Tristan Stevens
Zachary Schallenberger

PatternDemo

+ main(): void

In both patterns

- - Use - - -> (to Context)
- - Use - - -> (to Subject)

Context

- strategy: Strategy

+ executeStrategy(): int

<<Interface>> Strategy

+ doOperation(): int

Startegy A

+ doOperation(): int

Strategy B

+ doOperation(): int

Subject

- observers: List<Observer>
- state: int

+ getState(): int
+ setState(): void
+ attach(): void
+ notifyAllObservers(): void

Observer

- subject: Subject

+ update(): void

Extends | Extends

Observer A

- subject: Subject

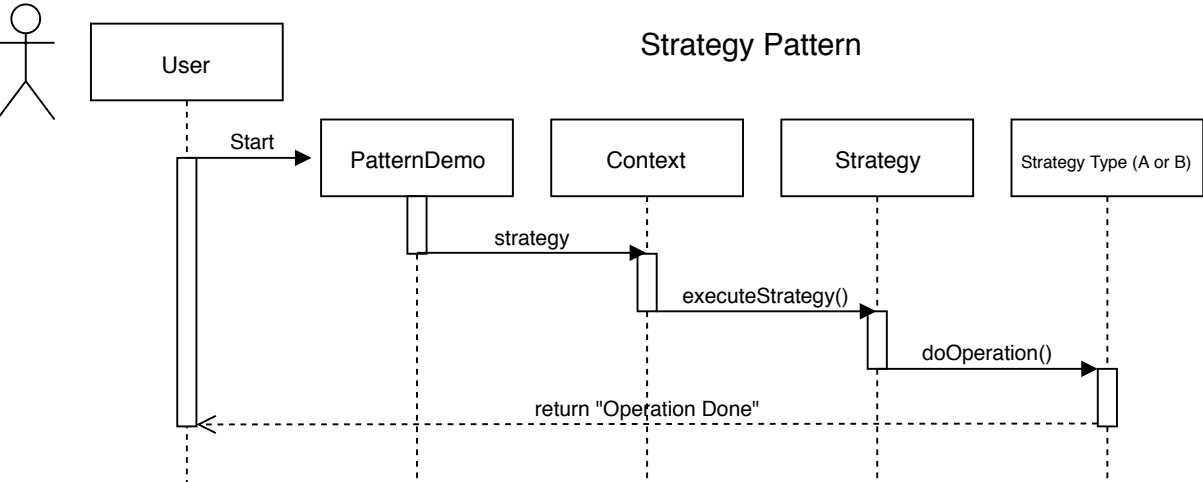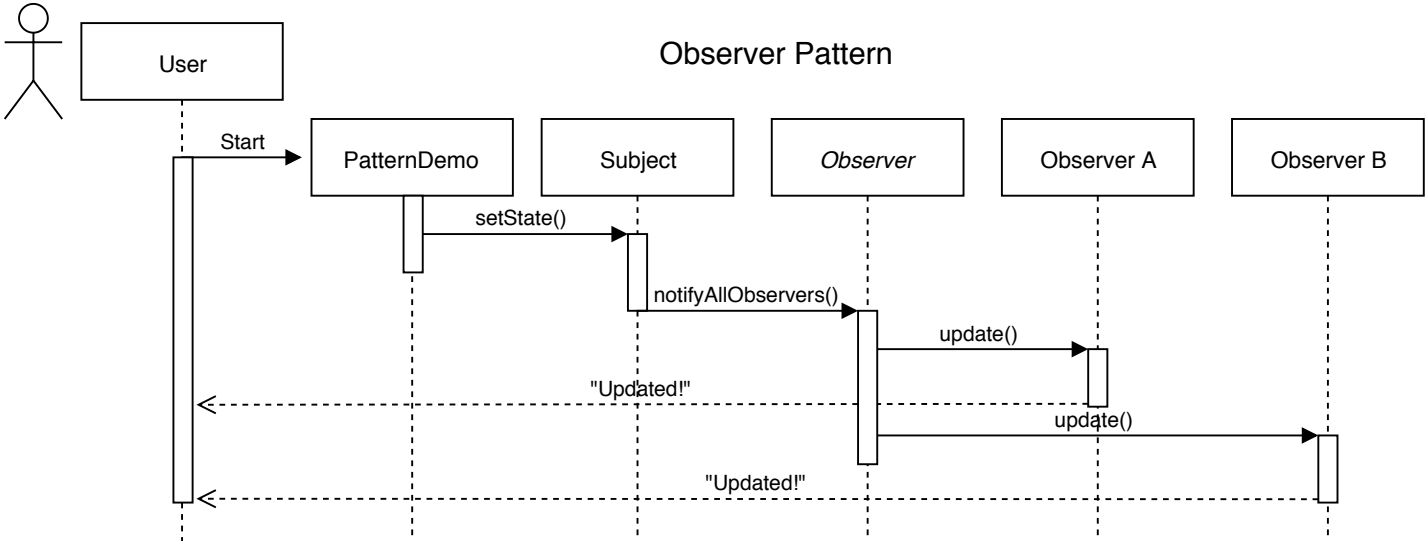+ update(): void

