Software Design Document (SWDD)

Widgie: A Productivity and Organization App

(Group 16: Melissa Pinto, Melisa Hayalioglu, Tanisha Mehta, Jordan Angst)

Date: (09/12/2021)

Table of Contents

1.0 INTRODUCTION

This is the software design document for Widgie. The document will outline the software design, specifications, user interface, and workflow task management, and particularly focus on system architecture, system components, and software requirements as agreed upon by the project team.

1.1 Purpose

This project will be called the Widgie, an application that will host and manage data and information inputted by the user. This Software Design Document will provide an understanding of how the system is designed and built, at both the high-level architectural design and the detailed decomposition description. This SDD provides the necessary information of details and requirements of the software and system that is to be built.

1.2 Scope

The app will organize and manage student course information, using the concepts of multithreading and producer-consumer architecture. This software design Document is focused on the critical parts of the course management system, building out the necessary objects and their interactions. This document will specify the structure and the modules that are being built for the system. This design document hosts information on the narrative documentation of the software design using subsystem models, UML diagrams, object behaviour models, hierarchy diagrams, class and sequence diagrams, and other supporting information.

2.0 SYSTEM OVERVIEW

Widgie helps a user manage their schoolwork and classes through multiple organizational tools. Users are able to get easy access to their course information, store notes, add tasks to a to-do list, and much more. The Widgie application should house all the ‘widgets’ on one main screen - the to-do list, the notepad, and the calculator. When the application is opened, the user will be able to access and use any widgets as they need. Widgie is designed to enhance student productivity; since all the widgets are on one single screen, the user will not need to exit out of the application in order to access anything they may need. By allowing the user to stay on one application, it reduces the chance of distractions. In order to supplement the organizational tools and focus on user productivity, the application should be clean, simplistic, and minimal in design. This will allow for minimal distractions and allow the user to focus on their goals and tasks.

3.0 SYSTEM ARCHITECTURE

3.1 Architectural Design

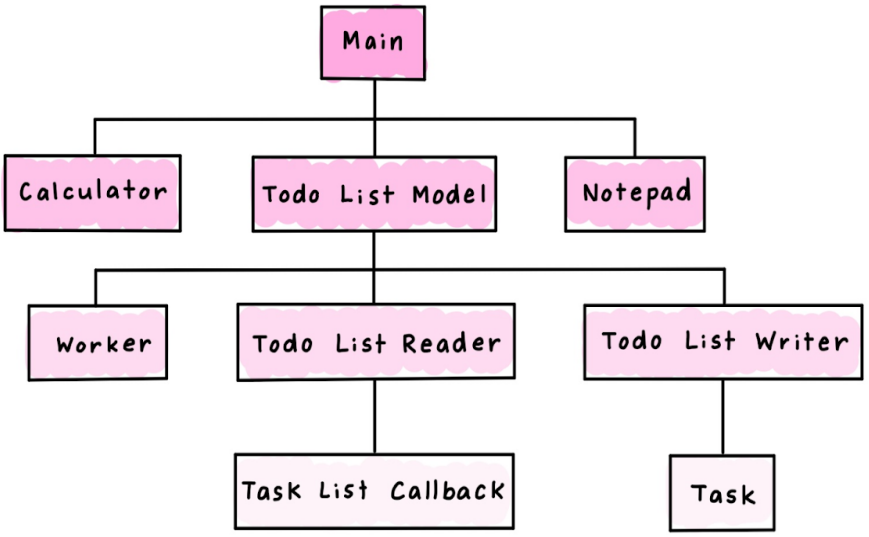


Figure 1: Hierarchy Diagram showing the major subsystems and the relationship between the individual parts. All of the listed subsystems are different classes, and the lines represent a hierarchical structure between each.

The system is broken into three major modules: Calculator, Todo List Model, and Notepad. Calculator is the module that will contain the calculator GUI and functionality. The calculator should be able to handle basic arithmetic operations. The Todo List Model will contain the GUI for running the Todo List functionality. The module should be able to house and store information for a to do list for the user. The Notepad class module will contain all functionality and GUI necessary to run the notepad. The notepad should function to allow the user to both access previously saved notes and store new notes. The Todo List Model is connected to three other classes: Worker, Todo List Reader, and Todo List Writer. Each of these will be involved in the multi-threaded processes when the user creates a new task in the to do list. The Todo List Reader is then going to be connected to the Task List Callback, which houses the previously stored information in the to do list so the user can access items they previously added to the to do list. The Todo List Writer class is connected to the Task class, which implements the auto-save mechanism of the app. Each time a user updates the to do list, the Todo List Writer class writes the changes to a JSON file. The worker class is where multithreading is primarily run, where new threads are created and run through a queue based on scheduled decisions.

