matils Documentation

Release 0.1

Mathias Santos de Brito

May 04, 2018

Contents

Matils is a personal project intented to catalog utillities functions.

These function and classes that I came accross during my coding activties. One of the major target is to catalog the Design Patters implemented by me (not exactly utilities but let's say so! :P).

CHAPTER 1

matils package

This package is the central point to find nice utilities in Matils.

Check out the code documentation it is very rich and provides informations and valuable examples to start playing with Matils.

1.1 matils

1.1.1 matils package

Subpackages

matils.patterns package

Submodules

matils.patterns.observer module

Implementation of the Observer Pattern.

In this module you will find the necessary classes to use the Observer Pattern. It provides both the Observer and Observable classes that you must use to derive your classes.

This Implementation allows you to observer general events acurred in a object that extends the *Observable* class. To do that you need to implement the interface *Observer* in your class and register it againts the Observable that you want to track. Below an usage example, let's say that we have an Observable reading values from sensors, and notify observers for new values, the Observable code is shown below:

```
import time
import random
from matils.patterns.observer import Observable, Observer
```

(continues on next page)

(continued from previous page)

```
class SensorsReader(Observable):
    def read_sensors_data_loop(self):
        while True:
            # reads new sensors data each 2 seconds.
            temperature_data = self.get_temperature()
            self.notify(temperature_data, 'temperature')

            humidity_data = self.get_humidity()
            self.notify(humidity_data, 'humidity')
            time.sleep(2)

    def get_temperature(self):
        # return some Random Value between 0-40
        return {'value': random.uniform(0,40)}

    def get_humidity(self):
        # return a random value between 20-100
        return {'value': random.uniform(20,100)}
```

We want them to have an observer that will take action everytime a new reading is done in the sensors. For that we implement the following observer:

```
class SensorDataAnalizer(Observer):
    def update(self, sensor_data, event_name):
        print('Sensor data observerd, I will do some nice analysis from '
              'received data type: {}, data: {}'.format(event_name,
                                                        str(sensor_data)))
```

In our main code we have:

```
if __name__ == '__main__':
    sensors_reader = SensorsReader()
    sensors_analyzer = SensorDataAnalizer()

    sensors_reader.register(sensors_analyzer, 'temperature')
    sensors_reader.register(sensors_analyzer, 'humidity')

    sensors_reader.read_sensors_data_loop()
```

After you implement your observer you can register it in Observables, let's take the following *Observable*

class matils.patterns.observer.**Observable**

Bases: object

The changes in a Observable object can be observed by Observer classes.

The Observable manages a list of observers and have to make sure that all registered observers will be notifyed when the status of one of the observed attributes change.

One characteristic of this implementation is that Observers need to inform what they want to observe. The Observers optionally can inform the type of messages as a list of string so that its callback will be called only when the specific event occurs.

If an observer registers to observer a message unknown by the Observable no error will be generated, the observer will simply not receive notifications.

notify(*data*, *event*)

Notify observers registered to be update about the event.

All the observers registered to get all events must be notified.

Important: Be sure to document the types returned by each event, so that the Observers can implement correctly their update functions. To maintain the flexibility we decided to allow the Observable to send any kind of object.

Caution: This method do not check if an observer is registered at all and other event in the same :class:=‘Observable‘, if you do so in your code you will get notified more than once!

observers
:attr:Observable._observers getter.

register(*observer*, *event='all'*)
Register an observer to listen to events from this Observable.

This function manipulates the attribute self.obseervers by addying the ‘observer’ to the list in the right entry taking into consideration the event of interest. If ‘all’ is given to the parameter ‘event’, than the observer MUST be inserted in the ‘all’ entry of self.observers.

If an observer is already registered to observe to all events, if a request to observe an specific event arrive, the request will be ignored and the request will be considered successful.

reset()
Remove all registered observers.

Clean *Observable.observers*. This means that the attribute after a reset, must be exactly as it was when the obect was created.

The reference to the original dictionary object is kept and the ‘all’ entry are kept...

unregister(*observer*, *event='all'*)
Unregister an observer, to all or specifc events.

This method will unregister an observer, the default behavior is to unregister from all events. If a event name is given it will be unregistered only from the specified event.

The unregistering process is done by manipulting the ‘self.observers’ removing the entries to the given observer from the different events lists.

Returns True if the unregistration was successful, and False if the unregistration failed, or no action was taken (maybe observer not registered).

class matils.patterns.observer.**Observer**
Bases: abc.ABC

Intended to be implemented if the objects are whiling to observe.

If your class wants to observe an *Observable*, needs to implement this abstract class.

Observers can register multiple times, in different Observables if they want to do so. However, only the method update will be called, the logic to be executed based on the event needs to be handled inside the update method.

Important: Check the implementation of your Observable to understand what kind of data it is send on each notification. An notification can send any type of data.

---

update(*data*, *event='all'*)
To be called by an Observable if object is registered.

Module contents

This Package contains the implementation of different Desing Patterns.

The *patterns* package is intended to be a collection of useful Design Patterns ready to be used in your projects. We intend to be extremly pythonic in the implementation and provide scalability and performance.

Fell free to report any issue regarding performance or lack od scallability if you find some.

Module contents

This package is the central point to find nice utilities in Matils.

Check out the code documentation it is very rich and provides informations and valuable examples to start playing with Matils.

Python Module Index

m

Index