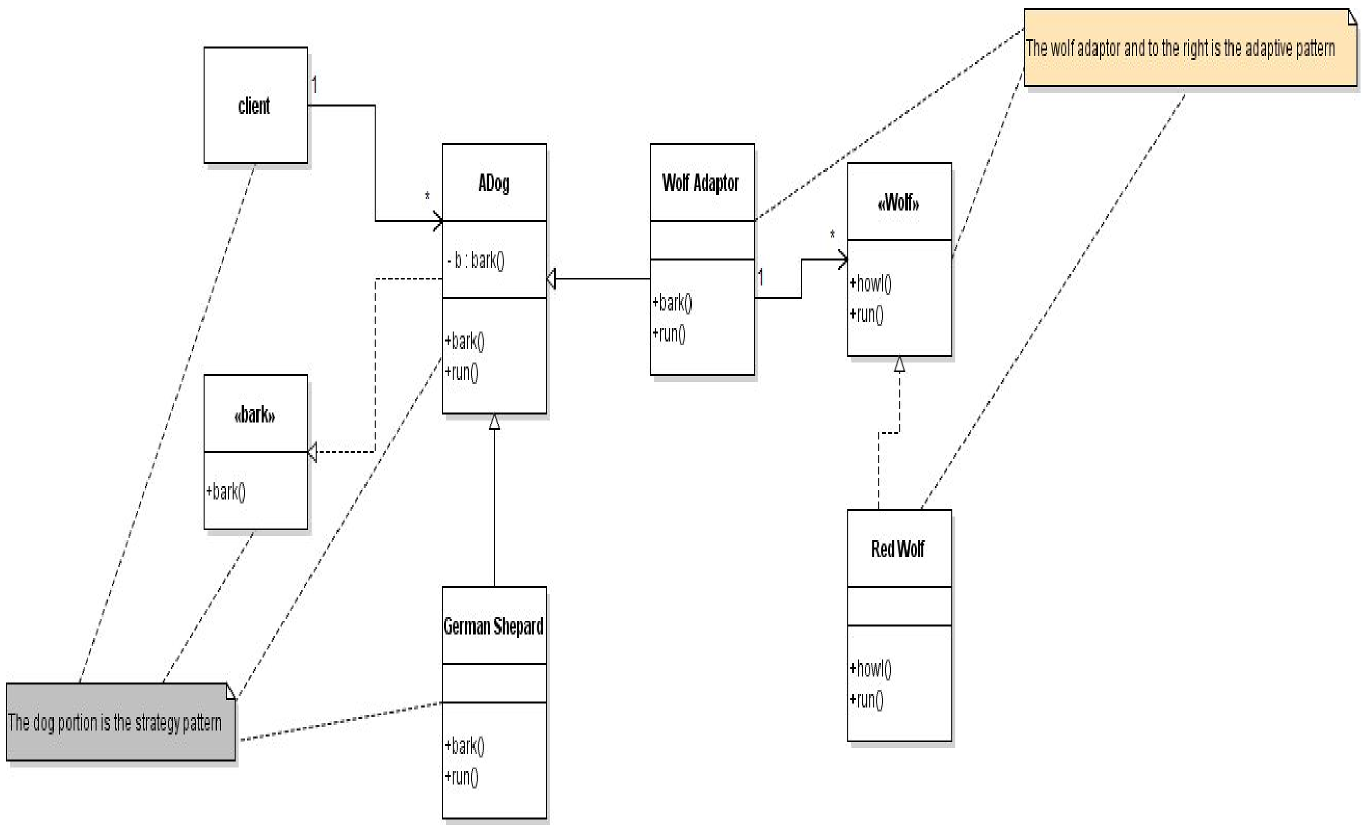
Names: Gabrielle Strong & Anthony Nardiello



