Converter Project (Third Software Project/Lab 6) **EECS3311**

Usama Taqikhan, 215414758, Section A **TAs: Kazi Mridul (kmridul@yorku.ca)**

Muhammad Abbasi, 215614555, Section B **Alireza Naeiji (naeiji@yorku.ca)**

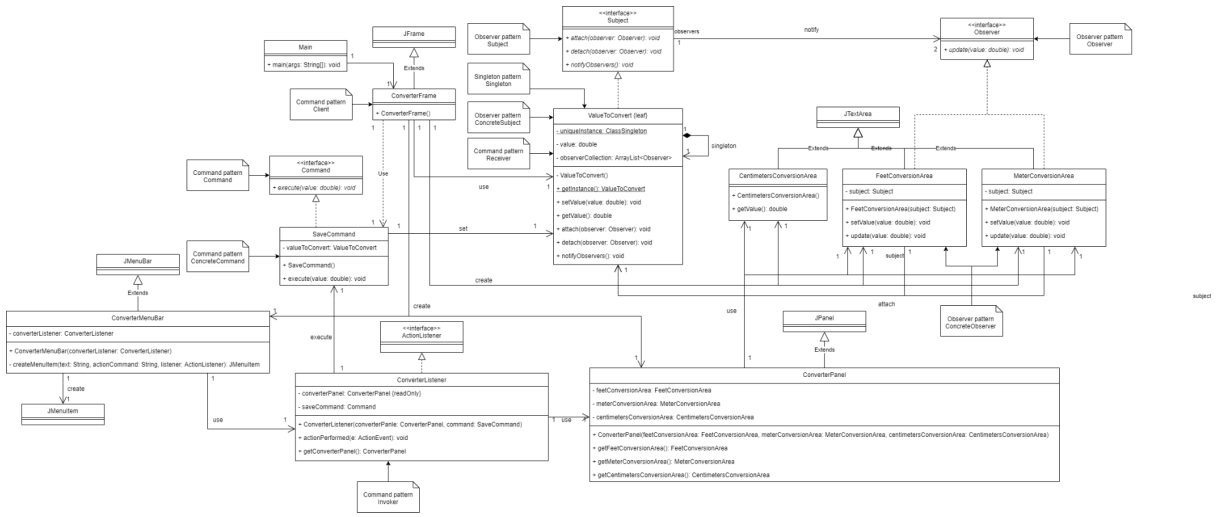
Muhammad Zaighum, 214947022, Section A **Due: December 6, 2021, 11:59PM EST**

Dongwon Lee, 215805260, Section B

Introduction:

For this project, we were assigned the task of creating an application that lets the user convert a value from centimeters to feet and meters. The application consists of a GUI with three panels; a green panel for feet, a yellow panel for centimeters, and an orange panel for meters. The user will input a value into the yellow panel and after saving the value, it will be converted to feet and meters. Saving can be done by either using the mouse to select the save option on the top menu bar or by using your keyboard and typing ALT + F. This is the first project in this course that we are doing completely from scratch; no starter code or source code was provided. A general challenge we faced was creating all the classes from scratch. Using JTextArea was also a little difficult, as we needed to use these for displaying and for inputting and it's a Java class we didn't have much experience with. We will be using the four OOP principles, abstraction, encapsulation, inheritance, and polymorphism, that we learned in our design and also incorporate the Observer, Command, and Singleton design patterns. This report is broken up into five sections: the introduction, the design, the implementation, the conclusion, and the roles. The design section will include an image of the UML diagram and will dive deeper into the design patterns and principles we used. The implementation section will consist of descriptions of each class and what the roles of each class are as well as technical details like what IDE we used, the version of Java we used, etc. Finally, the conclusion will consist of the pros and cons of the project; what went well, what didn't and some final thoughts from us. To accompany this written report, a video demonstrating the code as well as the source code and images of the UML diagram are included within a Github repository.

