Matthew O'Brien

Term Project: *The Task Manager*

Design Document

Table of Contents

1 Introduction

The project is called the Task Manager. Nothing clever or exciting about the name, but the purpose of it becomes very straight forward. This program is designed to help managed a given set of tasks. This is done through given categories that the user can use to better manage a set of tasks, and allow for multiple users to have access to their own set of tasks. Making this program ideal for a group of people within a household. The purpose of this document is to set the framework for the actual program itself. The inner workings of what the Task Manager will do within the scope of the programming itself. This document will give a clear representation of the actual work the program does when being used. There are clear representation for the given classes, their purposes, and how they correspond to the rest of the program. The idea of this document is to show the purpose of each portion of the project, on a programming level, without just showing the code. The document contains the considerations for the design of the project, an simple overview of the system, the project's architecture, and a detailed system design. Each of the portions will give explain the purpose and goals of each portion of the project.

1.1 Purpose and Scope

The focus of this document is to show how the system itself works for the project. Explaining the design, and the architecture of the system. There will be explanation for each portion of the project, giving each class, component, or subsystem a purpose or goal for which they are created. The use of diagrams and graphs will allow for a better understanding of the connections for the system on both the lower and higher levels of the project. Ideally, when after reading through this document, there should be a full understanding on some level of how the project is designed to work, and why it works the way that it does, leaving no useless components of the project.

1.2 Target Audience

The individuals this document is made for are those who will either be working on the project on a coding level, or those who want to have an understanding of how the system works on a higher-level. This document is not designed for someone outside of the actual production of this program. It is possible those not involved with the actual implementation of the project to read through this documentation, at which the figures will be the most helpful to have a better understanding at the purpose of the different aspects of the design. The focus for this audience is to be able to see the design of the system, how each different component is used, and show the architecture of the system.

1.3 Terms and Definitions

Any of the terms or definitions that are used within this document will be explicitly explained or given throughout the document as they are needed.

2 Design Considerations

This portion of the document contains the different considerations and constraints that went into the design of the project. Each of the constraints will be explained as to why they were implemented the way they were, and how they change the actual project itself. The different methodologies that were used for this project will also be explained, why they were chosen, and how they were implemented into the project itself.

2.1 Constraints and Dependencies

There were a few different constrains and dependencies that were considered with the design of the project. The very first has to do with the need for an external file and the user portion of the system. All of the users and passwords for the program will be made prior to use. As well as all the information for the tasks will be saved externally to a file. This does not really allow for much security towards the users and their passwords. It also allows for easy changes to the external files without actually accessing the program itself. There is also limitation from moving the tasks between lists. Each list only has one direction it may move a task to, and only when it reaches the final list does it allow for deletion. And the opposite, the tasks my only be created in the beginning list. The use of external files to account for the saved information and users will be more limiting as well. Each user, password and tasks will be saved to their own file. This will allow for easier implementation of the saving and loading features, but will not allow the user to view multiple accounts at the same time. Each account will be loaded when selected, and able to view. When the user wishes to view another account, the previous account must be saved and closed before the next one will be able for the user to select.

2.2 Methodology

The methodology used for this project is/was modeled after the waterfall model. This model is used in an incremental steps allowing for focus on each portion of the project over the course of the term. This is due to the structure of the course itself. Each portion of this project has scheduled due dates, and requirements that are due at certain times along the term. This structure made the waterfall methodology fit best for the project itself. The waterfall model is split into six general steps for the project. The first, system and software requirements have already been completed. This was done within class when the assignment was actually given to the class, with due dates, and the requirement document. The second part of analysis was split into two portions. The first portion of the business rules and models were included in the requirements document, but models and schemes are also included for this document. The third step is the design of the project. That step is the current one, and will be completed with this end of this document. The forth step is the actual development of the project, or the coding phase. This will be the biggest portion of the project. It is the actual implementation of the project. The fifth step is the testing phase. Another part of the project clearly laid out with the requirements of the project itself from the requirements documents. This phase will come after the project is implemented, and a testing schedule will be made to ensure that the program runs to the required specifications. The final stage is the operations stage. It is the installation, support, and possible migration of the project. This final stage will correspond will the final due date of the project. Though, the stage is an ongoing step. There will be continues maintenance of the project as the program ages.

The object orientated paradigm will be implemented as well for this project. The focus of each object having a purpose and function that interacts with the rest of the project. Certain lists will be derived from a general list, but each performing a more specific task. Each class' purpose and functions will be explained in this document.