3.2 Decomposition Description

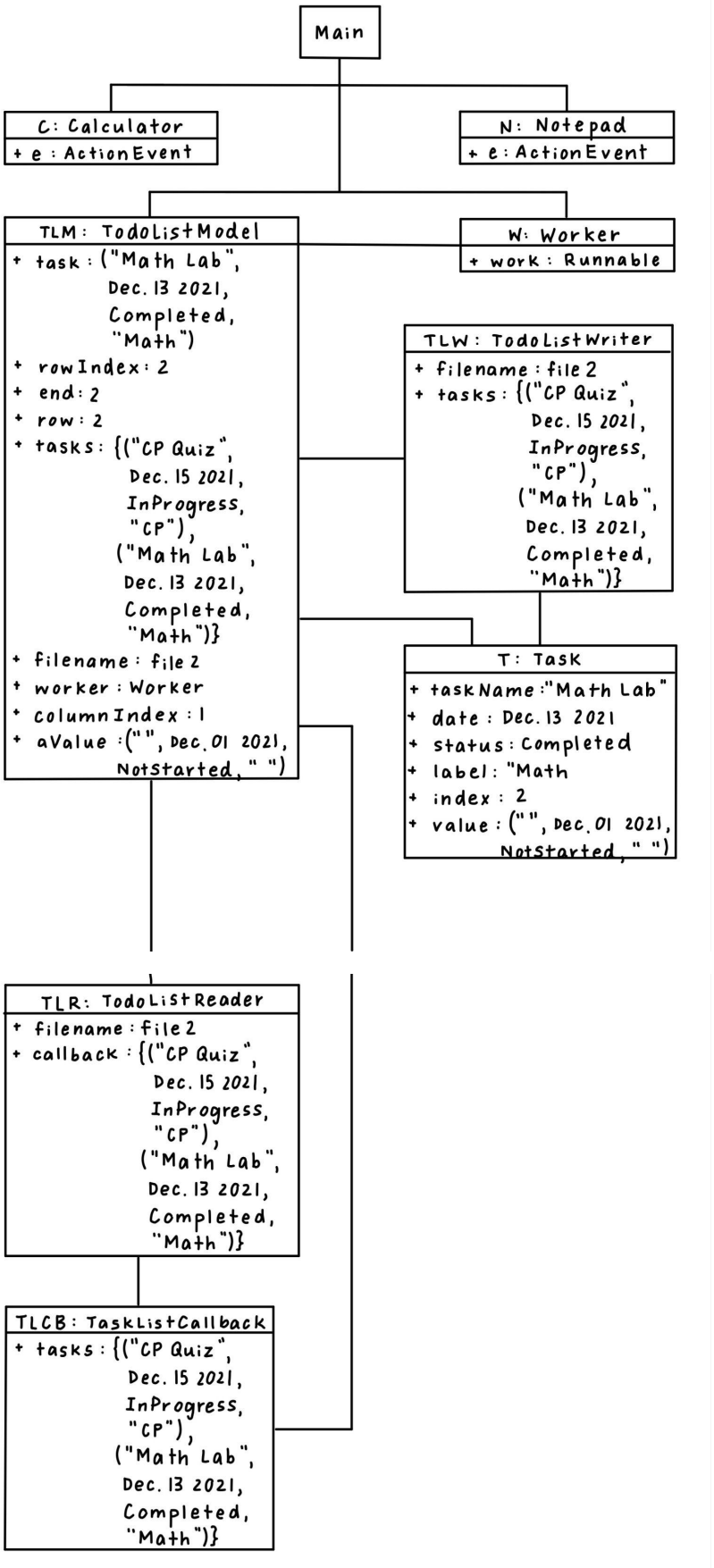


Figure 2: Object diagram describing the different objects, describing the relations by the different

classes and objects in the system. Each object is labelled with a descriptive title of its role in the software, methods, and attributes and information about the object. Some examples of information are also given in the diagram.

