Names:

Hassan Hassan - Section A – 215216930

Andrew Ngov – Section B – 217227331

Khalid Gobin – Section B – 216853483

Sabawon Stanikzai – Section B - 214233910

Part 1: Introduction

This software project is to develop an application that displays three JTextArea components each representing the length in feet, meter and centimeter. Whenever the user enters a centimeter value and clicks on the menu item labelled “save input centimeters”, the observer notifies the respected JTextArea components and then the input centimeters are converted to the feet and meter values respectively and shown in their respective JTextAreas.

The goal behind the development of this project is to able to figure out:

- How to implement the MVC pattern.
- How to design the GUI window.
- How to design and place JTextAreas using the layout manager.
- How to make the Observer that receive notifications when there is some changes in the model.
- How to fire and manage the ActionEvent.

The following challenges associated to the software project:

- Figure out what classes need to be defined to achieve the goal.
- Figure out how all the classes are integrated to meet the requirements.
- Determining how the GUI is designed.
- Determining how to draw various components and what should be their sizes and locations.
- Choosing the event to trigger the clicking on menu.
- Determining when to update the model.

The following concepts of OOD is used:

- Encapsulation

  All the classes are designed using the concept of encapsulation, they have their own data sets that is needed to design an object of that particular class. For example: Each object of MyJWindow class will have its own set of JTextArea, MenuBar, Menu, MenuEvent and Controller objects.
- Abstraction

  All the instance variables of the classes are made private, and we have provided setter and getter methods to access or modify the properties of an object indirectly.
- Inheritance

  Inheritance is a tool to reuse a software component as parent/child relation. We have made use of inheritance, and an example is the class FeetConversionArea class that inherits the JFrame class and implements the Observer interface.
- Polymorphism

  “One name many forms”
  In this project, the FeetConversionArea and MeterConversionArea both implement the Observer interface and these classes define the abstract method named update() of their own, according to their particular need.

Design pattern

The MVC (model-view-controller) is an architectural pattern that allows separating a system into three main subsystems: the Model, the View, and the Controller.
MVC supports a separate implementation of the model from the view.
The model, the controller and the view respectively consist of groupings of elements (e.g. Classes).
Each subsystem of the MVC can be further structured using design patterns.

Inheritance pattern

This pattern specifies the reuse of the software components as parent-child relationship. The child gets all the features from its parent automatically, without having to specify them again.

Observer Pattern

This pattern facilitates the sending of the notifications to the registered observers, whenever any special condition or situation arises.

Part 2: Design of the solution

Basic components of a UML class diagram

- Upper section: Contains the name of the class.
- Middle section: Contains the properties or attributes of the class.
- Bottom section: Includes class operations (methods).