Design:



The UML class diagram file (UML.drawio) and image (UML.drawio.png) are included in the Github repository. The elements of the UML diagram are as follows: CentimetersConversionArea, FeetConversionArea, and MeterConversionArea are in an inheritance relationship with JTextArea. Similarly, ConverterFrame, ConverterPanel, and ConverterMenuBar are in an inheritance relationship with JFrame, JPanel, and JMenuBar respectively. These three classes are in an aggregation relationship with ConverterPanel. FeetConversionArea and MeterConversionArea are in a realization relationship with Observer. Similarly SaveCommand, ConverterListener, and ValueToConvert are in a realization relationship with Command, ActionListener, and Subject respectively. The ConverterFrame is in a composition relationship with the ConverterPanel, the ConverterMenuBar, and the CentimetersConversionArea, FeetConversionArea, and MeterConversionArea classes. ConverterMenuBar and ConverterListener are in an association relationship. SaveCommand and ConverterListener are in an association relationship. SaveCommand and ValueToConvert are in an association relationship. ValueToConvert is in an association relationship with CentimetersConversionArea,

FeetConversionArea, and MeterConversionArea. Main is in a composition relationship with ConverterFrame.

The design patterns we used are Observer pattern, Command pattern, and Singleton pattern. Observer pattern is used when a class changes its state, and its dependent classes need to be notified of that change in state. For this project, we want to notify FeetConversionArea and MeterConversionArea anytime a new value is inputted in CentimetersConversionArea. The participant classes for the Observer pattern are FeetConversionArea, MeterConversionArea, ValueToConvert, Subject, and Observer. The Subject and Observer classes are interfaces and act as they are named; Subject knows its observers and proposes operations to attach and detach them and Observer observes a subject and is notified any time there is a change in Subject. ValueToConvert acts as the ConcreteSubject, it stores the state which the ConcreteObservers are interested in. In this case the state it stores is the state of CentimetersConversionArea and what value is currently being inputted/stored in it. FeetConversionArea and MeterConversionArea act as the ConcreteObserver and update themselves to keep consistent with the Subject. In this case the two ConcreteObservers will update themselves any time a new input is saved in CentimetersConversionArea.

Command pattern is used to encapsulate a request as an object, which lets us parametrize clients with different requests. For the Command pattern, the participant classes are ConverterListener, SaveCommand, Command, ConverterFrame, and ValueToConvert. The Command class is an interface and handles the execution of an operation using its execute() method. The SaveCommand class is the ConcreteCommand, and it implements the execute() method from Command and uses it on the Receiver. The ConverterFrame is the Client and it instantiates a SaveCommand object and will specify the Receiver. The ConverterListener is the Invoker and will ask the SaveCommand to do the request.

ValueToConvert is the receiver and will perform the action associated with the request. Because ValueToConvert is used in various other classes, we implemented it as a Singleton, so only one instance is available to use, getting rid of the need to create multiple instances in different classes.

We have used the OOP principles in our design. Abstraction is used in the Observer interface and FeetConversionArea and MeterConversionArea. The update() method that is common to both FeetConversionArea and MeterConversionArea is abstracted into the interface Observer. Similarly, if we were to add more commands to the project, they would need to use the execute() method from Command in order to adhere to the Command pattern, and so the Command interface and its related classes is another example of abstraction. Encapsulation is used in ValueToConvert. It has a private field called value and it can only be accessed using the getValue() and setValue() method. Inheritance is used as CentimetersConversionArea, FeetConversionArea, and MeterConversionArea are children of JTextArea. ConverterPanel is a child of JPanel, ConverterMenuBar is a child of JMenuBar, and ConverterFrame is a child of JFrame. Polymorphism is used in many instances. FeetConversionArea and MeterConversionArea implement the method update() from Observer. ValueToConvert implements the abstract methods attach(), detach(), and notifyObservers(). SaveCommand implements the abstract method execute() from Command. ConverterListener overrides the method actionPerformed() from the ActionListener class. All of these are examples of polymorphism. Thus, we were able to implement all four OOP principles within our design.

