Gbemisola Akerele (Student# 216167041)
Domenico Calautti (Student# 216 374 530)
Nancy Hao (Student# 217120197)
Misato Shimizu (Student#215310246)

The Converter Project
EECS3311 F2021
Naeiji Alireza (naeiji@yorku.ca)

PART Ⅰ (Introduction)

What is the software project about and its goals?

This software project is about implementing a functional centimeters-to-meters and centimeters-to-feet converter using Java graphical user interface (GUI) using *javax.swing* objects and components, and object-oriented programming (OOP) and object-oriented design (OOD), while using the Observer and Command design patterns. The goal for this software project is to allow users to input a measurement in centimeters inside a yellow square, and the GUI will display the conversion in meters inside an orange square and the conversion in feet inside a green square.

What are the challenges associated with the project?

As there is no initial code provided for this project, one of the challenges would be to structure the whole project from scratch. It would require sufficient discussion in the beginning to complete the project efficiently. Moreover, a group project with mandatarized design patterns is different from the last project, and it would be challenging for us not being able to use other familiar patterns.

What are the concepts we will use to carry out the project?

The OOD principles we will use are:

- Inheritance: This is utilized in ConverterFrame. Extending JFrame enables the former class to inherit methods, setJMenuBar and setDefaultCloseOperation.

The design patterns we will use are:

- Observer: This pattern is used to inform FeetConversionArea and MeterConversionArea when ValueToConvert, inherited from Subject, changes its value. With this design pattern, there is no need to notify each class for a change as all the Concrete Observers will be notified by Subject and will update their values automatically.
- Command: Although this pattern is useful for programs with multiple commands, our project contains only one command which is to save data. Thus, we set MenubarListener as a Concrete Command, use ConverterFrame as a Client, and make ConverterMenuBar work as a Invoker to set a save command. When the program receives an action, clicking a save button, MenuBarListener recognizes the action, executes an implemented actionPerformed() method, and ConverterPanel receives a signal to make an action.

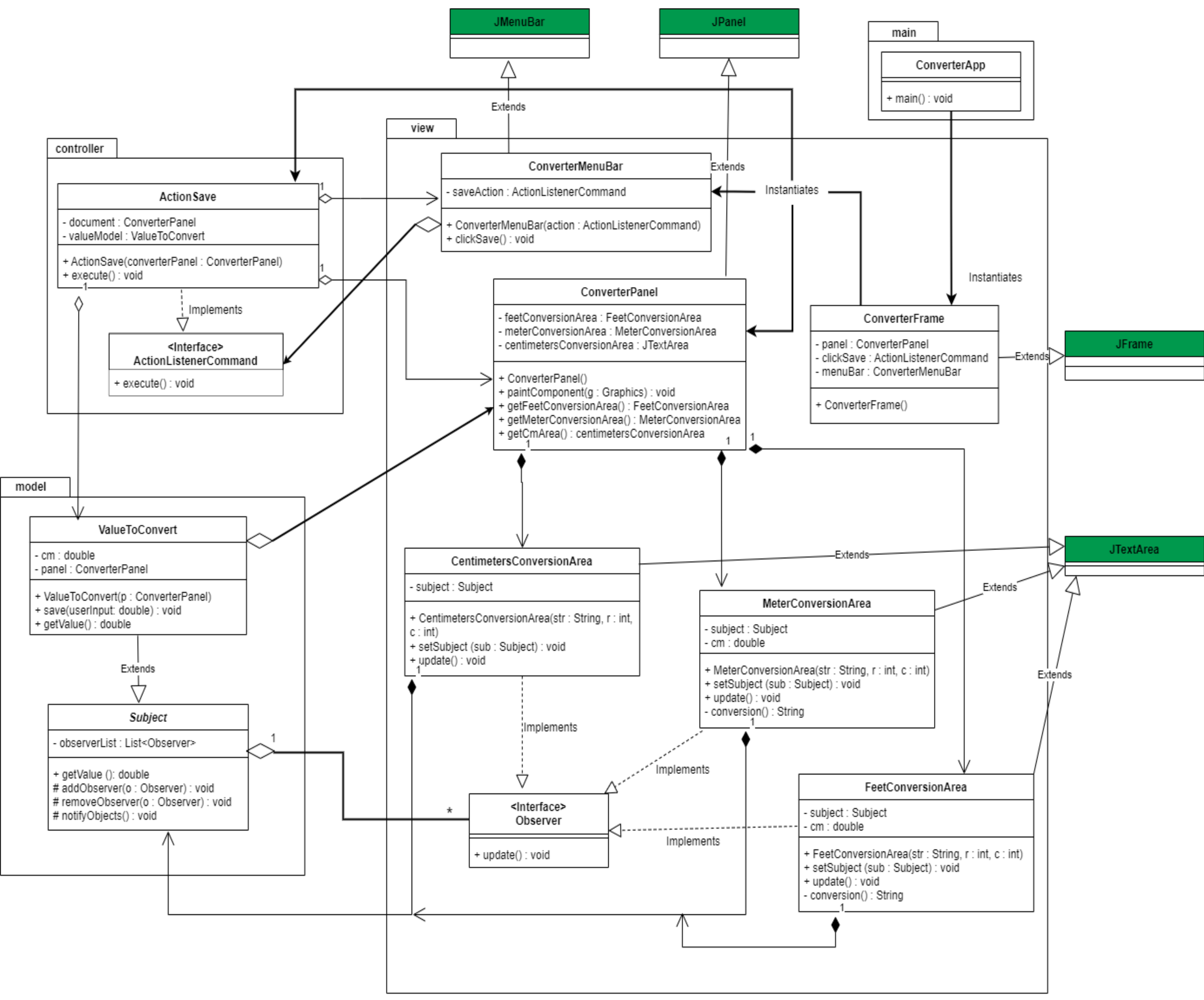
How are we going to structure our report?

