PA1458 Example

Mikael Svahnberg

March 1, 2021

Contents

`file:///Users/msv/Documents/Teaching/PA1415_software_design/Material/202102-HomeExam-Example-1-en.org`

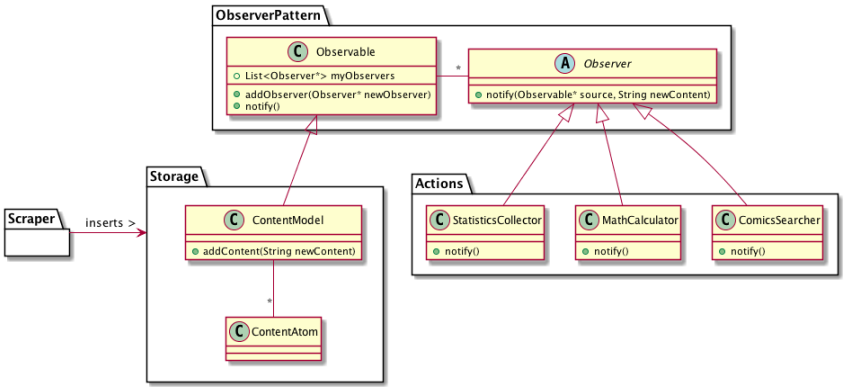
1 System Description

A web scraper that collects posts from social media platforms and when certain conditions are met, actions are taken.

2 Class Diagram

```
package Scraper {
' Not done in this exam
}
```

```
package Storage {
class ContentModel {
 +addContent(String newContent)
}

class ContentAtom

ContentModel -- "*" ContentAtom
}

Scraper -> Storage : inserts >

package ObserverPattern {
class Observable {
 +addObserver(Observer* newObserver)
 +notify()
 +List<Observer*> myObservers
}

abstract class Observer {
 +notify(Observable* source, String newContent)
}

Observable - "*" Observer
}

package Actions {
Observer <|-- StatisticsCollector
Observer <|-- MathCalculator
Observer <|-- ComicsSearcher

StatisticsCollector : +notify()
MathCalculator : +notify()
ComicsSearcher : +notify()
}

Observable <|-- ContentModel
```



3 Description of Class Diagram

The class diagram consists of a couple of packages:

Scrapers collect information e.g. from social media. Puts the data into Storage by calling the `ContentModel::addContent()` method.

Storage Creates new ContentAtoms based on the given input and stores them. Then it calls the `notify()` method to announce that there is new contents.

Actions Reacts to new contents.

ObserverPattern Contains the classes necessary for a generic Observer pattern.

The Observer pattern is used so that when new content is added via the `addContent()` method, it calls the `Observable::notify()` method. This method will run through all elements in `myObservers` and call their corresponding `notify()` method. The Observers (or the concrete instances, to be specific) will decide whether to take action or not.

4 Pseudocode

4.1 Observable::addObserver()

```
void Observable::addObserver(Observer* newObserver) {
  myObservers.add(newObserver);
}
```

4.2 Observable::notify()

```
void Observable::notify() {
 myObservers.forEach( function(o) {
  o.notify(this, newContents); // newContents is magically available.
});
}
```

4.3 ComicsSearcher::notify()

```
void ComicsSearcher::notify(Observable* source, String newContent) {
  String key = newContent.split()[0];
  if (myKeywords.find(key)) {
     // Do relevant stuff
  }
}
```

4.4 ContentModel::addContent()

```
void ContentModel::addContent(String newContent) {
  ContentAtom atom = new ContentAtom(newContent);
  DBHandler::store(atom);
  this->notify(newContent); // This is where the Observer pattern is used
}
```

5 Discussion of GRASP Patterns

- Information Expert
- Controller

The `Observable` (and sub-classes that inherit from `Observable`) are information expert on which `Observers` to call when the `notify()` method is called. It is also a controller, that delegates the responsibility of *acting* on new information to each of the observers (the classes that inherit from `Observer`). It offers an opportunity to each Observer to do whatever they please when notified.

The sub-classes to `Observer` are information experts on exactly what action to take when new information arrives via the `notify()` method.

6 Usage of GRASP patterns

ContentModel is an information expert on how to store new content. It is also an information expert on when to call the `Observers`.

ContentModel is (via the `Observable` class from which it inherits) an information expert on which Observers are available. See discussion of GRASP patterns above.

The sub-classes to `Observer` (e.g. the ComicsSearcher) is an information expert on which keywords that should trigger it, and what should happen when these keywords are mentioned.

ContentAtom is an information expert on one particular piece of contents.