Classes:
The classes and their descriptions are important to the overview because it allows the user to see what each class's role is in the program and how each class is going to fit in the program.
Main - A class that starts and runs the program
GUI - A class that creates a visual representation of the program and allows the user to interact with the program in order to use it
ChoreComponent - A class that represents a Chore object graphically and works with GUI to create the overall UI representing the chore of the program
Time - A class that represents a time of a chore
Split - represents a split in a Time object
SplitList - A class that stores all the times of a split and is used in getBestSplit() and getWorstSplit() of the ChoreMath class
Chore - A class that represents a chore and stores the chore's times
ChoreMath - A class that has all the methods to analyze the times of chores
FileFormatter - A class that would format a file and store any information regarding the program on a file
Stopwatch - A class that would represent a real world stopwatch and is useful in setting times of chores
InvalidSplitException - A class that throws a custom exception for when an incorrect split time has been added to a Time

UML Diagrams for each class
The UML diagrams are important to the design overview because they allow the user to see how each class is made and what its role is in the program. The UML diagrams show what each class is going to have and what they are going to do and it shows the building blocks of the program.

GUI
- frame: JFrame - card: cardLayout - panel: JPanel - button: JButton - addChore: JButton - field: JTextField - removeField: JTextField - arr: ArrayList<Chore>
+ actionPerformed(ActionEvent e): void

ChoreComponent
- button: JButton - stopwatch: JButton

- split: JButton
- time: JLabel
- bestSplit: JLabel
- recommendation: JLabel
- average: JLabel
- percent: JLabel
- standardDeviation: JLabel
- field: JTextField
- s: Stopwatch
- counter: int
- arr: ArrayList<Split>
- listChores: ArrayList<Chore>
- stopwatchcounter: int
- splitcounter: int

+ actionPerformed(ActionEvent e): void

Time

- time: float
- date: int
- splits: ArrayList<Split>

+ getTime(): float
+ getDate(): int
+ setSplit(int ind, Split s): void
+ addSplit(Split s): void
+ toString(): String

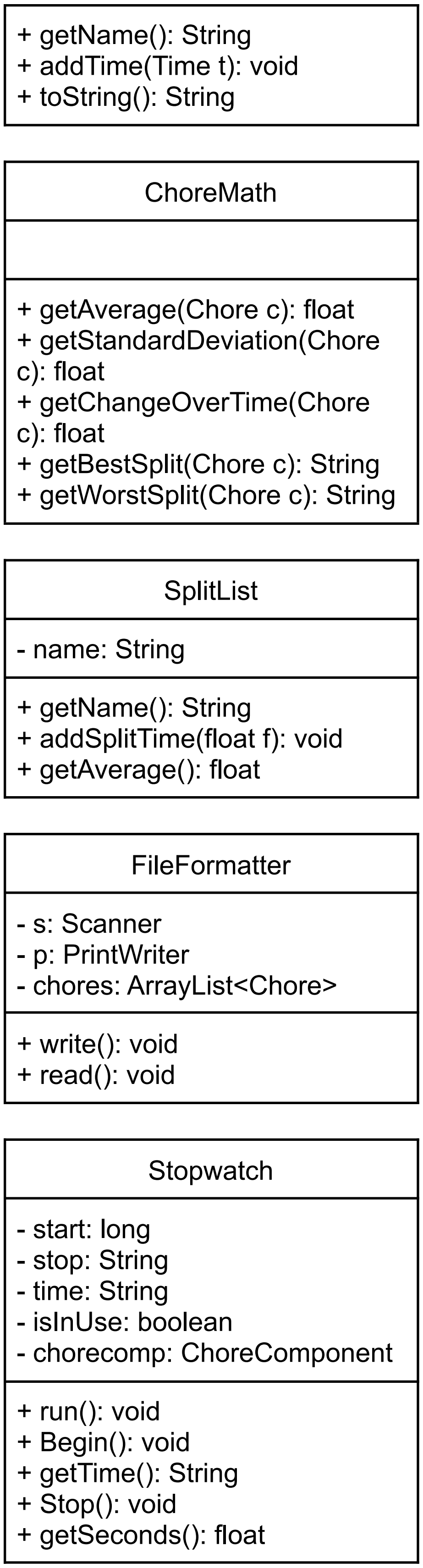
Split

- name: String
- point: float
- position: int

+ getName(): String
+ getPoint(): float
+ getPosition(): int
+ getDifference(int pos, Time t): float
+ toString(): String

