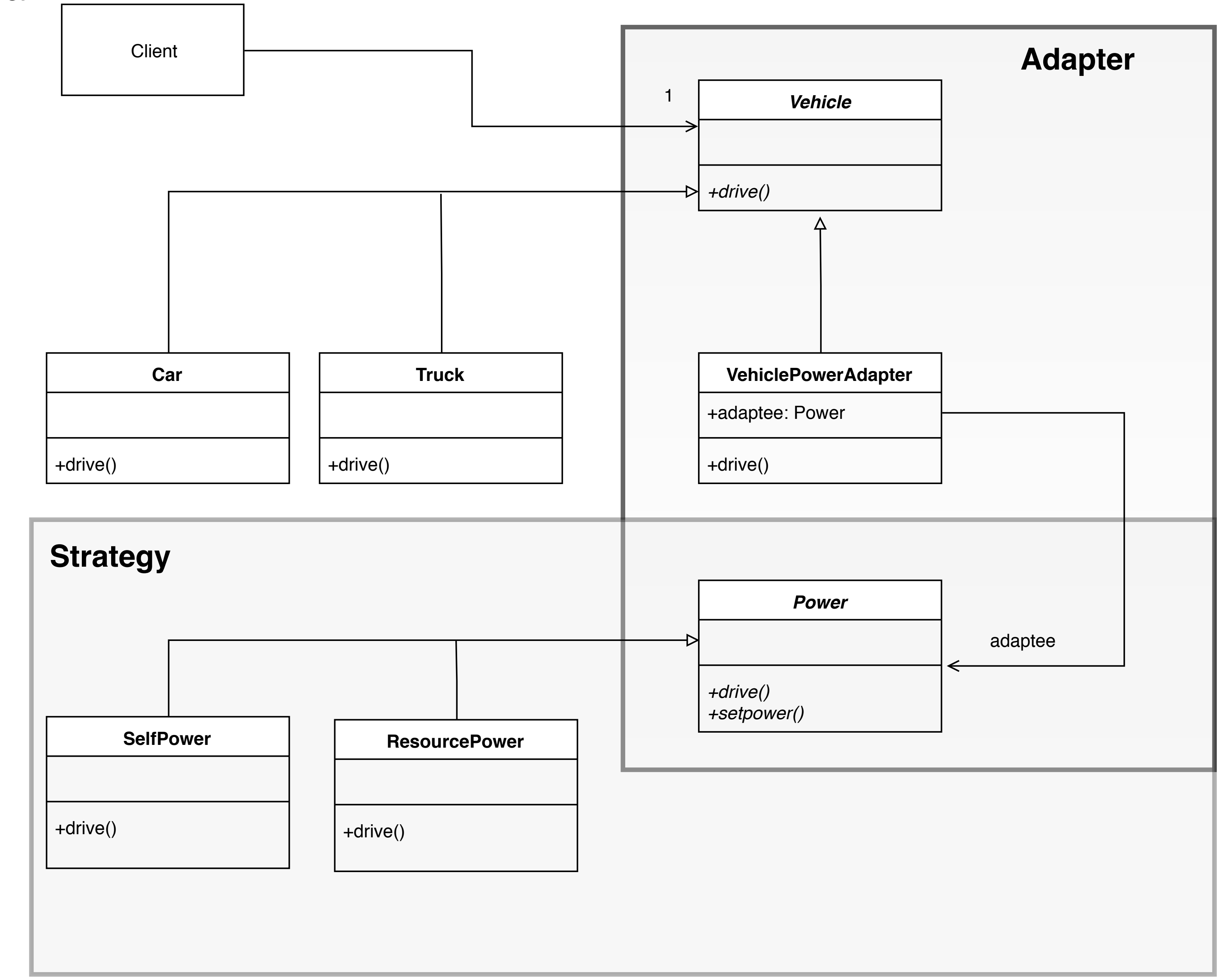
Cole Eddie & Johnny Egan
ESOF 322
HW 3