Class Name: Main.java	
Brief description: Main.java will be the executable file for the program. It will contain objects and references to all the different classes for each of the widgets that the user must have access to.	
Attributes (fields)	**Attribute Description and Program Description Language**
Worker worker;	This is a declaration of a Worker object to be used throughout main. It creates a worker thread. Worker worker = new Worker();
TodoListModel model;	This is a declaration of a Todo List Model object, which will be displayed on the JFrame that is seen by the user. TodoListModel model = new TodoListModel(todoListFilename, worker);
TodoListPage todoPage;	This is a declaration of the Todo List Page, which will be displayed on the JFrame that is seen and interaced with by the user. TodoListPage todoPage = new TodoListPage(model, worker);
JFrame frame;	This is a declaration of the JFrame object, which is where the application will be displayed on. This is part of the application GUI. JFrame frame = new JFrame();
JLabel lblTodoList;	This is a declaration of the JLabel that is part of the application GUI. It will display the image and the information that is stored within the To do list widget. JLabel lblTodoList = new JLabel("To-Do List");
JLabel lblNewLabel;	This is a declaration of the JLabel that is part of the application GUI. It will display the image and the information that is stored within the notepad widget. JLabel lblNewLabel = new JLabel("Notepad");
Notepad notepad;	This is a declaration of the Notepad. It will display on top of the JFrame, and the user will be able to interact with it. Notepad notepad = new Notepad();
Calculator calculator;	This is a declaration of the Calculator. It will display on top of the JFrame, and the user will be able to interact with it. Calculator calculator = new Calculator();

File f;	This is a declaration of the file. This file stores the logo for the Widgie app, and will be displayed on the app. File f = new File("Widgie_Logo.png");
JLabel logoLabel;	This is a declaration of the JLabel that will actually display the logo on the Widgie app. JLabel logoLabel = new JLabel(new ImageIcon(new ImageIcon(f.getName()).getImage().getScaledInstance(112, 112, Image.SCALE_DEFAULT)));

Table 1: Object descriptions for the class main.java

Class Name: Task.java	
Brief description: Task is the file that contains the constructors and methods for when the user wants to add a new task to their To Do List in the Todo List Model. This class extends the Object class.	
Attributes (fields)	**Attribute Description and Program Description Language**
new Task("", new Date(), Status.NotStarted, "");	This is a declaration of a new task. new Task("", new Date(), Status.NotStarted, "");

Table 2: Object descriptions for the class task.java

Class Name: TaskListModel.java	
Brief description: TaskListModel.java is the class that has the constructors and methods for the task list model that the user will interact with.	
Attributes (fields)	**Attribute Description and Program Description Language**
TaskListCallback cb;	This is a declaration of a new task list call back, inherited from tasklistcallback.java. It will be used to read from the file of saved tasks. TaskListCallback cb = new TaskListCallback()
TodoListReader reader;	This is a declaration of a new to do list reader object. This will read from the to do list in the load from file method. TodoListReader reader = new TodoListReader(filename, cb);
TodoListWriter writer;	This is a declaration of the a to do list writer object. This will be used to write a new task. TodoListWriter writer = new TodoListWriter(filename, tasks);

Table 3: Object descriptions for TaskListModel.java

Class Name: TodoListReader.java	
Brief description: This class will handle taking in task input from the user and reading it from to do list.	
Attributes (fields)	**Attribute Description and Program Description Language**
List<Task> tasks;	This is a declaration of a new task list. This list will be used to read the pre-existing tasks that the user has created. List<Task> tasks = new ArrayList<Task>();
StringBuilder builder;	This is a declaration of a new string builder. It will be used to store the input from the user. StringBuilder builder = new StringBuilder();
File myfile;	This is a declaration of a new file. This file is where the user's tasks will be stored. File myfile = new File(filename);
Scanner keyboard;	This is a declaration of a new scanner object. It will capture the user input. Scanner keyboard = new Scanner(myfile);
JSONObject obj;	This is a declaration of a new JSON object. The JSON file will store to-do list items. JSONObject obj = new JSONObject(builder.toString());

