Pattern-based Group Report

JIACHENG LI 10307457 ZONGCHEN YU 10310004 WENJIE LUO 10141127

1. The URL linking to where your source code is stored (GitHub or Gitlab).

https://github.com/Vintony/patternbased.git

1. A brief description of what your application system is about and what its main functional features are.

This application is used for word analyzation of politicians in twitter. When user enter his/her username or password, it will redirect into the politician page, which user could choose one or more politicians. Once clients choose the person, this will load and show data from Twitter. Statistics in this application can be sorted and filtered. Moreover, user can get the statistics analysis results based on current state.

The main features are:

1. User could choose one or more politicians
2. Present data from Twitter based on the choices of politicians
3. Search specific data and sort by filters
4. Analyze the data
5. UML Class Diagrams of the patterns used in your application.

* Abstract Factory Pattern

图片包含 屏幕截图

描述已自动生成

* Observer Pattern

图片包含 屏幕截图

描述已自动生成

* Singleton

图片包含 屏幕截图

描述已自动生成

* MVC

图片包含 屏幕截图

描述已自动生成

* Adapter

图片包含 屏幕截图

描述已自动生成

* Decorator Composite Strategy

图片包含 屏幕截图

描述已自动生成

4. A brief description of what problem each pattern aims to solve, why the pattern is needed and if there is a better solution without using the pattern.

The aim of utilizing strategy is to switch the interchangeable search algorithms without using a lot of switch sentences in a class. Strategy pattern can be replaced with using State pattern. The advantages of State pattern are approximately similar to the Strategy pattern while State pattern needs to define all states at beginning. Changing states can be more flexible and easier when using State pattern. Comparatively, strategy pattern can be easier for implementation and meets the principles of low coupling.

The purpose of adapter pattern is to solve the problems of transfer the format of tweets to make the clearer to see in the screen page. The advantage is that using this pattern can make the tweets data compatible. If this pattern is not used, a new class needed to be implemented in order to make the data compatible to current program. Therefore, using adapter pattern is the best solution.

For decorator pattern, it aims to add more function to the existing object. In this case, decorator is designed to add more filter conditions. Decorator pattern is more flexible than inheritance. We could use inheritance instead of decorator pattern, high flexibility means more options which can make program error-prone. However, decorator pattern is also a good choice on the level of code reuse.

Considering composite pattern, this can clearly show the hierarchy and composite among components and classes. Adding new component is easy. For users, the call of high-level modules is simple as users do not need to consider the processing of low-level modules, which means it is difficult to manage the whole part. We can use simple object composition method to replace composite pattern, but it can make the program structure complex. Thus, using composite pattern could be better.

When it comes to MVC pattern, it is necessary to have this pattern whereas it may take long time for separation of model, view and controller. We need MVC pattern since objects can clearly perform which task they could do. MVC makes code more reusable and easier to modify when requirements changed. Although in the early time, it can save time, but developers will spend more time on meeting changeable requirements and bug fixing. Overall, this pattern is a good solution.

The aim of abstract factory pattern is to instantiate TweeterObject which has different types of TwitterObject like tweeterabstract, tweeterdetail and tweeterlist that object creation can change without the effect of creation method. Therefore, once a specific object needs to be created, we just need to use that factory class. Without this pattern, this application can still be implemented. However, it will make code redundant and increase the abstract complexity of software. Therefore, no better solution can be used unless this pattern is applied.

The purpose of singleton pattern is to ensure this instance could only be created from this method of that class so that other classes instantiate it via that constructor. For Singleton pattern, it points to control each class can only have one object. Moreover, object can only be created from that public method in the class. For example, in this application, UserAuth class can save memory for database connecting. Whereas in UserAuth class, it has some objects, only single object could be created. It is not necessary to use Singleton pattern since this program is not a big application. Meanwhile, this program does not consume a lot of memories for database connection. Thus, it makes little difference whether use the Singleton pattern or not.

We use observer pattern to monitor the state of filter word, once it changes the state, this can be notified. Multiple tests can be used to authenticate if program meets the requirements. Therefore, use this pattern can somehow save the cost.

5. A brief description of how you have used different patterns together and what their relationships are.

Each pattern has different relationships based on the implementations and functions. The decorator pattern is used to implement the filter algorithm selected by strategy pattern. Strategy pattern, is implemented by class FilterInterpreter and interface FilterState, has an aggregation relationship with decorator pattern.

The method “notifyObservers(filterContent)” and class RegexAccelerator fulfill observer pattern, which associates with strategy pattern.

Class Datadisplay performs the adapter pattern, inherits class DataPrepreocess, that has an aggregation relationship with strategy pattern.

Strategy pattern has dependency on composite pattern, which is performed by class Analyze\_screen and analysisResult.

Singleton pattern implemented in class UserAuth has dependency on Abstract Factory pattern.

Abstract Factory pattern associates with other patterns.

MVC contains above patterns.

6. A brief description of how you have implemented each pattern.

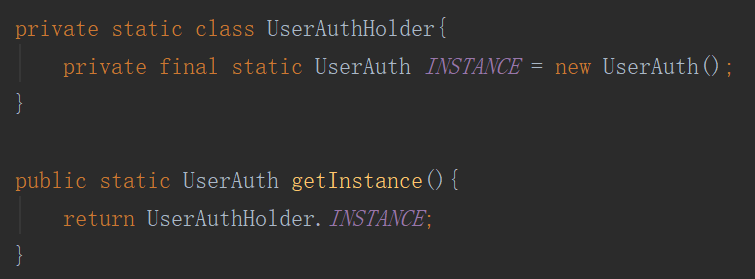
When considering the implementation of MVC. 3 packages (model, view and control) are created. We separate the UI into view, encapsulated data from the database in the model package and methods like search, sort are included in the controller package.

