Table of Contents

|  |  |
| --- | --- |
| **Section A: Planning** . . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . | **1** |
| Proposed Product. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . | 1 |
| Success Criteria. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . | 1 |
| **Section B: Design** . . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . | **2** |
| System Flow Chart. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . | 2 |
| Screen Prototypes. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . | 3 |
| Data Table. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . | 7 |
| Algorithms. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . | 9 |
| Test Plan. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . | 11 |
| **Section C: Development** . . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . | **13** |
| Input. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . | 13 |
| Processing. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . | 14 |
| Algorithms. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . | 15 |
| Data Structures. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . | 20 |
| Existing Libraries. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . | 21 |
| Output. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . | 21 |
| **Section D: Evaluation** . . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . | **23** |
| Success Criteria Status. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . .. . . . . . . | 23 |
| Recommendations for Further Development/Improvement. . . . . . . .. . . . . . . .. . . . . | 23 |

Section A: Planning

**Proposed Product**

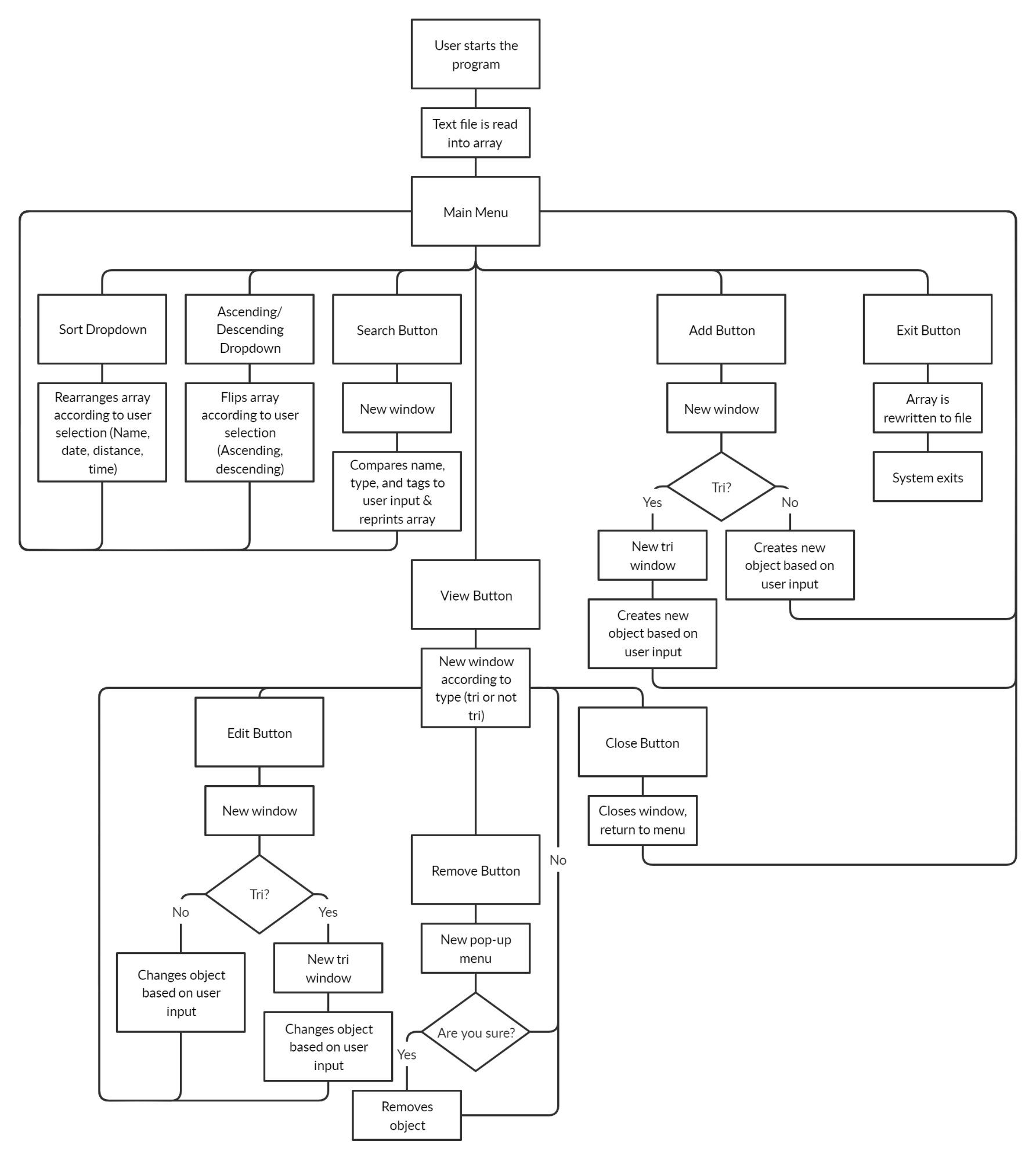
The proposed product is a program that will contain race information in one file and be able to perform different functions on them. This program will be written in Java in order to make use of Java libraries, like Swing for GUI. At the beginning of the program, a file containing all the clients' race information will be imported into the program, and at the end, the information will be rewritten into a file. The information will include a tag feature, which will allow the client to tag races with weather, location, or other niche information they desire. The client can also interact with the program GUI to perform the additional functions that they expressed in the interview. The client will be able to sort (ascending and descending) by date, distance, total time, and alphabetical title. The search function will show the reader races that match their search input. After searching, they can edit the information of that race to fix mistakes or add new information. As the client competes in more races, they can also create new races to add to the list.

