***The Converter Software Project***

*Asanka Gunatilaka (Section B) 217499450*

*Bansari Panchal*

*Faith obadun*

*Janki Jadeja (Section B) 215624182*

*T.A: Alireza Naeiji*

***Part I - Introduction***

*The Converter Project requires us to create an application which allows us to convert a value from centimeters to feet and meters on a JPanel with 3 views and a JMenuBar stating “Update Model”; which consists of a JMenu named “Save Input Centimeters”. The 3 views are yellow, green, and orange with values in centimeters, feet, and meters respectively; each view is a JTextArea which displays all the text. Once the user enters a value in cm, the user then clicks “Update Model” and “Save Input Centimeters” to store the value entered. Once that is complete, the green and orange views are automatically updated to the converted value in ft and m from cm. The goal is to design a UML diagram which interrelates with design patterns and design principles learnt in this course. Implement the code which follows the designs. This project also helps us understand how the design patterns and principles work. And allows us to familiarize ourselves within the concept of group environment, learn OO analysis and reflect on our work through this report.*

*The main challenge for this project were having to combine different parts of the implementation that each member worked on, to create the final application. It was also difficult to identify correct and accurate design patterns as well as to create the UML diagram.*

*The 2 important design principles used were the behavioral patterns, command (action) and observer. The command pattern is used when you have an object, and you need to encapsulate all information required to perform an action or trigger any event later. The observer pattern consists of an object, usually named the Subject, which maintains a list of its own dependents (observers) and notifies them whenever there is a change in the state (by calling one of their own methods).*

*This report will address each question, from each section, as required in the lab 6 pdf, provided by the professor on eclass. The report will be in the form of small paragraphs, where each section will have multiple paragraphs. There will be 4 main sections, introduction, design, implementation, and conclusion.*

***Part II – Design***

Diagram

Description automatically generated

*Please find document named Diagram.drawio for the UML diagram.*

*This is the direct link to the UML diagram:* [*https://tinyurl.com/ybsexayz*](https://tinyurl.com/ybsexayz)

*In this implementation we have used the Command and the Observer design patterns. For the Observer design pattern, we have an interface named ConversionAreaObserver that declares a method named update (double n) and enforces its concrete classes, which are also the observers (MeterConversionArea, FeetConversionArea) to define them. There is also an interface named Subject that enforces its concrete class ‘ValueToConnect’ to register observers and notify observers when there is an update. The ‘ValueToConnect’ class resides in the Model package as it also works as the intermediatory class between the View and the Model. This design pattern allows the observers to be updated every time the subject has a change and therefore, they are always aware of any changes.*

*For the command pattern, we have an Invoker, which is the CentimerterConversionArea class, that invokes a specific command. In our case, the command is encapsulated in the Conversion class. The receiver is the ValueToConvert class that executes the required action. The action in our case is to update the value and notify all the observers. The Invoker and the Command is not aware of the what the actual execution is, and it therefor allows other commands to be plugged in when required at a later stage.*

*For this project, we used a couple of design principles. We used inheritance to have access to the methods of JMenuBar, JPanel and JFrame so one could be able to convert from centimetres to feet and metres. Inheritance was also by ValueToConvert from the subject interface, the feetconversionarea method and the meterconversionarea method inherit from the ConversionAreaObserver interface. We also used encapsulation to display the shapes and for the conversion from Panel. Aggregation was used from the main Conversion class. Cohesion was used through the relationships within the ConversionArea classes making the functions related.*

***Part III - Implementation***

*Please see Github repository to find the implementation of the Converter application.*

*Following is a documentation of all the classes (their methods, global variables) ) to explain what each class does, their description, the invariants of each class, the preconditions and the postconditions of each method, when applicable.*

***Controller***

* *Conversion.java*
* *The Conversion class updates the value of centimeter using ValueToConvert class*
* *Global Variables: n*
* *Methods: execute()*
* *MenuBarListener.java*
* *The MenuBarListener class creates a listener for the Menu Bar.This class monitors the activity of the user and initiates saving the value of the user entered.*
* *Methods: actionPerformed()*
* *Global variable: value*

