2a. The programming language that I used is Java. The purpose of my program is to be an adaptable inventory management system that could work in almost any field. The program has the capability to add items, edit items, display items, search through items, sell items, view gross income, and delete items. The user's information is stored in various text files, a save-like idea. The main purpose of the program is to help businesses organize and inspect certain inventory elements. By allowing user inputs to be saved and organized, businesses can easily keep track of items and incomes. My video shows the use for the program, and displays some of the functions in action. The video also shows the easy to use GUI and how it is used, and my password/security code that I implemented into our project (at the beginning).

2b. A difficulty that I encountered while writing the PassMan class was the ability to save/store the passwords in a specific place. I knew that I wanted to have the user create a password and have that password be able to work the next time that the user wanted to use the program. What I found is that in Java, I could create a text file into which I could save the user's password. By doing this, I could write another function to check if the text file existed, and if so, ask for the pre existing password. This sparked an opportunity to save all of the necessary components of the project onto text files. This idea worked very well for me. Another difficulty that I had was encoding passwords and security questions into the text file. As I was beginning the project, I realized that it was a terrible idea to write passwords and security questions onto a text file without encoding them, as they could easily be accessed. To solve this, I created a encoding function to encode the password and the security question in the text file. These problems were incorporated in an incremental and iterative way.

2c.

```
static String letters[]={"a","b","c","d","e","f","g","h","i","j","k","l","m","n","o","p","q",
        "r","s","t","u","v","w","x","y","z","1","2","3","4","5","6","7","8","9","0","~","`",
        "!","@",")","#","$","%","^","&","*","_","-","=","+","A","B","C","D","E","F","G","H",
        "I","J","K","L","M","N","O","P","Q","R","S","T","U","V","W","X","Y","Z"," ","?"};
static String shifted[]={"m","l","8","q","A","P","R","W","C","h","n","k","1","7","z","B","o",
        "9","2","g","f","6","p","O","0","a","e","3","j","Q","V","-",")","4","=","b","d","J",
        "N"," ","X","","M","?","I","U","J","i","5","y","c","w","*","T","$","%","S","v","^","L",
        "+","&","Y","u","#","D","_","","r","Z","H","!","s","@","F","t","x","G","E"};

  public static String encode(String msg){
       String output="";
       for(int x=0;x<msg.length();x++){
           output+=shifted[indexOf(letters,msg.substring(x,x+1))];
       }
       return output;
  }
  public static String decode(String msg){
       String output="";
       for(int x=0;x<msg.length();x++){
           output+=letters[indexOf(shifted,msg.substring(x,x+1))];
       }
       return output;
  }
```

The algorithm's purpose in our program is to encode the security question and password. Because this information is saved on a text file and is private, an encode algorithm is necessary to protect this valuable info. The algorithm simply takes the original message and changes it into another encrypted message that is saved on the text file. It takes each individual character from the original input (from the *letters* String) and changes it to the corresponding letter/symbol in the *shifted* String. By doing this, the information saved in the text file gives no indication of the true password or security question. The two algorithms that are involved in the main algorithm are the *encode* algorithm and the *decode* algorithm. The decode algorithm works in the opposite way that the encode algorithm works. It does this by shifting the characters in the text file back to the original input from the user. It does this in the same was as the encode algorithm, but in reverse. With the combination of these two algorithms, I can create a main algorithm that implements both of these algorithms. For example, checking if the existing password is correct.

2d.

```
else if (p.exists()){
    try{

        Scanner scanner3=new Scanner(new File("pmem.txt"));
        System.out.println("Welcome back to Oxi Inventory Management! Please enter your password!");
        oldPass=scanner.nextLine();
        String readPass=decode(scanner3.nextLine());
        if (oldPass.equals(readPass)){
            try{
                new ProcessBuilder("cmd", "/c", "cls").inheritIO().start().waitFor();
            }
            catch(InterruptedException e){
                System.out.println("InterruptedException");
            }
            catch(IOException e){
                System.out.println("IOException");
            }
            Console console=new Console();
        }
        else{
            System.out.println("Password Incorrect.");
            reset();

        }
    }
    catch(FileNotFoundException e){
        System.out.println("Password does not exist.");
    }
catch(Exception e){
    }
    System.out.println("Password incorrect.");
    reset();
```

This abstraction was created so that if the text file *p.txt* exists, the program will automatically begin by asking the user for their password instead of asking the user to create a new one. The purpose of this abstraction is to save the inputs that the user put in the inventory manager if they wish to exit the program. By saving the password, the inventory manager retains all of the user's inputs. This saves the user time and tremendously helps with inventory management, as it will save all of the inputs from the previous session. It helped manage the complexity of the

program by allowing a previous password to be used, rather than needing a new password every time the program needed to be used. The abstraction begins by checking if the *p.txt* file exists. If so, a new text file is created. Then the old password is brought in with a scanner. After this, the user puts in their password. The *if* statement states that if the oldPass and the readPass equal the same input, the program starts. If the password is incorrect, the reset function is called.