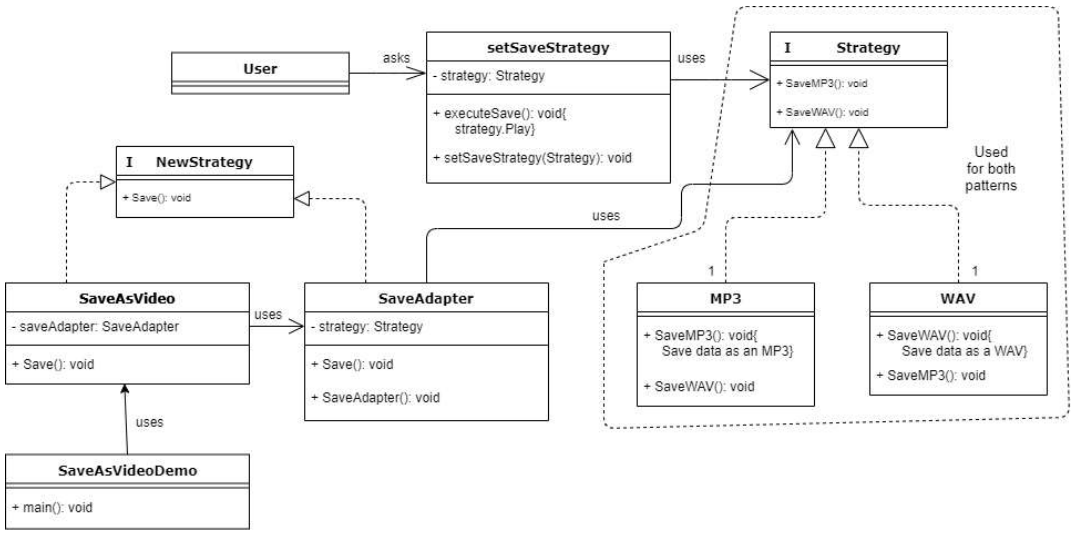
John Lambrecht and Skylar Smoker

ESOF322

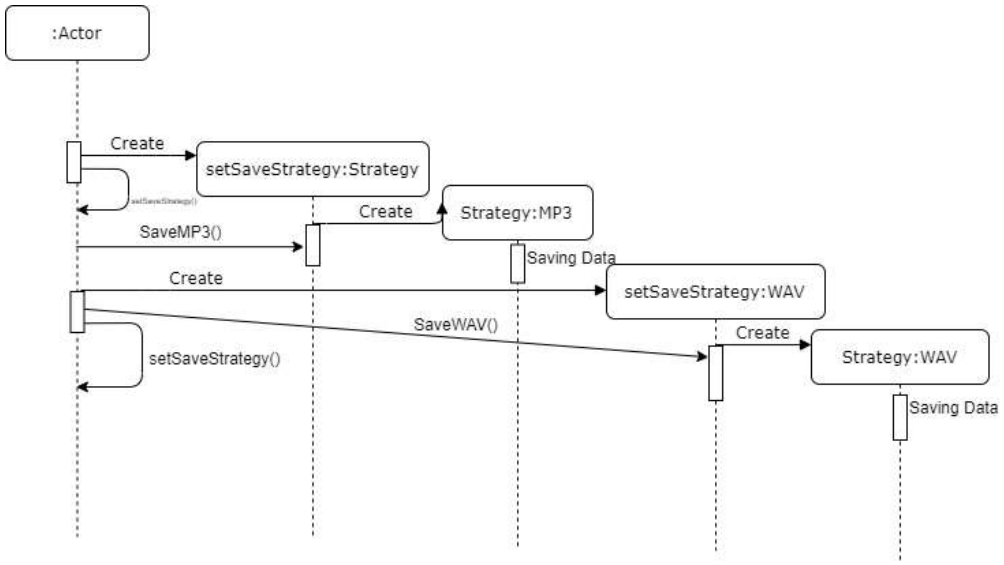
HW3

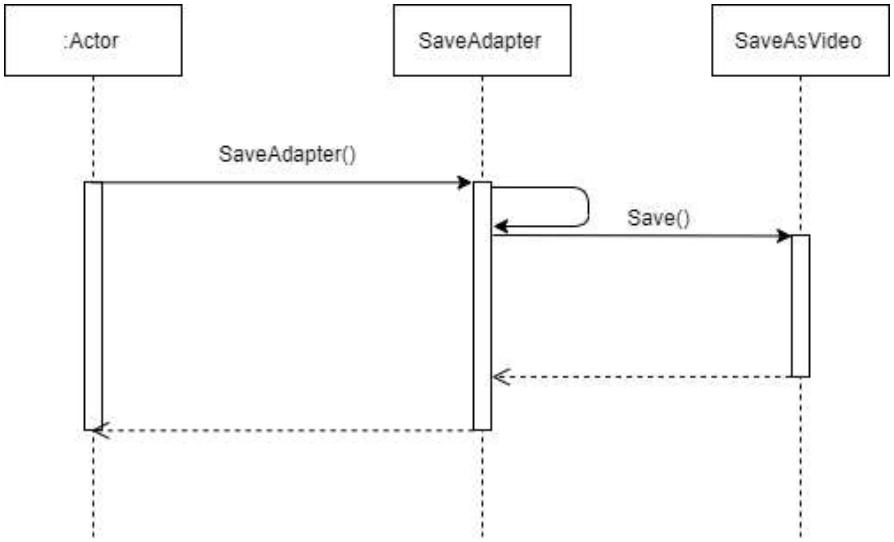
Exercise 1)

a)



b)





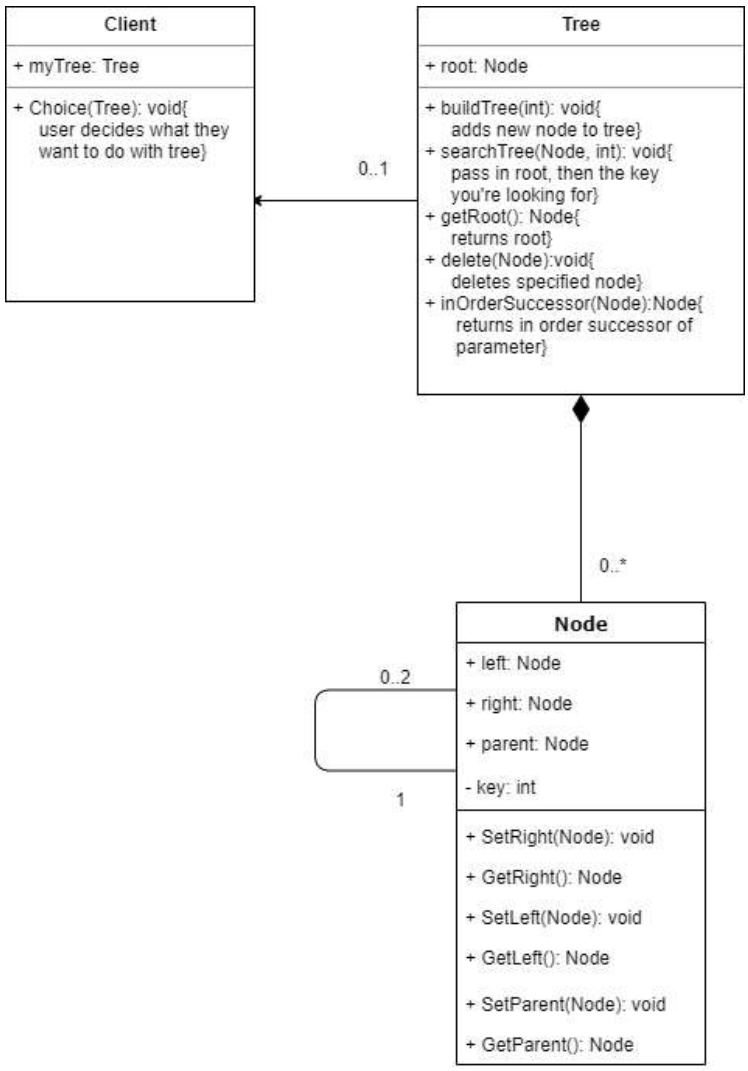
Exercise 2)

a) 32/45=71% 45+15+(15*.8)=72 72*.71=51.2 story points is the estimated velocity for the next sprint

b) Take the average of the focus factors from their previous teams

c) You could do something similar to planning poker, still using the semi-Fibonacci sequence. You would flip the cards, but instead of discussing why you chose what you chose, you just take the average of what everyone put, and that's how many story points it would get. I think this is worse than planning poker because it doesn't allow for discussion of why a task would get that many story points.

d)



e) package program;

```
import java.util.Scanner;
import java.util.Collections;

public class Program {
   public static void main(String[] args){
      tree mytree;
```

```
mytree = new tree();
int choice = 0;
while (choice != -1){
   System.out.println("What would you like to do?");
   System.out.println("Press 1 to insert a node");
   System.out.println("Press 2 to delete a node");
   System.out.println("Press 3 to search for a node");
   System.out.println("Press -1 to exit");
   Scanner reader = new Scanner(System.in);
   choice = reader.nextInt();
   switch(choice){
      case 1:
         System.out.println("Please enter the number to enter");
         Scanner in = new Scanner(System.in);
         int input = in.nextInt();
         mytree.buildTree(input);
         break;
      case 2:
         System.out.println("Enter the node you want to delete");
         Scanner del = new Scanner(System.in);
         Node delroot = mytree.getRoot();
         Node delete = mytree.searchTree(delroot, del.nextInt());
         mytree.delete(delete);
         break;
      case 3:
         System.out.println("Please enter the number to search");
         Scanner searchit = new Scanner(System.in);
         int searchint = searchit.nextInt();
         Node root = mytree.getRoot();
         Node search = mytree.searchTree(root,searchint);
         if (search == null){
            System.out.println("The node is not in the tree");
         }else{
            System.out.println("The node is in the tree");
         }
         break;
      case -1:
         System.out.println("exit");
         break;
      default:
         System.out.println("Invalid choice");
         choice = 0;
         break;
   }
}
```

```
    }
}

class Node {
    Node parent;
    Node leftChild;
    Node rightChild;
    private int key;
    public Node(int input){
        key = input;
    }
    public void setRightChild (Node temp){
        rightChild = temp;
    }
    public void setLeftChild (Node temp){
        leftChild = temp;
    }
    public void setParent (Node temp){
        parent = temp;
    }
    public void setKey(int temp){
        key = temp;
    }
    public Node getParent (){
        return parent;
    }
    public Node getLeftChild(){
        return leftChild;
    }
    public Node getRightChild(){
        return rightChild;
    }
    public int getKey(){
        return key;
    }
}

class tree {
    Node root;
    tree(){
        parent = null;
    }

    void buildTree(int key){
        Node n = new Node(key);
```

