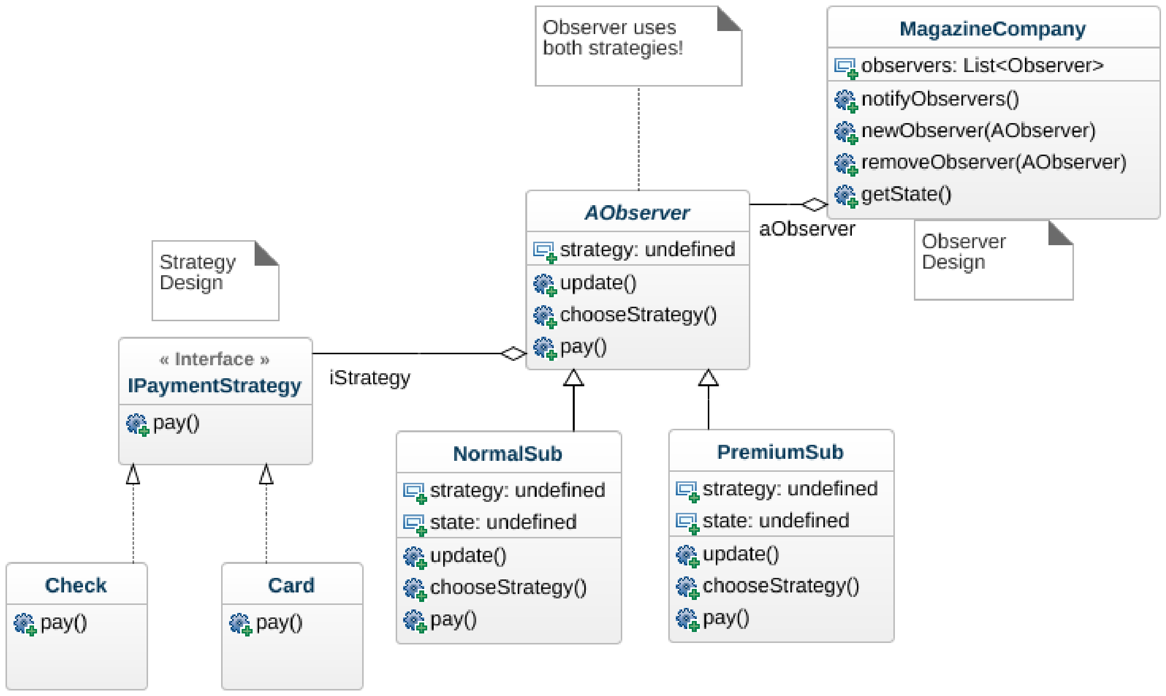
Homework 3

Software Engineering

Anna Jinneman and William Roberts

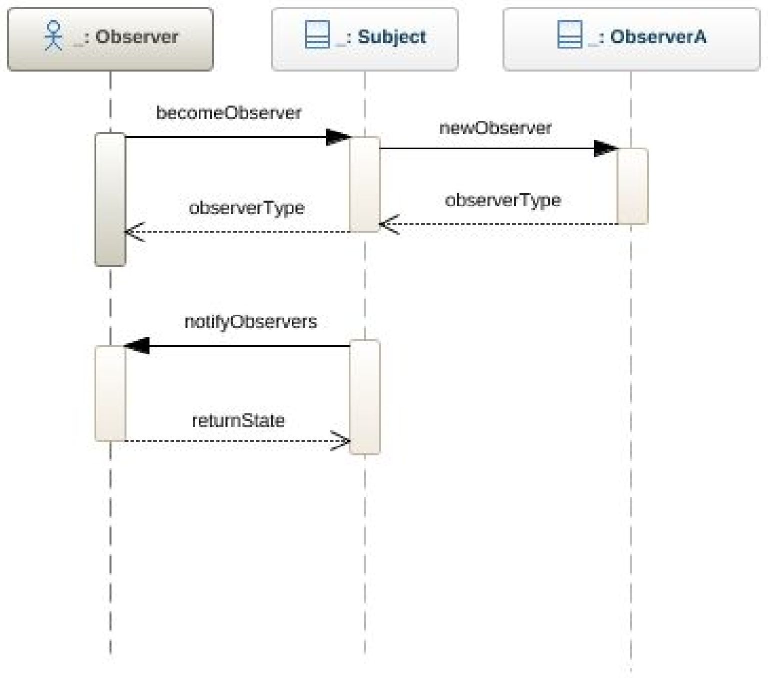
Problem 1

a) Coupling of Observer and Strategy design patterns:

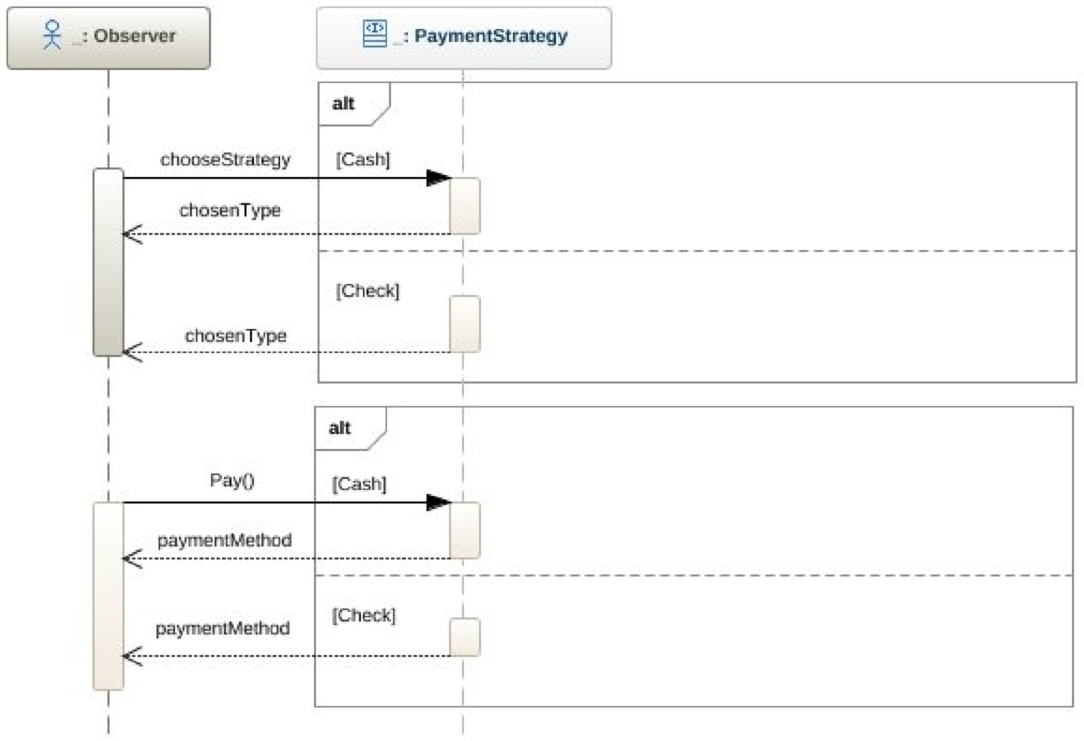


Problem 1

b) Observer Sequence Diagram:



b) Strategy Sequence Diagram:

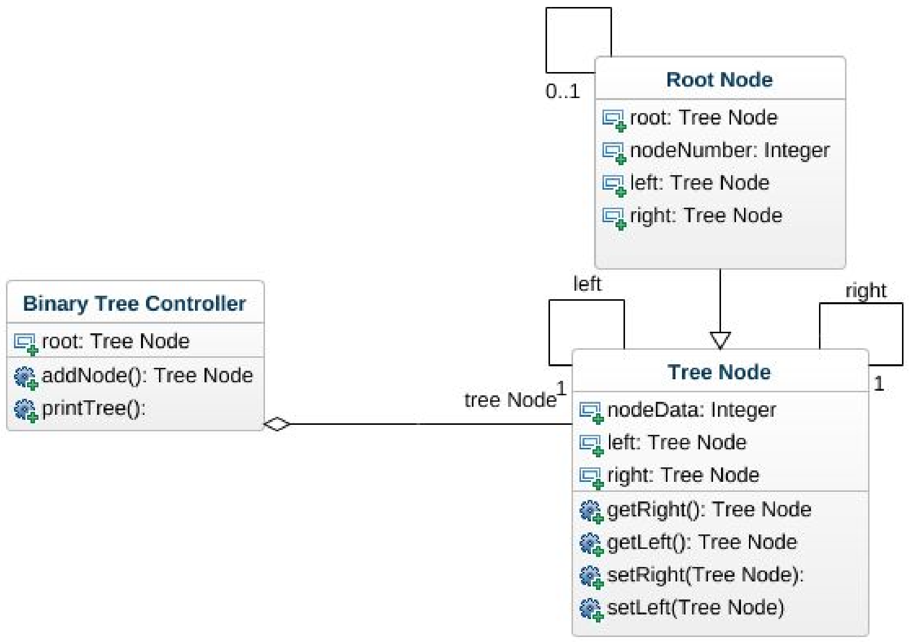


Problem 2

a. After Adding the two new employees that puts at 72 available Man Days. In the last sprint, our team (with just the original 3 engineers) completed 32 Story Points in 45 man days, so they had a Focus Factor of 32/45 = .71.
    i. Focus Factor = .71
    ii. Available Man Days = 72
    iii. The estimated velocity for the next sprint = Available Man Days * Focus Factor so: 72 * .71 = 51
    iv. So the estimated velocity for this team is 51 Story Points.
b. You do not estimate a Focus Factor for a brand new team, the general rule of thumb would be to just use the value of .7 for the very first sprint, then afterwards you could calculate the team's focus factor from a finished sprint. This is because .7 is a good benchmark and a decent estimation. Once a team drops below .7 Focus Factor you want to start watching for unnecessary behavior.
c. You, as the team manager, ask each engineer individually how long they think a certain task is going to take. You then compile all the answers and select the largest informed answer out of the group. This is how many story points that task has.
    i. This method ensures that you don't ever underestimate the story points for a certain task, while also ensuring that every team member has their voice heard in the story point decision. Of course it is rather time consuming, and you would have to be careful to make sure each person gets asked individually or their number could become biased.
    ii. For further enhancement of this process you could then come to the group and explain the decision to see if it is a common consensus among members. Perhaps the person who had the maximum Story Point estimation knows something about the task that everyone else did not. Make sure to disregard people who knew nothing about the task <u>and</u> answered (?) or some large number!
    iii. This method is probably not as strong as the Poker method, mainly due to the fact that it takes the MAX instead of the average of the story point values. This makes it so that there is a large chance you are overestimating the story point value and thus planning to finish less tasks. In the other sense however you would be more likely to not underestimate story point values and potentially miss important tasks! Overall the strength is probably on par with Poker, but of course I haven't tested it on a real team like Poker has been!

Problem 2

d) Binary Tree UML:

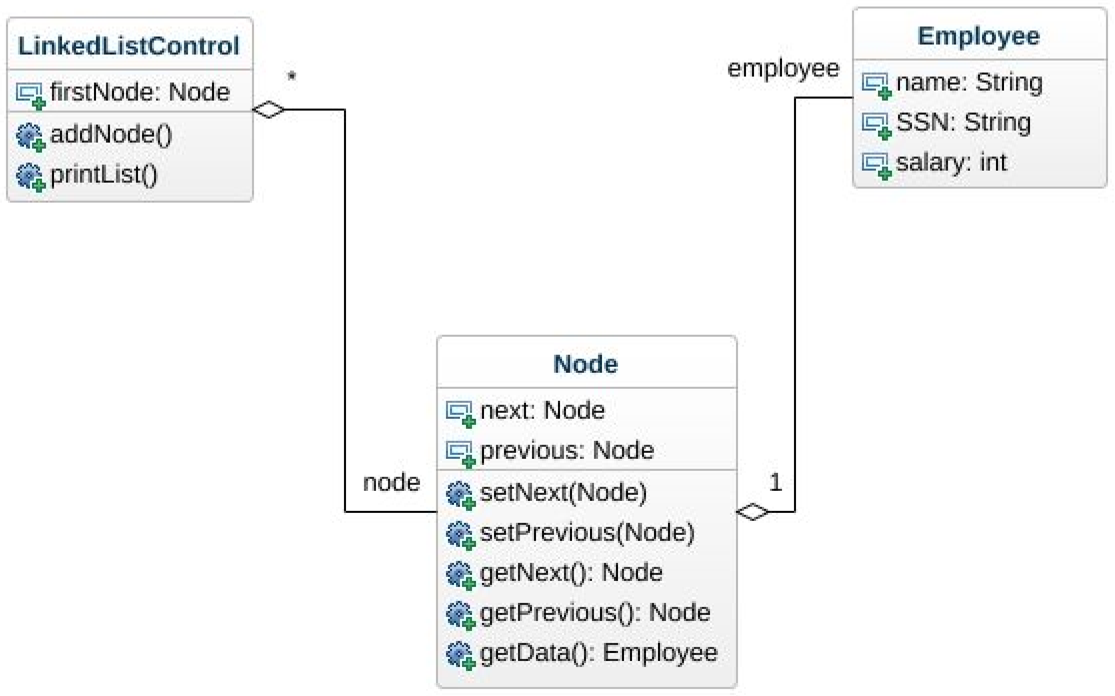


e) Binary Tree Code:

```
package binarytreeesof;

/**
 *
 * @authors Anna Jinneman and William Roberts
 */
public class BinaryTreeESOF {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        // TODO code application logic here
    }

    //-----------------------------------------------------
    //Binary Tree Code Below!

    public class treeNode{ //This class contains the code for individual nodes. Standard stuff.
        int nodeNumber;
        treeNode left;
        treeNode right;

        treeNode(int nodeNumber){
            this.nodeNumber = nodeNumber;
            left = null;
            right = null;
        }

        public treeNode getRight()
        {
            return this.right;
        }

        public treeNode getLeft()
        {
            return this.left;
        }

        public void setRight(treeNode newRight){this.right = newRight;}
        public void setLeft(treeNode newLeft){this.left = newLeft;}

    }
```

```
  public class binaryTree{ //Then this code actually links the nodes together and creates the tree. It
adds new nodes and prints the tree out.
    treeNode root;

    public treeNode addNode(treeNode parent, int number) //Adds a node with the given number
(always give root as first param)
    {
      if (number < root.nodeNumber) {
        root.left = addNode(root.left, number);
      }
      else if(number > root.nodeNumber)
      {
        root.right = addNode(root.right, number);
      }
      else
        return root;

      return root;

    }

    public void inOrder(treeNode data){ //This is printTree in order!
      if(root != null){
        if(data != null){
          inOrder(data.getLeft());
          System.out.println(data.nodeNumber);
          inOrder(data.getRight());
        }
      }
    }

  }
```

f)
Linked List UML:



g) Linked List Code:

```
/**
 *
 * @authors Anna Jinneman and William Roberts
 */
  //This code is for the Linked List!

  public class Employee{ //The data that gets stored in each Node.
     public String name;
     public String SSN;
     public int salary;

  }

  public class Node //This class holds the data and references to each of its neighbor nodes in the list.
  {
     private Node next;
     private Node previous;
     private Employee data;

     //Pretty Standard stuff below...
     Node(Employee newData)
     {
        this.data = newData;
        this.next = null;
        this.previous = null;
     }

     public void setNext(Node ne)
     {
        this.next = ne;

     }

     public void setPrevious(Node ne)
     {
        this.previous = ne;
     }

     public Node getNext()
     {return this.next;}

     public Node getPrevious(){return this.previous;}

     public Employee getData(){return this.data;}

  }
```

```
    public class LinkedListControl //This class controls the linked list by adding new nodes and
printing the list out.
    {
        Node firstNode;

        public void addNode(Node newNode)
        {
            Node before = firstNode;
            Node after = firstNode.getNext();

            before.setNext(newNode);
            after.setPrevious(newNode);
            newNode.setNext(after);
            newNode.setPrevious(before);
        }

        public void printList()
        {
            Node temp = firstNode.getNext();
            while (temp != firstNode)
            {
                System.out.println(temp.getData());
                temp = temp.getNext();
            }
        }
    }

}
```