EECS 3311

Lab6

Converter Project

Report

Group7

Abbyrahm Harrymanoharan 215099369

Boho Kim 217303033

Deep Patel 216519126

Yixing Cao 216775140

PART-I

1. This software project is creating an application (using MVC) that allows converting a value (specified in centimeters). This value should be converted in feet and meters respectively. Here are the goals of this software project: The View should comprise a JMenuBar ("Update model") **and** a JPanel (that again comprises three views).
   (a) The JPanel should have three views (each view is a JTextArea that can display text):
   - A yellow view where the user should enter (type) a value in centimeters (e.g., 165).
   - A green view should display the entered value in feet (e.g., 5.41338582675 ft).
   - An orange view should display the entered value in meters (e.g., 1.65 m).

   (b) The JMenuBar should comprise a JMenu named "Update model". The JMenu should comprise a JMenuItem named "Save input centimeters".
   - The user can enter (type) a value in centimeters (e.g., 165) in the yellow view.
   - To store value, the user clicks "Update model" and "Save input centimeters".
   - The controller retrieves input value from the yellow view and stores it in the model.

➢ The green and the orange views should observe the Model. When the Controller stores the input value in the Model, the Model should automatically notify the green and the orange views. Both views are then updated:
- The green view should display the input value in feet.
- The orange view should display the input value in meters.

2. Challenges associated to the software project:
   - Finding relevant/best resources for learning technologies and enhancing skills.
   - Designing the roadmap/design (like UML class diagram) of the complete project.
   - Implementation of the class diagram properly with complete coverage.
   - Meeting the expectations and complete requirements of the project with accuracy.

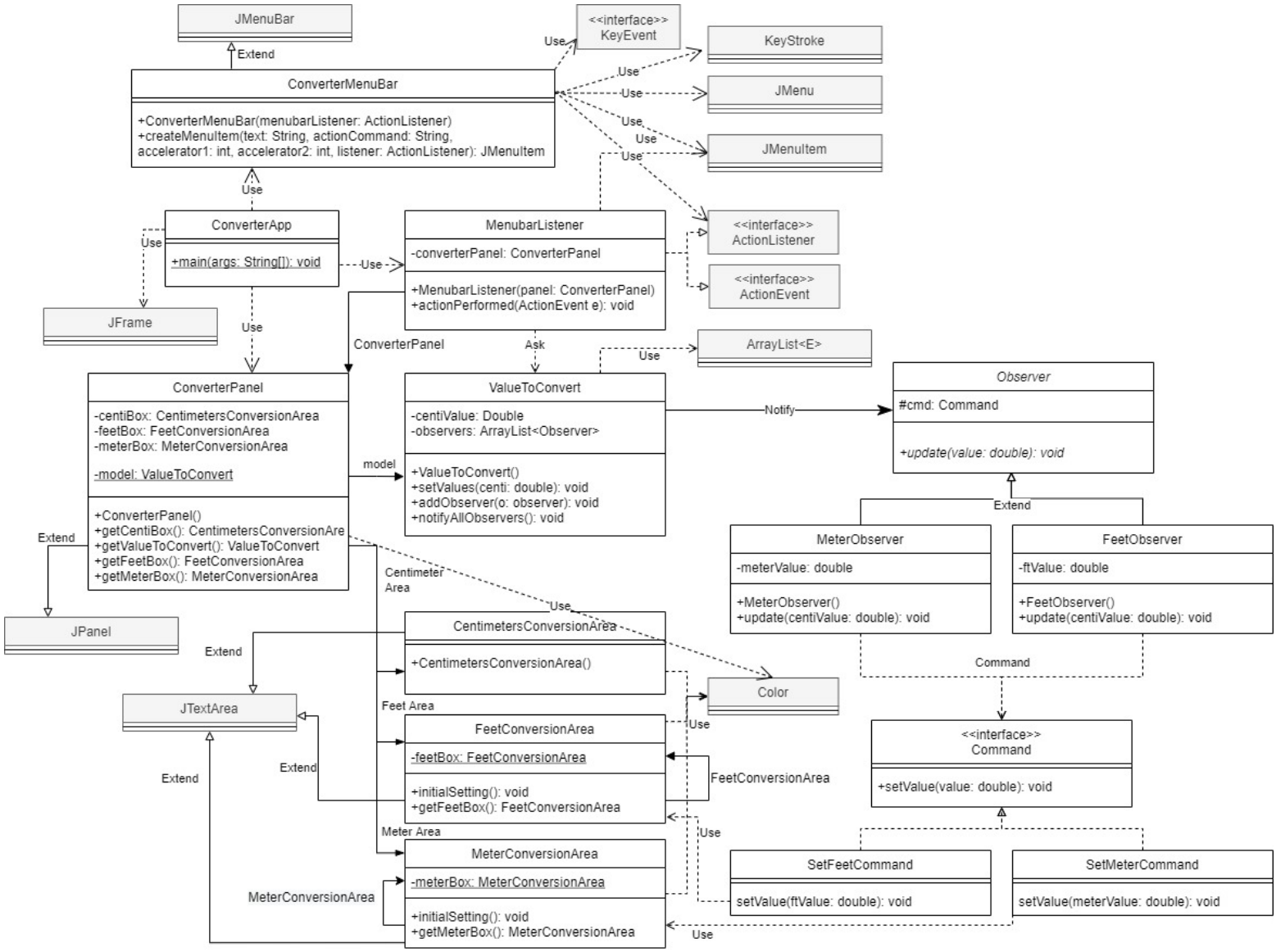
3. We will use the below concepts while working in this project:

- **Object/Class:** For representing real world thing (ConversionAreas, MenuBar, ConverterPanel, ValueToConvert, etc) in the software project.
- **Interface:** For achieving abstraction with Command and Observer design patterns.
- **Inheritance:** Inheritance will embody the generic concept of JPanel, JTextArea, and JMenuBar so that all other classes like ConverterMenuBar, FeetConversionArea CentimetersConversionArea, MeterConversionArea, and ConverterPanel can use them.
- **Encapsulation:** Bundled methods and attributes related to a particular entity/class in its body so that it could hide the internal representation/state of an object from outside.
- **Class Diagram:** For describing the structure (attributes and methods of a particular class), and maintaining relationships between classes of the project/system.
- **Design Pattern:** For analysing, designing the system with reusability and transparency and clarifying the system architecture to build a better system/project solution. In this project, we will use **MVC**, **Singleton**, **Observer** and **Command** design patterns.

4. Our reports will be structured in this way: We introduced about the project briefly in Part1. In Part2 and Part3, we will show how we designed and implemented with specific design patterns using OOD concepts. In Part4, we will explain about challenges, pros and cons of working in group, what we learned and show a table for allocated tasks.

PART-II

1. UML class diagram



(You can see the JPEG file in Report directory in Github repository)

- **ConverterApp**: The main class. It constructs panel components for user interface and runs application using main method.
- **ConverterPanel**: A view of MVC. It constructs the converter window.
- **CentimetersConversionArea**: A view of MVC. It constructs yellow window for unit cm's.
- **FeetConversionArea**: A view of MVC. It constructs green window for unit feet.
- **MeterConversionArea**: A view of MVC. It constructs orange window for unit meter.
- **ConverterMenuBar**: A view of MVC. It constructs the menu bar.
- **MenubarListener**: A controller of MVC. It processes the menu items click and keyboard events.

- **ValueToConvert**: An observer class type of MVC pattern. It gets centimeter value from view and notifies the value to observer classes.
- **Observer**: An observer class type of MVC pattern. It is an abstract class a parent class of specific observer class and is notified to update a centimeter value.
- **MeterObserver**: An observer class type of MVC pattern. It observes centimeter value and when it is updated, commands to set the changed value meter value.
- **FeetObserver**: An observer class type of MVC pattern. It observes centimeter value and, when it is updated, commands to set the changed feet value.
- **Command**: A command class type of MVC pattern. This interface is implemented by specific command classes to receive commands.
- **SetFeetCommand**: A command class type of MVC pattern. It commands to set value which is changed to feet on feet conversion area in panel.
- **SetMeterCommand**: A command class type of MVC pattern. It commands to set value which is changed to meter on meter conversion area in panel.

2. Design patterns

- ## MVC Pattern:

Model classes are as follows: -

Command.java, SetFeetCommand.java, SetMeterCommand.java, FeetObserver.java, MeterObserver.java, Observer.java and ValueToConvert.java

View classes are as follows: -

CentimetersConversionArea.java, ConverterMenuBar.java, ConverterPanel.java, FeetConversionArea.java, MeterConversionArea.java

Controller class is as follows: -

MenubarListener.java

- ## Observer Pattern:

The project consists of observer pattern. Observer pattern can be seen in the class *observer.java*. This class waits for the value (cm's) as an input and update the view accordingly by passing the value to the command class. It uses the method update; this method converts the value and updates it using command class.

- ## Command Pattern:

The project consists of command pattern. Command pattern can be seen in the class *command.java*. The class type is an interface type. This interface is implemented by specific command classes to receive commands. It uses the method setValue; This method receives value and set it on panel. Examples for other classes are *SetFeetCommand.java* and *SetMeterCommand.java*.

- **Singleton Pattern**:

Feet and meter text areas do not need to be generated whenever centimeter value is sorted. So, to keep and use only one object of text areas for feet and meter while running app, we used Singleton pattern. In *FeetConversionArea* and *MeterConversionArea* classes, they have their own class object as a field which is final static.

