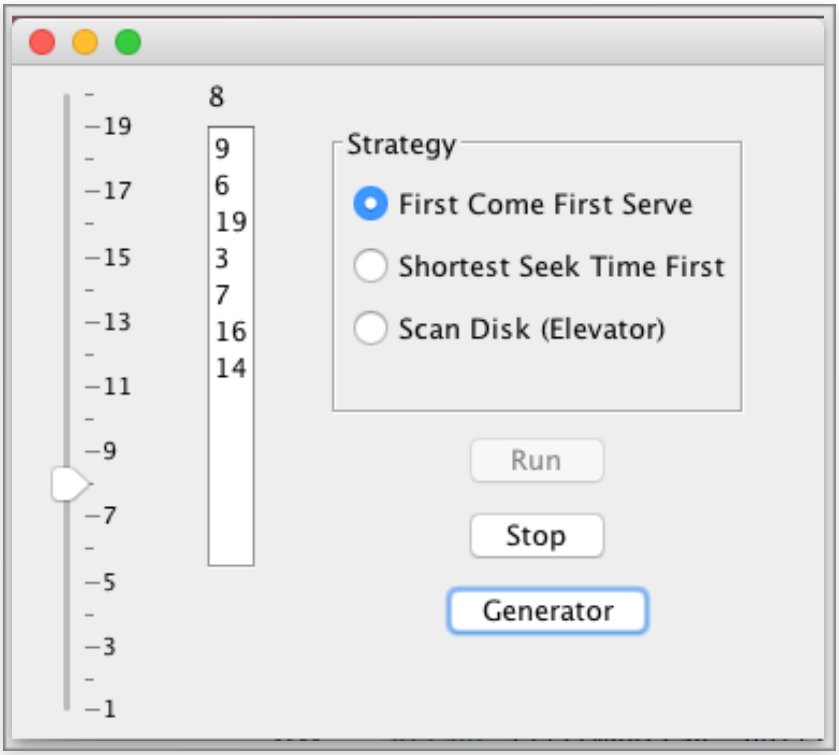
ANTHONY GUILLIER



Strategy Patern

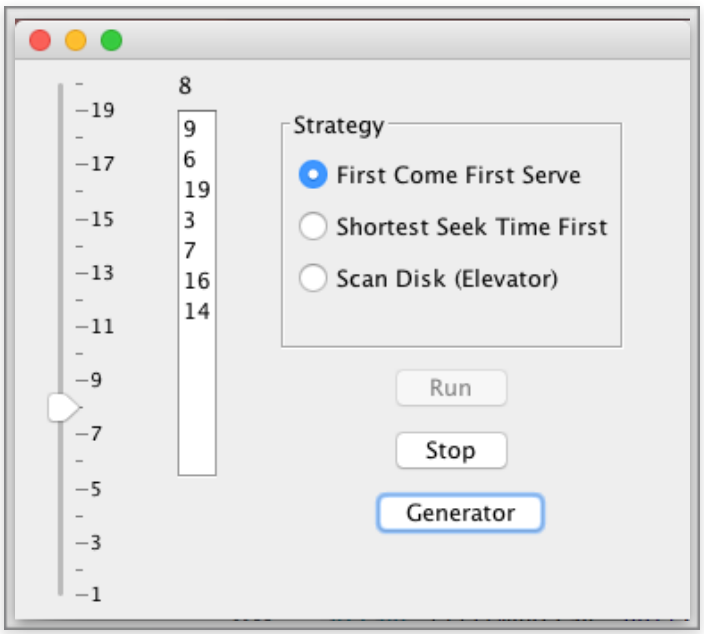
Class : Design Patern
Student : Anthony Guillier
11/02/2017
Group : 10

ANTHONY GUILLIER

STRATEGY PATTERN

Description of the application

This is an object-oriented class design for the **Disk Scheduling** of an (simplified) Operating System. An Operating System can have several different algorithms for handling the requests for a hard disk (like First Come First Serve, Shortest Seek Time First, etc.).
In our application, the read requests are given via the GUI (a list of numbers which represent the sectors on the disk).

