The ConverterProject (Lab 6 Project)

Group Members:

Neel Rathod (214914329)

Kashyap Patel (216785339)

Ayush Sharma (217581075)

Course: EECS 3311

Section: Section A&B

TA= Kazi Mridul, Naeiji Alireza

Date: December 06, 2021

Professor: Alvine Boaye Belle, Ph.D.

Part I: Introduction

The purpose of this software project is to successfully create a Java application using different design concepts such as UML that were taught in the class. The application will allow the user to convert a value from centimeters to feet and meters. The application will have three views in the JPanel, where in the yellow view user can enter the value, the green view will display converted value in feet, and the orange view will display converted value in meters. There is also a jmenu bar at the top of the application displayed as “Update model” and it will contain “save input centimeters”, which will allow the user to convert the input value to feet and meter. Alternatively users can also use “Alt + F” keys from the keyboard to convert the value.

There are two main goals to complete this software project, the first part of the project is to analyze the requirements of the project and based on that we will identify what OOP principles we can use to implement object-oriented software design. To make the software object oriented, we need to focus on 3 aspects: analyze the requirements of the project, design the project based on those requirements, and finally implement that design in actual software.

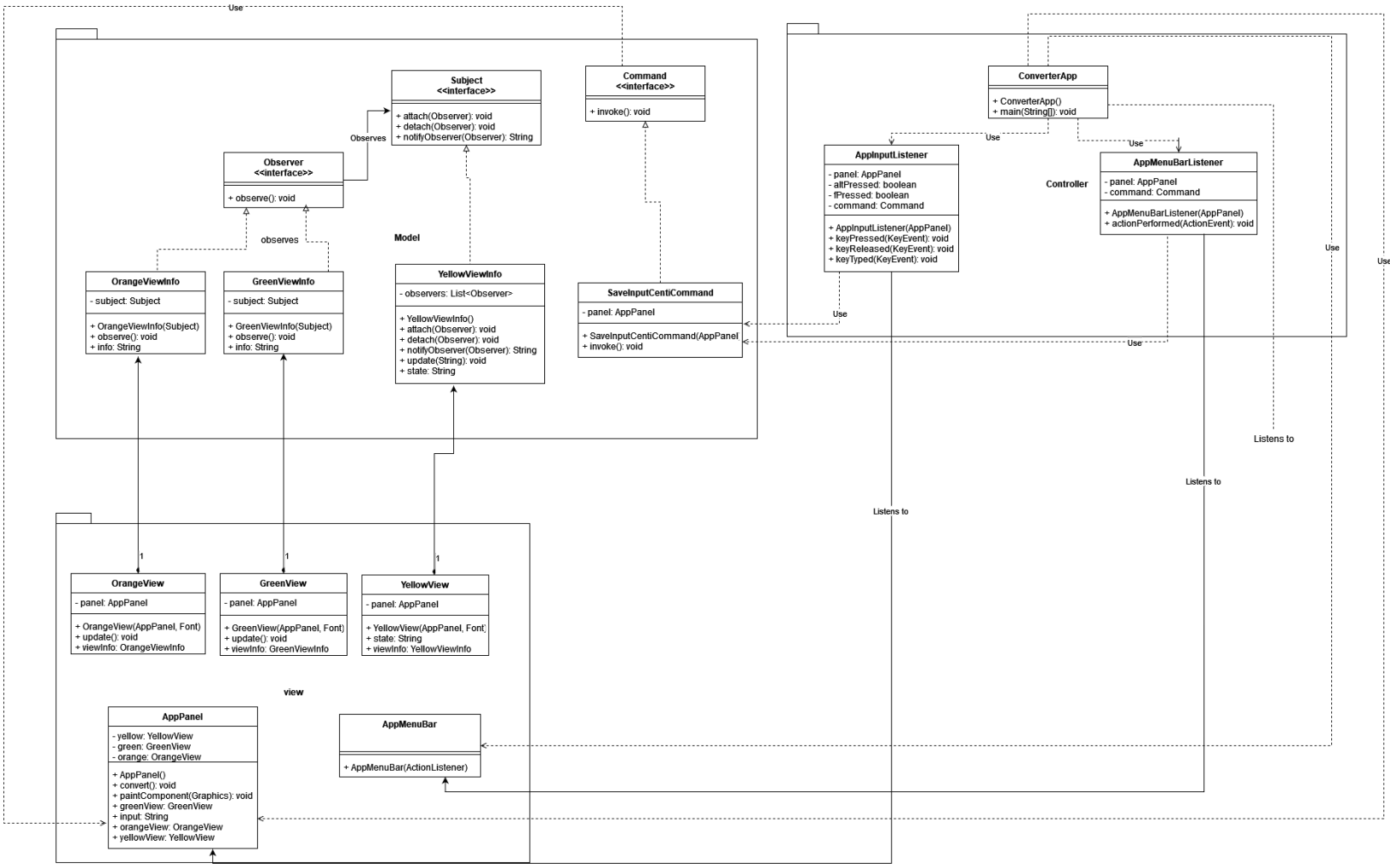
The main challenge with the project is to design the software in the model,view,controller (MVC) design. The separation of the MVC aspects of the code make it hard to design it in this way. However, once the design is finalized the implementation part gets easier since everything is outlined and clear.