This report begins with an introduction, showing the description of the project and mandatory design patterns, provides detailed explanations for all the classes and a UML class diagram, and concludes with the evaluation of the project.

The Converter Project
EECS3311 F2021
Naeiji Alireza (naeiji@yorku.ca)

Gbemisola Akerele (Student# 216167041)
Domenico Calautti (Student# 216 374 530)
Nancy Hao (Student# 217120197)
Misato Shimizu (Student#215310246)

PART Ⅱ (Design)


JMenuBar
JPanel
main
ConverterApp
+ main() : void
Extends
view
controller
ActionSave
- document : ConverterPanel
- valueModel : ValueToConvert
+ ActionSave(converterPanel : ConverterPanel)
+ execute() : void
ConverterMenuBar
- saveAction : ActionListenerCommand
+ ConverterMenuBar(action : ActionListenerCommand)
+ clickSave() : void
Extends
Instantiates
Implements
<Interface>
ActionListenerCommand
+ execute() : void
ConverterPanel
- feetConversionArea : FeetConversionArea
- meterConversionArea : MeterConversionArea
- centimetersConversionArea : JTextArea
+ ConverterPanel()
+ paintComponent(g : Graphics) : void
+ getFeetConversionArea() : FeetConversionArea
+ getMeterConversionArea() : MeterConversionArea
+ getCmArea() : centimetersConversionArea
Instantiates
ConverterFrame
- panel : ConverterPanel
- clickSave : ActionListenerCommand
- menuBar : ConverterMenuBar
+ ConverterFrame()
Extends
JFrame
model
ValueToConvert
- cm : double
- panel : ConverterPanel
+ ValueToConvert(p : ConverterPanel)
+ save(userInput: double) : void
+ getValue() : double
Extends
Subject
- observerList : List<Observer>
+ getValue (): double
# addObserver(o : Observer) : void
# removeObserver(o : Observer) : void
# notifyObjects() : void
CentimetersConversionArea
- subject : Subject
+ CentimetersConversionArea(str : String, r : int, c : int)
+ setSubject (sub : Subject) : void
+ update() : void
Extends
JTextArea
Extends
MeterConversionArea
- subject : Subject
- cm : double
+ MeterConversionArea(str : String, r : int, c : int)
+ setSubject (sub : Subject) : void
+ update() : void
- conversion() : String
Extends
Implements
Implements
<Interface>
Observer
+ update() : void
Implements
FeetConversionArea
- subject : Subject
- cm : double
+ FeetConversionArea(str : String, r : int, c : int)
+ setSubject (sub : Subject) : void
+ update() : void
- conversion() : String
1
*

How have we used design patterns?

Observer Pattern containing classes:

Subject: This is a Subject which observes operations to add and remove Observers.
ValueToConvert: This is a Concrete Subject that stores and manages the user input value.
Observer: This is an Observer which observes Subject and receives a notification from Subject when it makes any change.

FeetConversionArea: This is one of the Concrete Observer for a feet conversion. This also updates the obtained value by itself.
MeterConversionArea: This is one of the Concrete Observer for a meter conversion. This also updates the obtained value by itself.

Command Pattern containing classes:
ConverterFrame: This is a Client which instantiates ConverterPanel and specifies it to ActionSave, and instantiates ConverterMenuBar to add ActionListenerCommand.
ConverterMenuBar: This is an Invoker which asks ActionListenerCommand to perform tasks.
ConverterPanel: This is a Receiver which receives requests and performs specified actions.
ActionSave: This is a ConcreteCommand which utilizes a method inherited from ActionListenerCommand.
ActionListenerCommand: This is a Command which deals with the main action, execute().

How have we used OO design principles in the class diagram?

In the class diagram, inheritance is used in ValueToConvert class to inherit several methods in Subject class, such as addObserver method, adding each Observer to a list, and notifyObjects method, asking each Observer in the list to update its values.

PART Ⅲ (Implementation)

Regarding all the classes included in the class diagram
Main
- ConverterApp: This is a client class which triggers the whole program.

Controller
- ActionListenerCommand: This behaves as a Command for a Command design pattern. This contains an execute method which can be inherited by a Concrete Command, ActionSave.
- ActionSave: This class implements ActionListenerCommand and cares for all the user actions in the program such as clicking a save button.

Model
- ValueToConvert: This class extends Subject, containing two conversion areas. This class encapsulates an input value in centimeters and notifies FeetConversionArea and MeterConversionArea that a user changed a value in the main panel.
- Subject: This class stores a list of Observers for an Observer design pattern. This plays an important role to notify two areas when a user changed and saved an input.

View
- ConverterMenuBar: This class deals with the interface of the menu bar on the top left of the main panel. The class inherits the JMenuBar class for some features as well.

The Converter Project
EECS3311 F2021
Naeiji Alireza (naeiji@yorku.ca)

Gbemisola Akerele (Student# 216167041)
Domenico Calautti (Student# 216 374 530)
Nancy Hao (Student# 217120197)
Misato Shimizu (Student#215310246)

- ConverterPanel: This class sets up the basic interface with three colored panels.
- ConverterFrame: This class inherits the JFrame class to structure the whole program.
- FeetConversionArea: This class extends JTextArea for allowing texts on the panel and implements Observer for structuring an Observer design pattern. There is also a method to convert a saved value into feet.
- MeterConversionArea: This class extends JTextArea for allowing texts on the panel and implements Observer for structuring an Observer design pattern. There is also a method to convert a saved value into meters.
- CentimetersConversionArea: This class extends JTextArea for allowing texts on the panel and implements Observer for structuring an Observer design pattern. The update method does not take any action as the user input in centimeters does not have to be converted.
- Observer: This is an interface for an Observer design pattern. It holds a method to update values for a feet conversion and meter conversion.

Regarding the tools/libraries we have used during the implementation

Eclipse 4.18.0 with JDK 15.0.1 is used for this project. Diagrams.net is used to illustrate the UML class diagram. Google Doc is used for the members to collaborate on the report. Github is used to host our software project.

A short video showing how to launch the application and run it

As all the explanations fit in a shorter time, the length of the video is 1 minutes and 20 seconds.
https://drive.google.com/file/d/1fK-VzlzEzPUxvA6uZNQbOkOH2a56Ayw8/view?usp=sharing

PART Ⅳ (Conclusion)

What went well in the project?

The project took less time to complete as it included relatively fewer classes and it was a simple application, and with the experience from working on a project with the same members in a previous lab project, each task went smoothly without any issues.

What went wrong in the project?

To follow the requirements for design patterns, many modifications on code architectures were needed. Due to us focusing on short term software development goals, design debt added up, which took a lot of time to correct for restructuring to make sure each design pattern and requirement was included.

Gbemisola Akerele (Student# 216167041)
Domenico Calautti (Student# 216 374 530)
Nancy Hao (Student# 217120197)
Misato Shimizu (Student#215310246)

The Converter Project
EECS3311 F2021
Naeiji Alireza (naeiji@yorku.ca)

What have we learned from the project?

The payment for the design debt in this project taught us that it is important to follow a software development model, and to not focus on short term goals. Although for the previous lab project, we chose Singleton and Factory patterns, which all the members are most familiar with, the current project required two particular patterns, Observer and Command. It provided us with an opportunity to learn them in more detail instead of letting us use any design patterns which we are good at.

What are the advantages and drawbacks of completing the lab in a group?

As all the members are familiar with each other from the previous project, the whole process of completing the current project went smoothly, and there were no conflicts over task distributions. In the earlier lab, we have learned that miscommunication among members might cause overwriting of other members' completed code, resulting in not facing any drawbacks in this current project. An advantage of completing the lab in a group was learning how to collaborate on a software project with others, as well as how not to collaborate.

Indication of different tasks assigned to members

Gbemisola: Created a basic structure in Github, modified some fields, modified MVC structures, worked on a UML class diagram and created a video.
Domenico: Contributed to gui, arranged code to follow MVC, and created Command and Observer design patterns.
Nancy: Created the general appearance of the application, added a feature of conversions and modified a UML class diagram.
Misato: Created and added contents of a report and added Javadoc to the project.