3 System Overview

The system itself will be a group of three different lists of information. Each already made account will have three different lists of information that can be accessed through a use of a predetermined password that allows for access to the account. The user will interact with the GUI interface to select which account they want to use, once successfully in the account, the user can create tasks, view tasks, or delete tasks from the three different lists of information. The general set up of each list will be the same, and each list will be derived from a general list class. There will be small difference between each list class to better direct the flow of information from list to list depending on the user's inputs. The very brief diagram below shows the general overview of the program.

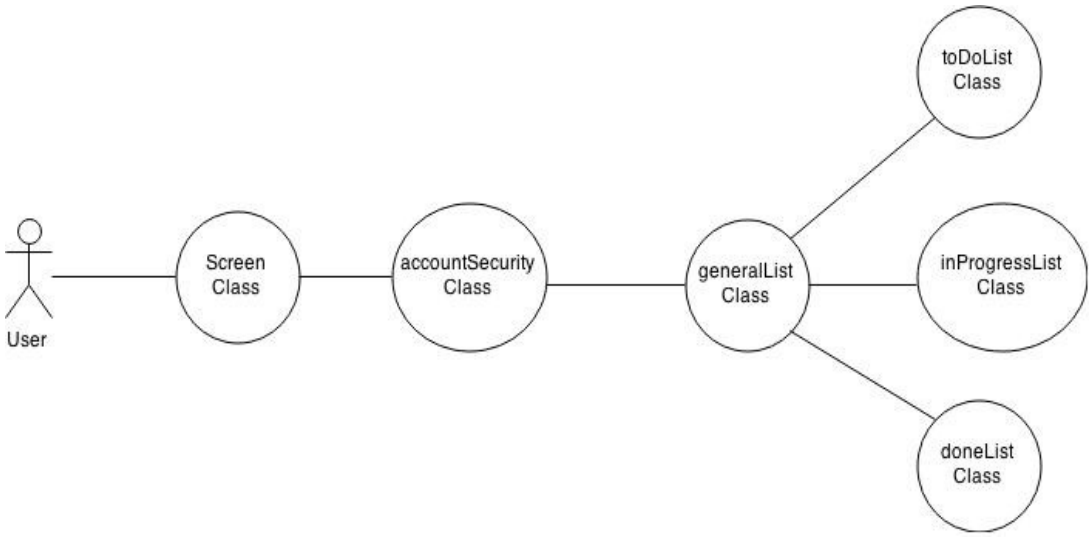

Figure 1. Statechart for user system-level flow.

Figure 1 shows a very simplistic flow of what will happen on a system-level with an interaction from a user. Each one of these classes will be explained later on, and given a purpose towards the project.

4 System Architecture

The system itself is broken down into different possible lists of information that the user can access depending on their inputs. Each class of the lists will be derived from a basic list class, each with a different role and setting depending on what type of task is actually be accessed by the user. There is a general GUI class to provide easier selection for the user, and a security setting to make sure the user can access the account they are trying to use. Refer to Figure 1, for a general idea of the flow of information the user will experience.

4.1 GeneralList Class

The GeneralList class will be used as a base for the three other possible classes that will be derived from it. Each of these classes will perform a more specific task, and have different functionality. This general classes purpose to hold a set amount of information loaded from the AccountSecurity Class, loaded from an external file. The follow table gives the set methods and functionality of the class. There is the connection with the Node Class as well, since it contains the actually information that each list will contain.

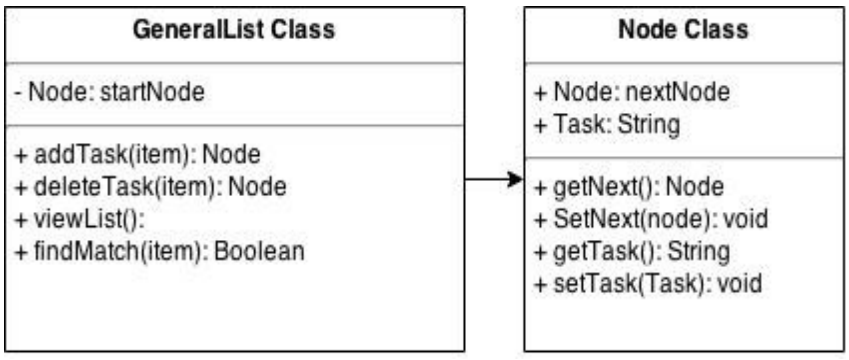

Figure 2. Class Diagram for GeneralList.

