|  |
| --- |
| Photo displaying partial image of two pie charts on a canvas-textured page |
| Strategy Pattern  Group 16 |
| |  |  |  | | --- | --- | --- | | Lyubomir Dimov , Nikola Nikushev | 2/21/16 | Design Patterns | |

Contents

[Introduction 2](#_Toc443847937)

[About Strategy Pattern 3](#_Toc443847938)

[What is strategy? 3](#_Toc443847939)

[When is used? 3](#_Toc443847940)

[What is Strategy pattern? 3](#_Toc443847941)

[Class Diagram 4](#_Toc443847942)

[Our Application and how it works 5](#_Toc443847943)

[Graphical user interface 5](#_Toc443847944)

[Program to Interface 7](#_Toc443847945)

[Operating System 9](#_Toc443847946)

[The form 9](#_Toc443847947)

[Conclusion 10](#_Toc443847948)

# Introduction

This document gives explanations about Assignment 1 from the Design Pattern course. You will find explanation of what is Strategy Pattern, how people used it and in which situations is used.

To present how actually Strategy Pattern works, an application was implemented. The whole process of work was in the following sequence - research about the pattern, understanding the pattern, making proper class diagram of the application, implementation, testing, documentation.

The application is about Disc Scheduling of an Operating System, using three types of scheduling FCFS, SSTF, SCAN.

In this document you will find:

* Strategy Pattern – What is it, how it’s used and why.
* Class diagram of Disc Scheduling application with a brief explanation of it.
* Pictures and explanation of Disc Scheduling application.
* The graphical user interface and what is each button used for.
* How we implemented and tested the scheduling methods
* Multithreading in the application

Let’s first start with the concept of Strategy Pattern.

# About Strategy Pattern

## What is strategy?

Strategy is a plan of action to achieve a specific goal.

## When is used?

A situation in which you are given context in which an algorithm is used, and you want to use different algorithms depending upon the situation or to avoid having to change the context. In this kind of situations is preferably to use Strategy pattern.

## What is Strategy pattern?

“Define a family of algorithms, encapsulate each one, and make them interchangeable. Strategy lets the algorithm vary independently from clients that use it.” – (Internet)