Access attributes are specified as

+ public access

- private access

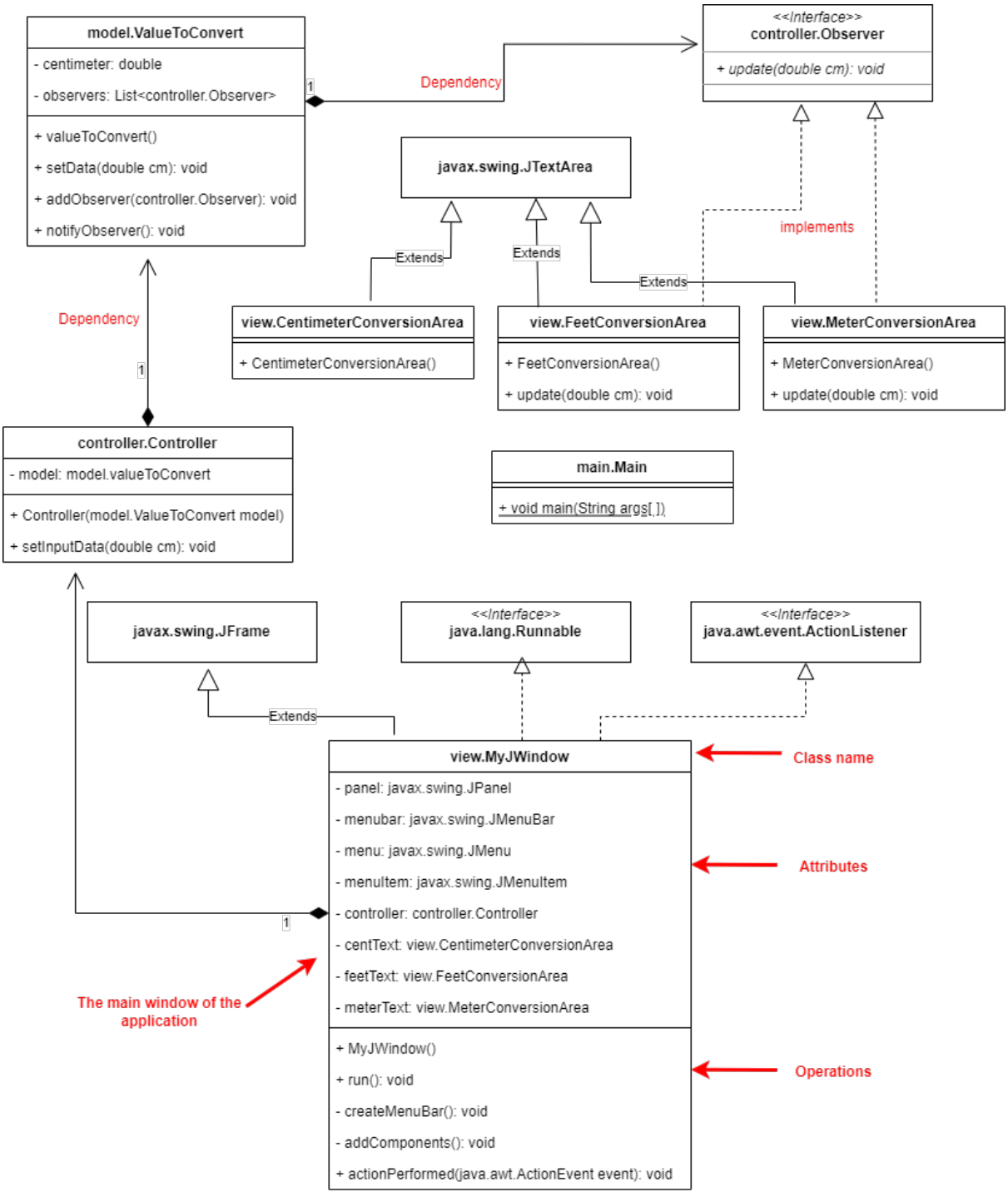
protected access

Basic elements of a UML class diagram

Class: set of objects having identical responsibilities. A class embodies a concept which encapsulates state(attributes) and behaviour(operations).

Interface: set of operations specifying the responsibility of a class.

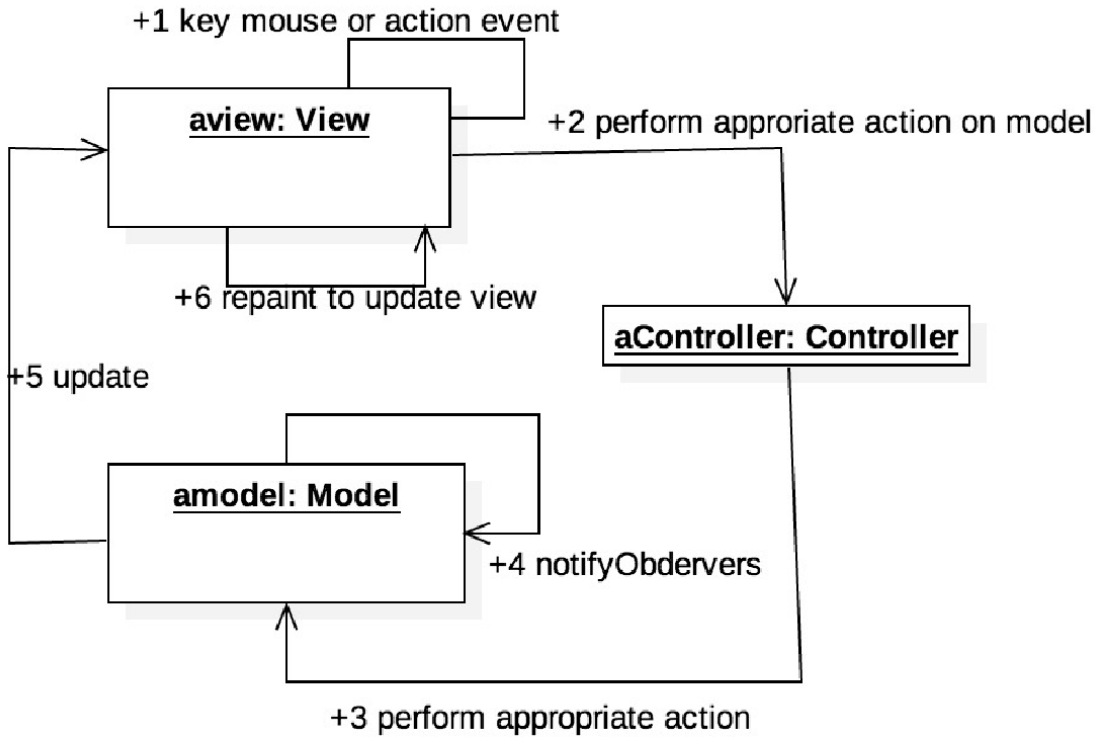
UML class diagram



Design Patterns used in the project:

1. Observer Pattern

   The FeetConversionArea and MeterConversionArea classes are Observers (implements Observer interface) that are registered with the ValueToConvert model, that gets notifications from the Controller, whenever the user clicks on the respective menu item.