3. OOD Principles Used:

- Encapsulation:

Encapsulation is implemented in the classes *FeetObserver.java, MeterObserver.java and ValueToConvert.java*. The *FeetObserver.java* has private class variable *ftValue*. The *MeterObserver.java* has private class variable *meterValue*. The *ValueToConvert.java* has private class variables like *centiValue* and *observers*. These variables can only be accessed and modified within their respective classes. Classes that access and modify these variables must use getter and setter methods instead of directly accessing the variables.

- Inheritance:

Inheritance is evident in the relationship between the *Observer.java* superclass and *FeetObserver.java* subclass. The *FeetObserver.java* extends the functionality and inherits the methods of the superclass.

- Abstraction:

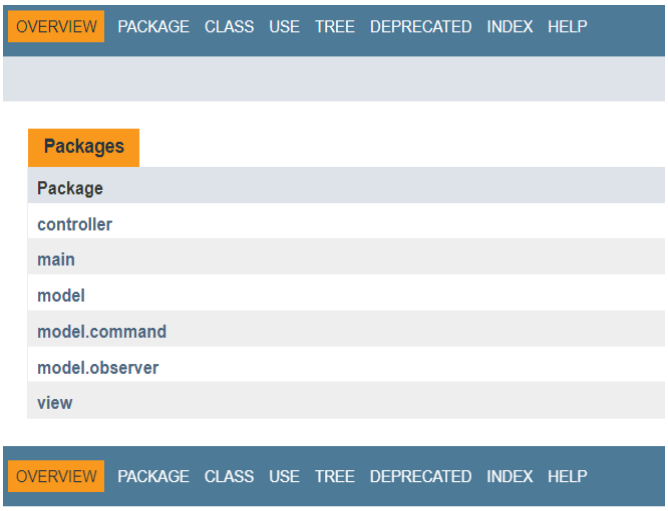
Abstraction is used in a few places in this project. In *ValueToConvert.java* attributes like *centiValue and observers* are made completely private. There is no way to access either of them outside of the *ValueToConvert.java* class itself. They can be only be modified indirectly through other public methods like *add* and *remove*.

PART-III

1. Implementation:
   You can see our implementation(codes) in Github repository
   https://github.com/kim-boho/ConverterProject_EECS3311.git

2. Javadoc is available in our Github repository in ConverterProject/doc/index.html

   Click the package name for more documents in index.html.



3. Tools/Libraries:
   Eclipse IDE, JRE JavaSE-15, draw.io, Github.

4. Short video to show how to launch and run application:

   https://photos.google.com/share/AF1QipM-c-KumzEJeBiGuJePSJ06T59_9_Nku2Rnr8tq6NH2AuzBk-JzMTj-lEqPB4vsag?key=TjFRWTAzX3N0bzhZcHBPelRZRmR1VGEteVR3M1dR

   (Also, it is available in Report directory in Github reposigory.)

PART-IV

We were able to come up with a good design based on the design patterns and principles. The design patterns used in our design were MVC, Observer, Command and the singleton design patterns. We were also able to make good use of OOD principles such inheritance, abstraction, encapsulation, and polymorphism. We were also able to work well as a group and take on assignments and collaborate on different assignments. The group also had a good understanding of the software requirements that needed to be met to complete the project. Other things that went well with the group is that we were able to divide the work up without issue and each member performed there assigned tasks.

Though some things went well while working on our software project we also faced some difficulties. One such thing was adding comments for class invariants as well as pre and post conditions. There were also difficulties with completing the implementation with meeting all the requirement without exceptions.

Through the software project we were able to gain knowledge about design patterns and its involvement in both the design if software systems and in implementation. One of the software patterns we learned about is the MVC architectural design pattern. Working on the software project gave us a better understanding of the three subsystems model, view, and controller. It also provided insight on how they interact with each other. We also learned more about the observer design pattern which allows for objects to share updates with dependant objects while maintaining a one-to-one relationship. We also used the command pattern the command pattern allows you to create queues, undo requests, allows for simple creation of new commands, and simplifies the testing process. Through this project we were able to learn a lot about these design patterns in how it helps in design and implementation.

Working in a group has a few advantages and disadvantages. Some advantages of working in a group is that every member has strengths that they use to help with the project. It also helps when a member is stuck or does not understand something they can ask for help form another member and learn from them. Some disadvantages of working in a group is that sometimes schedules conflict, and it is hard to get together and work on things at the same time or have regular meetings. Other issues with working in a group is that issues can arise with multiple people working on the same thing at the same time.

Group member	Tasks	Collaborative	percent
Abbyrahm Harrymanoharan	Report part 4, Check project launch/codes.	Yes	20%
Boho Kim	Report part 2(class diagram), part 3(Java doc), Code implementation, Readme page	Yes	35%
Deep Patel	Report part 2(explanation), part3(Short video)	Yes	25%
Yixing Cao	Report part 1, Group coordinator.	Yes	20%