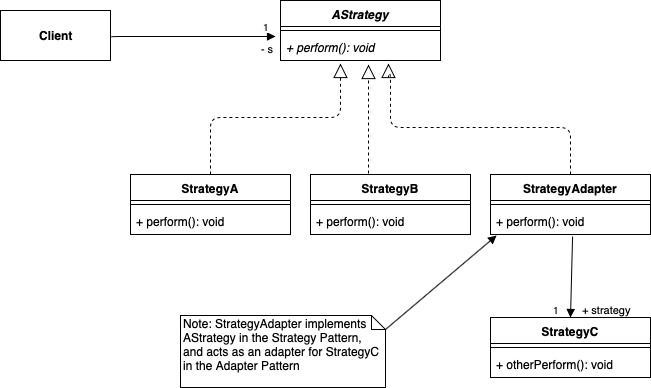
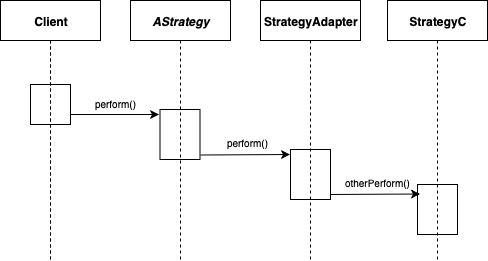
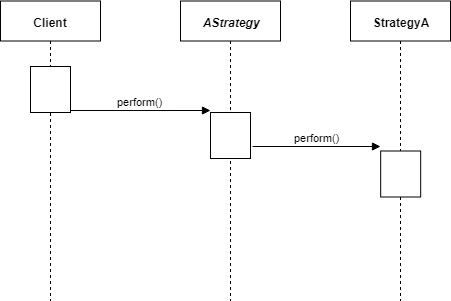
**ESOF 322: Homework 3**

**( Exercise I )**

****

****

****

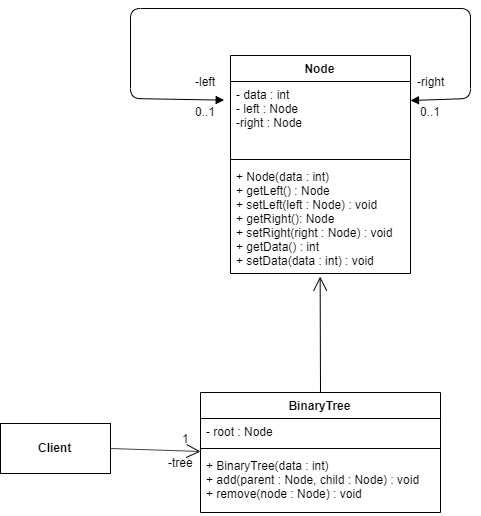
**( Exercise II )**

1. Focus Factor = (32 story points / 45-man days) = .71

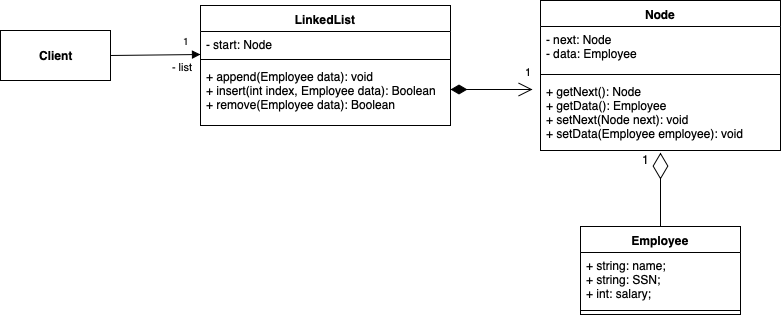
Estimated Velocity = Available man days \* Focus Factor = 72-man days \* .71 = 51.12

= **51 estimated story points**

1. You can estimate a focus factor by estimating how much time each day would be “wasted”. A focus factor of around 70% is typical for most professional companies. This theoretically means that out of a typical work day, around 70% of total man hours are spent being productive and contributing to the project, while the rest is either distracted work or relaxing.
2. One other way of estimating story points would be bucketing tasks into “Long”, “Medium”, “Short”, and “Tiny” groups. This way of estimating would involve group members sorting the tasks physically into groups by how long they think that specific note will take. This approach is a little different from poker, but no better and no worse. The benefit of having four groups and no numbers, is that sorting the notes is generally easier and there will likely be less debate on the length of the note. The downside of this approach compared to poker is that the poker approach has associated numbers which are easier to correlate to real-world days.



|  |
| --- |
| public class Client {  private BinaryTree tree;  public static void main(String[] args){  //make an instance of BinaryTree and use it to   //add and manipulate the nodes.  } } public class BinaryTree {  private static Node **root**;   //Nested class for Node  **class** Node **{  private** int data;  private Node **left**;  private Node **right**;   public Node **(int** data) {  this.data = data;  right = null;  left = null;  }  //Getters and Setters for nodes and data  public int getData() {  return data;  }  public void setData(int data) {  this.data = data;  }  public Node **getLeft**() {  return left;  }  public void setLeft(Node **left**) {  this.left = left;  }   public Node **getRight**() {  return right;  }  public void setRight(Node **right** ) {  this.right = right;  }   }    //BinaryTree constructor, takes initial data and makes a root node  **public** BinaryTree(int data)  {  root = new Node**(data**);  }   //method for adding a new node**, checks** to see if the left node **of** the parent  //is full and if so adds to the right instead  public void add(Node **parent**, Node **child**)  {  if (parent.getLeft() == null)  {  parent.setLeft(child);  }  else  {  parent.setRight(child);  }  }    //Method for deleting node **from** tree. Just checks to see if there are no   //children and then deletes the node **assuming** it is not the root.  public void delete(Node **node**){  if(node.**getLeft**() == null && node.**getRight**() == null){  if(!(node.**equals**(root))){  node **= null**;  }  }  else if(node.**getLeft**() != null || node.**getRight**() == null){  node.**setLeft**(null);  node.**setRight**(null);  }  else{  System.out.println("Something else happened");  }  } } |

f)

g)



|  |
| --- |
| **public** **class** Client {  **private** LinkedList list;  **public** **static** **void** main(String[] args)   {  *// Create LinkedList with new Employee*  *// Add, Insert, or Remove nodes in LinkedList*  } }  **public** **class** LinkedList {    **private** Node start;    **public** LinkedList(Employee employee)  {  start = **new** Node();  start.setData(employee);  }    **public** **void** append(Employee employee)  {  *// Code for appending node with employee onto end of LinkedList*  }    **public** Boolean insert(**int** index, Employee employee)  {  Boolean wasSuccessful = **false**;    *// Code for finding index and inserting node with employee*    **return** wasSuccessful;   }    **public** Boolean remove(Employee employee)  {  Boolean wasSuccessful = **false**;    *// Code for finding and removing employee*    **return** wasSuccessful;  } }  **public** **class** Node {    **private** Node next;  **private** Employee data;    **public** Node getNext() { **return** next; }  **public** **void** setNext(Node next) { **this**.next = next; }    **public** Employee getData() { **return** data; }  **public** **void** setData(Employee data) { **this**.data = data; } }  **public** **class** Employee {  **public** String name;  **public** String SSN;  **public** **int** salary; } |