Assignment 1 : Strategy pattern

Mark Ehrhart  
Johri van Eerd

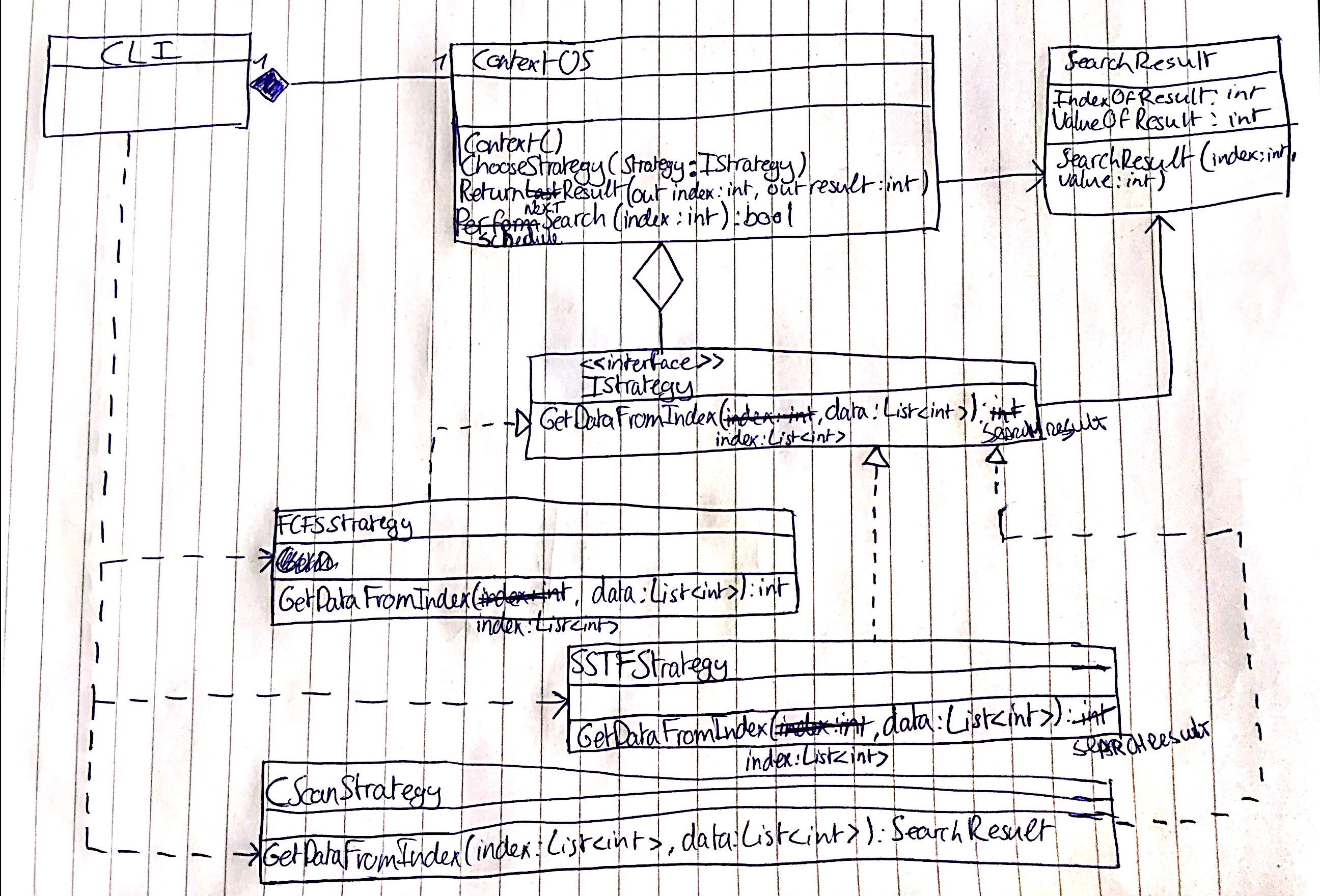
# Wat is de opdracht:

In de opdracht passen we het Strategy pattern toe op een probleem. Het probleem is dat een operating system uit verschillende zoek algoritmes wilt kunnen kiezen voor het opzoeken van data.

Om het probleem en opdracht te simuleren hebben we in C# een command line interface programma geschreven. In de CLI kan de gebruiker uit verschillende zoekalgoritmes kiezen: Shortest Seek Time First, Circular-scan en First Come First Serve. De gebruiker kan ook diverse zoekacties invoeren en uit laten voeren.