**Success Criteria**

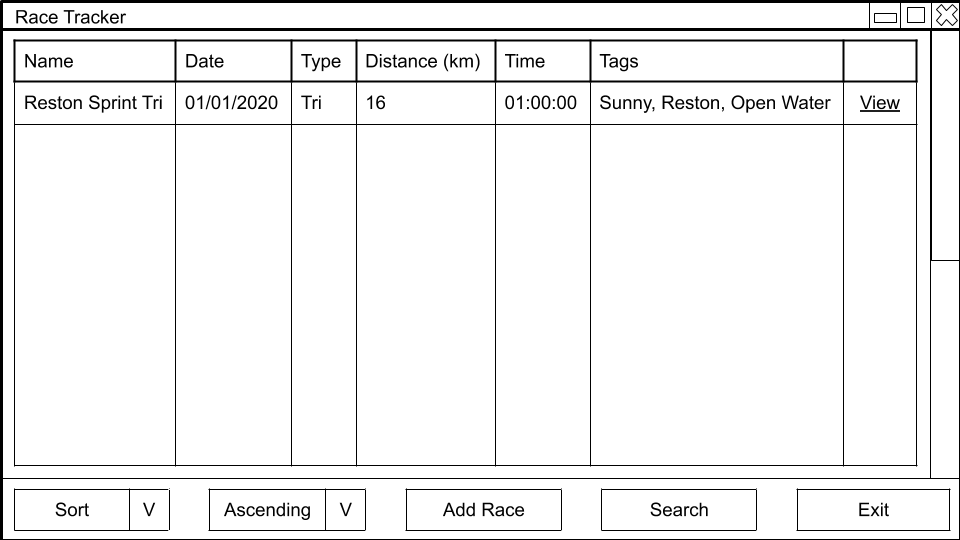
* The system can read existing race data from file into program
* The user can sort by name, date, distance, or total time
* The user can toggle sort by ascending and descending
* The user can search by name, type, or tags
* The system can save edited race information
* The user can create a new race
* The user can delete races
* The system can save array into file