Table 4: Object descriptions from the class TodoListReader.java

Class Name: TodoListWriter.java	
Brief description: TodoListWriter is a class that will take tasks from the reader to add to the to-do list.	
Attributes (fields)	**Attribute Description and Program Description Language**
JSONArray todoList;	This is a declaration of a JSONArray, which will be used in writing the tasks to a string. Tasks will be appended to this array. JSONArray todoList = new JSONArray();
JSONObject obj;	This is a declaration of a new JSON Object, which will hold new to do items. JSONObject obj = new JSONObject();

FileWriter writer;	This is a declaration of a new file writer item. This object will be converted into a string to display on the to do list model. FileWriter writer = new FileWriter(filename);

Table 5: Object descriptions for the class TodoListWriter.java

Class Name: Worker.java	
Brief description: Worker is going to handle the producer-consumer architecture and multithread handling of the program.	
Attributes (fields)	**Attribute Description and Program Description Language**
t = new Thread(this);	This is a declaration of a new thread. t = new Thread(this);

Table 6: Object descriptions for the class Worker.java

Figure 3: Subsystem model describing the various classes for each of the different modules. The different attributes and methods are listed in each subsystem.

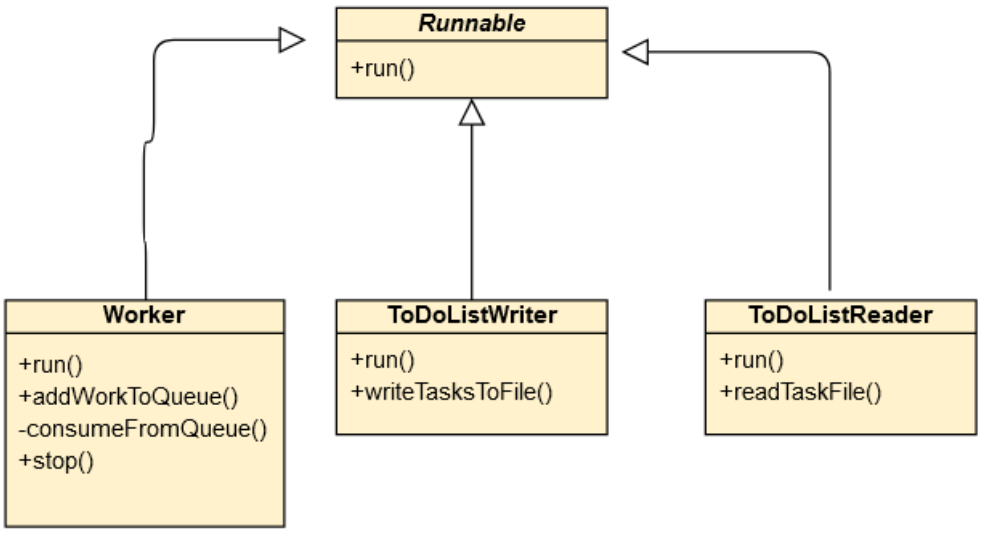
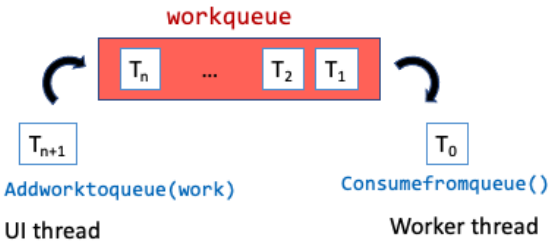


Figure INSERTNUMBERHERE: diagram showing inheritance relationship and polymorphism between the Runnable interface and classes: Worker.java, ToDoListWriter.java, and ToDoListReader.java

4.0 DATA FLOW

4.1 Data Description

A queue is used to store work items (a work item is something that implements Runnable) based on insertion order. It acts as a shared resource between the UI thread and the worker thread. There are then critical sections in the code– methods protected by 'synchronised' that read or write to this resource.



We use JSON to store each to-do list Task as key-value pairs. Here, a Task has attributes corresponding to the to-do list columns: "date", "to-do-task", "label", "completion status". Each Task object then corresponds to a specific instance (a row) in the to-do list. Using JSON to store information in this way helps implement the 'auto-save' feature of the to-do list. To clarify, each time the user adds, changes a to-do task to the to-do list, the JSON file is updated to mirror the changes. If no such JSON file exists (such as when a new user adds a to-do task for the first time), then one is created. Similarly, whenever the user removes a task from the to-do list, the JSON file is updated to reflect this change. Furthermore, each entry in the JSON file corresponds to a single Task so parsing the file and converting entries into Task objects to be visualised in the to-do list is simple and is handled by the program each time the user loads the application.