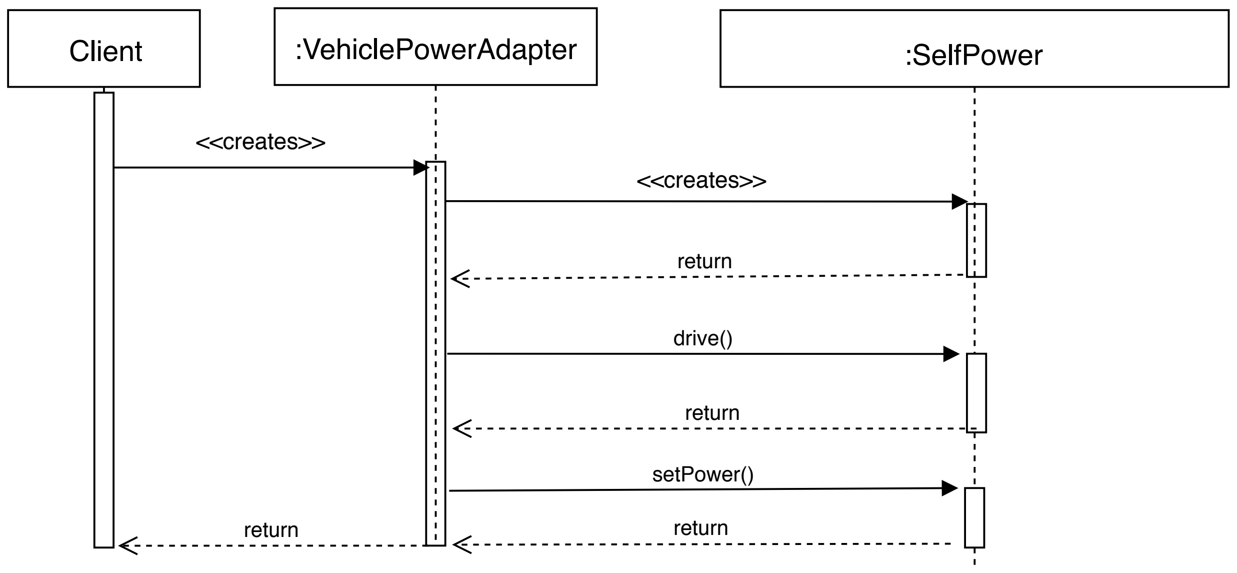
Exercise 1:

a.



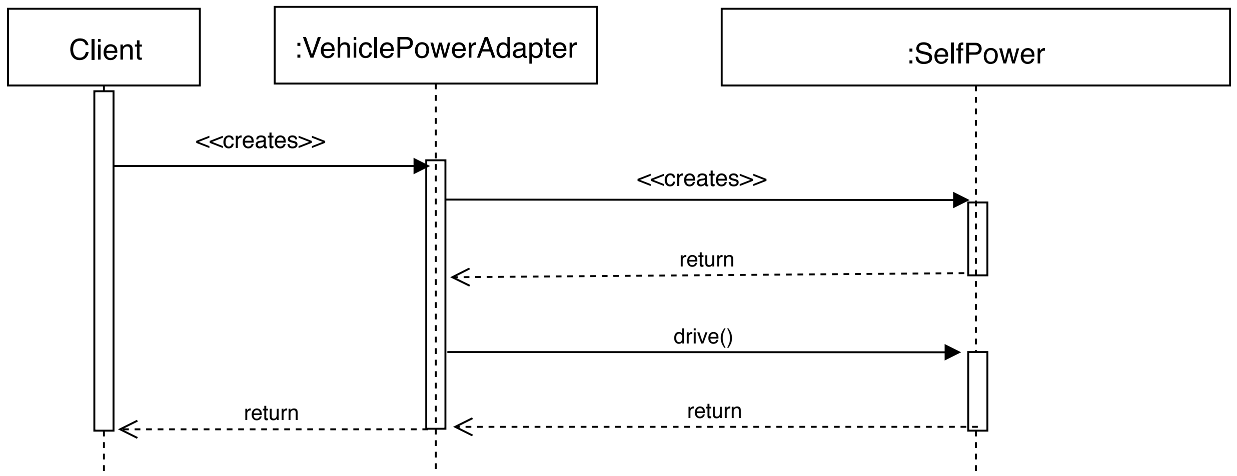
b.