# Je eigen implementatie:

Op de volgende pagina is het handgeschreven UML-diagram voor onze implementatie te vinden. Het Strategy Pattern is terug te vinden in de componenten ContextOS, IStrategy en uiteraard de implementaties van de verschillende strategieën. We hebben ervoor gekozen om de CLI volledige macht over het beheer van de Context te geven, en het voor de CLI mogelijk te maken om verschillende strategieën toe te wijzen aan de Context. Er is een extra klasse SearchResult later nog bijgekomen om de resultaten van de zoekacties op een overzichtelijke manier over te kunnen brengen van Strategy naar Context.



# Discuss the 3 “problems”:

**Reusability**Als basis is het strategy pattern zonder de CLI klasse eenvoudig opnieuw te gebruiken. Dit heeft wel als voorwaarde dat een vervangende user interface voldoende heeft aan de data die vanuit onze Context klasse geleverd wordt, of dat een vervangend strategy pattern data in hetzelfde formaat aan kan leveren.

Ook kan onze Context oneindig gebruik blijven maken van een Strategy, omdat het opnieuw toekennen van een Strategy buiten het beheer van de Context ligt.

**Extensibility**

Na onze eerste implementatie met twee patterns hebben we nog een Strategy eenvoudig kunnen toevoegen. Er bestaan geen harde afhankelijkheden zodat alles wat aan IStrategy voldoet door de CLI aan Context toegekend kan worden. Vanaf dan is het alleen een kwestie van het algoritme implementeren en zorgen dat de nieuwe Strategy aan onze output voorschriften voldoet.

Voor het toevoegen van een nieuwe Strategy hoeven geen klasses uit het pattern aangepast te worden. Hiermee voldoet onze implementatie aan het Open/Closed Principle.

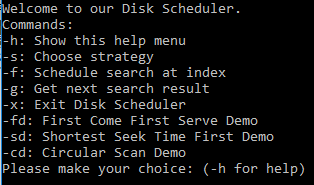
**Maintainability**

Elke Strategy is individueel aan te passen van elkaar, hierdoor hebben wijzigingen in een Strategy geen gevolg of invloed op een andere Strategy. Omdat we, zoals voorgeschreven voor het Strategy Pattern, gebruik gemaakt hebben van aggregaties in plaats van inheritance van klasses, kan elk stukje gedrag aangepast worden zonder dat klasses die gebruik maken van het gedrag kapot gaan, en andersom kunnen ook stukken gedrag gewijzigd worden terwijl de klasses die daar gebruik van maken daardoor onaangeroerd blijven.

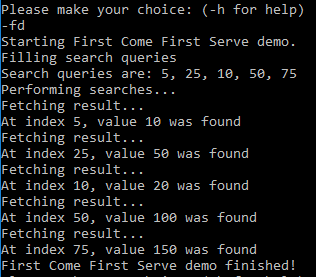
# Tests die je hebt gedaan:

Voor elk zoekalgoritme dat we geïmplementeerd hebben, hebben we een demo optie in het programma gemaakt. Voor elk zoek algoritme hieronder een screenshot van het resultaat.  
Onze data bestaat uit een lijst van 100 integers, welke gevuld zijn met waarde (index\*2). Op deze manier is makkelijk te verifiëren of het gevonden resultaat ook daadwerkelijk klopt.

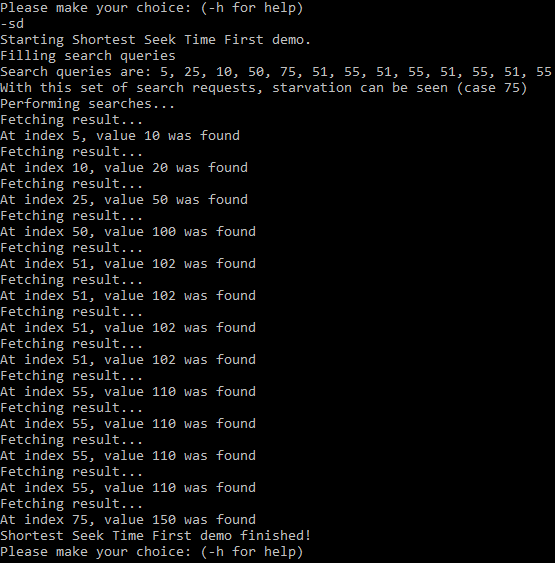
Application start:



First Come First Serve



Shortest Seek Time First:



Circular scan:

