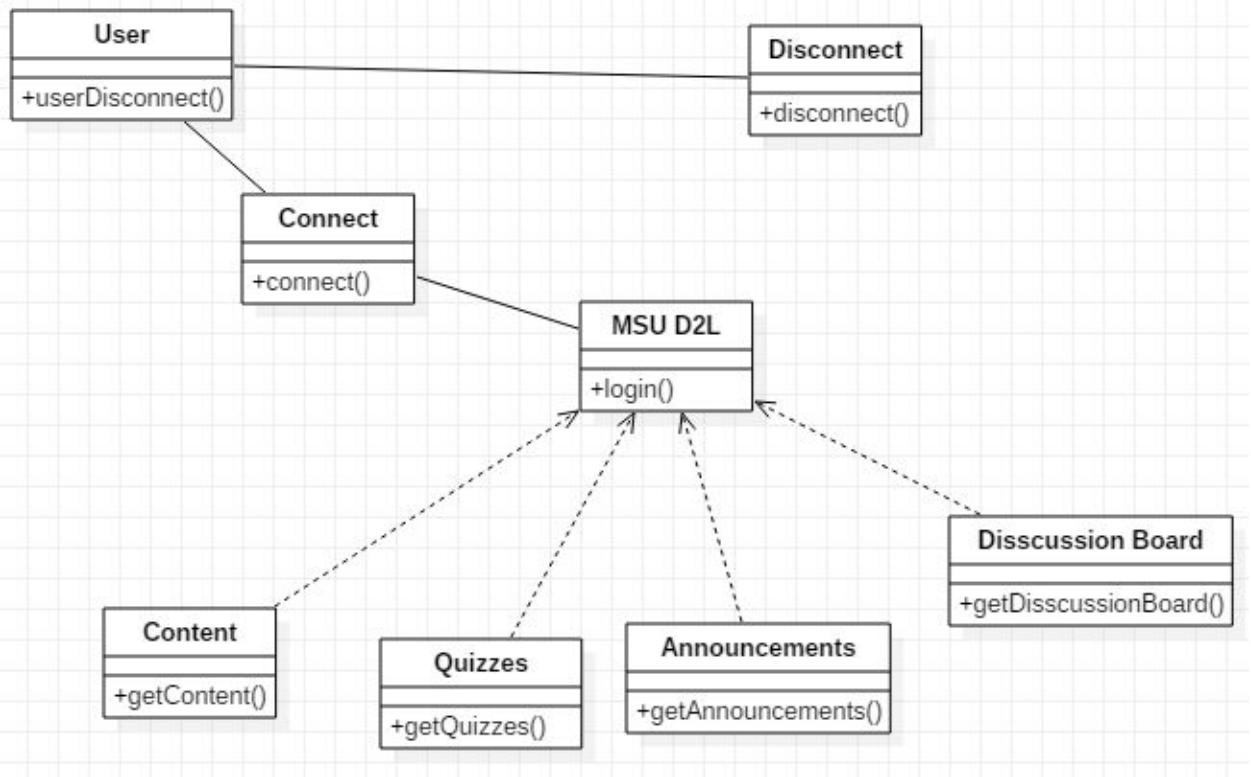
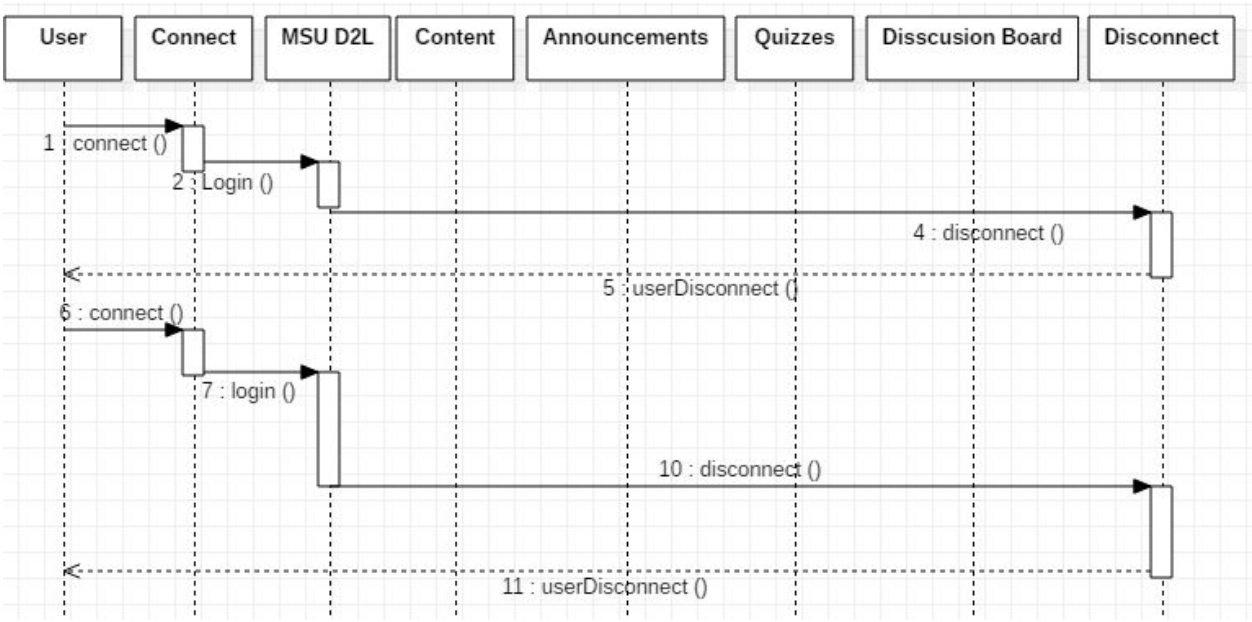
Exercise 1 Part A Jonah Gilbertson and Grant Baker



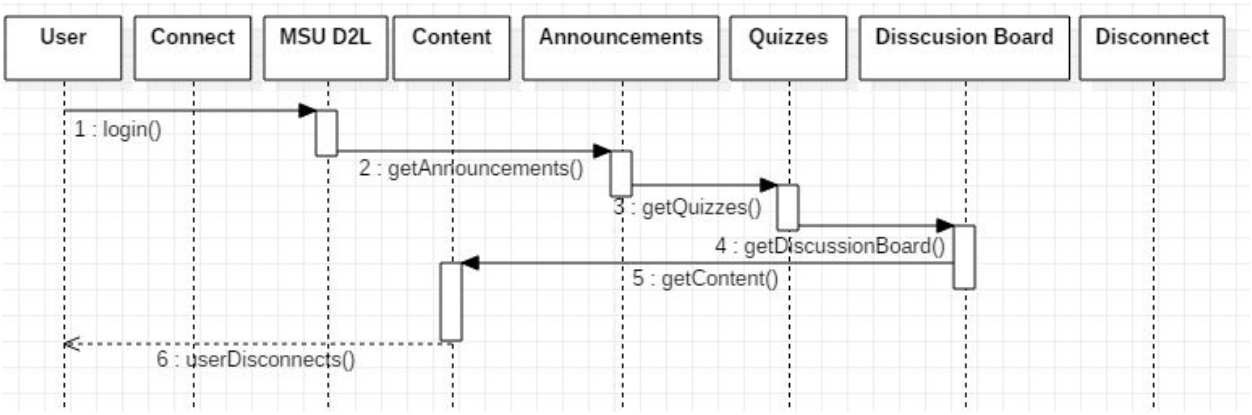
We chose the Command and Observer Pattern. The classes that participate in multiple patterns are MSU D2L. Connect and Disconnect are part of the command strategy by passing in commands to connect or disconnect the user from the Class D2l. The observer pattern are all the classes on D2L that rely on the class. (Content, Quizzes, Announcements, Discussion Board.

Exercise 1 Part B

Command



Observer



Exercise 2 part A

if 3 people finish 32 story points that's 32/3 = 10.666666 stories a person per sprint so in a 3 week period of time for 4.8 people (because one works 80%) that should be
10.66666 * 4.8 = 51.199999 story points for 4.8 people with 1 sprint

Exercise 2 part B

Focus Factor = velocity /(# of team members * productive days)

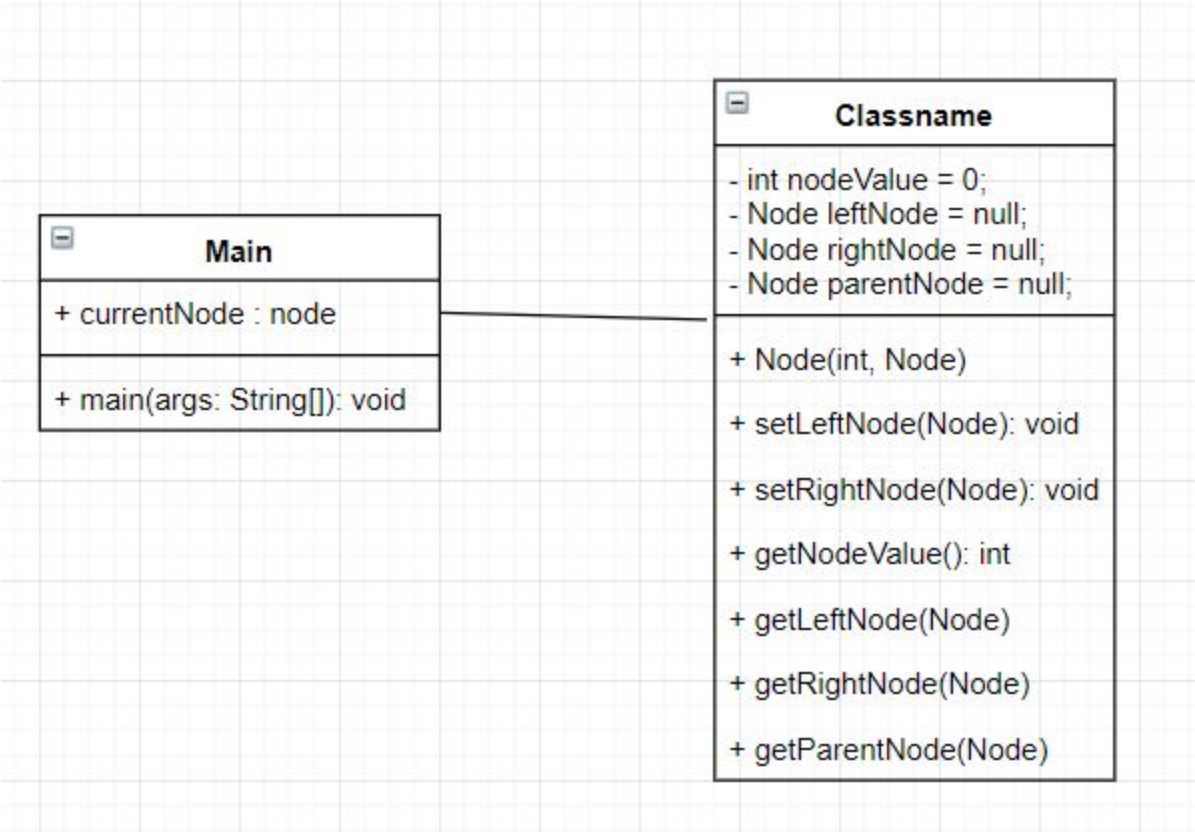
51.199999 /(5*15) = .682

Estimated Focus Factor of new team would be = .682

Exercise 2 part C

Another way to estimate story points is to have the most experienced team member decide the difficulty of each story and assign the correlating amount of points. This is a worse method than poker because the user doesn't necessarily have the same experience per-task which could skew the amount of story points users are getting.