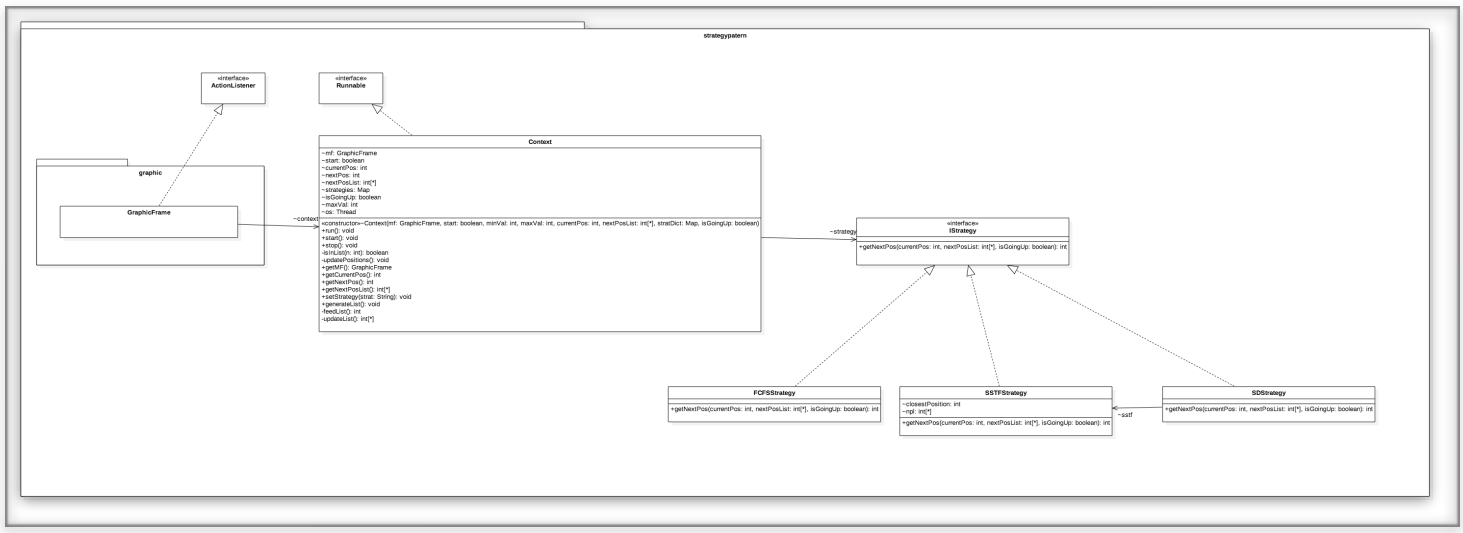


Here, three strategies are implemented:

- First Come First Serve: Your next position is the next position in the list.
  - Really Simple: only have to return the number at the position 0 of the list.
- Shortest Seek Time First: Your next position is the closest in the list.
  - In aim to find the closest position you have to sort your list in a function so you know that if n > n-1, n-1 is the closest position in the list, you don't need to go further.
- Scan Disk: If the cursor is going up your next position is the closest biggest in the list, if you are going down is the closest smallest.
  - First you need to use the SSTF strategy to get the closest number in a sorted list, then you check if this number is bigger or smaller than the current one. If you cursor is going up and your next number is bigger you return this one, else you return the biggest number of the list. You do the exact inverse to going down. The algorithm check if the new number is bigger that the current one to know if we are going up.

ANTHONY GUILLIER

UML



Discussion about Reusability

Using this Pattern you only have to change getNextPos() or add new functions and you have a whole different application doing something completely different but with the exact same pattern.
It is also possible to create a new Frame with a new design using the Context class as a scope.
Accessor of the “Context“ class allows to reuse this class where you want, whiteout knowing how it works.
This pattern in fact already reused itself: the class “SDStrategy“ contains a “SSTFStrategy“ class so the algorithm first used “SSTFStragegy.getNextPos()“ to get the closest number in a sorted list of “nextPosList“ and after checking if the number is bigger or smaller, you can set the next position knowing if your cursor is going up or down.

Discussion about Extensibility

The different strategies are implementing an "IStrategy" interface, that means you can have as many strategies as you want. Furthermore, the class “Context“ contains strategies in a Map variable: “strategies“, which means you just need to add your strategy to this Map and you will be able to use it in Context class without changing anything in the constructor or functions.
Adding new elements in the frame (buttons, list, label etc.) that change context’s element is possible if you bind the frame’s element with a new function you made in “Context“ class.

Discussion about Maintainability

Thinking this way, if you have a better algorithm, you only have to change the getNextPos() of your class in aim to improve it.
Mutator of the “Context“ class allows to change Context’s elements, so is behavior.
Everything is separate in little functions making bigger ones, so you can improve or change the way a function is working, without changing the entire algorithm.