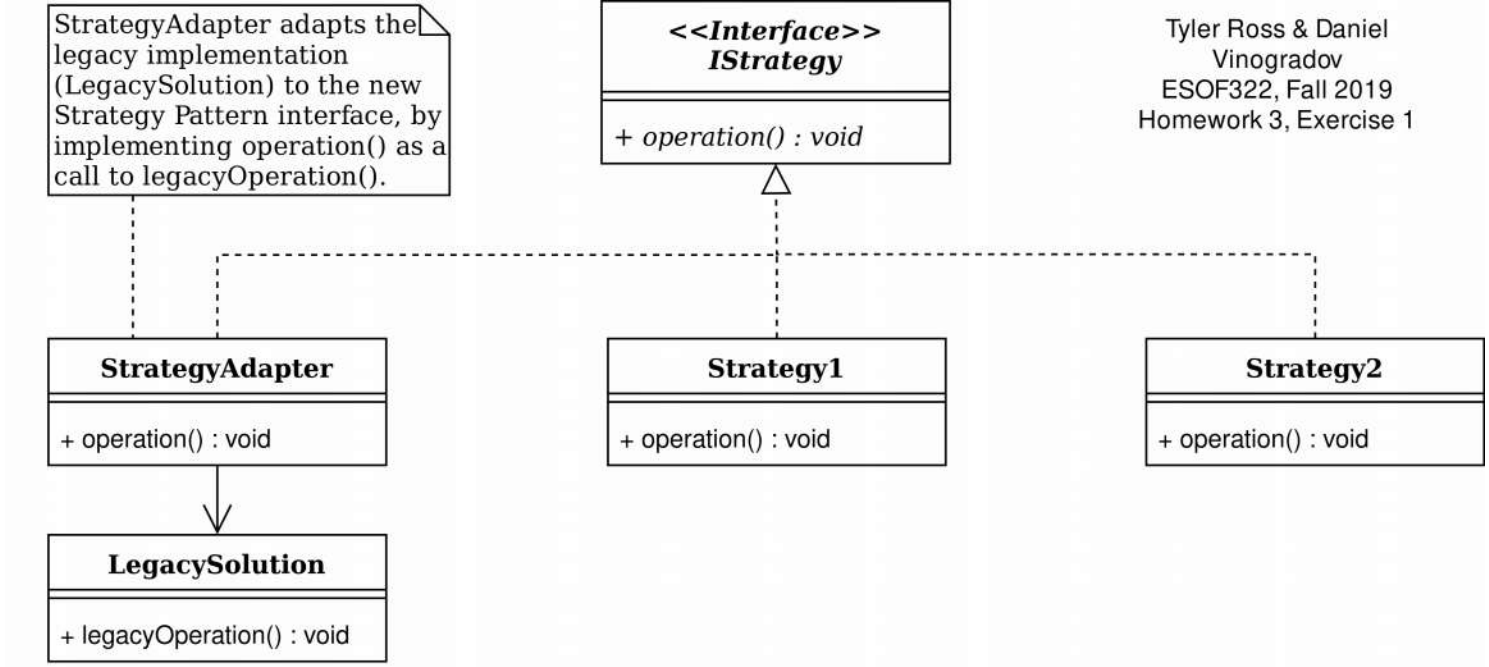
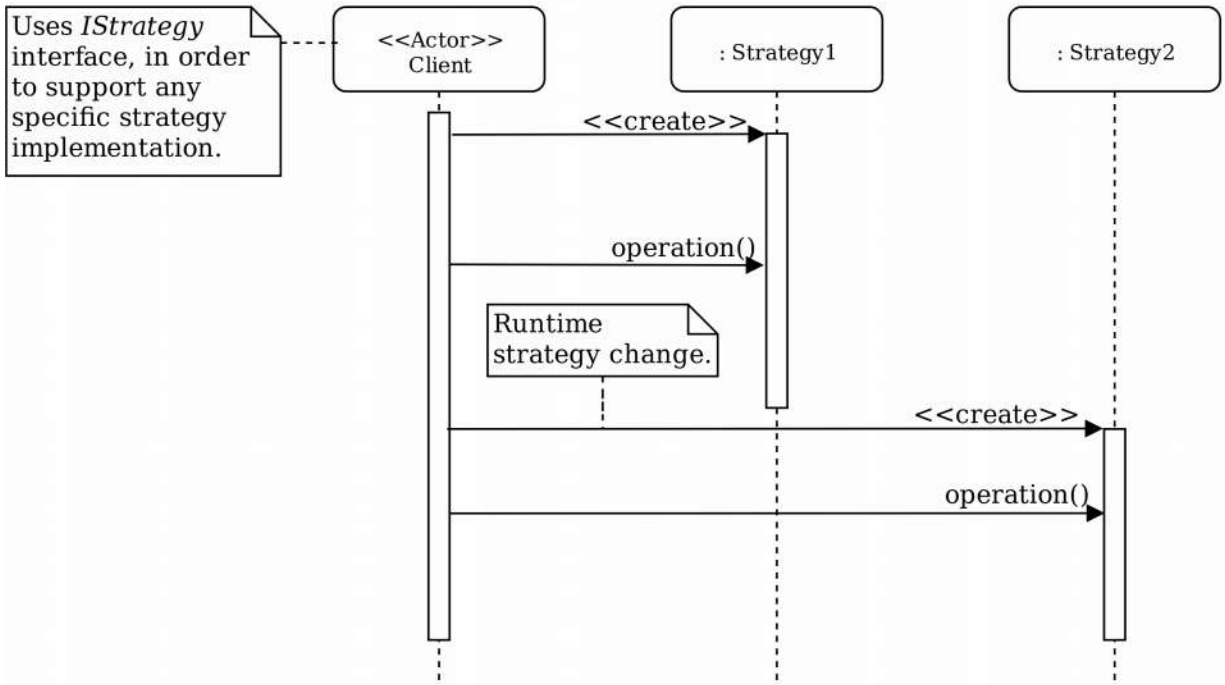
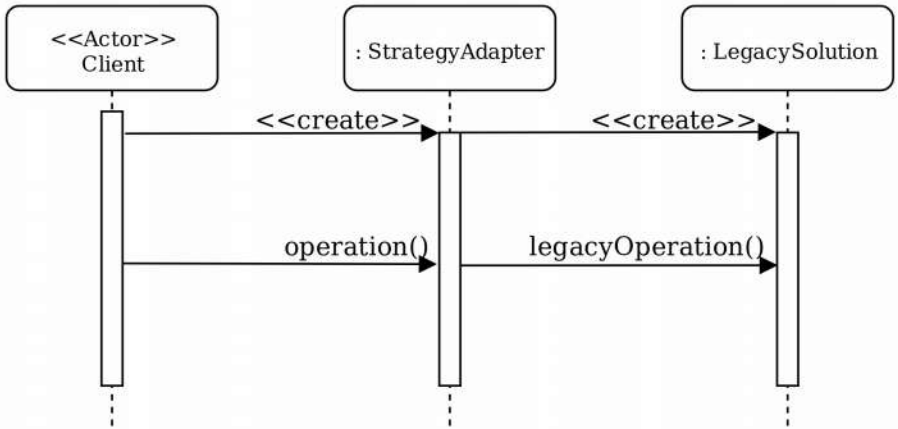
Tyler Ross & Daniel Vinogradov
ESOF322, Fall 2019
Homework 3, Exercise 1