Chore

- times: ArrayList<Time>
- name: String



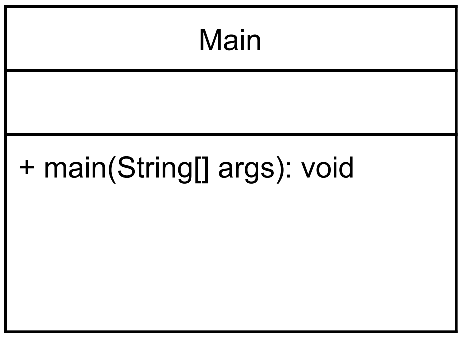
+ getName(): String + addTime(Time t): void + toString(): String

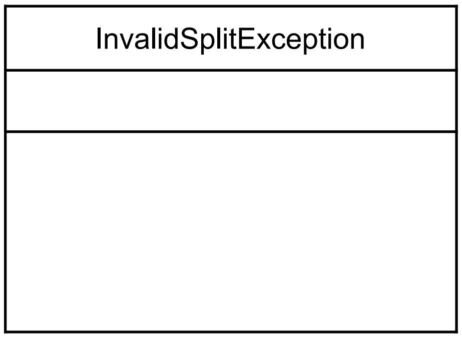
ChoreMath
+ getAverage(Chore c): float + getStandardDeviation(Chore c): float + getChangeOverTime(Chore c): float + getBestSplit(Chore c): String + getWorstSplit(Chore c): String

SplitList
- name: String
+ getName(): String + addSplitTime(float f): void + getAverage(): float

FileFormatter
- s: Scanner - p: PrintWriter - chores: ArrayList<Chore>
+ write(): void + read(): void

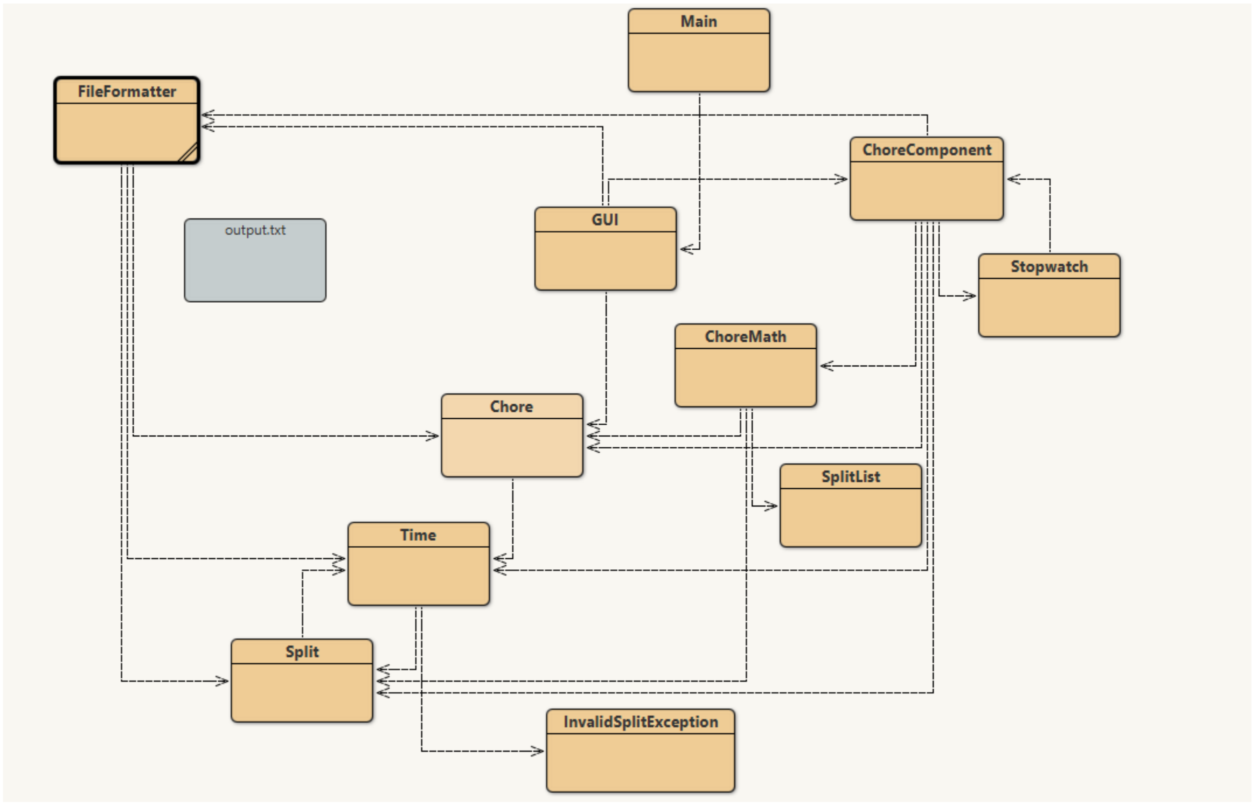
Stopwatch
- start: long - stop: String - time: String - isInUse: boolean - chorecomp: ChoreComponent
+ run(): void + Begin(): void + getTime(): String + Stop(): void + getSeconds(): float



