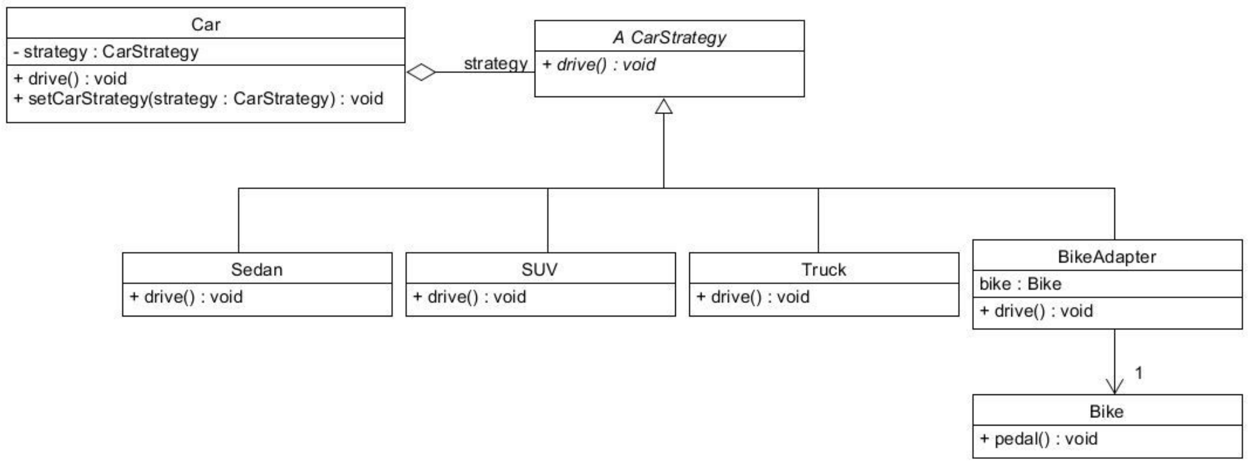
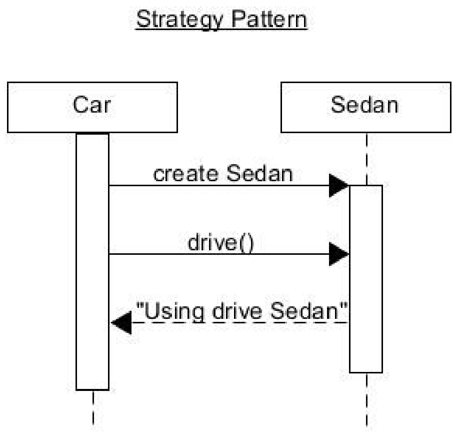
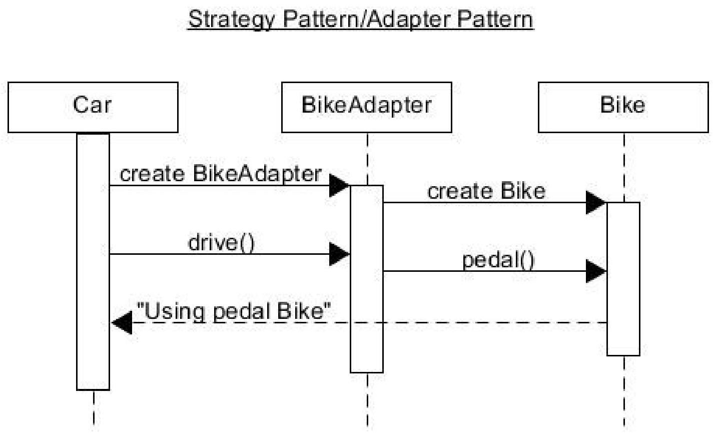
Software Engineering - Homework 3

Partners: Arynn Collins and Kevin Kleiman

Exercise 1:





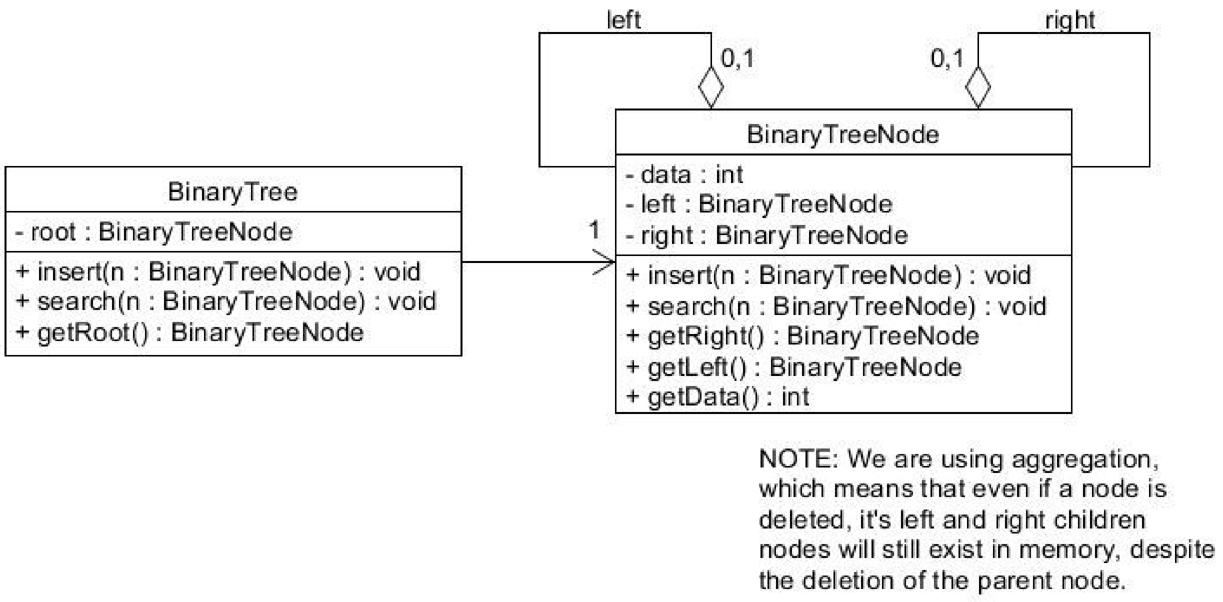


Exercise 2:

A. 32 story points / 3 sprints = 10.67 average velocity
   (4.8 person team * 10.67 avg. velocity) / 3 weeks per sprint = 17.07 est. velocity
B. If we have a brand-new team, the focus factor would be estimated to be close to 70-75 percent, since that accounts for missed or sick days, or appointments the team members will encounter throughout the sprint period.
C. In estimating story points, a useful technique could be reflecting upon previous projects/tasks as well as considering the story points of the team upon completion of the aforementioned tasks, then rating those tasks on a spectrum of colors, blue to red, where blue is an easier task and red is more difficult. Once the previous projects have been rated, the current tasks at hand could be compared to the difficulty of the older projects/tasks, thus allowing for a more accurate story point calculation through a relatable difficulty scale and ease of scaling via the color spectrum. This would be better

than poker because it provides insight to tasks, as well as accounting for the previous team's story points, resulting in a more accurate way to calculate story points for the current team.

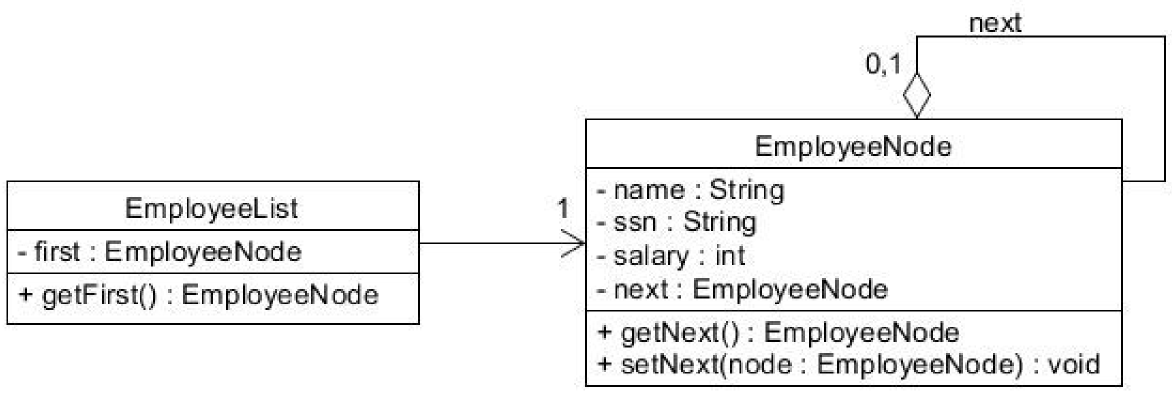
D.


left
0,1
0,1
right
BinaryTree
- root : BinaryTreeNode
+ insert(n : BinaryTreeNode) : void
+ search(n : BinaryTreeNode) : void
+ getRoot() : BinaryTreeNode
1
BinaryTreeNode
- data : int
- left : BinaryTreeNode
- right : BinaryTreeNode
+ insert(n : BinaryTreeNode) : void
+ search(n : BinaryTreeNode) : void
+ getRight() : BinaryTreeNode
+ getLeft() : BinaryTreeNode
+ getData() : int
NOTE: We are using aggregation, which means that even if a node is deleted, it's left and right children nodes will still exist in memory, despite the deletion of the parent node.

E.
```
public class BinaryTree {
        private BinaryTreeNode root;
        BinaryTree() {
                // initializing root node
                root = new BinaryTreeNode(15);
        }
        public void insert(BinaryTreeNode n) {
                // insert node n
        }
        public void search(BinaryTreeNode n) {
                // search tree for node n
        }
        public BinaryTreeNode getRoot() {
                return root;
        }
}
```

```
public class BinaryTreeNode {
        private BinaryTreeNode left;
        private BinaryTreeNode right;
        private int data;
        BinaryTreeNode(int data) {
                // initializing new node with the integer data
                private n = new BinaryTreeNode(data);
        }
        public void insert(BinaryTreeNode n) {
                // insert node n
        }
        public void search(BinaryTreeNode n) {
                // search for node n
        }
        public BinaryTreeNode getRight() {
                return right;
        }
        public BinaryTreeNode getLeft() {
                return left;
        }
        public int getData() {
                return data;
        }
}
```

F.



NOTE: We are assuming a singly linked list, since it was not specified on the assignment.

G.

```
public class EmployeeList {
        private EmployeeNode first;
        EmployeeList() {
                // initializing first node
                first = new EmployeeNode(‘Bob Burns’, ‘555-55-5555’, 50000);
        }
        public EmployeeNode getFirst() {
                return first;
        }
}

public class EmployeeNode {
        private EmployeeNode next;
        private String name;
        private String ssn;
        private int salary;
        EmployeeNode(String name, String ssn, int salary) {
                // initializing the new node
                node = new EmployeeNode(name, ssn, salary);
        }
        public EmployeeNode getNext() {
                return next;
        }
        public void setNext(EmployeeNode node) {
                this.next = node;
        }
}
```