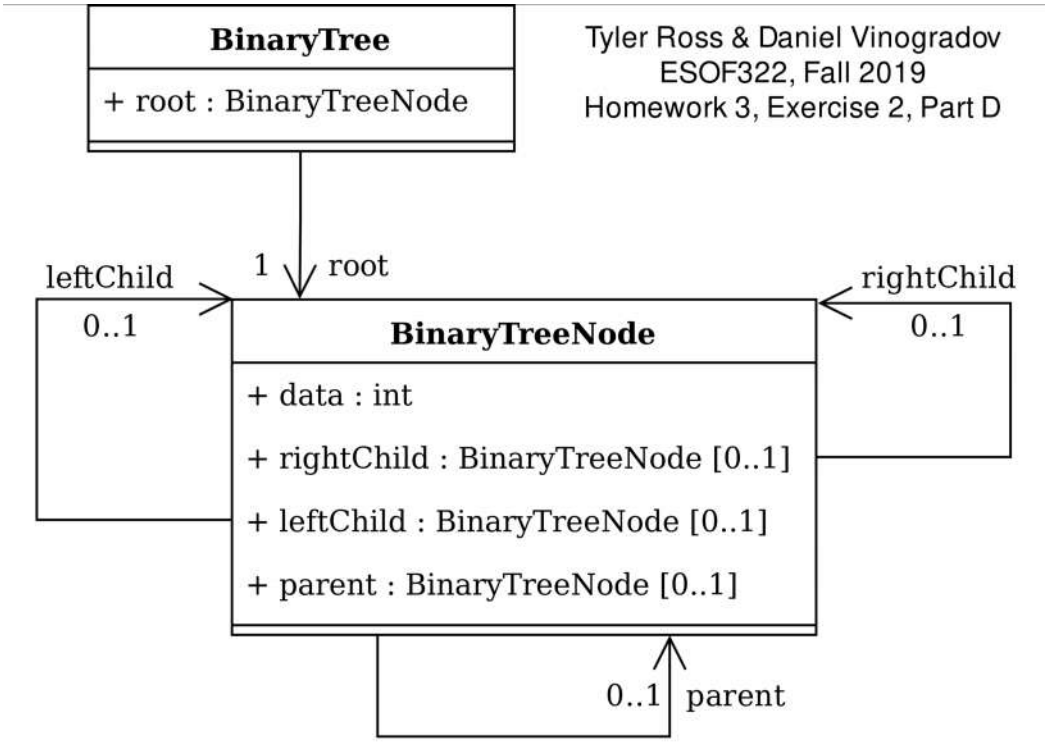
ESOF322, Fall 2019, Homework 3, Exercise 2,
Parts A-C, Tyler Ross & Daniel Vinogradov