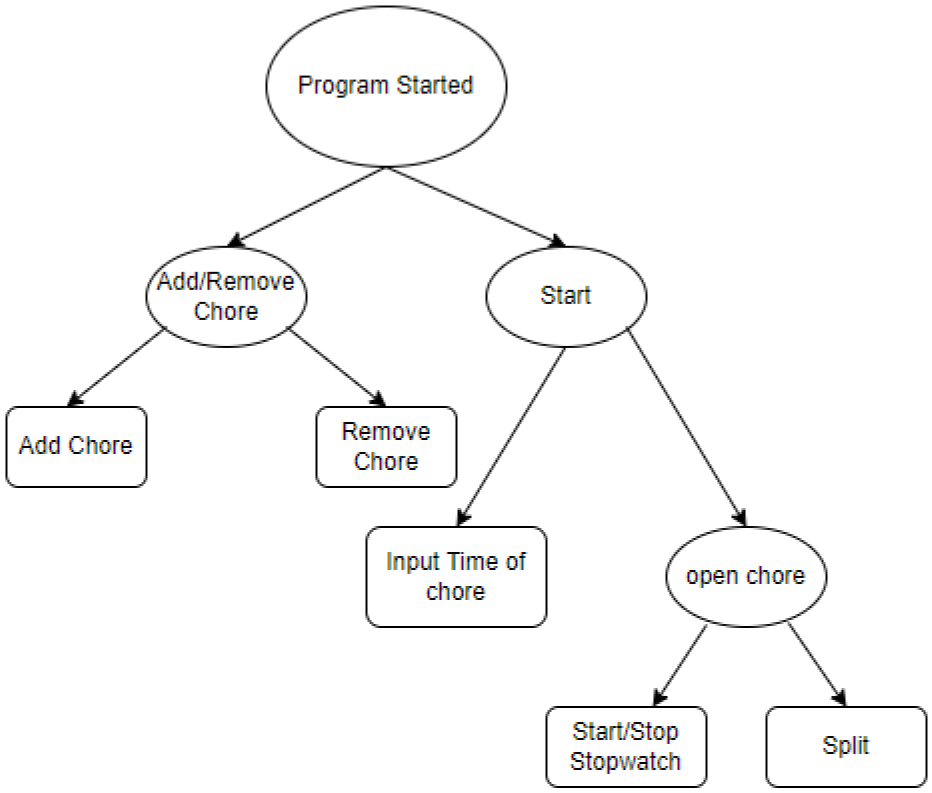


Class relationships

The class relationships are important to the design overview because it allows the user to see how each class relates with one another. Seeing how the classes relate with one another allows for the user to see which objects of the program depend on or use other objects of the program, improving the understanding of how the solution works.



Menu Layout
The menu layout is an important part of the design overview because the diagram allows the user to see how the menus and panels are laid out in the program in a condensed and concise way. The menu diagram below shows what the user can interact with in the program when they click a button to get to a certain menu or panel.



Flowcharts of algorithms:
These algorithms are an important part of the design overview because the algorithms are the backbone of the data processing going on inside the program in order for it to work properly. These algorithms allow the user to add and remove Chores from the program, finding the average of the Times of a Chore, and adding a Time to a chore which are some of the main and important functions of the solution.

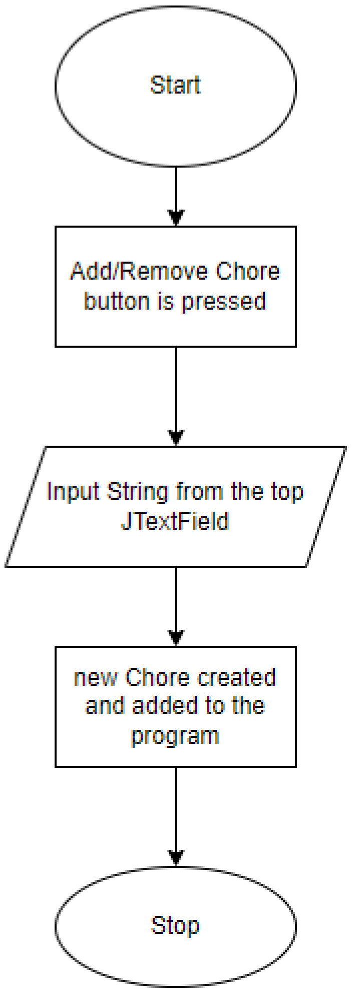

Start
Add/Remove Chore button is pressed
Input String from the top JTextField
new Chore created and added to the program
Stop