Section B: Design

**System Flow Chart**

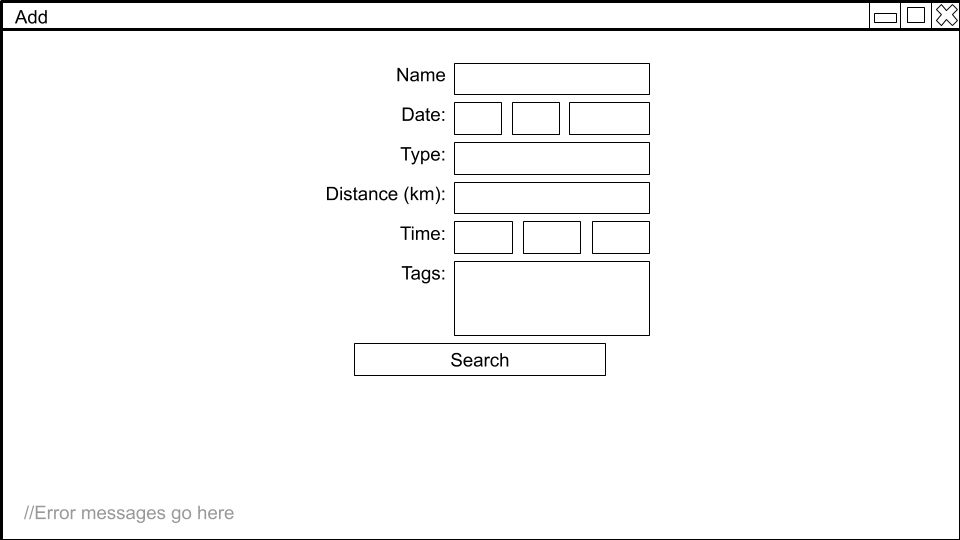


**Screen Prototypes**



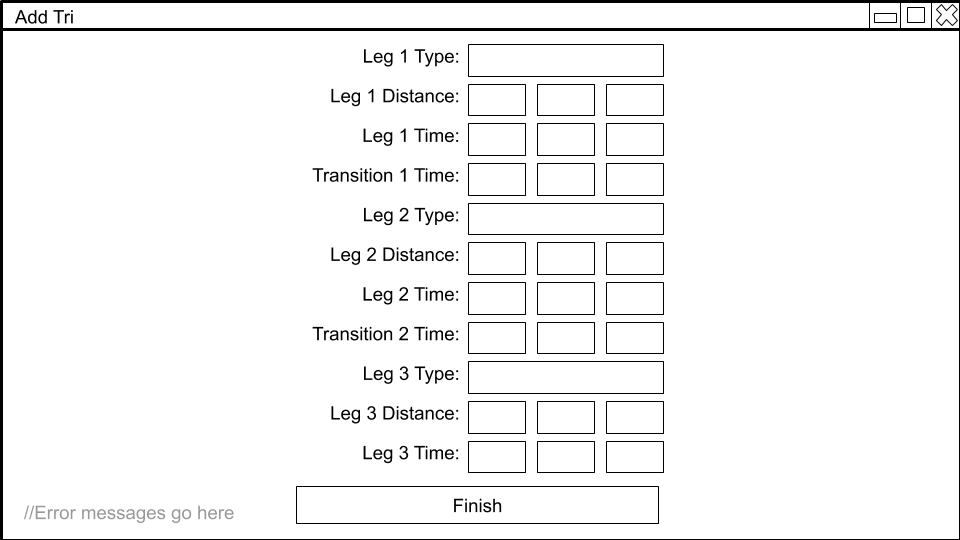
**Figure 1: Home Screen**

On the home screen the user sees a list of all their races and different operations they can perform.



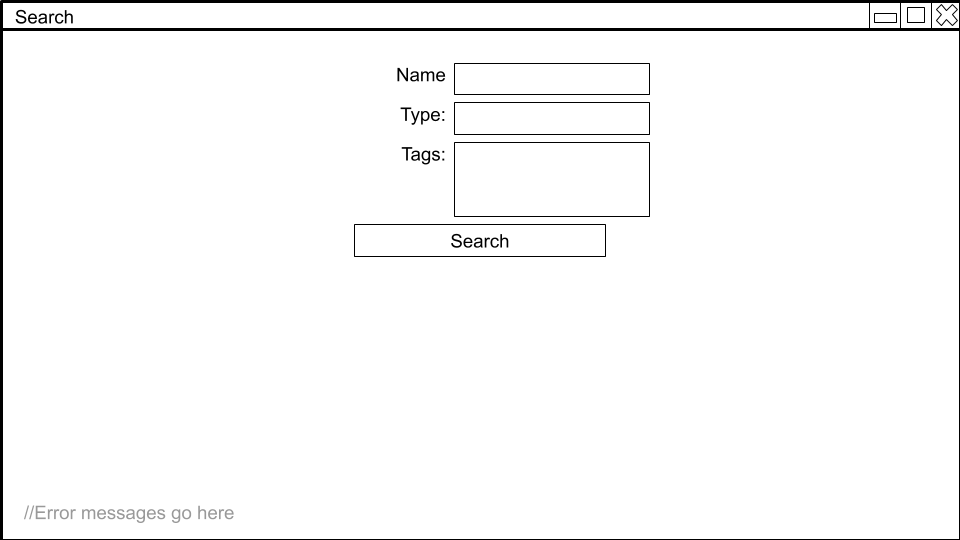
**Figure 2: Add Screen**

This is the screen where the user can add a new race, and is reached by clicking the “Add” button on the “Main” screen. If the user clicks “Add” and the type is tri, the user is brought to the additional “Add Tri” screen where they can add the relevant tri data.