Exercise 2 part D



Exercise 2 part E

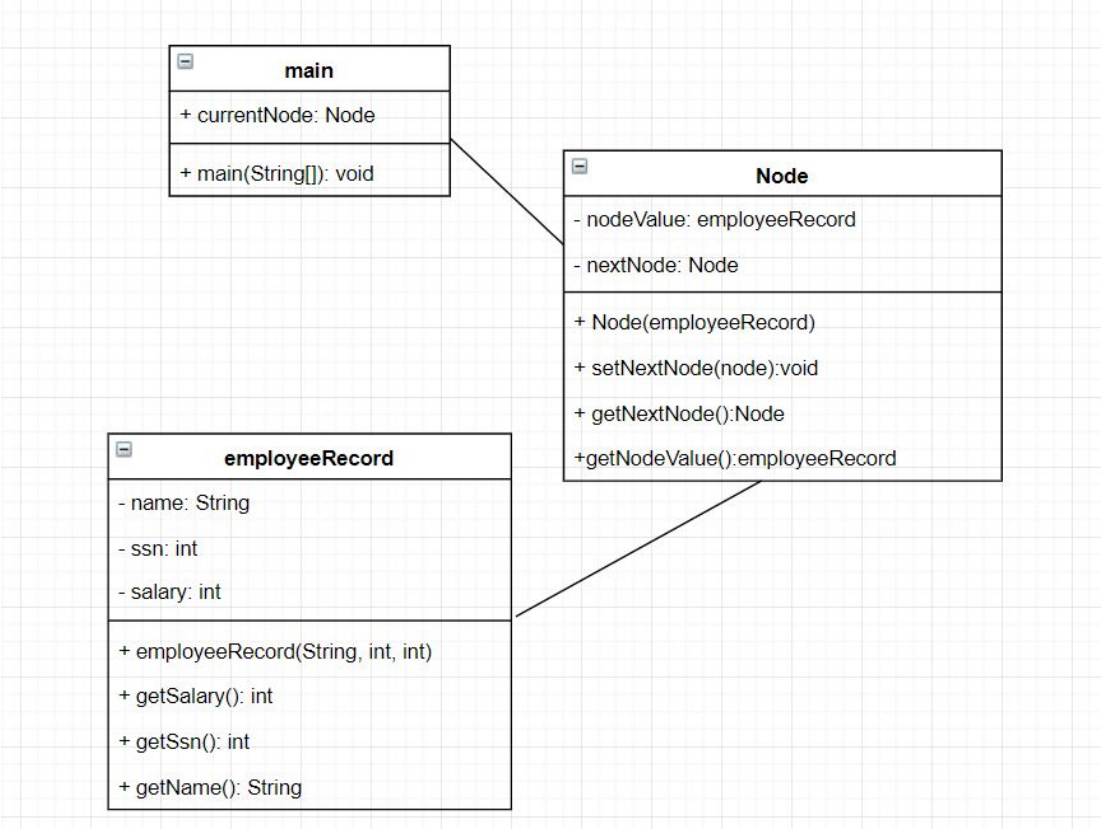
```
public class main {
    public static void main(String[] args) {
        //main method to show how to use linked list
        //setting current node passing in the nodes value 1 with a null parent
        Node currentNode = new Node(1, null);

        //setting the left node by running setLeftNode and creating node with value 2
        currentNode.setLeftNode(new Node(2,currentNode));

        //setting the right node by running setRightNode and creating node with value 3
        currentNode.setRightNode(new Node(3,currentNode));

        //should print 1 by getting the current nodes value
        System.out.println(currentNode.getNodeValue());

        //setting current node to left child
        currentNode = currentNode.getLeftNode();

        //should print 2 by getting the currentNodes value
        System.out.println(currentNode.getNodeValue());

        //set current node back to root node
        currentNode = currentNode.getParentNode();
```

```
        //setting current node to right child
        currentNode = currentNode.getRightNode();

        //should print 3 by getting the currentNodes value
        System.out.println(currentNode.getNodeValue());
    }
}

public class Node{
    //initializing variables nodeValue and left and right nodes
    private int nodeValue = 0;
    private Node leftNode = null;
    private Node rightNode = null;
    private Node parentNode = null;

    //constructor setting nodevalue
    public Node(int nodeVal, Node parent){
        nodeValue = nodeVal;
        parentNode = parent;
    }

    //function to set the leftNode
    public void setLeftNode(Node node) {
```

```
    leftNode = node;
}

//function to set the rightNode
public void setRightNode(Node node) {
    rightNode = node;
}

//function to get the nodes value
public int getNodeValue(){
    return nodeValue;
}

//function to get the left nodes
public Node getLeftNode(){
    return leftNode;
}

//function to get the nodes
public Node getRightNode(){
    return rightNode;
}

public Node getParentNode(){
```

```
        return parentNode;
    }
}
```

Exercise 2 part F



Exercise 2 part G

```
public class main {
    public static void main(String[] args) {
        //main method to show how to use linked list
        //setting current node passing in the nodes value with greg's employee record
        Node currentNode = new Node(new EmployeeRecord("greg", 123456789, 40000));

        //setting the next node by running setNextNode and creating node with steve's employee
record
        currentNode.setNextNode(new Node(new EmployeeRecord("steve", 923456789,
100000)));
```

```
        //setting the 3rd node by running setNextNode and creating node with value mike's
employee record
        currentNode.getNextNode().setNextNode(new Node(new EmployeeRecord("mike",
555555555, 20000)));

        //should print greg's name
        System.out.println(currentNode.getNodeValue().getName());

        //utilized linked list by getting following node in list and setting to currentNode
        currentNode = currentNode.getNextNode();

        //should print steve's SSN
        System.out.println(currentNode.getNodeValue().getSsn());

        //utilized linked list by getting following node in list and setting to currentNode
        currentNode = currentNode.getNextNode();

        //should print mike's salary
        System.out.println(currentNode.getNodeValue().getSalary());
    }
}

public class Node{
    //initializing variables nodeValue and nextNode
    private EmployeeRecord nodeValue
    private Node nextNode = null;

    //constructor setting nodevalue
    public Node(EmployeeRecord nodeVal){
        nodeValue = nodeVal;
    }

    //function to set the next node
    public void setNextNode(Node node) {
        nextNode = node;
    }

    //function to get the next node
    public Node getNextNode() {
        return nextNode;
    }

    //function to get the nodes value
```

```
    public EmployeeRecord getNodeValue(){
        return nodeValue;
    }
}

public class EmployeeRecord{
    //initializing variables
    private String name;
    private int ssn;
    private int salary;

    //constructor taking in the name SSN and salary
    public EmployeeRecord(String n, int s, int sal){
        name = n;
        ssn = s;
        salary = sal;
    }

    //method to get salary
    public int getSalary() {
        return salary;
    }

    //method to get ssn
    public int getSsn() {
        return ssn;
    }

    //method to get name
    public String getName() {
        return name;
    }
}
```