Figure 1 - Adding a new Chore to the program

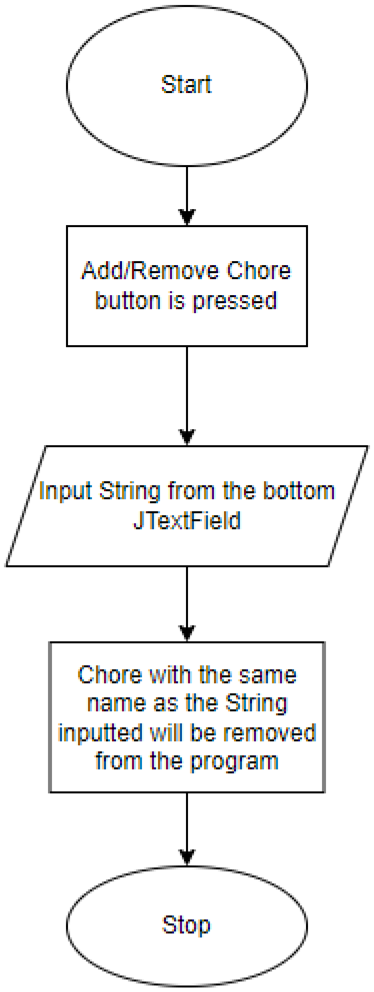

Start
Add/Remove Chore button is pressed
Input String from the bottom JTextField
Chore with the same name as the String inputted will be removed from the program
Stop