Strategy Pattern



Note: Uses the strategy pattern by allowing drive() to be implemented by each class in a way that best suits that class. The SelfPower class is shown implementing it's own method of drive() in the diagram as defined by the code within. In addition, the client is able to change the power by calling setpower(), to dynamically change the drive() method called at run-time.

Adapter Pattern



Note: Uses the adapter pattern by allowing the user to use the abstract vehicle class interface, via the VehiclePowerAdapter, to interact with the abstract Power class intererface and still perform the drive() method as the client expects.

Exercise 2:

a. Known: 32 story points last sprint with 3 person team and 45 man days.
New: +2 persons, one that can only work 80%

Last Sprint Focus Factor = 32/45 approx: 71%

One engineer will add 15 man days, and the other will put in 12 (15*80%).

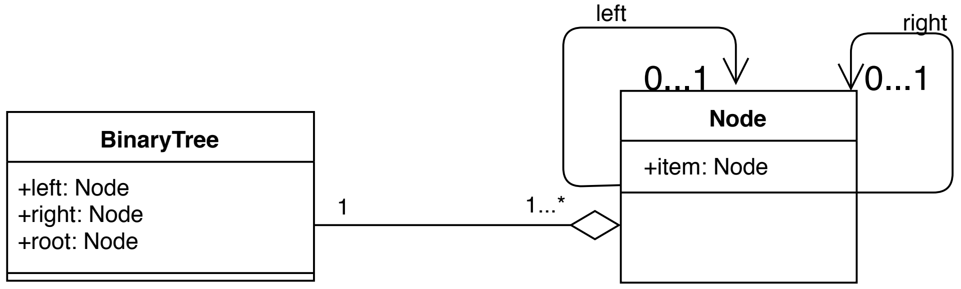
Estimated Velocity = 72 * 71% = $\boxed{\text{51 story points}}$

b. If you have previous sprint data, you could still estimate a focus factor based on that data, which is why this method is useful. Teams are always changing as people come and go.

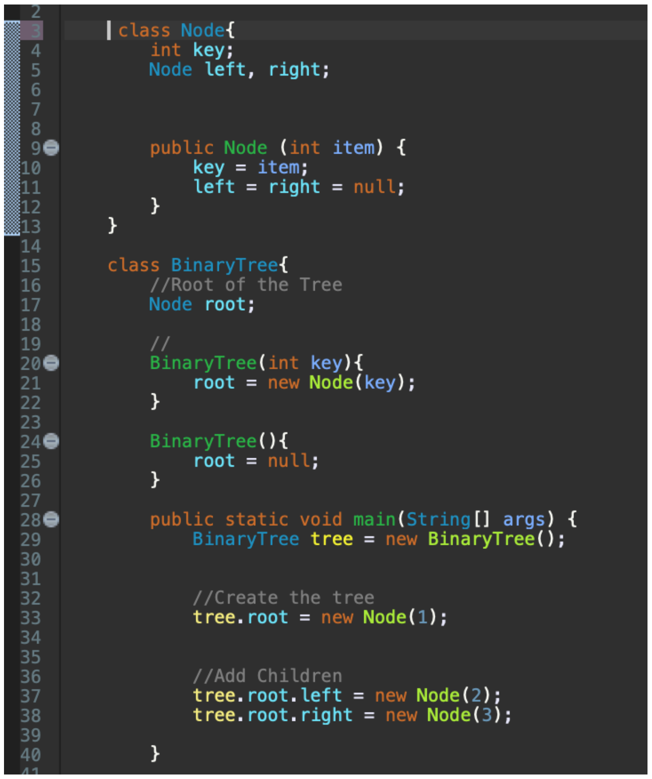
Nevertheless, if no data was available, you could estimate a focus factor based on the working environment and perhaps the timing of the sprint as far as incoming holidays. I would estimate an actual focus factor to be between 60% & 80% as most environments and employees are fairly distracted. Starting at a focus factor of 75%, I would then increase or decrease it. If I felt like the work environment was full of distractions, i.e. the Christmas season was right around the corner I might decrease the FF. I would likely increase the focus factor during a month like June, since there really aren't very many holidays celebrated during this time. The work environment could also dynamically change. For example, I would expect production to be higher when a major company deadline is approaching as opposed after a major deadline.

