CSE326: Software Engineering

Assignment 2 - Design Patterns and OOD

Cole Johnson - cole.johnson.@student.nmt.edu

April 9, 2025

Problem-Solving

(1.) You are designing a Time and Talent Survey website for a local church. Suppose the administrator would like to be notified whenever a new survey is completed. What design pattern is suggested? Why did you choose this design pattern? (15 points)

The observer / subject-object pattern would be an ideal design pattern to choose for this problem. I choose this design pattern because it's an effective tool when you have two aspects of a system that are both dependent on the other. Here, the subject could potentially be the survey itself and participants could act as observers.

(2.) You are developing a graphics editor, within which shape can be basic or complex. An example of simple shape is a line object; an example of a complex shape is a rectangle object. The rectangle object consists of four-line objects. Because all shapes have many common operations and can be represented in the hierarchy, you wish to treat all shapes uniformly. What design pattern is suggested? Why did you choose this design pattern? (15 points)

The composite pattern would be an ideal design pattern to choose for this problem. This design pattern is good when you want to represent part-whole hierarchies–components that are composed of simpler items.

Stack Implementation

Here's the implementation of the stack:

```
public class StackImpl1 implements Stack {
    private int[] stack;
    private int pointer = 0;
    private int capacity;
    // create a new stack with an initial capacity
    public StackImpl1(int capacity) {
        this.capacity = capacity;
        stack = new int[capacity];
    }
    @Override
    public void push(int i) {
        if(isFull()) {
            grow(5); // grow stack by 5 if it's already full
        }
        stack[pointer] = i;
        pointer++;
    }

    @Override
    public int pop() {
        if (isEmpty()) {
            throw new RuntimeException("[ERROR]: Stack is empty!
Cannot pop an element!");
        }
        int val = stack[pointer-1];
        // rewind pointer
        pointer--;
        return val;
    }

    @Override
    public int top() {
        return stack[pointer-1];
    }

    @Override
    public boolean isEmpty() {
        return pointer == 0;
    }

    public boolean isFull() {
        return pointer == capacity;
```

```
        }

        void print() {
            StringBuilder s = new StringBuilder();
            s.append("{ ");
            int i = 0;
            for(; i < pointer - 1; i++) {
                int val = stack[i];
                // collect values except last, append comma and space
                s.append(val).append(", ");
            }
            // get last value, append with ending curly brace instead
            int val = stack[i];
            s.append(val).append(" }");
            System.out.println(s);
        }
        // grow the stack by a given amount
        public void grow(int amount) {
            int[] stack = new int[capacity + amount];
            // copy old elements to new larger stack
            System.arraycopy(this.stack, 0,
                             stack, 0, this.stack.length);
            this.stack = stack;
            this.capacity += amount;
        }
    }
```

The stack implementation just has two parts: an array and a pointer to the next unoccupied address inside the array. To push, just use the pointer to assign a new element and increment the pointer. To pop, get the value that is pointed to, decrement the pointer, and return the value. We need some additional helper functions not described in the interface, like `grow()` and `isFull()` to help us grow the stack when needed.

Decorator Pattern