Figure 2 - Removing an existing Chore to the program

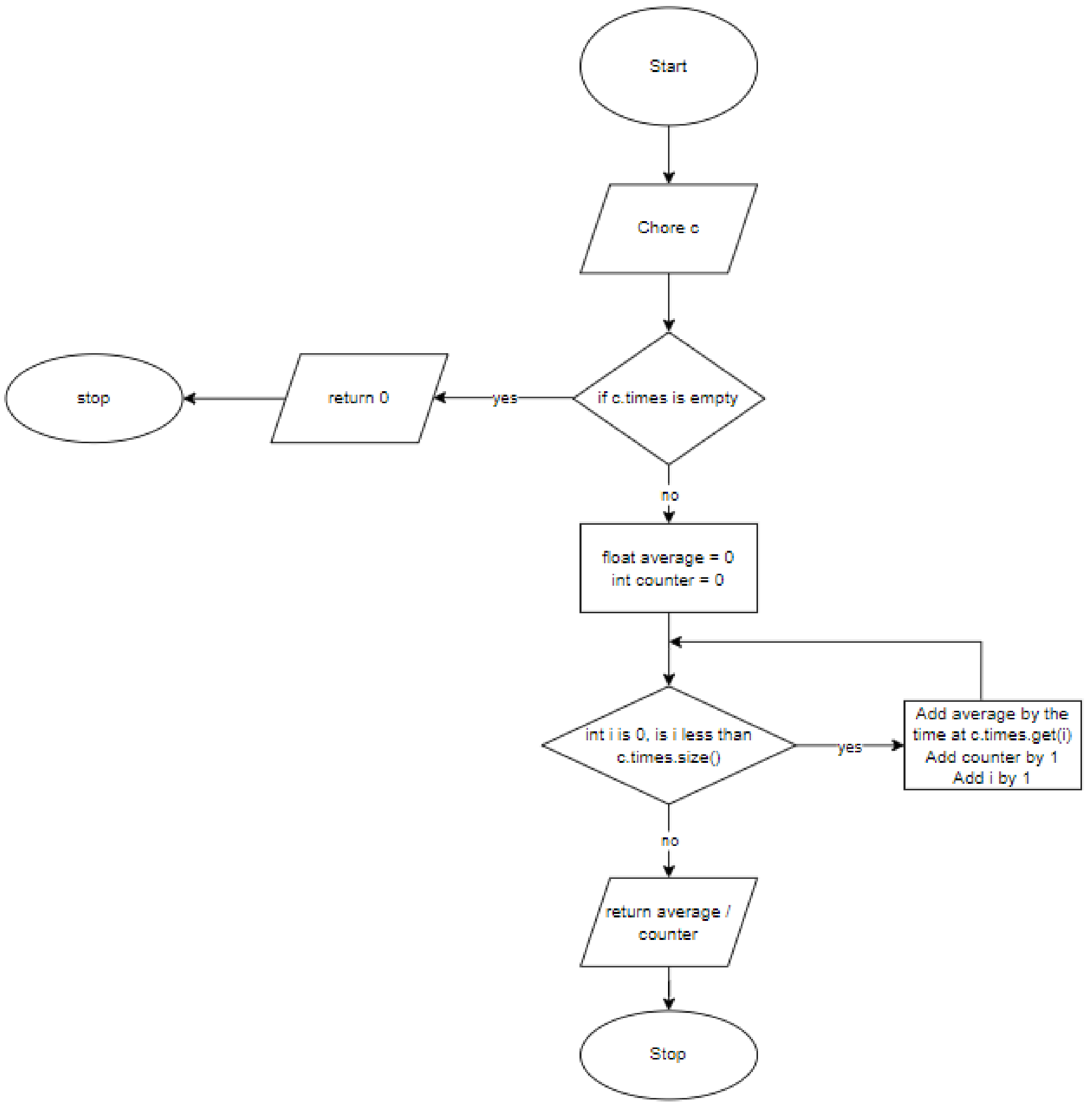


Figure 3 - Finding the average of the times of a Chore

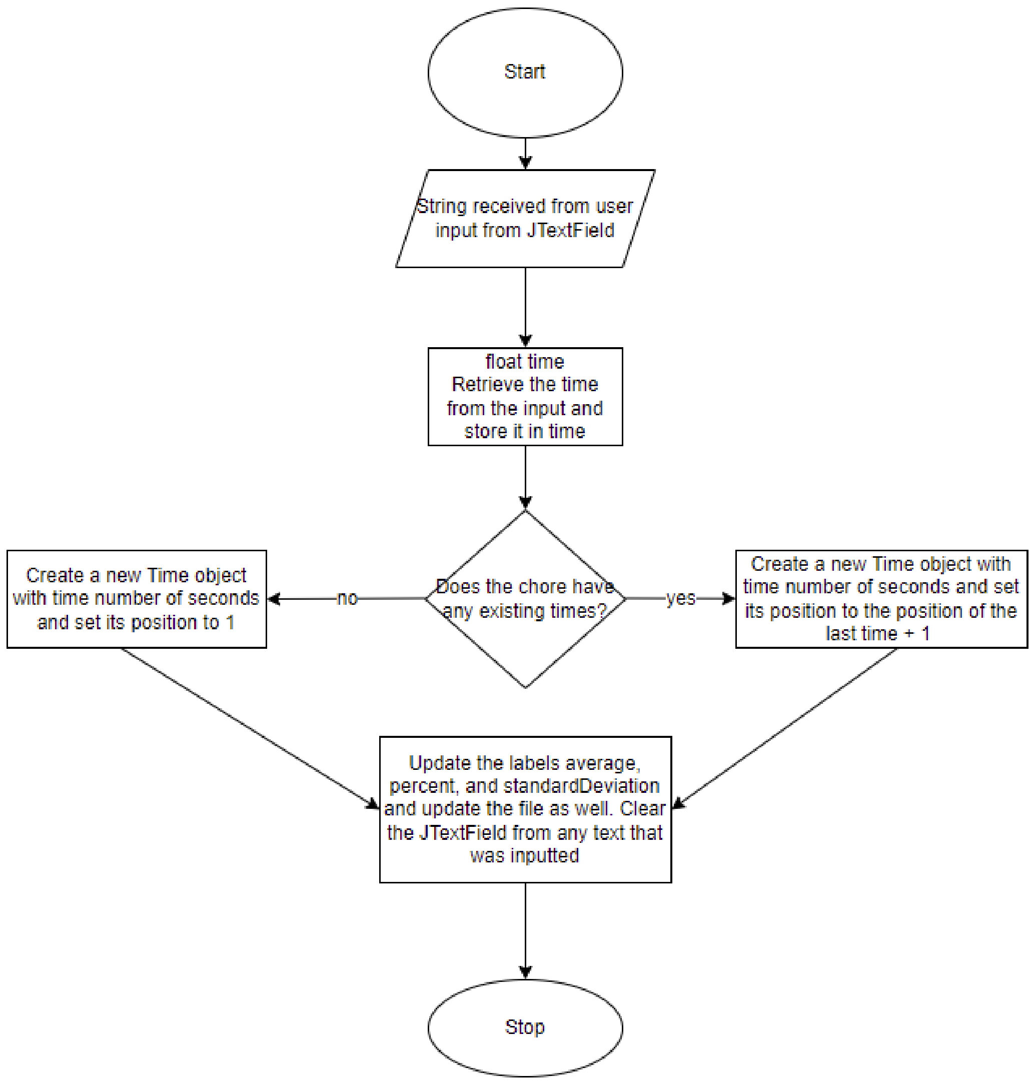

Start
String received from user input from JTextField
float time
Retrieve the time from the input and store it in time
Does the chore have any existing times?
no
yes
Create a new Time object with time number of seconds and set its position to 1
Create a new Time object with time number of seconds and set its position to the position of the last time + 1
Update the labels average, percent, and standardDeviation and update the file as well. Clear the JTextField from any text that was inputted
Stop

Figure 4 - Adding a time to a Chore

GUI windows/screens

The GUI window screens are an important part of the design overview because they are a visual representation of the program and allows the user to interact with the program in order to use it. The GUI allows the user to start the program, add and remove Chores, input Times for Chores, view statistics on particular Chores, and be able to use a stopwatch directly in the program to able to create Times and Splits for those times and allows the user to see what Splits are good and what Splits are not.

