Kajia Coziahr & Logan Dolechek

Exercise 2: ESOF_HW1

(a) Estimated Velocity = Available Man Days x Focus Factor

Focus Factor = $\frac{Actual\ Velocity}{Available\ Man\ Days}$

Focus Factor = $\frac{32}{45} \approx 71\%$
(last sprint)

Now add 2 Engineers

Name	Available Hours
Tom	15
Adam	15
Cary	15
Lisa	**15**
Sam	**12 (can only work 80% of time)**

New Available Man Days = 72

Estimated Velocity for new Sprint = 72 x .71 ≈ **51 Story Points**

(b) The focus factor formula can be found in the problem above. However, in order to use this method, the previous velocity is needed. With a brand-new team, the velocity is not known. Because of this, many companies choose to set their focus factor to 70% until they have more information from their sprints.

(c) Let's consider rock, paper, scissors as a method of finding out how many story points a team would need to finish a project. Like the poker example, everyone would sit around a table and ultimately put down the number of the card they believe it would take them to complete a sprint in work days. This rock, paper, scissors method would work almost the same - Except the group would set time estimates to each hand symbol involved in the game, rather than using cards. You could potentially add additional hand symbols within the game if needed to create a broader spectrum of time. Arguably this rock, paper, scissors method

could be better than the poker method for smaller groups, but would be less efficient in larger groups.

// (d) and (f) UML diagrams are located in attached Draw.io

(e)

```
// Code referenced from CSCI 132 taken at Flathead Valley Community College

class Node
{

	//Node Data here
	public int iData; //key for searching
	public double dData;

	//node requirements
	public Node leftChild;
	public Node rightChild;

	public void displayNode()
	{
		System.out.print('{');
		System.out.print(iData + ", " + dData);
		System.out.print('}');
	}

}

class Tree
{

	private Node root;

	public Tree()
	{
		root = null;
	}

	public Node find(int key)
	{

		Node current = root;

		while (current.iData != key)
		{
			if (key < current.iData)
			{
				current = current.leftChild;
			} //going to the left child cuz it's less than key
			else
			{
				current = current.rightChild;
			} //going to right of key cuz it's greater
			if (current == null)
			{
				return null;
			} //didn’t find it at the bottom of the leaf
		}
			return current;
			//finds the key and it will stop and return the vallue
```

```
	}

	public void insert(int id, double dd)
	{

		Node newNode = new Node();
		newNode.iData = id;
		newNode.dData = dd;

		if (root == null)
		{
			root = newNode;
		}
		else
		{
			Node current = root;
			Node parent;

		while (true)
		{
			parent = current;

			if (id < current.iData)
			{
				current = current.leftChild;
			if (current == null)
			{
				parent.leftChild = newNode;
				return;
			}
			else
			{
				current = current.rightChild;

				if (current == null)
				{
					parent.rightChild = newNode;
					return;
				}
			}
		}
		}

	}
	}

}

	public class Ds17_trees {

		public static void main(String[] args) {

		}

	}
```

(g)

```
//Code reference from DataStructures and Algorithms 232 @ Flathead Valley Community College


class Link
{
	public Link next;
	public double dData;
	public int iData;

	public Link(int anInt, double aDouble)
	{
		iData = anInt;
		dData = aDouble;
	}

}

class LinkList
{
	private Link first;

	public LinkList()
	{
		first = null;
	}

	public boolean isEmpty()
	{
		return(first == null);
	}

	public void insertFirst(int intdata, double doubledata)
	{
		Link newLink = new Link(intdata, doubledata);
		newLink.next = first;
		first = newLink;
	}

	public Link deleteFirst()
	{
		Link temp = first;
		first = first.next;

		return temp;
	}
}

public class ESOF_HW3
{
	public static void main(String[] args)
	{
		LinkList theList = new LinkList();
	}
}
```