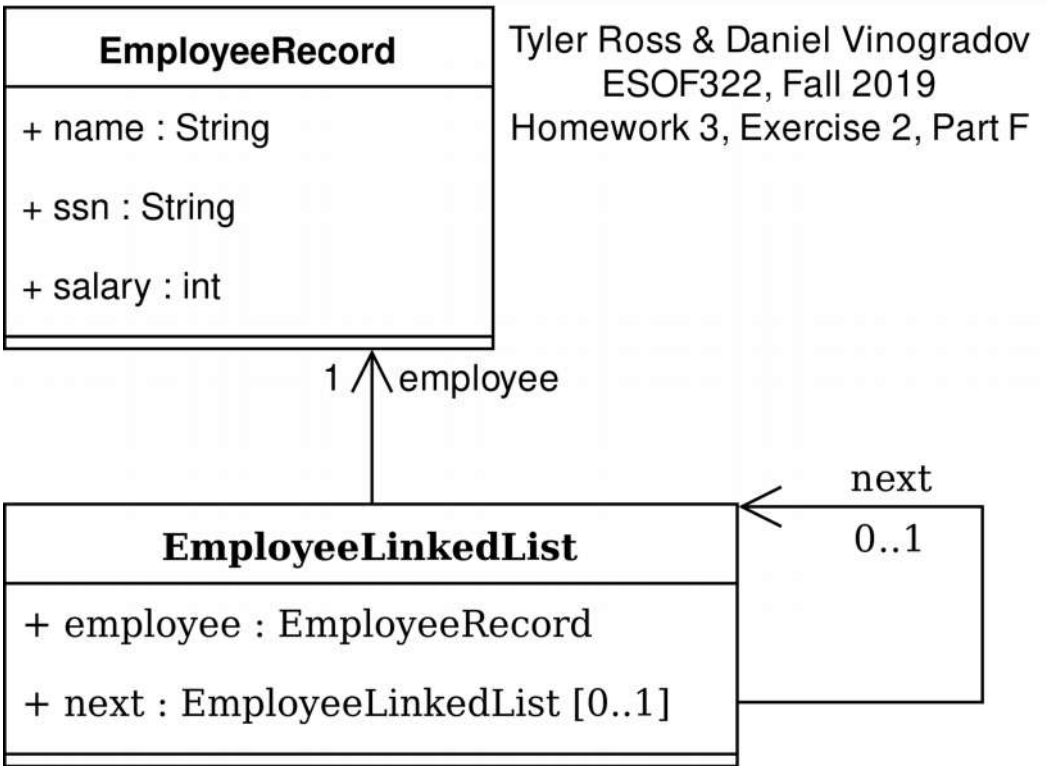
a) Given 45 man-days (3 engineers; 3 week sprint), a velocity of 32 story points has a focus factor of ~71%. If the team becomes 5 engineers, with one having only 80% availability, a 3 week sprint now consists of 72 (4*15 + 1*15*0.8) man-days. Using the team's previous focus factor of ~71%, with the two additional engineers, they should have a velocity of around 51 story points in the next sprint.