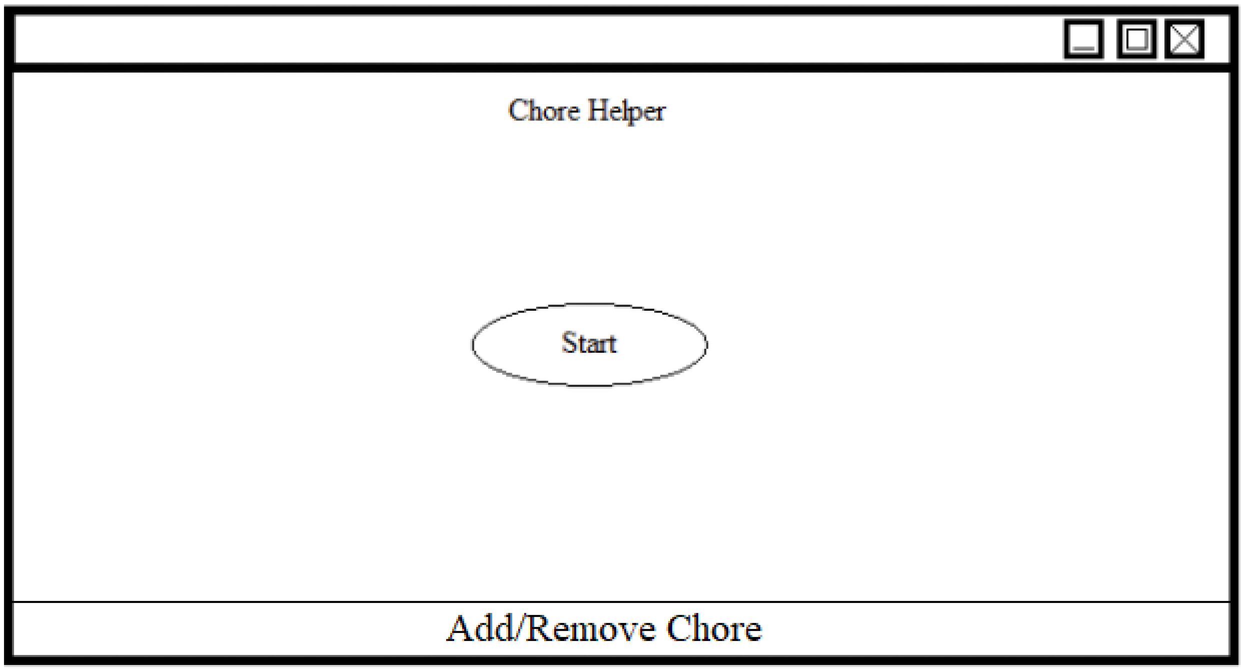


Figure 1 - Start window when program is opened

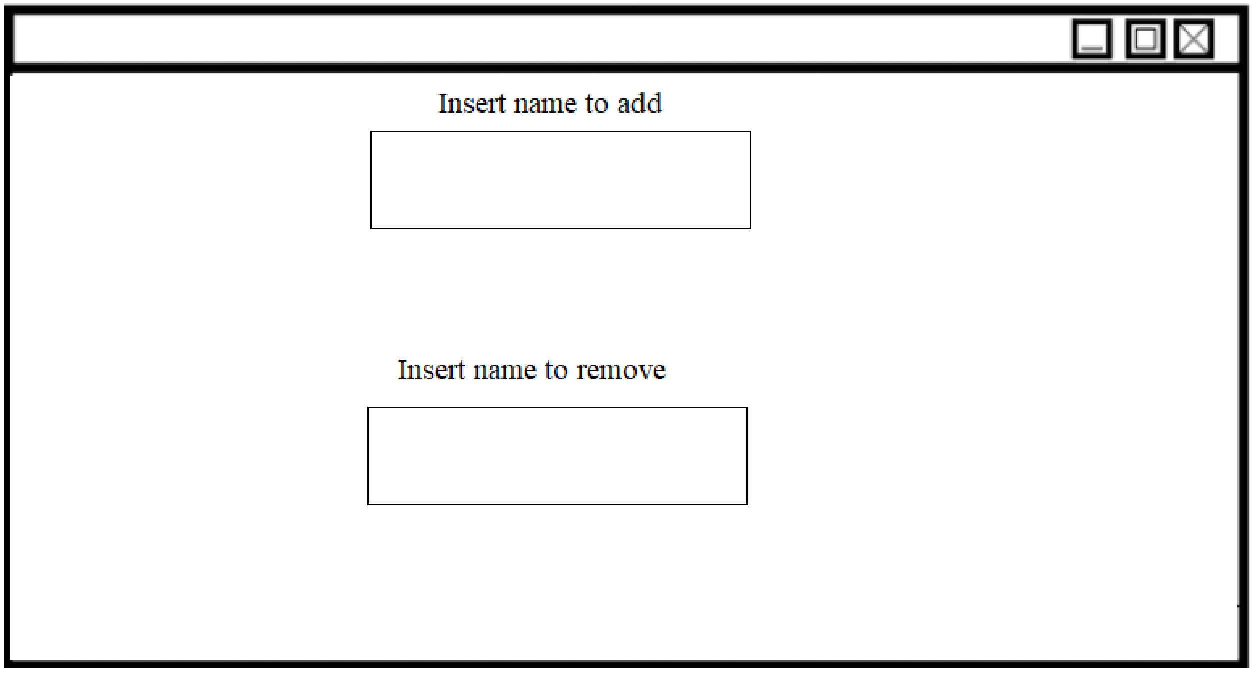


Figure 2 - Example panel of adding or removing a chore when the