This class along with the Node Class, which will be discussed later on, make up the basic framework for the information of the project. They will be doing the majority of the work, which will be implemented into another classes throughout the program. These

two are the classes that handle the tasks themselves, in the different lists. Each different list will be made up of the GeneralList class, and will do something different depending o which class the information is being processed in. These classes will now be explained.

4.1.1 ToDoList Class

The ToDoList class is a class derived from the GeneralList class. The purpose of this class is the contain all of the new tasks. The following flow chart will explain how tasks are implemented in order. This figure will also be referred to later on for sections 4.1.2 and 4.1.3.

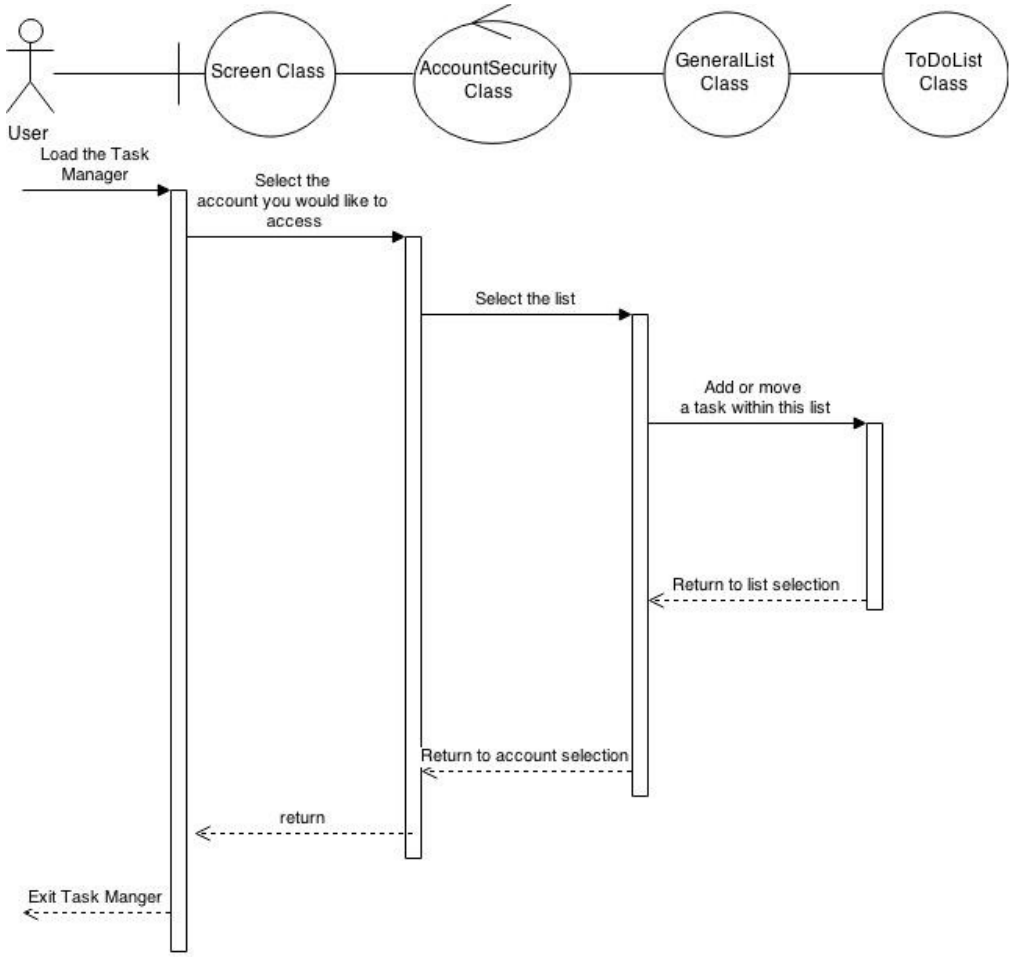

Figure 3. Work Flow for ToDoList Class

Figure 3 shows a very generalized version of the different steps for the flow of information if the users wanted to access the ToDoList Class. Each portion sending you to the next class until the ToDoList is reached. At which point the user is able to edit the list in two ways. Moving or adding to the list. Figure 4 shows a class diagram for the

ToDoList Class. The only method that is not showing is the addTask method. This was previously implemented in the GeneralList Class, and will be used from within that class if the user wants to add a task. This was done because the addTask method will be used in the other classes as well, but not in the same way as this class. That will be explained in the next sections.

