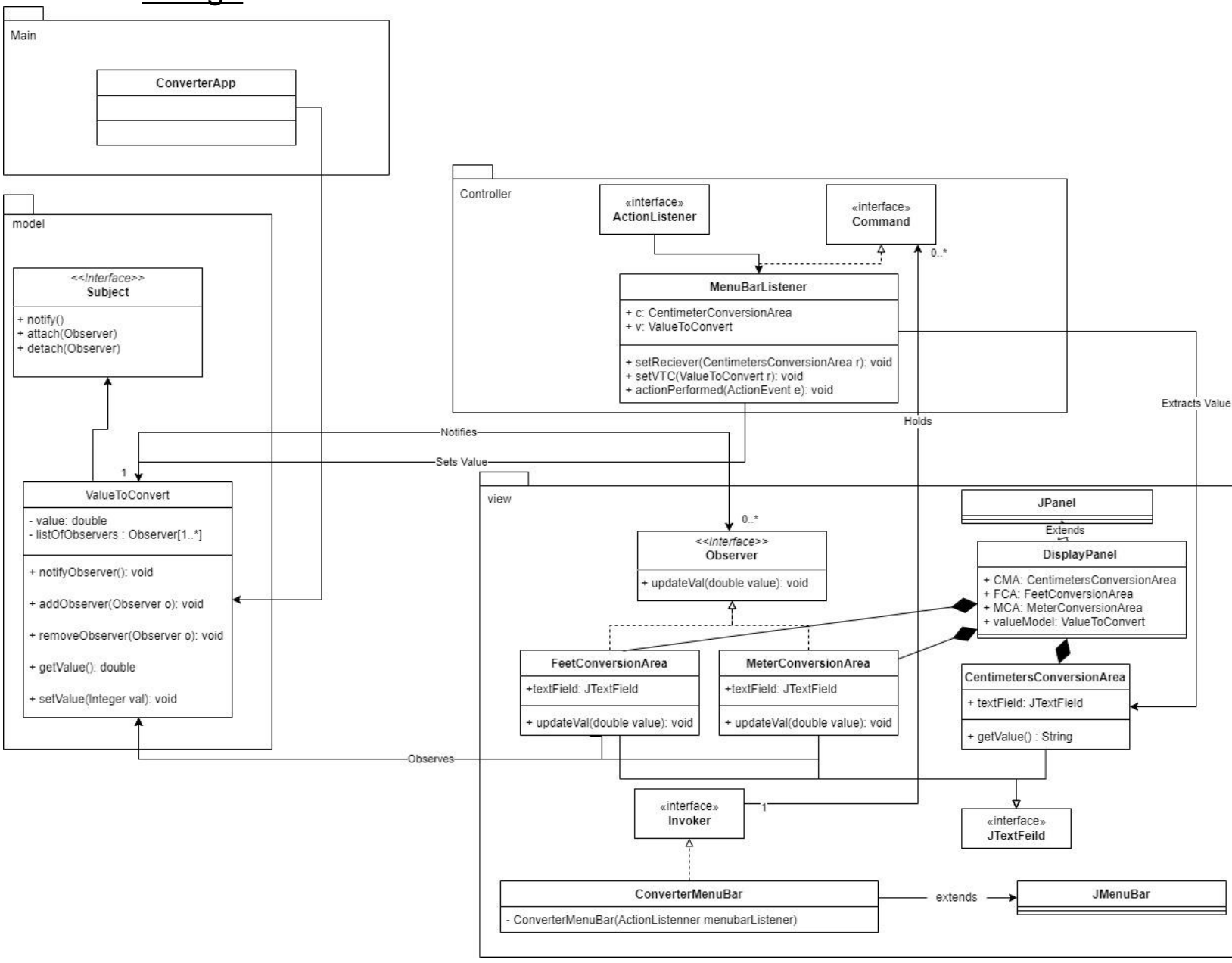
Some challenges the team will have to overcome first include: finding how we can apply the Observer and Command pattern to our Object Oriented System, as well as making sure the Model, View, and Controller parts of the system all work together correctly. In order to attack these problems our team will meet and plan a way to solve them using tools such as OOPs and design patterns.

This report is structured into four sections, the introduction section where we explain the goal of the project and how we plan to accomplish this goal, the design section where we show the Object Oriented Analysis and Design as well as all the UML Diagrams and Models used to provide structure for this application, an Implementation section going over how we decided to implement this application, and finally a conclusion where we will reflect on accomplishments and shortcomings of this project.

To create this application, we will be using Object Oriented Analysis to find distinct object concepts applicable to our application as well as object oriented design to specify software objects needed to satisfy our requirements. To create this application using Object Oriented Programming we will use basic principles such as Inheritance, Encapsulation, Abstraction, and

Polymorphism. The main design pattern we will use to structure the project is the MVC or Model View Controller design pattern and we will also be using other design patterns including the Observer and Command design pattern.

Design



Design Patterns:

The two design patterns used in this project are Observer and Command behavioural design patterns.

The Observer pattern is used to specify a one-to-many relationship between objects in the case where one object's state changes, all its dependents are notified and updated automatically. The concrete subject in this project is the ValueToConvert class in the model package. This class stores the state that interests the concrete observers. The concrete observers are the FeetConversionArea and MeterConversionArea classes from the view package. These classes implement the Observer interface and get notified whenever ValueToConvert changes its state and gets updated automatically. The ValueToConvert class maintains a list of observers and implements all the mutator methods of a Subject interface, which include notifyObserver, addObserver and removeObserver. The concrete observers FeetConversionArea and MeterConversion area implement the update method of the Observer interface which updates the values automatically when ValueToConvert state is changed.