Add/Remove Chore button is clicked

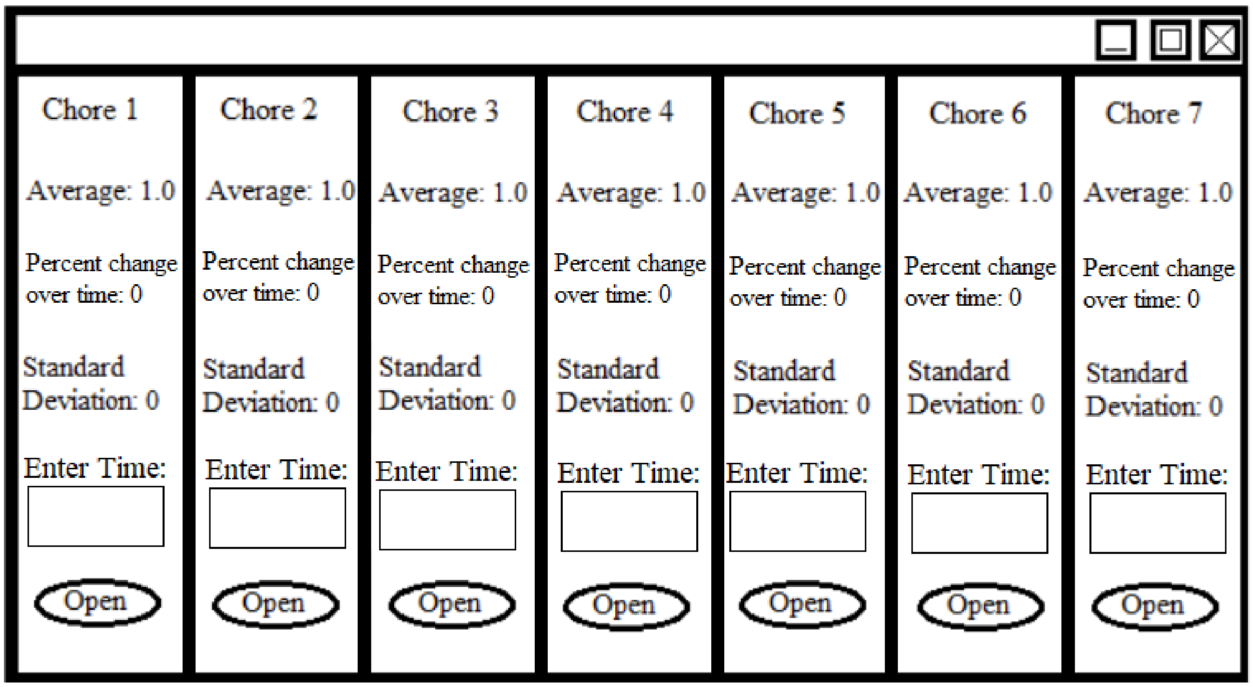


Figure 3 - Example panel of what the layout of the list of chores would look like when there are 7 chores on one page