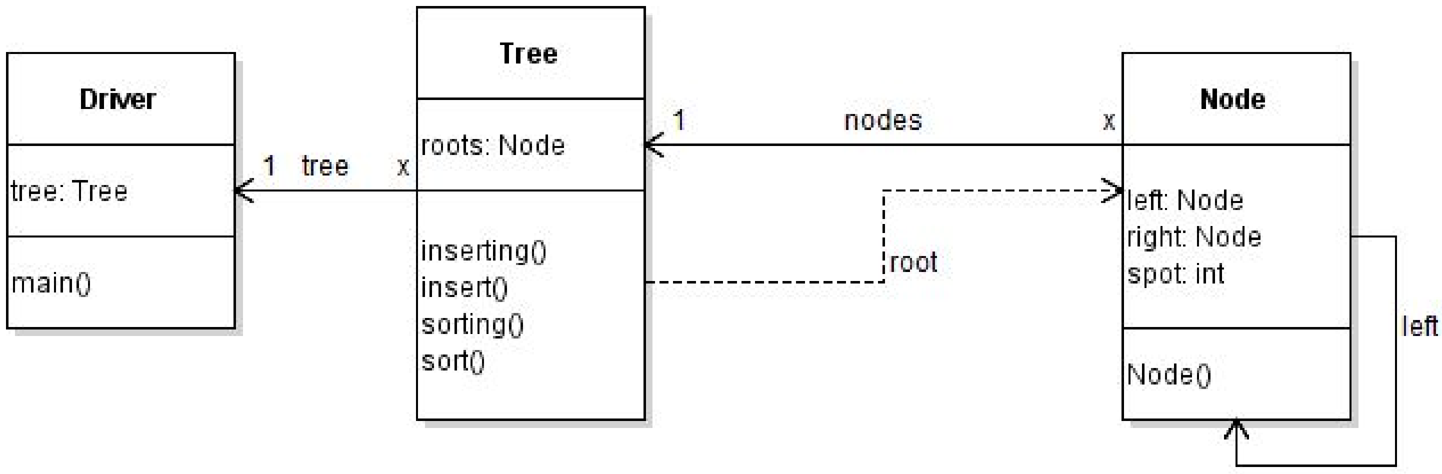
Exercise 2

a. Agile Velocity Estimation
    i. Previous team's man-days: 45
        1. Under assumption that old team is still here all 45 days
    ii. One new team member: 15 man-days
        1. Under the assumption that one new teammate is here all 15 days.
    iii. Second new team member: 12 man-days
    iv. Total current team man-days: 72
    v. Team's Focus Factor: (Actual Velocity)/(Available man-days)=(32)/45=71%
    vi. Team's Estimated Velocity: (Available man-days)*(Focus Factor)=(72)*(.71)=51 Story Points

b. Focus Factor for a brand new team
    i. For a brand new Development team, you won't have an actual velocity to work with in order to calculate the Focus Factor. Thus, it is up to the Team Manager to estimate how effective they think the team will be. Typically, most times Managers will choose a value between 70-80%, but this value can be changed based upon general experience levels of the group.

c. Chocolate bar
    i. Each person is given a chocolate bar. Each piece is a point towards a problem within the discussion. The more pieces given to the problem means the more amount of time spent on that particular project. Worse idea as not enough pieces to maybe emphasize the importance of a project. The other being engineers might eat it.

d. Tree Diagram on next page.