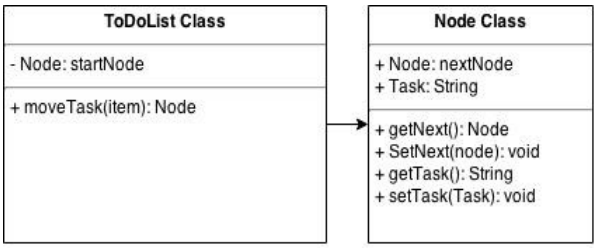

Figure 4. ToDoList Class diagram.

4.1.2 InProgressList Class

The InProgressList class will function very similarly to the ToDoList class. They are both derived from the same GeneralList class, and use the Node class for it's information storage. The general differences are the operations that it will perform. The following flow chart will give a description of how the user will use this class.

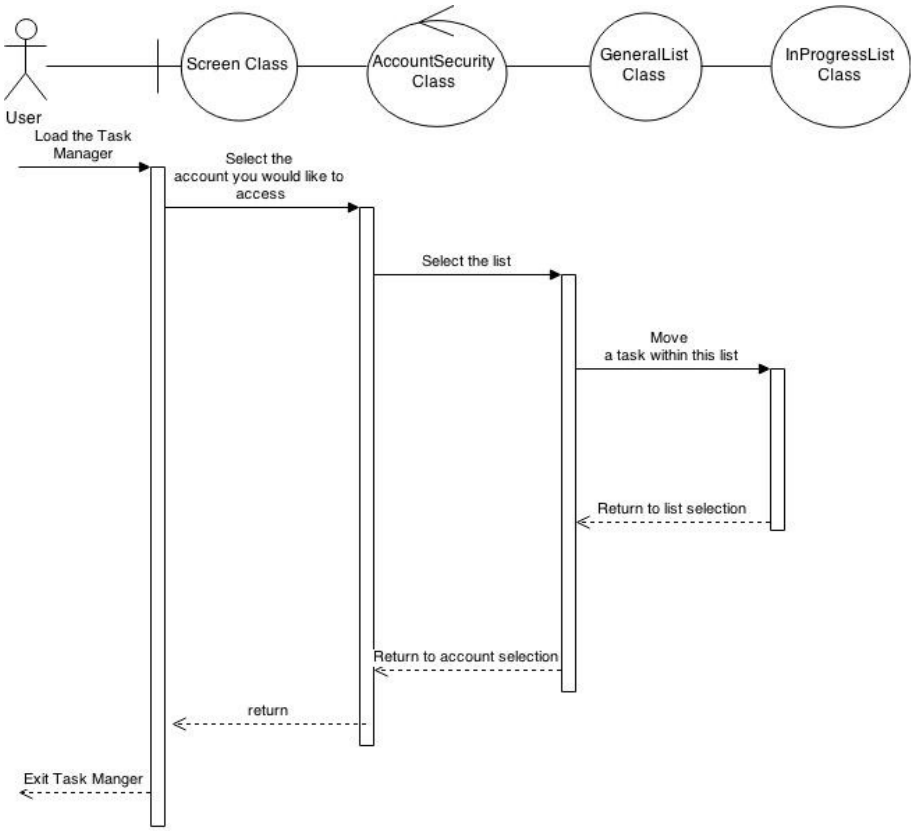

Figure 5. Flow chart for InProgressList

The key difference here is the move functionality of the class. Using the GeneralList's method for addTask and deleteTask, whichever task the user selects to move to the DoneList, will be deleted from this list, but not before using the add function to move the task into the other list. This was set up this way in order to not rewrite nearly the same method for three different classes but instead use the GeneralList class' methods to the advantage of the other three classes. Simply being able to move the nodes from list to list is a simpler implementation. The following figure is the class diagram for the ToDoList class.

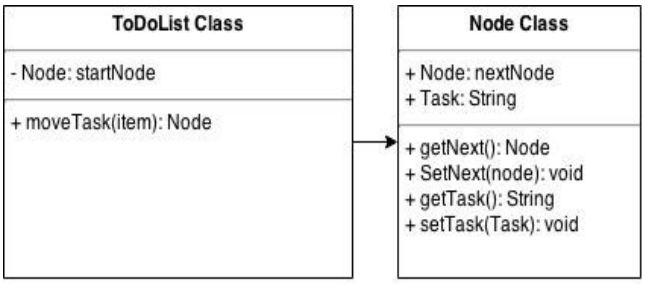

Figure 6. ToDoList Class Diagram

This is the very basic methods of the ToDoList Class with it's connection to the Node class. Again, very similar to the other list classes, but will use the GeneralList class' methods in a different way from the others to better suit the needs of the class.