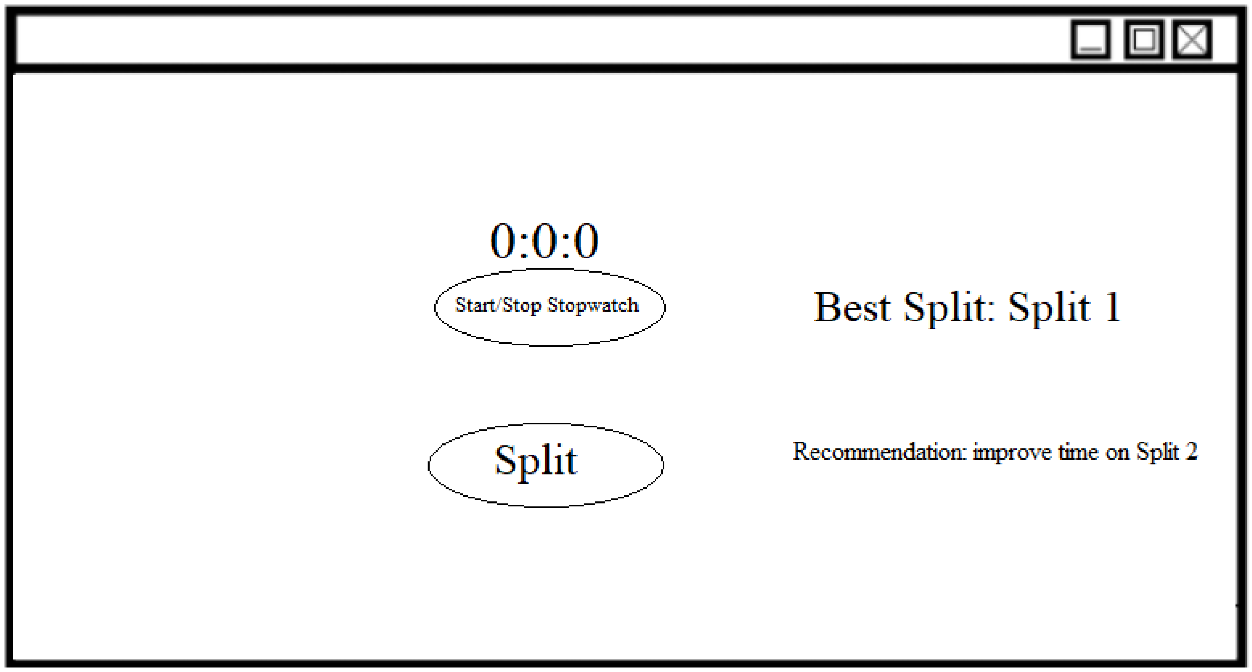


Figure 4 - Example of panel of what clicking the open button looks like

read/write file program uses - “output.txt”

Test Plan:

Criterion to be tested	Methods used to test
Adding and removing Chores from the program with Times	Add Chores with Times and check if they are all registered in arr in the GUI class and then remove them from the program and check arr again for verification that they were removed
Test the getAverage, getChangeOverTime, and getStandardDeviation which are algorithms that measure progression over time for a Chore	Make a few Chores and Times in the chore and test the methods. I would then see if the algorithms are correct by viewing the outputs of the methods and checking for accuracy
Adding Times to a Chore	Input Times for Chores and check the Chore to see if the Times have been added to times (times is the ArrayList Times are kept in for the Chore class)
Test the read and write methods of the FileFormatter class to see if the Chores could be written, read, and stored with a file	Input Times and Splits for a few chores and call the read and write methods and view the file and see if the Chores on the file match with the Chores inputted
Test the getWorstSplit method of ChoreMath to test if the recommendations of what actions needed to be taken to improve time on chores is working	I would input Times with Splits for a few Chores and execute the method and see if it gives the correct results which will be displayed through a ChoreComponent object
Go through the program and use the program as if I am the client of the program to test if the interface is simple and to the point	I would go through all the menus, press every button, and use the program in a similar manner to how the client would use the program and if there are any tedious or unnecessary GUI components I would remove them or add components to improve the overall structure and feel of the GUI
Making sure Split with all its methods works as intended and make sure Split objects are stored in splits (which is the ArrayList Splits are stored in for the Time class) correctly to check if being able to cut the timer for a chore into different sections to measure the time of a specific part of the chore works properly	I would add Times with Splits and test out the methods of a Split object and see if the methods work and see if the Split objects are being correctly stored in splits
Making sure the stopwatch and split buttons work	I would test the buttons by pressing them in multiple trials and seeing if the stopwatch gets activated and the stopwatch methods work and also see if the Splits are recorded in splits in a Chore object when the split button is pressed