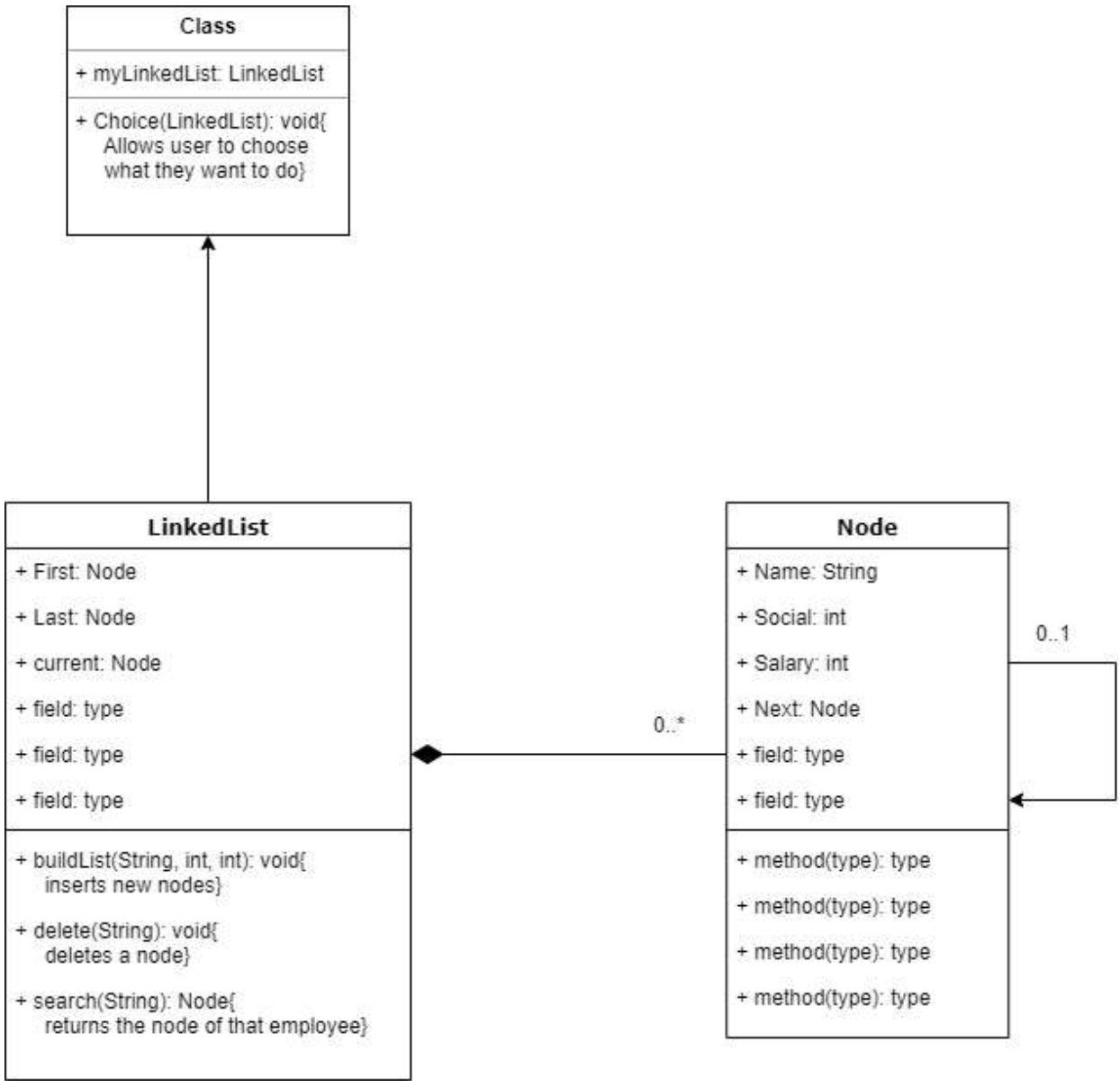
```
    if (root == null ){
        root = n;
        return;
    }
    while (root.getParent()!=null){
        root=root.getParent();
    }
    while(n.getParent()==null){
        if (root.getKey()>n.getKey()&&root.getLeftChild()==null){
            root.setLeftChild(n);
            n.setParent(root);
        }else if (root.getKey()>n.getKey()&&root.getLeftChild()!=null){
            root = root.getLeftChild();
        }else if(root.getKey()<n.getKey()&&root.getRightChild()==null){
            root.setRightChild(n);
            n.setParent(root);
        }else if (root.getKey()<n.getKey()&&root.getRightChild()!=null){
            root = root.getRightChild();
        }
    }

}
Node searchTree(Node current, int value){
    if (current.getKey()==value){
        return current;
    }
    else if (current.getKey()>value){
        return searchTree(current.getLeftChild(), value);
    }
    else if (current.getKey()<value){
        return searchTree(current.getRightChild(), value);
    }
    else {
        return null;
    }
}
Node getRoot(){
    while (root.getParent()!=null){
        root = root.getParent();
    }
    return root;
}
void delete(Node delNode){
    Node delRoot = getRoot();
    int rootkey = delRoot.getKey();
```

```
        int children = delNode.howManyChilren();
        if (children == 2){//If deleted node has 2 children
            Node Successor = inOrderSuccessor(delNode);//swaps keys of del and Successor,
then deletes successor
            int Temp = Successor.getKey();
            delNode.setKey(Successor.getKey());
            Successor.setKey(Temp);
            delete(Successor);
        }else if (children==1){//If deleted node has 1 child
            if(delNode.getLeftChild()!=null){//If child of deleted node is left child
                delNode.getLeftChild().setParent(delNode.getParent());
                if (delNode.getParent().getLeftChild()==delNode){//If deleted node is a left
child
                    delNode.getParent().setLeftChild(delNode.getLeftChild());
                }else if(delNode.getParent().getRightChild()==delNode){//If deleted node is a
left child
                    delNode.getParent().setRightChild(delNode.getLeftChild());
                }
            }else if (delNode.getRightChild()!= null){//If child of deleted node is right child
                delNode.getRightChild().setParent(delNode.getParent());
                if (delNode.getParent().getLeftChild()==delNode){//If deleted node is a left
child
                    delNode.getParent().setLeftChild(delNode.getRightChild());
                }else if(delNode.getParent().getRightChild()==delNode){//If deleted node is a
right child
                    delNode.getParent().setRightChild(delNode.getRightChild());
                }
            }
        }else{//If node has no children
            if(delNode.getParent().getLeftChild()==delNode){//if node is a left child
                delNode.getParent().setLeftChild(null);
            }else if(delNode.getParent().getRightChild()==delNode){//if node is a right child
                delNode.getParent().setRightChild(null);
            }
            delNode.setParent(null);
        }
    }
    Node inOrderSuccessor(Node node) {
        Node succ;
        if (node.getRightChild() != null) {
            succ = node.getRightChild();
            while (succ.getLeftChild() != null) {
                succ = succ.getLeftChild();
            }
        } else {
```

```
            succ = node;
        }
        return succ;
    }
}
```