The analysis of the project is very crucial as it helps us in finding and describing the object concepts that will be required to design the project. It will also help us specify software objects and the way they collaborate to satisfy the requirements.

The second aspect of OOD focuses on the design of the project. To create the successful design, we will create a UML diagram which will help us identify the main OOP principles (Abstraction, Polymorphism, Encapsulation) that are needed to make this project Object oriented. Usage of Encapsulation can be seen in all the classes by having private attributes and providing explicit getters and setters. Abstraction is shown by the presence of the two interfaces: the Command and Observer interface. We have used polymorphism in structuring the info classes.

The report will be structured to first show the requirements of the project and the analysis on what needs to be achieved for completion of the project. The second part of the project will focus on designing two UML class diagrams for the project based on the requirements and showing the relationships between the classes. The third part of the project will focus on implementing the project based on our UML class diagrams that we created in part 2 and highlighting the various software tools that we used to implement this project. The last part of the report will focus on sharing the learning aspects from the software project.

Part II: Design of the solution



NOTE: To get a better view of the UML go to UML.png!
EECS-3311-Lab-6/UML.png.

The UML diagram above shows the various classes which comprise the software project.

We have utilized the MVC architecture for the design of our software.

view: Starting with the view Package, we have 5 classes in it. The *AppPanel , AppMenuBar, OrangeView*, *GreenView*, and *YellowView*.

controller: Now in the controller Package we have 2 listener classes, namely *AppListener* and *AppMenuBarListener.* We also have the main class *ConverterApp*, where all the code is run from.

model: In the model package, we have the 3 interfaces namely, *Command*, *Subject* and *Observer*. We also have *GreenViewInfo*, *OrangeViewInfo*, and *YellowViewInfo* classes. We also have the *SaveInputCentiCommand* class.

Our UML class diagram uses 2 design patterns in total and they are as follows:

Observer Design Pattern: This pattern is shown by the *Observer* and *Subject* interface. We have the *GreenViewInfo, YellowViewInfo* and *OrangeViewInfo* classes that act as the observers and notify the controller of any state changes by calling the *changeState*() method.

Command Design Pattern: This pattern is shown by the *Command* interface. We have a **receiver** object that is the *SaveInputCentiCommand* class, this invokes what needs to be done for our app.

The main **OO design principles** used are **Abstraction, Encapsulation and Polymorphism**.

Abstraction: We have created two interfaces: the Command and Observer interface. These categorize the green and orange views accordingly. These views then in turn focus on doing exactly one thing and that is to convert the centimeters given in the yellow view, into feet and meters. This in itself is abstraction, where our meta-code is doing what the code is supposed to do.

