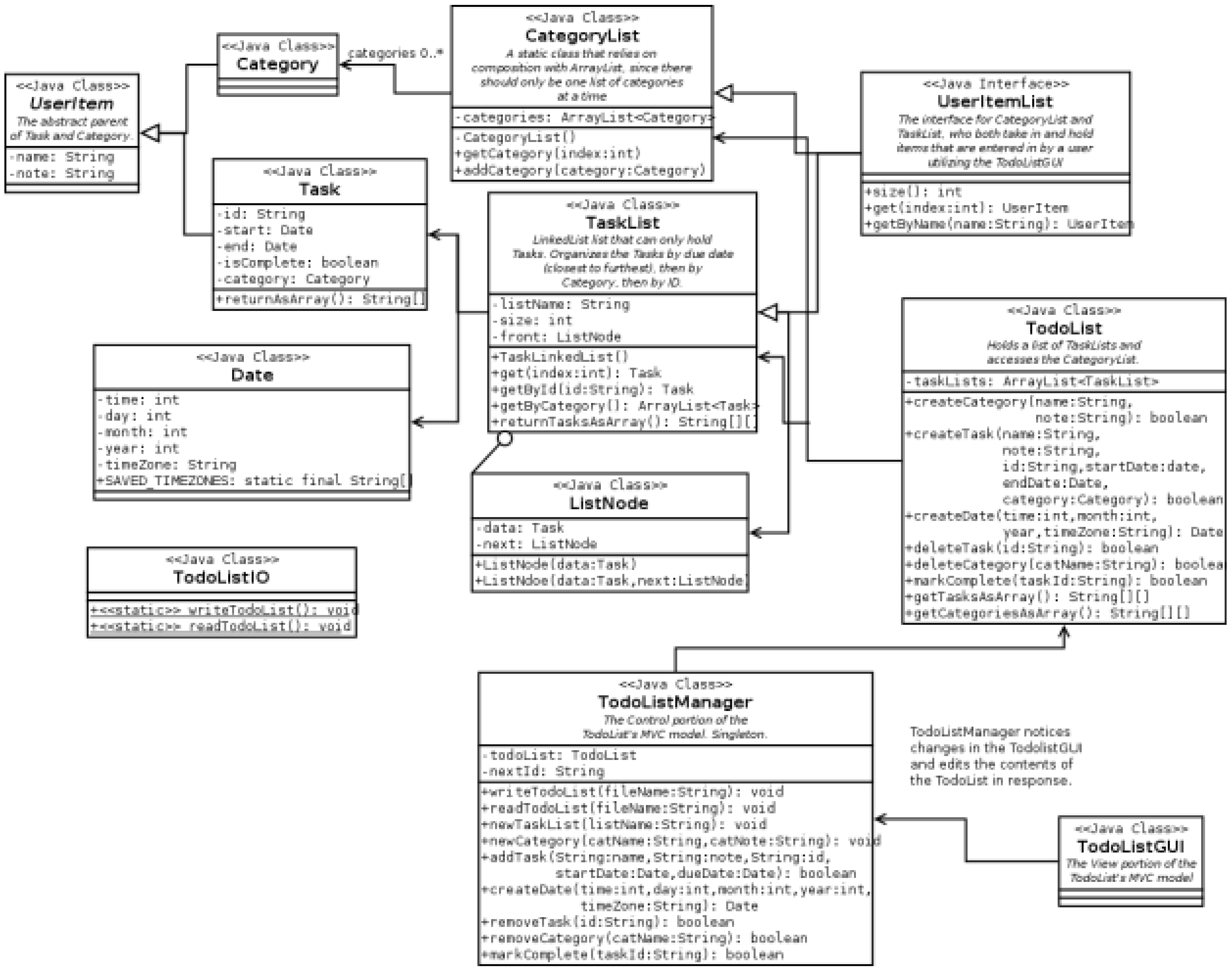
ToDoList Design Proposal

Document Author(s): Christian Byrnes

Date: November 10, 2016

Design Rationale

- What objects are important to the system's implementation and how do you know they are important?
    a. The most important objects to the design are most certainly the lists (CategoryList and TaskList), TodoList (which holds both of those lists), and TodoListManager (which relays the happenings in the TodoListGUI to the TodoList). Without these crucial parts of the list it wouldn't be able to exist, as there would be nothing to store and manage all of the Tasks and Categories the user inputted items of the TodoList program.
- What data are required to implement the system and how do you know these data are needed?
    a. The date required to implement the system, after the user begins to use the program, is the Task and Category object contents. These items are the base level of the data that the user can manipulate when using the TodoList program to take care of their various to-do items.
- Are the responsibilities assigned to an object appropriate for that object's abstraction and why?
    a. Yes- there are no methods assigned to a class that seem inappropriate. They are all able to manage themselves, and provide enough information about themselves to other classes that use them.
- What are the relationships between objects (such as inheritance and composition) and why are those relationships are important?
    a. There is a composition relationship between TodoList and TodoListManager; the manager uses the TodoList to store and manage the items handed down to it from the GUI.
    b. Task and Category are children of the abstract class UserItem, which can simplify how user-entered data is transformed into a Category or Task after being given through the GUI
    c. CategoryList and TaskList both implement the UserItemList interface.
- What are the limitations of your design? What are the constraints of your system?
    a. One of the actual limitations to my design, actually, were the requirements requested by the project brief. It required that we have at least one interface, one abstract class, etc etc. I believe that I could've compressed the classes down further if it weren't for the excess of requirements, though I know this is just to show that we know how to properly apply these ideas to our code and design.



<<Java Class>> UserItem
The abstract parent of Task and Category.
-name: String
-note: String
<<Java Class>> Category
categories 0..*
<<Java Class>> CategoryList