In addition, the Command pattern is used to encapsulate a request as an object, which enables us to parameterize objects by an action to perform. In our project, the ConverterMenuBar is the invoker that is passed a command (i.e. the request wrapped under an object), then asks the command to perform the request. The MenuBarListener in our system is the concrete command which implements the execute method by extracting the value from CentimetersConversionArea. The ValueToConvert is a receiver of the command design pattern in our system, it gets the value from MenuBarListener. In addition, the CentimeterConversionArea is also a receiver, as the command acts on this class by extracting and parsing a JTextField value.

OO design principles:

We have used several Object Oriented Principles in our class diagram. The DisplayPanel class composes objects of each FeetConversionArea, MeterConversionArea, and CentimetersConversionArea. These classes follow the principle of encapsulation by providing a mutator method specific to DisplayPanel such as getValue(). The other methods for these classes are inherited from java's JTextField, which abides by encapsulation as well. Another class that uses encapsulation is ValueToConvert. This class maintains a list of Observers that are encapsulated objects of Observer interface's subclasses. Each class that implements Observer has mutator methods to access its properties.

Polymorphism is used in the ValueToConvert class, the observers methods in this class such as addObserver, removeObserver, and notifyObserver all use polymorphism as they can take any specialization of the common interface Observer.

Abstraction is used a lot in our system as many parts work together and call each other in order to function.

Another object oriented design principle used in this design is inheritance, this is shown in FeetConversionArea, MeterConversionArea, and CentimeterConversionArea that inherit properties and methods from java's JTextField. Other classes that follow inheritance are ConverterMenuBar and DisplayPanel, which inherit Java's JMenuBar and JPanel.

Implementation

Class Descriptions

The ValueToConvert class is the model of the system that holds the inputted value in centimeters. It includes all the methods to the observers including add, remove and notify observer. It is used in the Observer pattern as a subject and also used in the command pattern as a receiver.

The CentimetersConversionArea is a class in the view package of the system which inputs a value into its text area. It is used in the command design pattern as a receiver that the MenuBarListener command acts on.

ConverterMenuBar is used as the Invoker in the command pattern and instantiates the menu bar of the interface, inheriting properties from the JMenuBar. It constructs a bar at the top of the interface and will receive a signal with button clicked or shortcut key pressed and invoke a command to act on CentimetersConversionArea and ValueToConver.

DisplayPanel inherits the properties and functions of java's JPanel concrete class. It generates 3 JText areas that represent centimeters, feet and meters each with distinct colors. These JText areas are objects of their own respective classes that extend the JText class. The FeetConversionArea and MeterConversionArea objects created are the observers used for the observer design pattern, which update and adjust their values accordingly whenever ValueToConvert's state is changed. In addition, it maintains the references to receivers used by MenuBarListener in the Command pattern.

FeetConvertionArea specialization of a JText class that displays the user input value in feet. It contains a method that updates and converts the centimeters value into feet whenever a change in state of ValueToConvert is notified.

MeterConvertionArea class is similar to the Feet conversion area. It's a specialization of a JText class that displays the user input value in meters and contains a method that updates and

converts the centimeters value into feet whenever a change in state of ValueToConvert is notified.

MenubarListener is used in the Command pattern as a command. This class will act on the receivers CentimetersConversionArea and ValueToConvert every time it receives an ActionEvent from Invoker(ConverterMenubar).

ConverterApp is the main application class which initializes the Model, View, and Controller of the system. This class also connects these three parts to form the application.

Some libraries, tools, and frameworks we have used during the implementation were Java Swing, Java Util, and Java AWT.

Conclusion

Some things that went well in the software project was when we implemented the GUI with the necessary text areas and menu bar. We had the same situation as the last software project so we were more comfortable with the process of creating those objects. Another thing that went well was addressing FeetConversionArea and MeterConversionArea as observers that inherit the JText Class.

Some things that went wrong in the software project were creating the corresponding Javadoc for the project. And when we implement the area part into other methods such as menubarlistener and ValueToConvert, at first we were unable to load the input from CentimeterConversionArea until we set a receiver for it. Another problem we had was making use of the command pattern in our system. We had trouble finding a way to encapsulate a command in a way that would help us build our application although through group brainstorming we were able to apply the command pattern in a way that fit our system.

Some things we have learned while completing this software project were how to practice object-oriented concepts. We also learned how to write a simple java program and basic debugging and running methods. There are also some techniques used in the GUI that were learned during this project that were also used in the previous project making us better versed in those areas. While researching for our command pattern we learned how the ActionEvents and ActionListeners of the Java Swing framework are actually implemented. We became more comfortable with the MVC pattern, observer and command pattern as well.

Some advantages of completing the lab in a group is that we can discuss/share our different ideas to complete certain tasks. We can quickly solve problems with less effort when working in a group versus working alone. Some drawbacks of working together in a group is that some of the group members might not be able to meet on a specific day to work on the project or

the communication was cut between group members. This would lead to the issue of some members not being able to finish their work on time. This would create a lot of problems within the project as well as creating stress on the other group members.

Tasks	Portion of Work	Collaborative
Controller Main Model View	Chun-Kit Chung Leong Li Isaiah Linares Yuming Yan	Yes
Writing report	Chun-Kit Chung Leong Li Isaiah Linares Yuming Yan	Yes