Observer B

- subject: Subject

+ update(): void

References
https://www.tutorialspoint.com/design_pattern/strategy_pattern.htm
https://www.tutorialspoint.com/design_pattern/observer_pattern.htm
https://www.geeksforgeeks.org/observer-pattern-set-1-introduction/

Observer Pattern

User | PatternDemo | Subject | Observer | Observer A | Observer B

Start
setState()
notifyAllObservers()
update()
"Updated!"
update()
"Updated!"

Strategy Pattern

User | PatternDemo | Context | Strategy | Strategy Type (A or B)

Start
strategy
executeStrategy()
doOperation()
return "Operation Done"

a) Velocity * Focus Factor = Estimated Velocity
   Focus Factor = (100% + 100% + 100% + 100% + 80%) / 300% = 160%
   Estimated Velocity = 32 * 1.6 = 51.2
b) Allow the team to estimate the time it will take to complete the highest priority task.
   Take (Estimated Story Points / Estimated Time) for the task to get Estimated Velocity / Capacity (Focus Factor)
   Could also consider that the industry standard is ≈70% and start with that
c) Simply use the first story point guess from each team member and average them. This is probably not better than Planning Poker
   as there may be variables that not all team members have considered leading to inaccurate estimates.
d)
e)

Node

left 1 | 1 right

1 | 1

+ left: Node

+ right: Node

+ key: int

```
class Node{
    //Integer to be stored in each Node
    int key;

    //Creates the left & right child (if exists)
    Node left;
    Node right;

    //Constructor for creating a Node
    public Node(int var) {
        key = var;
        left = right = null;
    }
}
```

Classname

+ parent: Node

+ key: int

0,1 | parent

f)
g)

LinkedList

+ firstItem(item: EmployeeRecords)

+ lastItem(Item: EmployeeRecords)

+Contains(item: EmployeeRecords)

1 | -list

EmployeeRecords

+ name: String

+ snn: int

+ salary: int

1 -list
-firstItem
-lastItem
-nextItem

```
public class LinkedList {
    //Creates an instance of the first/last items in the list
    EmployeeRecords firstItem;
    EmployeeRecords lastItem;
    //Contains a name, Social Security Number, and a Salary
    class EmployeeRecords{
        //Creates an instance of the next item in the list
        EmployeeRecords nextItem;
        String name;
        int ssn;
        int salary;
        //Constructor to create the object to be linked into the list
        EmployeeRecords(String n, int ss, int s){
            name = n;
            ssn = ss;
            salary = s;
            //Resets the nextItem to null
            nextItem = null;
        }
    }
}
```