f)



g)

```
package linkedlist;
import java.util.Scanner;
public class LinkedList{
   Node current;
   Node First;
   Node Last;
   public LinkedList(){
      First = null;
   }
   void buildList(String name, int social, int salary){
      Node n = new Node(name, social, salary);
      if (First==null){
         First = n;
      }
      current = n;
      while (current.getNext()!=null){
         current = current.getNext();
      }
      current.setNext(n);
      Last = n;
   }
   void delete (String employee){
      Node n = First;
      if (First.getName()==employee){
         First = n.getNext();
      }
      else{
         while (n.getNext().getName()!=employee){
            n=n.getNext();
         }
      }
      n.setNext(n.getNext().getNext());
   }
   Node search(String employee){
      Node n = First;
      while (n.getName()!=employee){
         n = n.getNext();
      }
      return n;
   }
```

```
}
class Node{
    Node next;
    private String Name;
    private int Social;
    private int Salary;
    public Node(String name, int SSN, int Sal){
        Name = name;
        Social = SSN;
        Salary = Sal;
    }
    public void setName(String name){
        Name = name;
    }
    public void setSocial(int SSN){
        Social = SSN;
    }
    public void setSalary(int Sal){
        Salary = Sal;
    }
    public void setNext(Node temp){
        next = temp;
    }
    public Node getNext(){
        return next;
    }
    public String getName(){
        return Name;
    }
    public int getSocial(){
        return Social;
    }
    public int getSalary(){
        return Salary;
    }
}
class Client{
    public static void main(String[] args){
        LinkedList myLinkedList;
        myLinkedList = new LinkedList();
        int choice = 0;
        while (choice != -1){
            System.out.println("What would you like to do?");
            System.out.println("Press 1 to insert an employee");
            System.out.println("Press 2 to delete an employee");
```

```
            System.out.println("Press 3 to search for an employee");
            System.out.println("Press -1 to exit");
            Scanner reader = new Scanner(System.in);
            choice = reader.nextInt();
            switch (choice){
               case 1:
                  System.out.println("Please enter the employee name");
                  Scanner emp = new Scanner(System.in);
                  String name = emp.next();
                  System.out.println("Please enter the employee SSN without dashes");
                  Scanner SSN = new Scanner(System.in);
                  int social = SSN.nextInt();
                  System.out.println("Please enter the employee salary");
                  Scanner sal = new Scanner(System.in);
                  int salary = sal.nextInt();
                  myLinkedList.buildList(name, social, salary);
                  break;
               case 2:
                  System.out.println("Please enter the name of the employee to be deleted");
                  Scanner del = new Scanner(System.in);
                  String delete = del.next();
                  myLinkedList.delete(delete);
                  break;
               case 3:
                  System.out.println("Enter the name of the employee");
                  Scanner employee = new Scanner(System.in);
                  String empl = employee.next();
                  Node search = myLinkedList.search(empl);
                  System.out.println(search.getName()+ search.getSocial()+
search.getSalary());
                  break;
               case -1:
                  System.out.println("exit");
                  break;
               default:
                  System.out.println("Invalid choice");
                  choice = 0;
                  break;
            }
         }
      }
}
```