4.1.3 DoneList Class

The final list class is the DoneList class. This is the final set of tasks the user will have access to in the project. The purpose of this class to contain the tasks that have already been done by the user, as a sort of archive. This will be the only place that the user can full remove the tasks from the lists. This again is from the GeneralList class' method of deleteTask. The next figure shows the class diagram for the DoneList. It has the same connection to the Node class as the other list classes.

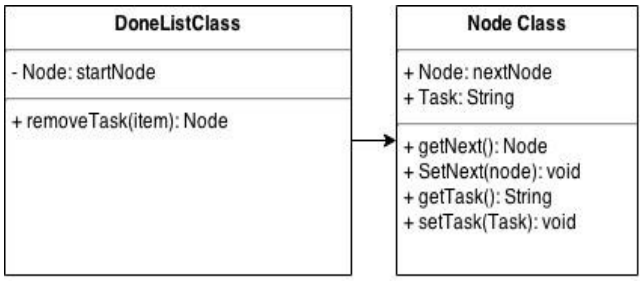

Figure 6. DoneList Class diagram.

This class will have the same functionality as the GeneralList class, but will not be using the addTask in a manner to create new tasks, instead it will only use that to take in a task from the InProgressList class. A general work flow diagram shows the basic flow of information from the user to interact with the class.