Encapsulation: Usage of Encapsulation can be seen in all the classes by having private attributes and providing explicit getters and setters. These getters and setters provide a consistent interface for retrieving and changing the attributes of a class. This effectively encapsulates the data within the class.

Polymorphism: We used polymorphism in the structuring of the *GreenViewInfo* class and *OrangeViewInfo* class. Since these classes implement the Observer interface, in *YellowViewInfo* class we can use a list that takes in the type Observer that can have elements that are for the orange or green view.

Part III: Implementation of the solution

Our code ensures a smooth run without any exceptions. We have created an **executable file** to make it easier for you to run our program.

In the **view** package we have:

The *AppPanel* class embodies the canvas on which the app is displayed. It is composed of three classes: *GreenView*, *OrangeView*, and *YellowView* classes. This class also extends the *JPanel* class.

The *AppMenuBar* which is used to draw the app menu which allows us to update the centimeter input. This class extends the *JMenuBar* class.

The *GreenView*, *OrangeView*, and *YellowView* classes are responsible for the green, orange and yellow squares drawn on the app-panel. This is where our input and calculations go.

In **controller** we have:

AppInputListener and App*MenuBarListener.* These classes listen for any events, like key presses, in the *AppPanel* and *AppMenubar* classes, respectively. The App*MenubarListener* implements the *ActionListener* class and the *AppInputListener* class implements the *KeyListener* interface.

For **model** package we have:

GreenViewInfo, OrangeViewInfo and *YellowViewInfo*. These classes hold onto the information that is displayed. This is the centimeters and feet info.

Observer and *Command* are the interfaces that implement the design patterns that are necessary to run the program.

SaveInputCentiCommand is the class that takes in the user input and then passes the information along to the relevant classes that need it to crunch the data and display the right amount.

The **Javadoc** can be found under the *EECS-3311-Lab-6/index.html (shortcut)*. The index.html file will give you the landing page of the Javadoc.

The **tools** that we used to create this project are: latest version of **draw.io** to create the UML class diagram, last version of **Eclipse** IDE, JDK & SDK version 17, git version control & **Github**.

The short informational video is labeled ***shortExplanation.mp4,*** located in *EECS-3311-Lab-6*/*shortExplanation.* It gives all the basic information to run the application.

Part IV: Conclusion

The thing that went well while creating this software project was the experience that we had after doing the mini-soccer-project. We had exposure to the MVC model through our previous assignments, so that helped us how to tackle the project and how to implement the logic behind the project based on the requirements.

The things that went wrong while creating the software project were, utilizing the specified design patterns, Observer and Command patterns respectively. As we had never gotten any kind of exposure to actually coding using these patterns before, so this was a learning curve for all of us. We tried a few things, failed; then tried again. The second challenge was completing the project with one less person. Due to certain circumstances we lost a team member half-way through the project. However, We learned how to handle extra workload well during this project.

The things that we have learned after completion of the project were the new design patterns which are Observer and Command patterns. We also learned how to code from the ground up, without any starter files.

The main advantage of working in a group is that the work gets done super quickly with a good morale, since your group members motivate you to do your tasks. The drawback is that every group member had other commitments in other courses that made it hard to schedule meetings to push the project along.

The following table denotes the different tasks performed by the group members.

Ayush Sharma	**Neel Rathod**	**Kashyap Patel**
Implementing the solution by writing several classes for the project.	Managing the project, by maintaining a line of contact between all the team members.	Recording of the video and creating a jar and exe file.
Designing the first draft of the UML	Finalizing the design of the UML	Implementing Javadocs for all the classes.
Contribution to the report writing.	Contribution to the report writing.	Polishing the code for final build and finalizing the report.

All in all every member has put in great efforts towards the completion of this project and we are proud and content with the outcome.