1705046
Iftekhar Hakim

(1.)

@Test
void test_function {

int a[];

a = new int [10];

a = {1, 2, 3, 4, 5, 6, 7, 8, 9, 10};

assertEquals (5.5, given_function(a));

a = new int [2];

a = {1, 2};

assertEquals (1.5, given_function(a));

a = new int [1];

a = {0};

assertEquals (0, given_function(a));

a = new int [3];

a = {-1, -2, -3};

assertEquals (-2, given_function(a));

a = null;

assertEquals (0, given_function(a));

}

(2).

```
void testing(int array[], int what) {
    int pos = given_function(array, what);
    int pos2 = -1;
    for(int i=0; i<array.size(); i++) {
        if (array[i] = what) {
            pos2 = i;
            break;
        }
    }
    assertEquals(pos, pos2);
}
void create_cases() {
    for(int cas=1; cas=10; cas++) {
        int array[];
        int n = random(1,50);
        array = new int[n];
        for(int i=0; i<n; i++) {
            array[i] = random(0,100);
        }
        sort(array);
        testing();
    }
}
```

3.

```
class Report implement Generator{
    @override
public String generate Report (String ClientCode){
    ....
    }
}

class adapter implements Generator {
@override
public String generate Report (String NID){
        String ClientCode = Map.get_
                            code(NID);

        Report report = new Report();
        return report.generateReport(clientcode);
    }
}

interface Generator {

    String generate Report(String s);

}
```

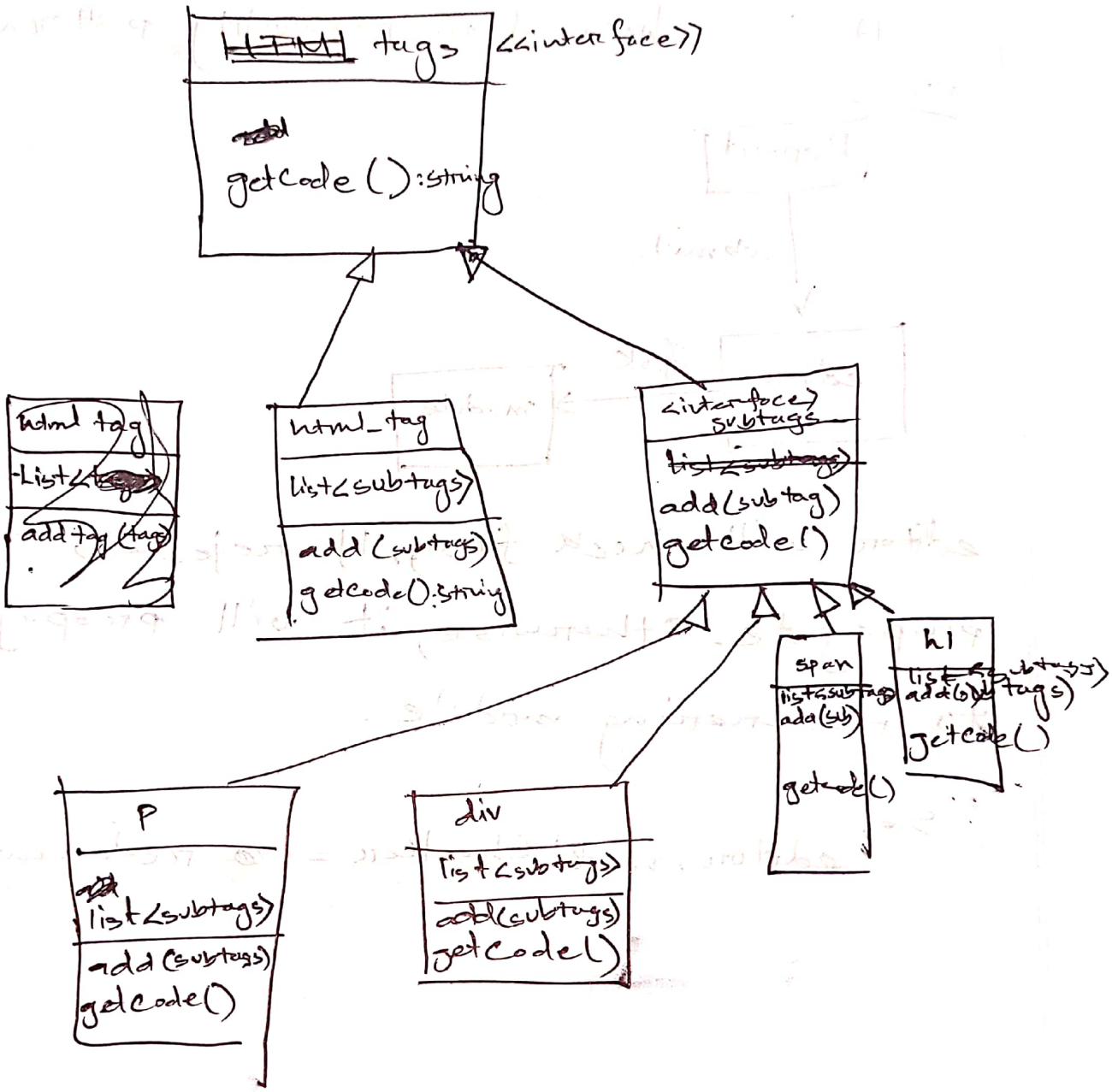
(9) Mediator pattern is suitable.

```
class AirTrafficControl {
    bool runway_empty = true;

    bool receive_request(String sender) {
        if (runway_empty) {
            print(sender + " is permitted");
            runway_empty = false;
            return true;
        }
        return false;
    }

    void plane_left(String sender) {
        runway_empty = true;
        print(sender + " has left");
    }
```

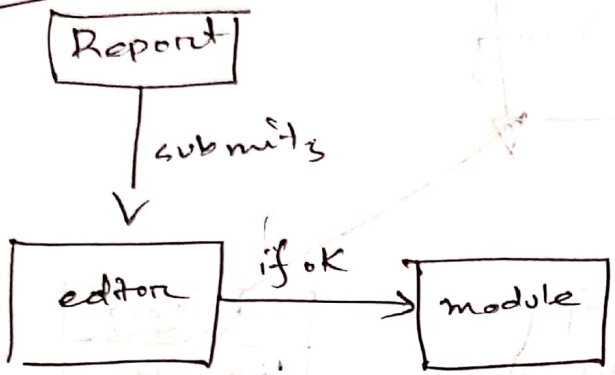
5. Composite pattern



6.

It is chain of responsibility pattern.

Flow:



editor will check first, if rejected, it will not propagate. Otherwise, it will propagate it to rendering module.

So, editor.nextChecker = rendering module.

7. Strategy pattern.

```
interface function {
    void execute (String input);
}

class normalization implements function {
    void execute (String input) {
    }
}

class stemming implement function {
    void execute (String input) {
    }
}

class ProcessingToolkit {
    function f;
    void execute() {
        f.execute(); } }
```