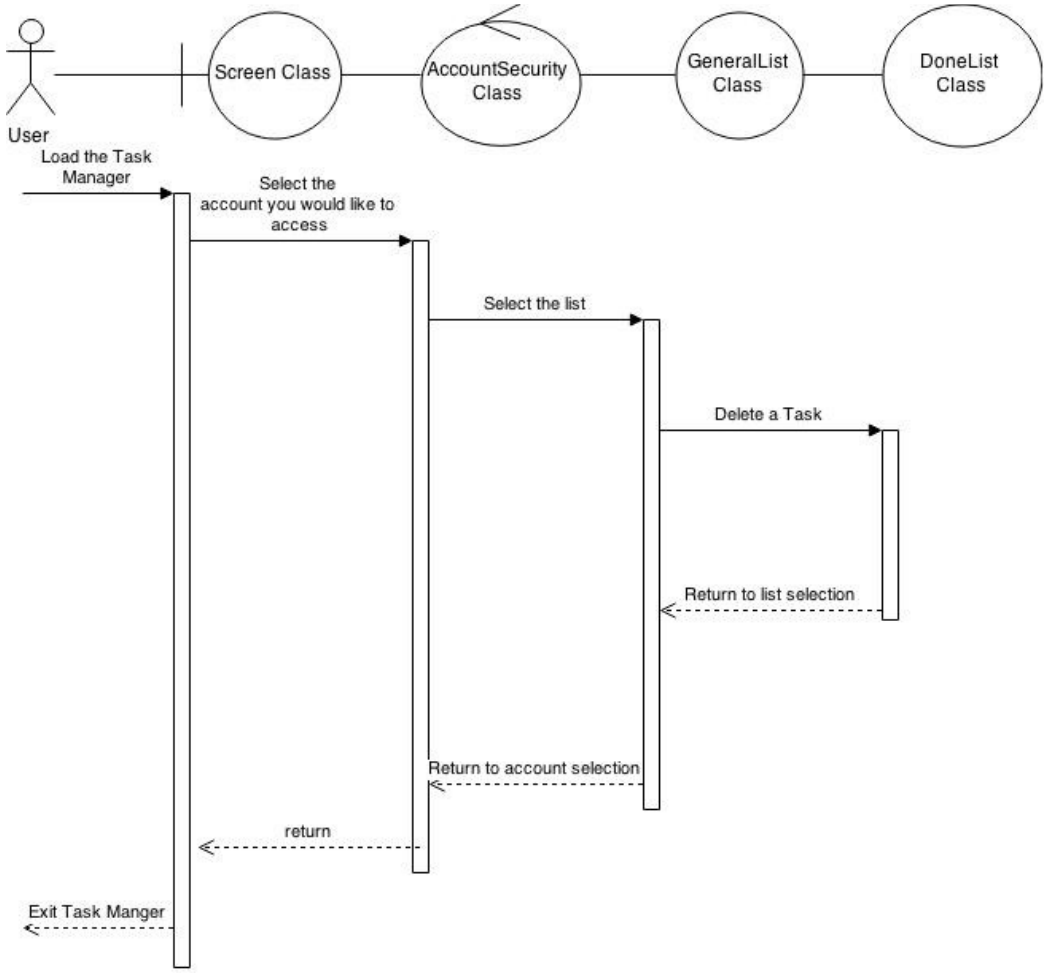

Figure 7. Workflow diagram for DoneList class

4.2 AccountSecurity Class

The AccountSecurity class purpose to allow for the user to select which account they want to access, and make sure they have the appropriate password to use that account. It'll be the general security for the overall project. The following class diagram will explain the AccountSecurity class.

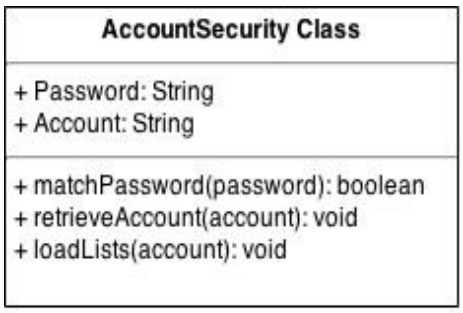

Figure 8. AccountSecurity class diagram.

The general purpose of this class for the security of the accounts, and to select a given account. The user will select an account they want to access, and enter a corresponding password, if they match the account's lists will load, and the user will be able to interact with the other classes. This will also act as the split between accounts. Allowing for the selection between accounts. The following diagram shows a set of steps that this class will make when picking an account.

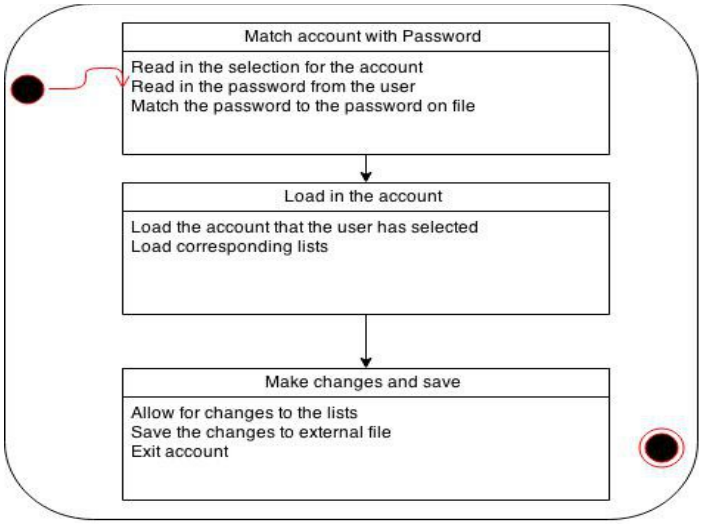

Figure 9. AccountSecurity Flow Chart.

Figure 9 shows a general work flow for that actions of the AccountSecurity class. Since it is both the boundary class for the project, and the handles the load/save

functionality of the project. The majority of the project will happen after the AccountSecurity class, but it handles whether the user will have access to the account, and saves any changes the user makes when they have access to the accounts.

4.3 Screen Class

The Screen class is a general GUI class that will be used through the project in order to create easily usable interfaces for the user to select from when navigating the program. Since this is a simple interface class, there are no need for the class or work flow diagrams as the previous classes. This class is only being implemented to simply the user's interactions with the project itself. Each stage of interaction will have their own interface created through this class, but the general functionality of it will change from stage to stage.

4.4 Node Class

The Node class is a simple storage device for the list. Since each list will handle the same information just in different places, a general object containing all of the information is useful. This class is design to hold the task's information, and allow for each list to easily add to themselves using just the object. The functionality of the Node class are all setters and getters of the information of each task.

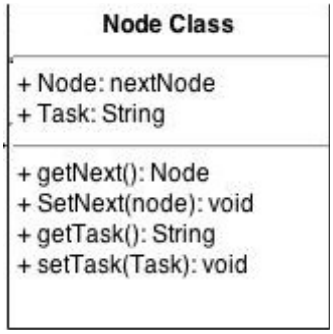

Figure 10. Node class diagram.