c. You could go to a single person and have them estimate the difficulty of each task on a semi-Fibonacci scale. Unlike using a semi-Fibonacci poker method, this way is more biased to the strength's and weakness's of the individual being asked. While one personmight find a particular task very difficult, another may not find the task very difficult.Because the individual asked has no outside input via a group discussion and rating likehe or she does in the poker strategy, the individual is likely to overlook some aspects of each task as well. However, by asking one person, this strategy could be much faster than the poker method. In addition, the poker method might be unreasonable if it is very difficult to get the "team" together at a single time. The poker strategy is likely to give you a much more accurate estimate of story points but there are several drawbacks and depending on the situation it may not be the best case, which is why there are multiple strategies to estimate story points.

d.



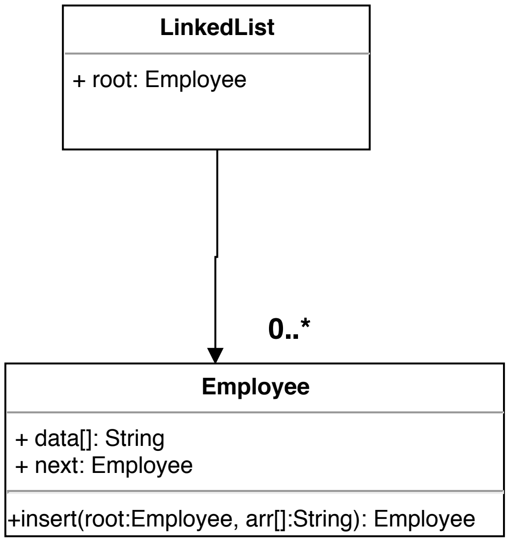
e.



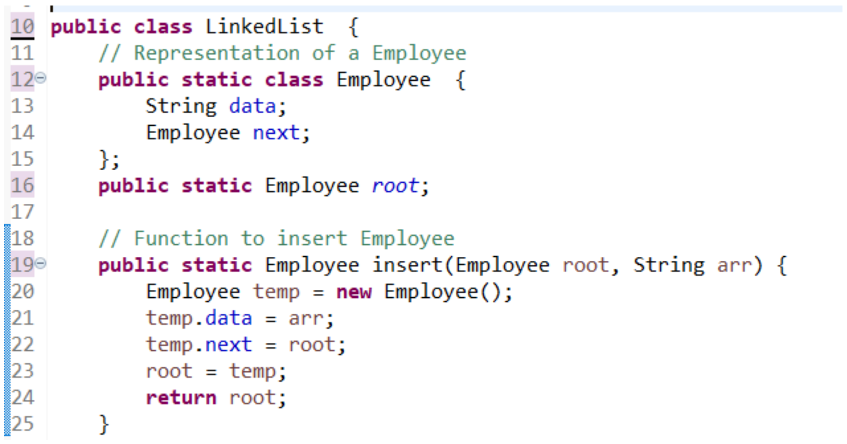
```java
class Node{
    int key;
    Node left, right;


    public Node (int item) {
        key = item;
        left = right = null;
    }
}

class BinaryTree{
    //Root of the Tree
    Node root;

    //
    BinaryTree(int key){
        root = new Node(key);
    }

    BinaryTree(){
        root = null;
    }

    public static void main(String[] args) {
        BinaryTree tree = new BinaryTree();


        //Create the tree
        tree.root = new Node(1);


        //Add Children
        tree.root.left = new Node(2);
        tree.root.right = new Node(3);

    }
```

Binary Tree code taken from GeeksforGeeks.org/binary-tree-set-1-introduction

f.



g. Used https://www.geeksforgeeks.org/create-linked-list-from-a-given-array/ but modified for the problem.



```java
public class LinkedList  {
    // Representation of a Employee
    public static class Employee  {
        String data;
        Employee next;
    };
    public static Employee root;

    // Function to insert Employee
    public static Employee insert(Employee root, String arr) {
        Employee temp = new Employee();
        temp.data = arr;
        temp.next = root;
        root = temp;
        return root;
    }
```