2. Inheritance Pattern:

   FeetConversionArea and MeterConversionArea class inherits the JFrame class and implements the Observer interface, so that both of those classes get the common characteristics from JTextArea class.

OO design principals in the class diagram

Observer interface

It is an interface that has an abstract method named update (double centimeters) that is defined by the subclasses named FeetConversionArea and MeterConversionArea according to their own requirements.

ValueToConvert class

It is a model class for MVC pattern, that has registered all the Observer objects that wish to receive notifications on some special event.

Controller class

This class has encapsulated the model and it issues notifications to the model, whenever it needs to.

Part 3: Implementation of the solution

controller.Observer interface

This interface has an abstract method named update (double centimeters). This abstract method is intended to be defined by the subclasses named FeetConversionArea and MeterConversionArea according to their own requirements.

view.FeetConversionArea

This class extends the JTextArea and Observer interface. It draws a JTextArea where some text can be shown to user and defines the update (double centimeter) method of Observer interface that updates the data converted to feet, in the JTextArea component

view.MeterConversionArea

This class extends the JTextArea and Observer interface. It draws a JTextArea where some text can be shown to user and defines the update (double centimeter) method of Observer interface that updates the data converted to meters, in the JTextArea component

view.CentimetersConversionArea

This class extends the JTextArea. It draws a JTextArea where some text as centimeters can be input by user.

model.ValueToConvert

This class represents the model of the MVC pattern. It has encapsulated centimeter data and a list of Observer objects. It has addObserver() that adds the given Observer object to the list. It has a method named notifyObserver() that calls the update() for all the registered Observer objects in the list. It has an additional method named setData(double cm) that sets the given centimeter value to the centimeter field and invokes the notifyObserver() to send those a notification to update.

controller.Controller

This class act as Controller in the MVC pattern. It has encapsulated the model named ValueToConvert in it. It has a method named setData(double cm) that calls the setData(double cm) of the model.

view.MyJWindow

This class extends the JFrame and implements the Runnable and ActionListener interfaces.It creates the Window , FeetConversionArea , MeterConversionArea , CentimetersConversionArea , MenuBar , Menu , MenuItem and a Controller object. It has createMenuBar() method that creates the menubar , menu and menu item , and then registers the menu item with the same class same object , to listen the ActionEvent. adComponent() method is used to design the window by positioning all the JTextArea components on it. It has defined actionPerformed() method of ActionListener interface , that sets the centimeter data to the controller object.

All the classes are compiled and executed on Eclipse IDE.

Eclipse IDE for Java Developers

Version: 2020-12 (4.18.0)

Build id: 20201210-1552

OS: macOS 11.1

Java version: 12.0.2

Part 4: Conclusion

The MVC pattern helped here a lot like whenever user clicks the menu item of view, the input centimeter data is set to the Controller object that further issues notfications to the model and then that model calls the update() method of Observer objects.

Inheritance and polymorphism principles of OOD also helped here a lot. The classes were created using the principal of inheritance so that FeetConversionArea and MeterConversionArea classes inherited the Observer interface . Observer interface provided an abstract method named update() , that is defined by all of those subclasses according to that particular need and while creating an object of that particular Observer, the respective method is chosen automatically due to the polymorphism.

Initailly we did not encapsulate the Model object in the Controller class, so we were having problems designing the solution but we learned how to use the MVC, Observer pattern, how to manage the action events and it helped us to design the solution.

Teamwork has lot of advantages because everyone has different strengths, so we were able to split the work evenly and help each other if needed. A big problem is broken down into several small problems using the principles of OOP and then each team member is assigned a piece of the problem that he can do quite efficiently.

Abstraction, inheritance, and polymorphism are the top recommendations to solve a bigger assignment in much more efficient way.

Each member did the task assigned to them successfully and were split as follows:

1. Design the Observer interface and MeterConversionArea and FeetConversionArea.

2. Develop CentimetersConversionArea and MyJWindow class as view that maintains overall look of the application.

3. Develop the ValueToConvert class for model.

4. Design the Controller class that act as the controller.