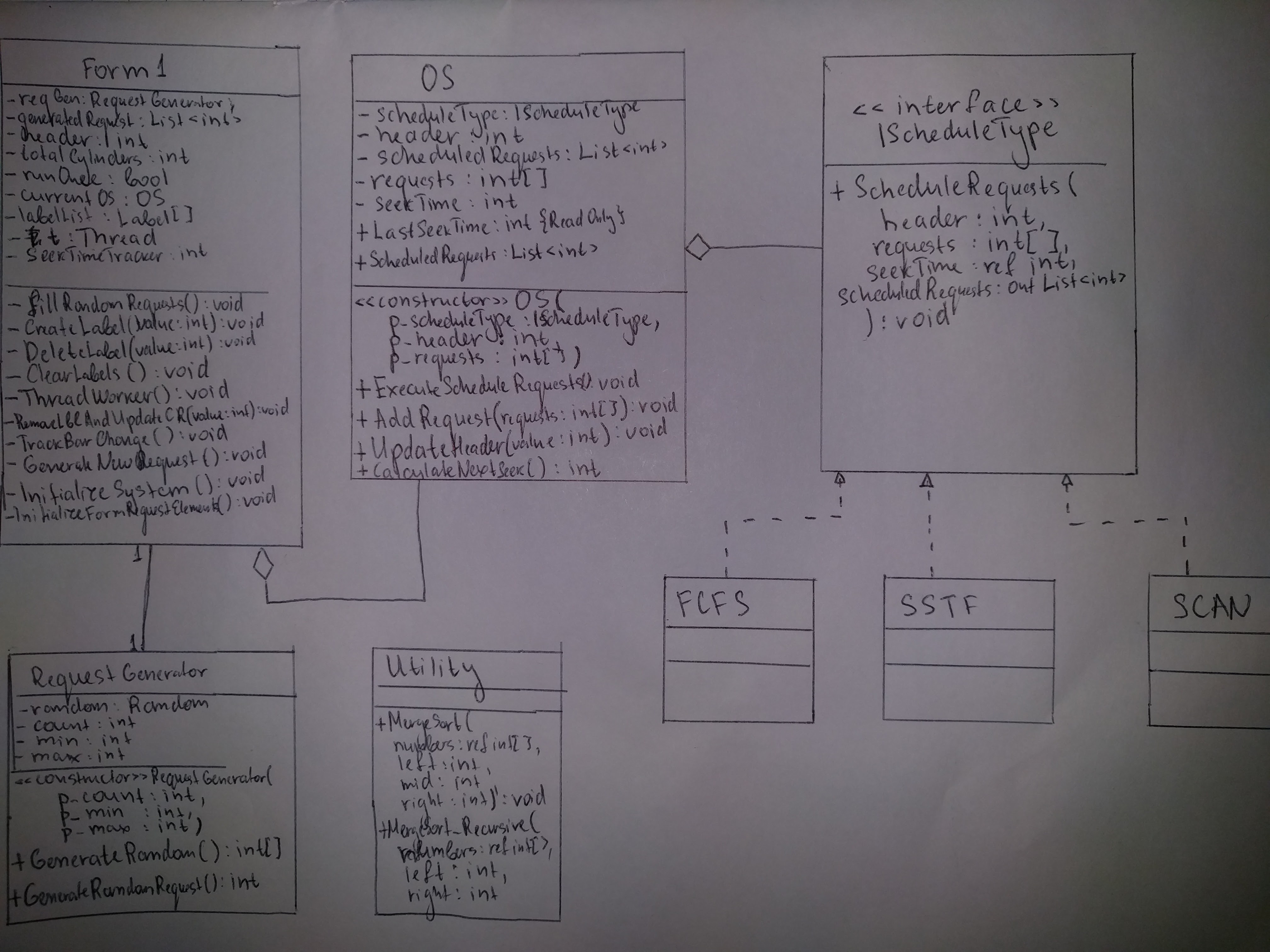


# Class Diagram

The class diagram was the initial part of the development of the application.

To apply properly the Strategy pattern we had to:

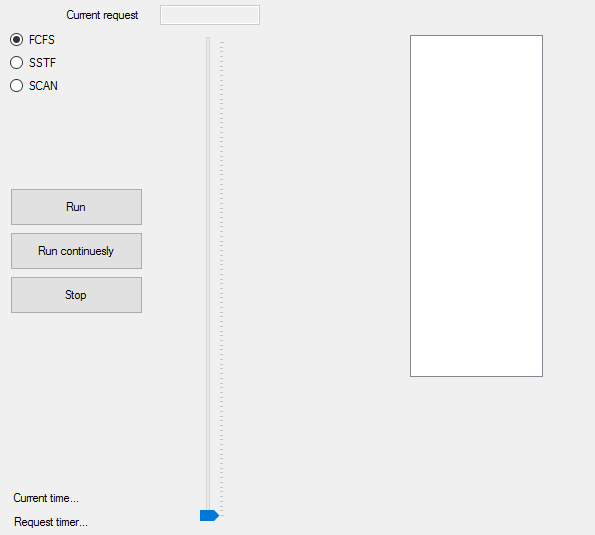
* Identify the aspects of our application that vary and separate them from what states the same.
* Program to an interface, not implementation
* Favor aggregation over inheritance



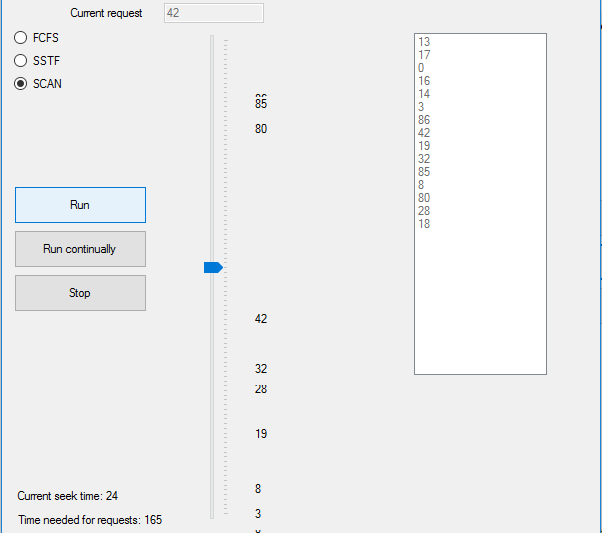
(note: there is an attached picture of the class diagram in the submission archive)