图片包含 天空

描述已自动生成

Singleton:

Considering the implementation of Singleton pattern, as we have database connection. Therefore, in order to save memory of that, a class named UserAuth is created in the following image to ensure that single object could only be created in this class. It can be suitable for the utilization of Singleton pattern.



Adapter:

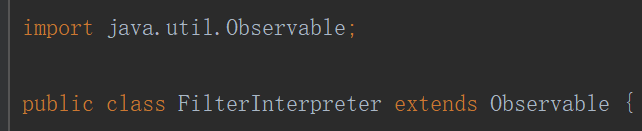
Class Datadisplay inherits Class DataPreprocess. Class Datadisplay not only implement the original function, but also override previous method to adapt new requirements. Class Datadisplay is the adapter of Class DataPreprocess.

图片包含 屏幕截图

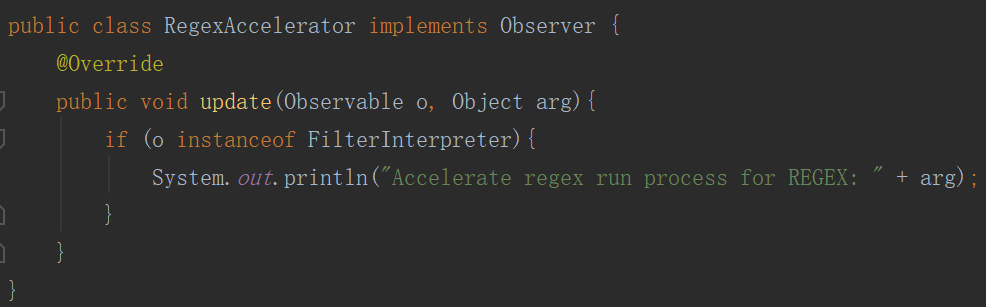
描述已自动生成

Observer:

Observer is implemented in the method “notifyObservers(filterContent)” in the class FilterInterpreter. There is another class called RegexAccelerator which has method “public void update ()” helping the implementation of observer pattern. implementations are shown in following images:







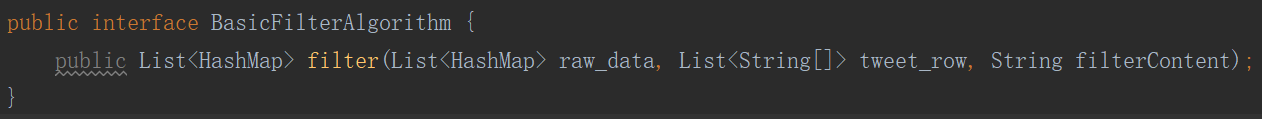
Abstract Factory:

We use an interface TwitterFactory as an object creation interface, through which client code can access different concrete Factory classes for creating different types of TwitterObject (e.g. TweeterAbstract, TweeterDetail and TweeterList).

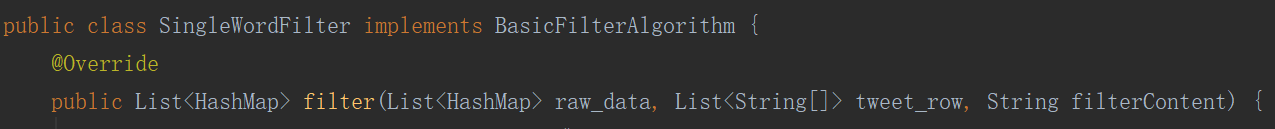
Decorator:

Decorator1:

BasicFilterAlgorithm interface defines the methods that will be implemented:

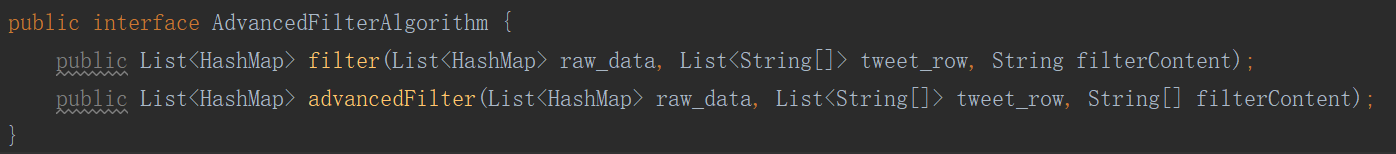


SingleWordFilter Class implements BasicFilterAlgorithm interface:



Decorator2:

AdvancedFilterAlgorithm interface defines the methods that will be implemented:



MutilwordFilter and RegexFilter Class implements AdvancedFilterAlgorithm interface:

图片包含 屏幕截图

描述已自动生成

图片包含 屏幕截图

描述已自动生成

Strategy:

Class SinglewordFilter, MultiwordFilter and RegexFilter are interchangeable algorithms decided by interface FilterState. In class FilterInterpreter, it judges the length of content and then transfer the results into the interface FilterState. After that, that interface will choose the algorithms accordingly. If filtercontent meets the requirement of Regex, then Regex state will be used.

Composite:

Class AnalysisResult is the leaf, Class Analyze\_screen is the component and the rationale of classes FilterInterpreter are composite. It modifies and shows the different results on the screen page. This pattern contains a lot of components such as filterInterpreter, filterState and filterAlgorithm.