***Main***

* *Main.java*
* *The Main class contains the graphical user interface for Converter application that converts the centimeter value into feet and meter.This class builds the frame, the panels and the objects required to launch the application*

***Model***

* *ValueToConvert.java*

*- ValueToConvert class observes the value of centimeters specified by user in CentimetersConversionArea and updates and notifies FeetConversionArea and MeterConversionArea*

*- Variables:value*

*- Methods: register(), unregister(), notifyObserver(), updateValue()*

*- precondition: Value to be converted is not negative*

* *Subject.java*
* *Subject class is an interface that has a register, unregister and notifyObserver and updateValue method. Subject class uses observer design pattern to register observers and notify observers when there is an update.*
* *Methods: register(), unregister(), notifyObserver(), updateValue()*

***View***

## *CentimetersConversionArea.java*

* *The CentimeterConversionArea is a yellow text view where users can input values in centimeters. This class uses the Command pattern and works as invoker to invoke specific command*
* *Methods: saveValue()*
* *Variables: n*
* *Precondition: centimeter value is not negative*

## *FeetConversionArea.java*

* *The centimeter input value is converted to feet and displayed in Green view.*
* *Methods: update()*
* *Variable: n*
* *Precondition: centimeter value is not negative*
* *Postcondition: Feet value is not negative*

## *MeterConversionArea.java*

## *The centimeter input value is converted to meter and displayed in orange view*

* *Methods: update()*
* *Precondition: centimeter value is not negative*
* *Postcondition: Meter value is not negative*
* *MenuBar.java*
* *The JMenuBar manages the status of the application with features like Update Model and Save input centimeters. This class creates menubar and sets up required features*
* *MyPanel.java*
* *The Panel class contains the graphical user interface for Converter application.JPanel class has three views. The yellow view is where the user enters value in centimeters. The entered value is displayed in feet and meters in green and orange view respectively.*
* *ConversionAreaObserver.java*
* *ConversionAreaObserver class is an interface that updates the value from centimeters to feet and meters. ConversionAreaObserver class uses observer design pattern to register observers and notify observers(MeterConversionArea and FeetConversionArea) when there is an update using update method*
* *Methods: update()*

***PART IV - Conclusion***

*As taught throughout this course, we were able to tackle this project by creating use cases, domain model, sequence diagrams and a UML diagram. Using design patterns gave an idea of how the classes should be created and structured; the UML diagram created, provided us with the structure of the entire project and their dependencies. We got another chance to understand the overall concept of OO analysis and design. The time management and the distribution of the workload within the group members was much better for this project.*

*The advantages of working in a group is that you get to learn different things from each member as well as a chance to view different ideas and opinions. Everyone gets to work at their own pace and time and then the work is put together at the end and the workload is divided equally. The drawbacks of working in a group is that there might not be enough collaboration, everyone is not available at the same time. The work done may be very slow and consist of errors which takes time to fix.*

*Below table indicates the different tasks that were assigned to each team member of the team.*

|  |  |  |  |
| --- | --- | --- | --- |
| *Name* | *Work Assigned* | *Work Completed* | *Collaborative* |
| *Asanka* | *MVC (Controller) & how and which design patterns are used (for the report)* | *UML/ Implementation/ Report* | *Yes* |
| *Bansari Panchal* | *UML & precise documentation for all the classes (for the report)* | *UML/ Implementation/ Report* | *Yes* |
| *Faith Obadun* | *MVC (View) & how and which design principles are used (for the report)* | *UML/ Implementation/ Report* | *Yes* |
| *Janki Jadeja* | *MVC (Model) & Introduction and Conclusion (for the report)* | *UML/ Implementation/ Report* | *Yes* |

|  |
| --- |
| *We all worked on the report together since each member required other members’ work in their own section.* |