**Raditya Surya Pratama**

**Radu Alexandru Stoica**

Disk Scheduling

Strategy Pattern

STRATEGY PATTERN

The strategy pattern is a software design pattern that enables an algorithm’s behavior to be selected at runtime. The principle of this class is to encapsulate the common behaviors of an object such that they can easily be interchanged at runtime by the user. The most usual approach is to use subclasses which inherits interfaces in order to achieve a better decoupling between the behavior and the class that uses the behavior. Then, the classes can switch between different behaviors by changing only the implementation of each behavior without having significant code changes.



Source: <https://en.wikipedia.org/wiki/Strategy_pattern>.

UML DIAGRAM

REUSABILITY

The reusability of our code is represented by the absence of redundancy, by the unit tests and all our classes are meant to do only one thing. Also, the presence of the interface makes the code reusable.

EXTENSIBILITY

The extensibility is represented by the abstract interface used to define the behavior for each concrete strategy. Each concrete strategy is overriding the strategy behavior. Therefore, if there will be more algorithms in the future, the programmer can create a new concrete strategy which inherits the behavior from the strategy itself. Each concrete strategy will have its specific implementation. Also, some small changes are required for the GUI in order to use the new concrete strategies.

MAINTAINABILITY

In order to add more concrete strategies to strategy, it should be easy for the programmer. The time spent will be for creating the additional class and the specific algorithm implementation. Also, it is required to add a new statement in the switch/case in the ‘Simulation’ class, which is the control class.

A new radio button should be placed on the GUI and a new statement in the ‘DecideWhichAlgIsSelected’ method in order to let the application to select the new concrete strategy.

PATTERN DOWNSIDE

There are a couple of disadvantages of using the strategy pattern:

* The application must be aware of all strategies to select the right one for the right situation;
* Sometimes the input data from Context to Strategy may be not relevant to all the Concrete Strategies;
* Strategy must expose all behaviors to the Concrete strategies, even if some concrete strategies don’t use all the behaviors.
* The application needs to create and maintain two objects in place of one.

Source: <http://bobcash20.blogspot.nl/2007/05/strategy-pattern.html>.

UNIT TEST