Traditional Iterative Implementation	Comparisons	Recursive Implementation

`public static void firmOutput(Worker[] lantrineIndustries){`
 	
`for(int i = 0; i<lantrineIndustries.length-1; i++){`
 	
`if(lantrineIndustries[i].equals(temp[1])){`
 	
`lantrineIndustries[i].workDay();`
`}`
`}`
 	
`}`
	Memory: (think about how many variables and function calls are created)
	Both implementations have the same amount of variables created, and both are integers meaning they'll take the same amount of bits in memory.
	The only argument I see for traditional implementation having a lesser number of instance variables is that the one variable that the traditional iterative implementation has lasts for slightly shorter than the one variable used for the recursive implementation. (It lasts for a count shorter and is initialised 4 lines later than the recursive implementation).
	If we are looking at the amount of time that the system is taxed for rather than how many instance variables exist period at any given time in either implementation, the traditional iterative implementation costs less memory.
		`private static int count = 0;`
		
		`public static void firmOutput(Worker[] lantrineIndustries){`
		
		`if(lantrineIndustries.length > count){`
		
		`lantrineIndustries[count].workDay();`
		`count++;`
		`firmOutput(lantrineIndustries);`
		`}`
		`else{`
		`count = 0;`
		`}`
		`}`

	Lines of Code: Both implementations have the same amount of code. However, that doesn't really respect the metric here: "How difficult is this code to write and read". In that case, the recursive method is a lot more difficult to read. The reason for this is whereas one can look at the for loop and see exactly how long it runs for, one has to manually trace the recursive implementation to see when it will reach its base case. This takes a lot more time and is far more taxing on brain power. It is also a lot more difficult to design a good recursive implementation than a good iterative implementation, making it harder to write. It's not efficient when it comes	

	to reading code. More or less lines of code does not necessarily mean a more or less difficult to run program. However, these two variables are positively correlated to some degree. Though, this information isn’t very useful because they have the exact same number of lines of code.	
	Efficiency: (consider the above two, as well as the number of operations, to give an overall evaluation of efficiency) The number of operations between the two methods is virtually identical. This means that there isn’t a necessarily better way to implement this method (in terms of these two possibilities) to favour processing power. In short, the memory is impacted only slightly less by the iterative implementation because although there are the same number of variables used, the iterative implementation used the variables for less time. The efficiency to read and write the iterative implementation is far better than its counterpart. The efficiency of the computer to go through the lines of code is the same because they have the same number of lines of code. And the processing power is the same on both ends of the spectrum because the number of operations in the two methods is virtually identical. Thus, because of the slight advantage of the iterative implementation in it taking up less memory as well as the advantage to read and write the code, I’m going to have to recommend using the iterative method. My other iterative/recursive overloaded implementation can be found below.	

One of my recursive methods (and it’s iterative version) are down here. The table above is based on my other recursive method.

```
public static void profit(Worker[] IantrineIndustries){

        if(IantrineIndustries.length > count){
            if(IantrineIndustries[count] instanceof Janitor)
            {
                profits = profits + ((Janitor)IantrineIndustries[count]).getProfit();
            }
            else if(IantrineIndustries[count] instanceof Designer)
            {
                profits = profits + ((Designer)IantrineIndustries[count]).getProfit();
            }
            else if(IantrineIndustries[count] instanceof Coder)
            {
                profits = profits + ((Coder)IantrineIndustries[count]).getProfit();
            }
            else
            {
                profits = profits + ((Designer)IantrineIndustries[count]).getProfit();
            }

            count++;
            profit(IantrineIndustries);
        }
        else if(profits < 0.0 || profits > 100000)
        {
            count = 0;
            System.out.println("The firm made $" + profits + " today. This number should be double checked before it is added to the checkbook as
it is unusual for a firm to make this amount of money in a day.");
        }
        else
        {
            count = 0;
            System.out.println("The firm made $" + profits + " today.");
        }
    }

public static void profit(Worker[] IantrineIndustries, String typeOfWorker){

        Worker[] temp = new Worker[1];

        if(typeOfWorker == "Janitor")
        {
            temp[1] = new Janitor("temp");
        }
        else if(typeOfWorker == "Designer")
        {
            temp[1] = new Designer("temp");
        }
        else if(typeOfWorker == "Coder")
        {
            temp[1] = new Coder("temp");
        }
        else
        {
            temp[1] = new SoundDesigner("temp");
        }
        for(int i = 0; i<lantrineIndustries.length-1; i++)
        {
            if(lantrineIndustries[i].equals(temp[1]))
            {
                if(lantrineIndustries[count] instanceof Janitor)
                {
                    profits = profits + ((Janitor)lantrineIndustries[count]).getProfit();
                }
                else if(lantrineIndustries[count] instanceof Designer)
                {
                    profits = profits + ((Designer)lantrineIndustries[count]).getProfit();
                }
                else if(lantrineIndustries[count] instanceof Coder)
                {
                    profits = profits + ((Coder)lantrineIndustries[count]).getProfit();
                }
                else
                {
                    profits = profits + ((Designer)lantrineIndustries[count]).getProfit();
                }

                System.out.println("The firm made $" + profits + " today.");
            }
        }

        if(profits < 0.0 || profits > 100000)
        {
            System.out.println("The firm made $" + profits + " today. This number should be double checked before it is added to the checkbook as
it is unusual for a firm to make this amount of money in a day.");
        }
    }
```