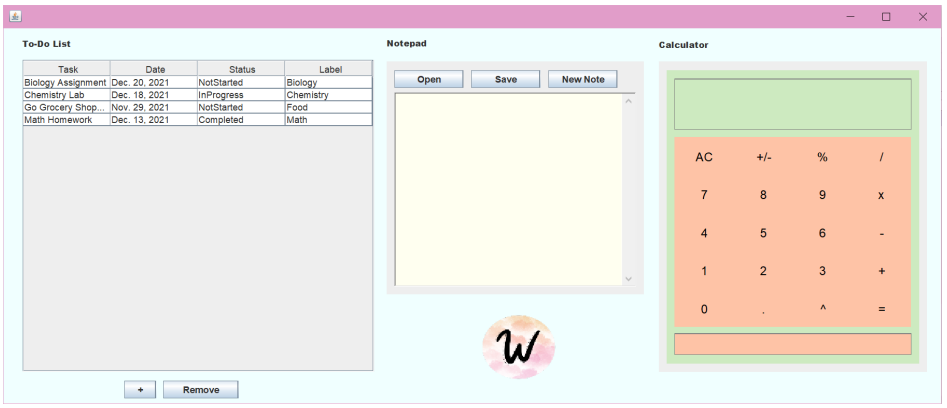
To elaborate on the point above, once the program extracts a task from the JSON file, it creates a new Task object and stores each Task object in a List. The List of tasks, List<Task> tasks is a means to organise and to temporarily (while the application runs) store the Task objects of the to-do list. While the user sees a to-do list much like a table on the application / UI side, this information is actually stored in the Task List on the backend. Since Lists allow easy access to elements and are mutable, each time a row in the to-do list is updated, the corresponding Task in the List is also modified. This Task List is also what is accessed to read and write to a JSON file each time the application is opened, closed or the to-do list is updated.

Notes created on the Notepad are saved using the JFileChooser which is part of the Java Swing Package. Using the showSaveDialog function, text written to the Text Area is saved to a .txt file named by the user using the dialogue box. By default, files are saved to the "Documents" folder of the user's Desktop. Using the showOpenDialog function the user is able to access the files they had saved. When the selected file is opened, it is written back to the Text Area.

5.0 HUMAN INTERFACE

5.1 Overview of User Interface

Describe the functionality of the system from the user's perspective. Explain how the user will be able to use your system to complete all the expected features and the feedback information that will be displayed for the user.



The interface is user-friendly, simple, and pleasant to the eye. The To-Do List is organized with column headers, easy to read colours, and with the "+" and "Remove" buttons users will have no issue adding to the To-Do List and removing old tasks from the To-Do List. The date column comes formatted when adding a new task for easy editing and the status column provides a dropdown selection for the prebuilt options for the column. The label column provides even more functionality by allowing the user to add a label to each task to organize them in a personal way. On top of all of that, the user can sort the whole table by just clicking the column header of choice to sort by that column; this application becomes apparent just by hovering the mouse over the header.

The Notepad takes the form of a sticky note for recognizability sake and implication. With the three large buttons at the top of the note, the user will have no trouble appropriately using the Notepad. The "Open" button implies opening a previously written note, the "Save" button implies saving the note just written, and the "New Note" button implies starting a new

note to write, all of which buttons do as they imply. These buttons have been added to make this process quick, easy, and simple for the user, all of which they accomplish.

The Calculator is elementary, straightforward, and convenient. The layout is typical of any calculator so the interface is familiar. The few buttons have been chosen by predicted frequency for compact but useful application. The boxes above and below the typing pad convey coordinated output based on input from the user. The top box relays what was just entered by the user back to them to ensure the correct number was inputted and at the end of the calculation provides the final answer to two decimal places for accuracy. The bottom box prints the whole inputted calculation and final answer to once again insure the desired data was inputted correctly. The interface plans to prevent user error by aiding the user in seeing their mistakes.

6.0 REFERENCES