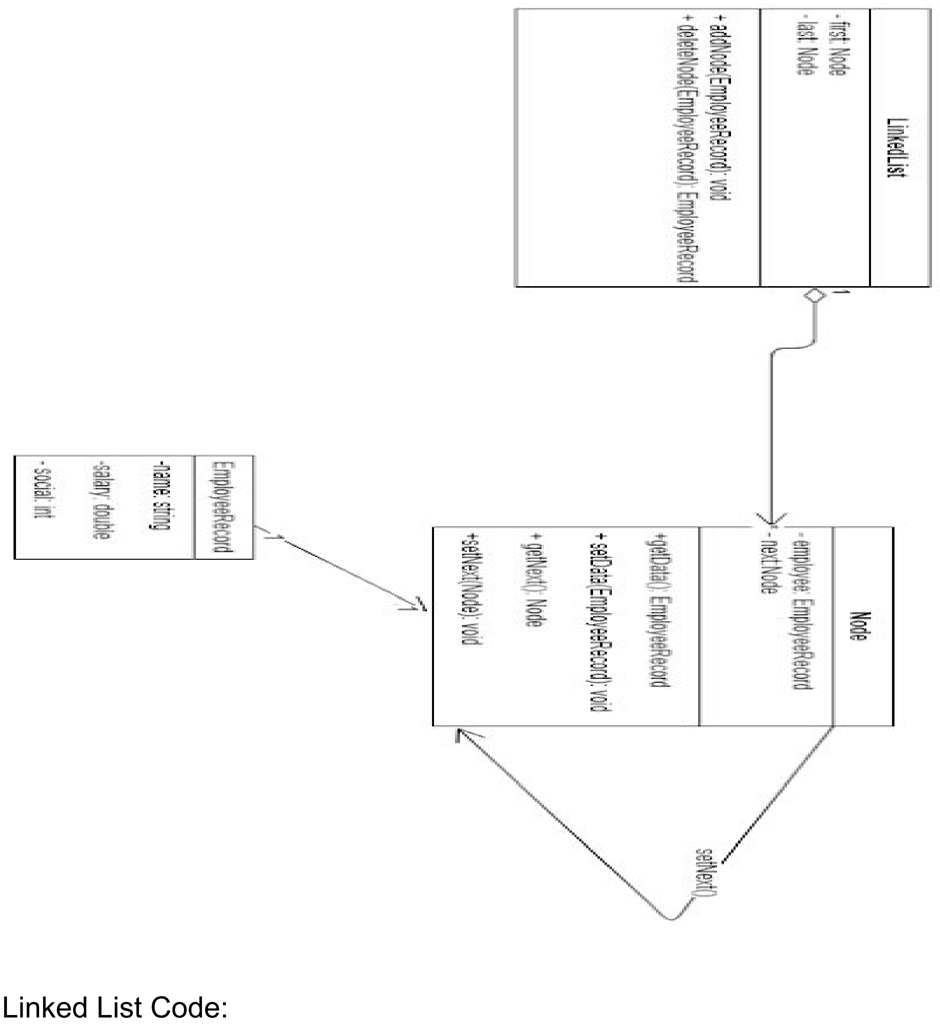


Tree Code:

```
public class Tree {
    Node roots;

    public Tree()
    {
        roots = null;
    }

    public void inserts(int spot)
    {
        roots = insert(roots, spot);
    }

    public Node insert(Node rooted, int spots)
    {
        //checking for empty tree
        if (rooted == null)
        {
            //creates nodes for empty tree
            rooted = new Node(spots);
            return rooted;
        }
        if (spots < rooted.spot)
        {
            rooted.left = insert(rooted.left, spots);
        }
        else if (spots > rooted.spot)
        {
            rooted.right = insert(rooted.right, spots);
        }
        return rooted;
    }
    //calls the sort function
    public void sorting()
    {
        sort(roots);
    }
    //orders the tree
    public void sort(Node roots)
    {
        if (roots != null)
        {
            sort(roots.left);
            System.out.println(roots.spot);
            sort(roots.right);
        }
    }
}
/*
 */
package hw3;

/**
 */
public class Node {
    public int spot;
    public Node left;
    public Node right;

    public Node(int num)
    {
        spot = num;
        left = right = null;
    }
}
/*
 * To change this license header, choose License Headers in Project Properties.
 * To change this template file, choose Tools | Templates
 * and open the template in the editor.
 */
package hw3;

/**
*/
public class Driver {
    public static void main(String[] args)
    {
        //creates a new tree
        Tree tree = new Tree();
        //numbers to insert
        tree.inserts(45);
        tree.inserts(33);
        tree.inserts(9);
        tree.inserts(21);
        tree.inserts(49);
        tree.inserts(71);
        tree.inserts(99);
        //sorts the numbers
        tree.sorting();
    }
}
```

Linked List Diagram:

Linked List Class Diagram


LinkedList
- first: Node
- last: Node
+ addNode(EmployeeRecord): void
+ deleteNode(EmployeeRecord): EmployeeRecord
1
Node
- employee: EmployeeRecord
- nextNode
+getData(): EmployeeRecord
+ setData(EmployeeRecord): void
+ getNext(): Node
+setNext(Node): void
setNext()
EmployeeRecord
-name: string
-salary: double
- social: int
1

Linked List Code:

```
public class LinkedList{
private Node first;
private Node last;
        public void LinkedList(){}//constructor for the linked list class
        public void addNode(EmployeeRecord node){
        //takes and adds the a new node to the list.
        Node newNode = new node();//sets up a temporary node to store the data.
        newNode.setData(node);//sets the data for the new node.
        if(first==null){
        //if there is no data in the list, sets the first node equal to the temp node.
                newNode.setNext(newNode);
                first=newNode;
                last=first;
                }
                else{//if the list exists, adds the data in between the first and last element.
                        last.setNext(newNode);
                        last=newNode;
                }
        }
        public EmployeeRecord removeNode(EmployeeRecord key){
        //takes out a specificed node from the list.
                Node temp= first;//this is the node we would like to remove.
                Node prev=null;//this is the iterator which goes through the list keeping track of the node behind it.
                if(temp!=null && temp.getData()==key){//if the first node holds the data, simply set the next node to
the first one.
                        first=temp.next;
                        return temp.getData();
                }
                while(temp!= null&& temp.getData()!=key){//Search for the specific record, while keeping track of
the previous nodes
                        prev=temp;
                        temp=temp.getNext();
                        if(temp.getData()==key){//if the data is found, goes through and returns that value while
setting the next value in the list
                                prev.setNext(temp.getNext());
                                return temp.getData();
                        }
                }
                if(temp==null){return null;}
                prev.getNext()=temp.getNext();
        }
}



public class Node{
        private EmployeeRecord employee;
        private Node next;

        public Node(){//constructor for the node class
```

```
			this.next=null;
		}
		public void setData(EmployeeRecord data){
			//sets the data for the node
			employee=data;
		}
		public Node getData(){
		//retrives the data from the node
			return employee;
		}
		public Node getNext(){//retrives the next node in the linked list
			return next;
		}
		public void setNext(Node temp){//sets the next node in the linked list.
			this.next=temp;
		}
	}

	public class EmployeeRecord{
		private string name;//variable used to store the name of the employee
		private int social;//the social security number of the employee
		private double salary;//the salary of the employee, set as a double in case of decimal values

		public EmployeeRecord(string empName, int security, double number){
			//the constructor for the employee records, which sets the values for name, social security, and
	salary
			name= empName;
			social=security;
			salary=number
		}
	}
```