ESOF 322, Homework 3

September 26, 2019

YuehChen Tsou and John Singleton

1 Exercises 1 (10pts)

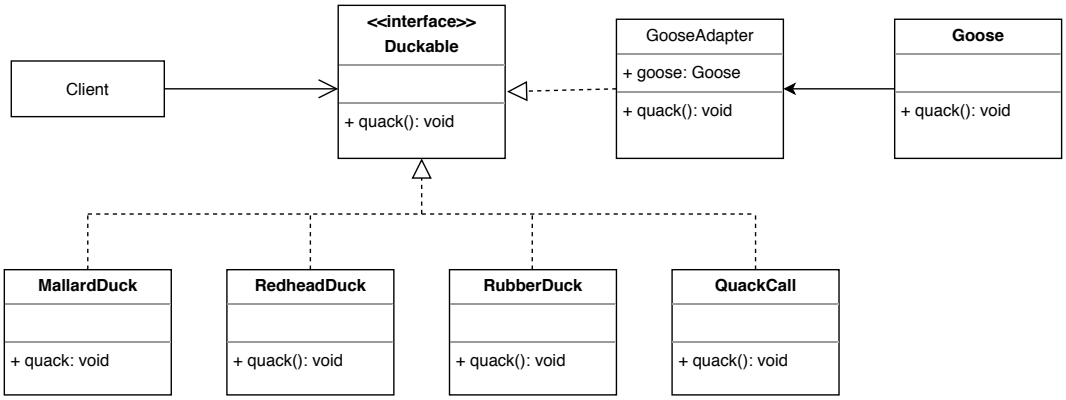
We have covered three design patterns in class(Strategy, Adapter, Observer). Pick two design patterns (any two that you want–you can even do research on your own and find another one you like!) and couple them. Two design patterns are coupled when at least one class is a participant in both design patterns.

Part a Draw a UML class diagram that clearly shows the coupling. Please identify which class(es) participates in multiple patterns.

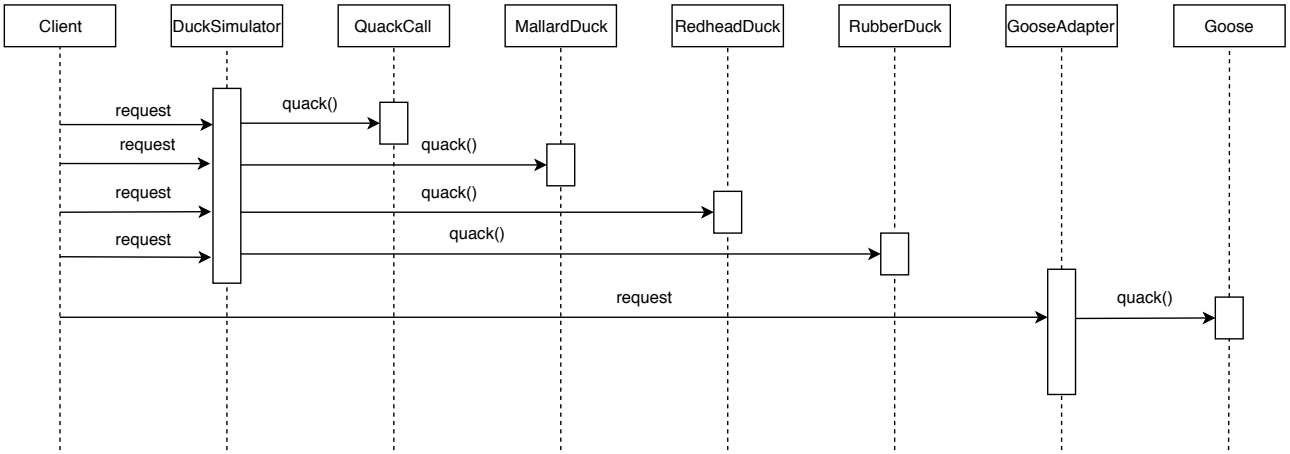
Part b Draw a UML Sequence diagram where you demonstrate the behavior of an instance of the coupled class from the perspective of one pattern, then from the perspective of the other pattern.

Solution :

Exercise 1: Class Diagram



Sequence Diagram



2 Exercises 2 (14pts)

Part a Assume during your team's last sprint, that they completed 32 story points using a 3-person team working in sprints of 3 weeks for a total of 45-man days. Calculate your team's estimated velocity for the next sprint if we still have 3-week sprints, but you now added 2 engineers to the team, and one of them can only work80% of the time

Answer :

First 3 weeks	available days	completed	Focus Factor
Engineer 1	15		
Engineer 2	15		
Engineer 3	15		
Total	45	32	32/45=71%

Second 3 weeks	available days	completed	Focus Factor
Engineer 1	15		
Engineer 2	15		
Engineer 3	15		
Engineer 4	15		
Engineer 5	12		
Total	72	$(32 \cdot 72)/45 = 51.2$	71%

For the first sprint, the actual velocity was 32 story points and there were 45 available man-days. The focus factor is given by the actual velocity divided my the number of available man-days, which in this case is 32/45. For the second sprint, we have (4*1 + 1*.8)*15 = 72 available man-days. We'll assume that the new team will have the same focus factor as the old team did during the first sprint (32/45). Then the estimated velocity is given by 72 * 32/45 = 51.2 story points.

