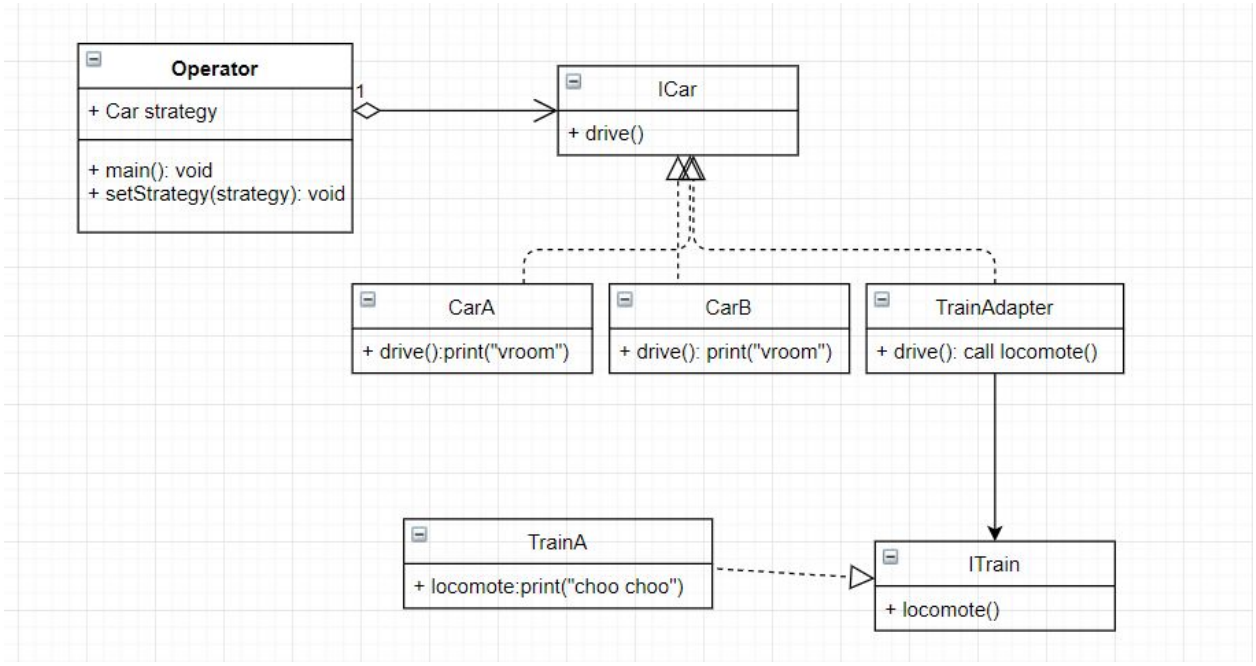
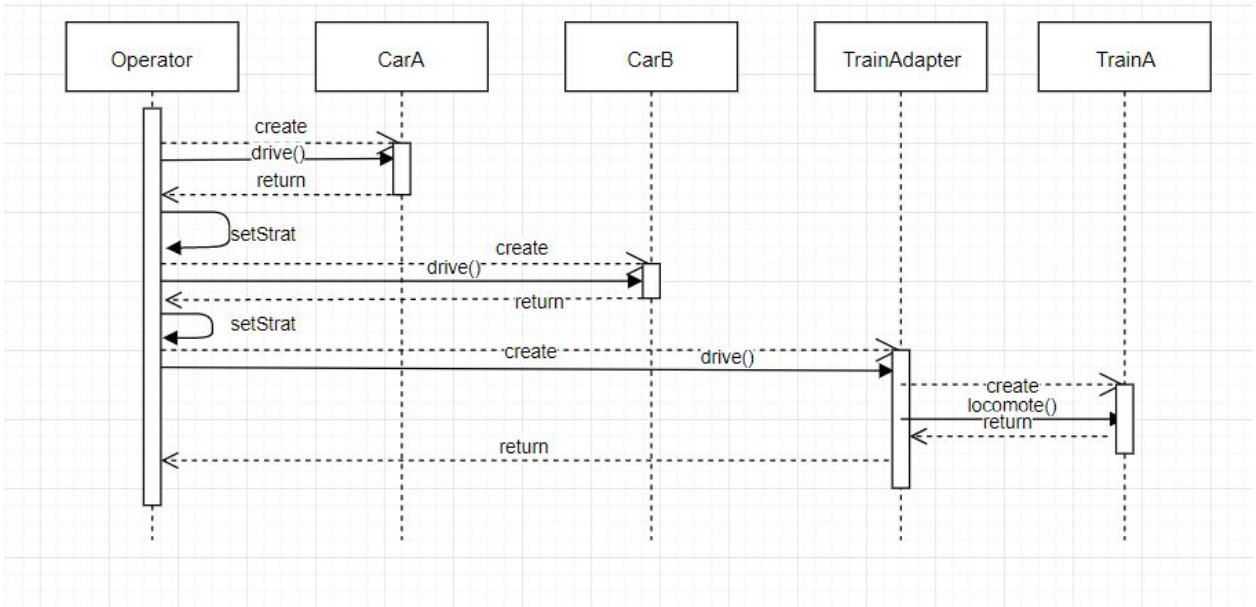
ESOF 322 HW3 Hunter Line, Logan Gilder

Part 1

A)



B)



Part 2

A)

Focus Factor is gained from the first sprint = (32 Actual Velocity*)/(45 Available Man-Days*)

Estimated Velocity = 72 Available Man-Days*(32/34) Focus Factor*

Estimated Velocity = 51.2

B)

It can be difficult to give an estimate for a new team, but there are a couple of methods that can help estimate. Set the high priority requirements from the backlog using story points. Break down the highest priority into smaller, more manageable tasks and determine how long in hours each of these smaller tasks will take. Proceed to add up all these hours to see if the team can make this part of the project work within the given time. Keep doing this cycle until the team reaches its full capacity.