b) Assuming no prior experience with any team members' performance, an initial focus factor of 70-80% is a common, safe estimate. Following the first sprint, the team's actual velocity can be used to determine a more concrete focus factor.

c) Ordering stories from smallest to largest, without assigning any story points, provides a relative comparison. Starting from the smallest, and using pseudo-Fibonacci values (e.g. 1,2,3,5,8,13,20,40,etc.), each story can be given a value based on size relative to smaller stories and previously completed stories.

Compared to story point poker, this helps put stories in context with each other. Working from the smallest up also provides a basis for sizing subsequent stories. On the other hand, this method does nothing to solve the issue of particular team members dominating the conversation - which poker does well.


BinaryTree
+ root : BinaryTreeNode
Tyler Ross & Daniel Vinogradov
ESOF322, Fall 2019
Homework 3, Exercise 2, Part D
leftChild
0..1
1
root
rightChild
0..1
BinaryTreeNode
+ data : int
+ rightChild : BinaryTreeNode [0..1]
+ leftChild : BinaryTreeNode [0..1]
+ parent : BinaryTreeNode [0..1]
0..1
parent


EmployeeRecord
+ name : String
+ ssn : String
+ salary : int
Tyler Ross & Daniel Vinogradov
ESOF322, Fall 2019
Homework 3, Exercise 2, Part F
1
employee
next
0..1
EmployeeLinkedList
+ employee : EmployeeRecord
+ next : EmployeeLinkedList [0..1]

```java
// ESOF322, Fall 2019, Homework 3, Exercise 2,
// Part E, Tyler Ross & Daniel Vinogradov

// Absolute barebones implementation.
// Provides no helper methods, convenient constructors, etc..
// Thus, the user of this data structure is expected to
// handle setting/getting, maintaining the tree,
// properly linking children/parents, etc..

class BinaryTree {
    public BinaryTreeNode root;

    BinaryTree(int rootNodeData) {
        this.root = new BinaryTreeNode(rootNodeData);
    }
}

class BinaryTreeNode {
    public int data;
    public BinaryTreeNode rightChild;
    public BinaryTreeNode leftChild;
    public BinaryTreeNode parent;

    BinaryTreeNode(int data) {
        this.data = data;
    }
}

```

```java
// ESOF322, Fall 2019, Homework 3, Exercise 2,
// Part G, Tyler Ross & Daniel Vinogradov

// Absolute barebones implementation.
// Provides no helper methods, convenient constructors, etc..
// Thus, the user of these data structures is expected to
// handle setting/getting parameters, validating data (e.g. SSNs),
// and properly linking the "next" entry of the linked list.

class EmployeeLinkedList {
    public EmployeeRecord employee;
    public EmployeeLinkedList next;

    EmployeeLinkedList(EmployeeRecord employee) {
        this.employee = employee;
    }
}

class EmployeeRecord {
    public String name;
    public String ssn;
    public int salary;

    EmployeeRecord(String name, String ssn, int salary) {
        this.name = name;
        this.ssn = ssn; // Data validity is assumed and left to the user.
        this.salary = salary;
    }
}

```