Implementation:

The project was implemented with four main packages: main, view, controller, and model. The main package contains the Main class, which is the client and the demo class that displays the GUI and allows the program to run. The view package contains the various classes that come together to form the GUI. These classes are CentimetersConversionArea, FeetConversionArea, MeterConversionArea, ConverterFrame, ConverterPanel, ConverterMenuBar, and Observer. CentimetersConversionArea, FeetConversionArea, and MeterConversionArea are implemented as JTextAreas. CentimetersConversionArea is the class that will take in the input in centimeters while FeetConversionArea and MeterConversionArea will convert that input into feet and meters respectively. The invariant for the CentimetersConversionArea class is either the input value is 0 or it is positive. This invariant also applies to FeetConversionArea and MeterConversionArea. The latter two classes are used for the Observer pattern, which we explained in the previous section of this report, and will update whenever an input value is saved in CentimetersConversionArea. These two classes implement the Observer class and thus implement the update() method from that class. The ConverterFrame instantiates the various aspects of the GUI for use in the Main class. The invariant of the ConverterFrame class states that an instance of this class should include instances of all the other classes within the view package. ConverterPanel is used to paint the three JTextArea classes onto the frame and the invariant for this class is that an instance of this class should have three JTextAreas corresponding to the three ConversionArea classes we have in this package. ConverterMenuBar paints the menu bar at the top of the frame allowing for the use of the “Save input centimeters” function, which allows the user to save an input and convert to feet and meters. The invariant for this class states that an instance of the class should have a JMenu along with a JMenuItem and an ActionListener. The controller package contains three classes, Command, ConverterListener,

and SaveCommand. The Command class is an interface, and is used for the Command pattern which we talked about in the previous section. The SaveCommand is used for saving/storing the value that is inputted during execution of the program and the invariant for this class states that a ValueToConvert instance should be initialized before a call to the execute() method is made and the pre condition for the execute method is that the value is either 0 or positive. This class implements the Command class. The ConverterListener implements the ActionListener, is connected to ConverterMenuBar, and allows the user to actually use the menu bar items during execution of the program. The invariant for this class states that a ConverterPanel and SaveCommand instance should be initialized before any action event happens. The model package consists of two classes, Subject and ValueToConvert. Subject is an interface and used for the Observer pattern implementation within our program. ValueToConvert is responsible for saving the input value from CentimetersConversionArea and notifying the other relevant classes about the change in state. The invariant for this class is the input value stored should either be 0 or positive. We used Eclipse to create the project and JDK version 1.8. We used the JRE System Library (JavaSE-1.8). Javadoc of the source code and a video demonstrating the code are included in the Github repository.

Conclusion:

Some things that went well in this project was the collaboration with the group and the type of project that was given. Firstly, having collaborative group members was essential for getting this project done, since we had to do everything from scratch. Secondly, since we were using GUI for this project, the group was eager to implement it, as it is an interesting topic. Not many things went wrong with the project, but to pinpoint one thing, it was the implementation for the Observer and Command pattern. It was a little confusing

implementing these patterns as we had not done it before and we had to adhere to certain principles and try to match the structure of the patterns given in the slides.

The things we learned from this project are how we can use the OOP principles and design patterns we learned throughout the course and use them as we see fit in this project as well as in any future projects we get.

There were many advantages for doing this project in a group. Firstly, the group we have is the same as for Lab 5, so we are already familiar with each other's strengths and weaknesses and can give tasks to group members accordingly. It’s also easier for us to communicate this time around as we are already familiar with each other, and we can share ideas or explanations freely. There were not many drawbacks for this project. One thing that could be considered a drawback was the fact that we had to do the whole project online because of real world circumstances. Not being able to meet in person may have deprived some group members from giving better ideas or explanations, as some ideas are best conveyed in real life rather than in text or during a video call.

Roles:

Usama Taqikhan (25%)	Video Report
Muhammad Abbasi (25%)	Report
Muhammad Zaighum (25%)	Report
Dongwon Lee (25%)	Coding UML Javadoc