# Our Application and how it works

## Graphical user interface



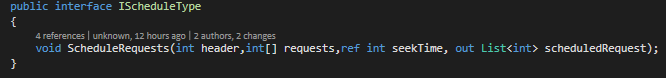
* Choice of Scheduling Type in the upper-left corner (FCFS, SSTF, SCAN)
* Run – A single 20 cylinders scheduling
* Run continually – The program generates new requests and schedules the current request.
* Stop – stops the scheduling
* Current Time…(label) – when the scheduling is running, on the place of that label the ‘current seek time’ will be shown. Current seek time means the cylinders it takes from previous request to current request.
* Request timer…(label) – When scheduling is running, on the place of that label the ‘Total time for requests’ will be shown (the number of cylinders the head will pass until it finishes the request)



* On the right side of the trackbar, numbers are illustrated which are the requests still not scheduled.
* Current request is illustrated in a text box on top of the trackbar.

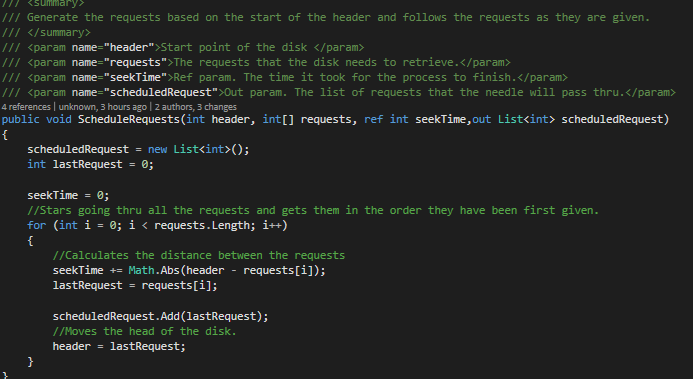
## Program to Interface

After we Identified the aspects of our application that vary (Scheduling types) we separated them from what states the same. And we encapsulated using the interface (IScheduleType).



Every Scheduling type implements this interface and what vary is the implementation of the ScheduleRequests(..).

Example: FCFS

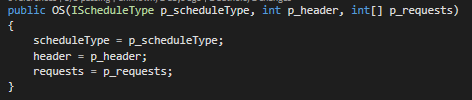


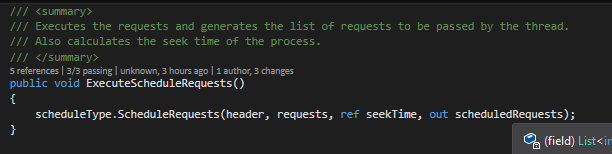
Benefits from this sort of implementation is that we eliminated conditional statements, we built families of related algorithms. This way is alternative to sub classing, and give us choice of implementation.

## Operating System

Our Operating System has to be simple. It has a given request and it has to be scheduled.

In the initialization of the OS(constructor), object of type IScheduleType (the interface) is also initialized. This object is used to call the specific scheduling type.





## The form

* An OS object is initialized and Scheduling type is specified.
* Requests are generated randomly by the RequestGenerator class object.
* Requests are depicted for better visual representation
* Multithreading is used for showing visually how the scheduling works. The trackbar is moving with delay of 30 msec in between scheduling request.
* The Scheduling can be run once for limited number of requests or it can be run continually until its stopped.

# Conclusion

This strategy is simple yet very effective. We never worked with patterns before and now we understood many of our previously mistakes, especially that object oriented programming is not that obvious and techniques like inheritance and polymorphism are not very efficient.

We look forward for learning some other design patterns.