Part b How would you estimate a focus factor for a brand-new team?

Answer :

It is common to make a conservative estimate of 70% for brand new team's focus factor.

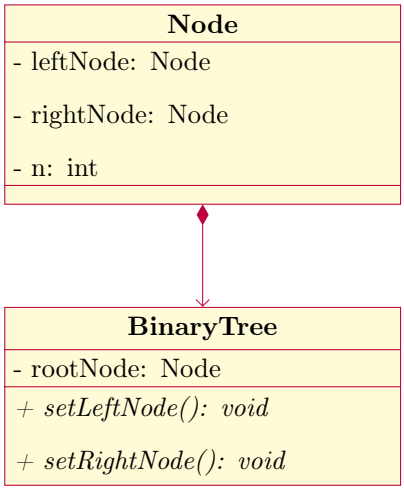
Part c We looked at using poker using semi-Fibonacci sequences to estimate story points. Think of another way to estimate story points and explain it. Is it better or worse than poker?

Answer :

We could use cards with natural numbers that increase by a constant factor of 2, i.e. 1,2,4,8,16,.... However, evidently these options do not provide as good of an estimate of how long it will take to complete a standard set of software engineering tasks (or how many resources they require).

Part d Draw a UML class diagram of a binary tree. Each node contains an integer.

Answer :



Part e Provide the corresponding object-oriented code that implements your binary tree designfrom part d.

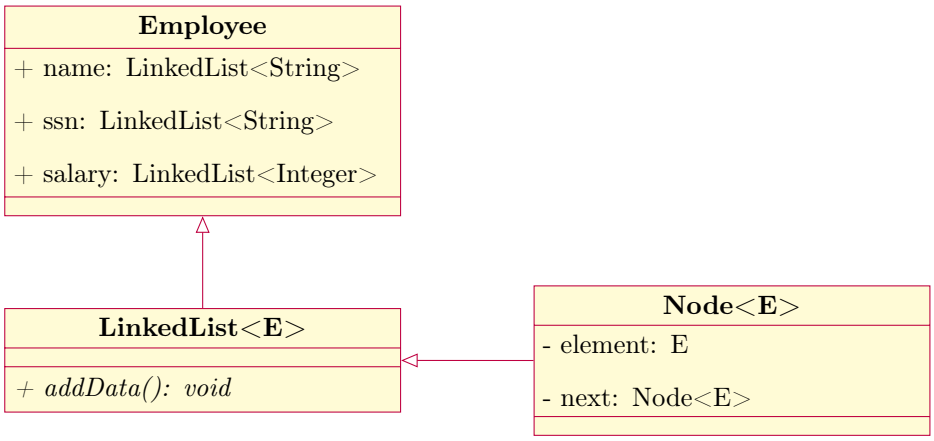
Answer :

```
public class Node {
    private Node leftNode;
    private Node rightNode;
    private int n;
    // setters and getters for each of the above
}


public class BinaryTree {
    private Node rootNode;
    public BinaryTree( Node rootNode) {
        this.rootNode = rootNode;
    }
        public void setLeftNode (Node node, Node left) {
            node.setLeftNode(left);
        }
        public void setRightNode (Node node, Node right) {
            node.setRightNode(right);
        }
}
```

Part f Draw a UML class diagram of a linked list that contains Employee records as data. An Employee record has a name, a social security number, and a salary.

Answer :



Part g Provide the corresponding object-oriented code that implements your linked listfrom part f.

Answer :

```
public class Employee {
    LinkedList<String> name = new LinkedList<>();
    LinkedList<String> ssn = LinkedList<>();
    LinkedList<Integer> salary = LinkedList<>();
}

public class LinkedList<E> {
    private static class Node<E> { // Node class
        private E element;
        private Node<E> next;
        public Node<E>(E e,Node n) { // constructor
            this.element=e;
            this.next=n;
        }
    }
    public void addData( E e) {}
}
```

```
public class Employee {
    public static void main(String[] args) {

        Employee employee = new Employee();
        employee.worker();
    }
    public void worker() {
        LinkedList<String> name = new LinkedList<>();
        LinkedList<String> ssn = new LinkedList<>();
        LinkedList<Integer> salary = new LinkedList<>();

        name.addData("Peter");
        ssn.addData("SN1234567");
        salary.addData(75000);
    }
}
public class LinkedList<E> {

    private static class Node<E> {
        // fields
        private E element;// reference to the element stored at this node
        private Node<E> next; // reference to the subsequent node in the list
        // constructor
        public Node(E e, Node<E> n) {
            this.element = e;
            this.next = n;
        }
    }
    public LinkedList() {}

    public void addData(E e) {
        System.out.println(e);
    }
}
// -----output-----
Peter
SN1234567
75000
```