C)

One idea we had for estimating story points was to extend upon the poker method. We thought that the workers should post their estimates anonymously to a poll at first. People would also be able to reply anonymously. A second wave of anonymously polling would happen 15 minutes later in case people changed their mind. Shortly after the second poll, people would meet up in a room and discuss from there.

We thought this method would be a little nicer because it gave people the chance to post their estimates without having someone come crashing down on them.

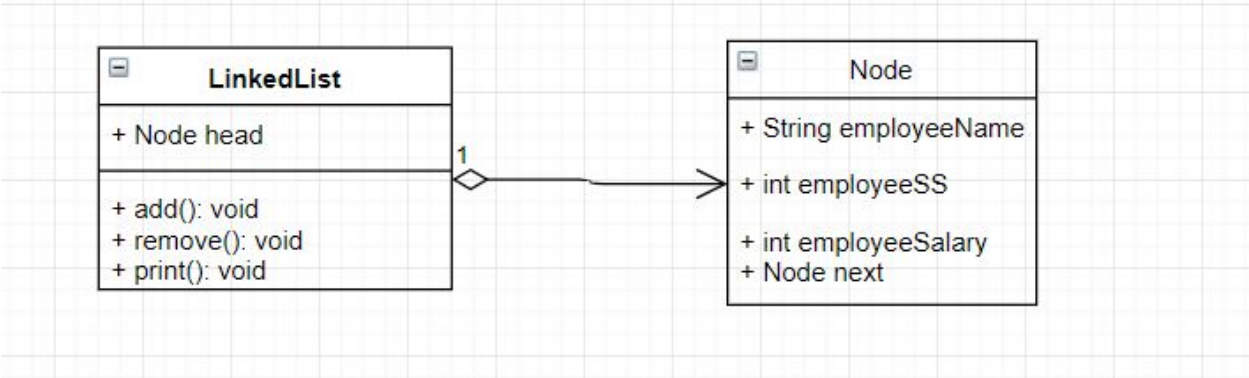
D)

E)

```
package BinaryTree;
import java.util.Random;
public class BinaryTree {

	Node root;

	public BinaryTree() {
	}
	public static void main(String[] args) {

		BinaryTree tree = new BinaryTree(); // No node is made when initialized.
		tree.add(5);
		tree.add(7);
		tree.add(56);
		tree.add(-2);
	}
	public void add(int value) {
	   root = recursionPlacement(root, value);
	}
	public Node recursionPlacement(Node current, int i) {
		if (current == null) {
			System.out.println(i + " Has been made");
			return new Node(i);
		}
		Random rand = new Random();
		int decision = rand.nextInt(2);
		 if (decision == 0) {
			System.out.println(i + " going left");
			current.left = recursionPlacement(current.left, i);
		} else if (decision == 1) {
			System.out.println(i + " going right");
			current.right = recursionPlacement(current.right, i);
		}
		else {
			return current;
		}
		return current;
	}
}
```

```
package BinaryTree;
public class Node {
	int number;
	Node right;
	Node left;

	public Node(int data) {
		this.number = data;
		left = null;
		right = null;
	}
}
```

F)



G)

```
package LinkedList;
public class LinkedList {
	Node head;
	public LinkedList() {
	}
	public static void main(String[] args) {
		LinkedList list = new LinkedList(); // No node is made when initialized.
		list.add("Jerry", 2, 3);
		list.add("Tom", 7, 9);
		list.add("Gary", 7, 9);
		list.print();
		System.out.println();
		list.remove();
	}


private void remove() {
```

```
		if (head == null)
			return;
		head = head.next;
	}
	private void print() {
		Node current;
		current = head;
		if(head == null) {
			System.out.println("Nothing Here");
			return;
		}
		while (current.next != null) {
			System.out.println(current.employeeName);
			current = current.next;
		}
		System.out.println(current.employeeName);
	}
	private void add(String string, int i, int j) {
		if (head == null) {
			this.head = new Node(string, i, j);
		} else {
			Node current;
			current = head;
			while (current.next != null) {
				current = current.next;
			}
			current.next = new Node(string, i, j);
		}
	}
}
package LinkedList;
public class Node {
	String employeeName;
	int employeeSS;
	int employeeSalary;
	Node next = null;
	public Node(String name, int sS, int salary) {
		this.employeeName = name;
		this.employeeSS = sS;
		this.employeeSalary = salary;
	}
}
```