A static class that relies on composition with ArrayList, since there should only be one list of categories at a time
-categories: ArrayList<Category>
-CategoryList()
+getCategory(index:int)
+addCategory(category:Category)
<<Java Interface>> UserItemList
The interface for CategoryList and TaskList, who both take in and hold items that are entered in by a user utilizing the TodoListGUI
+size(): int
+get(index:int): UserItem
+getByName(name:String): UserItem
<<Java Class>> Task
-id: String
-start: Date
-end: Date
-isComplete: boolean
-category: Category
+returnAsArray(): String[]
<<Java Class>> TaskList
LinkedList list that can only hold Tasks. Organizes the Tasks by due date (closest to furthest), then by Category, then by ID.
-listName: String
-size: int
-front: ListNode
+TaskLinkedList()
+get(index:int): Task
+getById(id:String): Task
+getByCategory(): ArrayList<Task>
+returnTasksAsArray(): String[][]
<<Java Class>> Date
-time: int
-day: int
-month: int
-year: int
-timeZone: String
+SAVED_TIMEZONES: static final String[]
<<Java Class>> ListNode
-data: Task
-next: ListNode
+ListNode(data:Task)
+ListNdoe(data:Task,next:ListNode)
<<Java Class>> TodoListIO
+<<static>> writeTodoList(): void
+<<static>> readTodoList(): void
<<Java Class>> TodoList
Holds a list of TaskLists and accesses the CategoryList.
-taskLists: ArrayList<TaskList>
+createCategory(name:String, note:String): boolean
+createTask(name:String, note:String, id:String,startDate:date, endDate:Date, category:Category): boolean
+createDate(time:int,month:int, year,timeZone:String): Date
+deleteTask(id:String): boolean
+deleteCategory(catName:String): boolean
+markComplete(taskId:String): boolean
+getTasksAsArray(): String[][]
+getCategoriesAsArray(): String[][]
<<Java Class>> TodoListManager
The Control portion of the TodoList's MVC model. Singleton.
-todoList: TodoList
-nextId: String
+writeTodoList(fileName:String): void
+readTodoList(fileName:String): void
+newTaskList(listName:String): void
+newCategory(catName:String,catNote:String): void
+addTask(String:name,String:note,String:id, startDate:Date,dueDate:Date): boolean
+createDate(time:int,day:int,month:int,year:int, timeZone:String): Date
+removeTask(id:String): boolean
+removeCategory(catName:String): boolean
+markComplete(taskId:String): boolean
TodoListManager notices changes in the TodolistGUI and edits the contents of the TodoList in response.
<<Java Class>> TodoListGUI
The View portion of the TodoList's MVC model

Figure 1: Class Diagram for TodoList

Document Revision History

Date	Author	Change Description
11/10/16	Christian Byrnes	• First iteration of design proposal