As seen in figure 10, the general working of the Node class is very basic, and simply used for the simplicity of the other classes. This object will be easily passed from list to list depending on what the user wants to do. Figures 3, 5, and 7 show the uses of the node classes with the other object.

5 Detailed System Design

5.1 GeneralList Class

The GeneralList class is the base level class for the other three list classes. As seen in figure 2 in contains the general methods that will be utilized by the other classes. The method for addTask is very self explanatory with itself. It is used to add a task to a given list. This method will be reused in different ways by the other classes, but in a general sense, it takes a node object and adds it to the list. For simplicity sake, the objects that are being added will always be added at the front, since no priority system was used for the tasks. The deleteTask method is again very general. It is used to remove a task from one list. This method will relate directly to the findMatch method. FindMatch()'s purpose to take task's name, and match it with one in the list. If there is no match, there is an error, and nothing will be removed. Finally the viewList method is used to see the entire list. These four methods will be used by the other three list classes. Some in different ways, others the exact way explained. Since this class is the general list class, not much information will be needed here, and the actual implementation of these methods will be explained in more detail with the next few classes.

5.1.1 ToDoList Class

The ToDoList class is a derived class from the GeneralList class. This class will manage one type of list for each account. This also makes use of the GeneralList addTask method in a unique way. This class will allow for the user to create a brand new task for this list. This will be the only list that allows for new creation of tasks, where as the other classes only allow for moving and deleting of each task. A general temporary Node object will be created, and the user will fill it with whatever information that want to for that give task. This node will then be passed to the addTask method, and copied into the ToDoList class' list. The other portion of this class will be for the specialized moveTask method. This method is specific to this class, and will utilize the deleteTask from the GeneralList class. The user will select a task that be moved, using the findMatch method from the GeneralList to make sure the task is actually in the list, it will then be copied

into the InProgressList class, and with the GeneralList deleteTask the task will be removed from the ToDoList class. The final method for this class is the viewList, but that is the same as what was explained in section 5.1.

5.1.2 InProgressList Class

The InProgressList is derived from the GeneralList class. It will have the same methods from that class but used in a way specific to this class. The viewList will work the same as previously mentioned in section 5.1. There will be no way for the user to add or delete any of the tasks within this class. Only for the user to move the task to the DoneList class. This is done through the method of moveTask. It will perform almost the same as the functionality as stated in 5.1.1 with the ToDoList, but instead will copy the node object to the DoneList. The removal and addition will happen in the same manner, but the location of the copy will be different as stated.

5.1.3 DoneList Class

The DoneList class is derived from the GeneralList class. It has the same functionality as it's derived class, but a specific way of using the methods. The viewList method will be used the same way as stated in section 5.1, but only to show it's own list of nodes. The difference with this class is that there will be no moving tasks. Once node object as reached this list, the only operation for it is to be removed and completely deleted from the program. This class is an archive for recently finished tasks. It will contain a list of node objects that the user has moved here. No task creation will be allow within this class.

5.2 AccountSecurity Class

The AccountSecurity class is used as a boundary class for the project. It is the point where the user will be allowed to select from the predetermined accounts, and enter a password for that account. The matchPassword method will be used to gain access to the account they want to select. A password will be entered for the given account, and if it matches the string on file, then the user will be given access to that account's lists. This will be done through the loadLists method. Which will load the three lists from an external file for the matching account. Using the three list classes and their node classes

the user will be able to perform the corresponding methods on each of the lists, and have full use of the program for that account. When exiting an account and going back into the AccountSecurity class, all of the information from that account will be saved back onto the external file. All of the information previously on file will be deleted, and the new information that the user had changed will be saved to that external file. Once back at the AccountSecurity class, the user will be able to select from the accounts again, or exit the program. This will be done through an interface generated by the Screen Class.

5.3 Node Class

The Node class is a storage class for the list classes. This class is used to contain the tasks for each list in the project. All of the task's information will be put into this class. This will also be the object that is moved from list to list to allow for a simpler way of adding or removing the tasks from list to list. There is no real need for great detail with this class, since it is only used for a storage object of the tasks.

5.4 Screen Class

This class is the general GUI class for the project. It will create the interface that the user has to make it's selections for accounts, passwords, lists, and tasks. The purpose of this class is to allow for a simpler way of interaction with the user. There won't be much of functionality of the project here. Most of this is for the benefit of the user.