Design Principles

Open/Closed

The classes are not able to be modified however they are able to send data which can't be changed and thus send a copy of the data. This is done using getters and setters, which is done in the WeatherTimeLapse class and the GUI by creating objects of the class to obtain access to its methods. These classes cannot be changed by the user from the GUI when the program is running.

Liskov Substitution

The GUI must be able to use objects of the WeatherTimelapse class without the class knowing that they are being used. Thus, we have created objects TempObject of the WeatherTimelapse class, however we never reference or change the object within the WeatherTimelapse class.

Dependency inversion

High level modules should not depend on low level modules. The entire project depends on the monitor which is an interface without it the classes would throw a NullPointerException. We are depending on the interface which is not a concrete class.

Reuse/Release

This principal is implemented in the GUI class as well as the other classes. The methods can be reused and released such as the createInternalFrames method that creates the basic internal features however it is used several times after each use, it is released and can be used again by another classes method. Stage 1 and Stage 2 are using different packages that are being reused and released such as the MelbourneWeather2Stub.

Common reuse

The packages that are used within the classes are all reused in that if you use one of them it is obtained and maintained within this program as only the necessary packages are called and are reused when the methods are called once more. There are no unnecessary packages being called or implemented. The classes that aren't used together are not grouped together. This is evident in the classes for weatherTimeLapse are it is separated from the stage 1 classes temperature and rainfall.

Stable dependencies

A package should only depend upon packages that are more stable than it is. Both the weathertimeLapse and the weatherMonitor Stage1 and 2 are only dependent on these two packages. The weather service itself is a stable package.

Interface segregation

The methods that implement the interface are not affected by unused methods. All the classes are independent and only have and use the methods they require in order to complete their tasks. The interface doesn't instantiate methods that are not required by both the weatherTimeLapse and weatherMonitor.

Common closure

The classes that would be changed at any given time are grouped together into their own packages such as the melbourneWeatherTImeLapse and the MelbourneWeather2. They each have their own classes that if effected would need to be changed together.

Acyclic dependencies

The construction of the graph all end at the display point and don't form any cycles between classes. There are no cycles being formed between the different packages within the classes. The flow of the program ends at one point waiting for the user to enter into the system in order to perform a function.

Stable abstractions

Most classes that were created are concrete classes the interface is inherently abstract. The interface is an abstract class and it is stable and interacts with the GUI of the system. None of the other classes required abstraction as there wasn't much in common between them to abstract.

Observer monitor

The program would be notified each time the user interacts with the system.

Design patterns

MVC

Model view controller

The model being the main core of the program

The controller would be the driver in this case as it controls the program and the outcome of it as well.

The view concept would be covered by the GUI and how the data is stored and displayed to the user.

Observer

The observer design pattern has been implemented. The program mostly designed around the GUI and the users input. So, the program observers if there is a change to the GUI and performs the necessary functions and display output once an event has occurred. The entire process for Stage 1 and Stage 2 are observed and recorded.

References

Shalloway,A., & Trott,R. (2008). Design patterns explain simply. A new perspective on object orientated design. ISBN: -13:978-0-321-24714-8