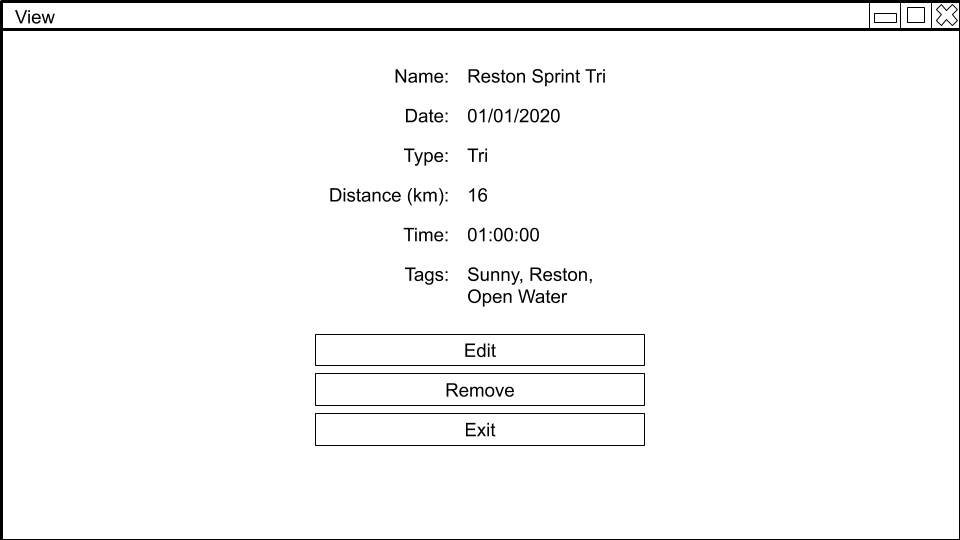
**Figure 3: Add Tri Screen**

If the user identifies a race as “tri” while adding, they are brought to this screen to add the tri data.



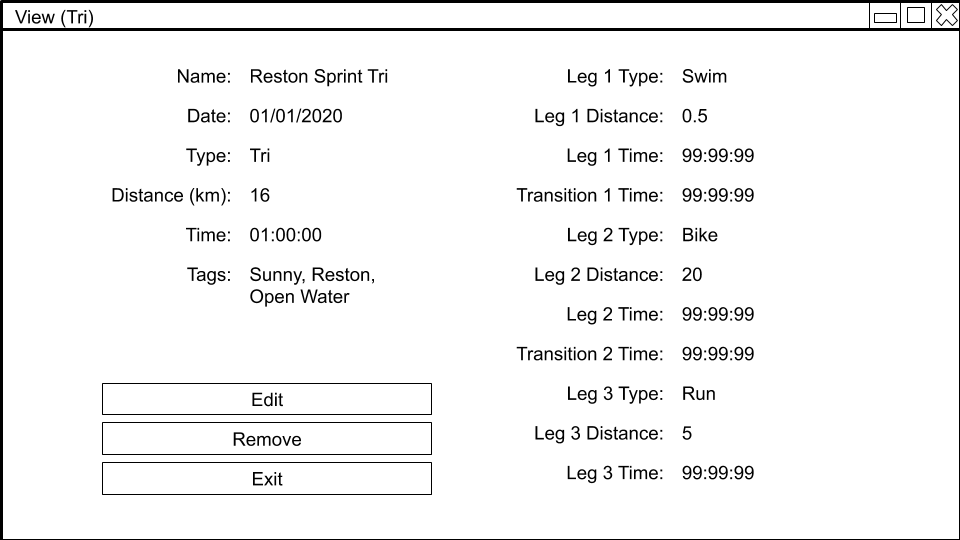
**Figure 4: Search**

This screen is used to input search parameters. The user is brought to this screen if they click “Search” on the “Main” screen.



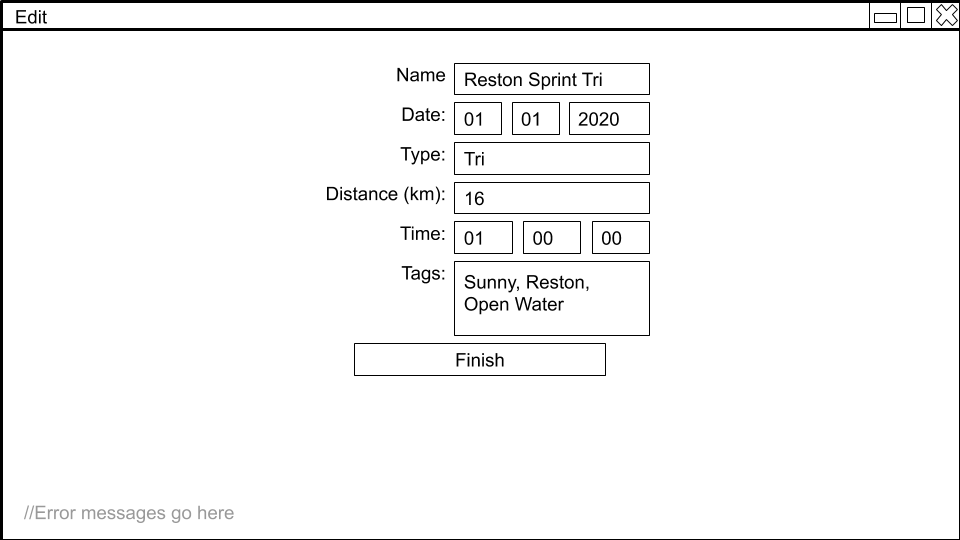
**Figure 5: View Screen**

This screen is used to view the details of a specific race and perform actions on it. The user is brought to this screen when they click “View” next to the race they want to view.



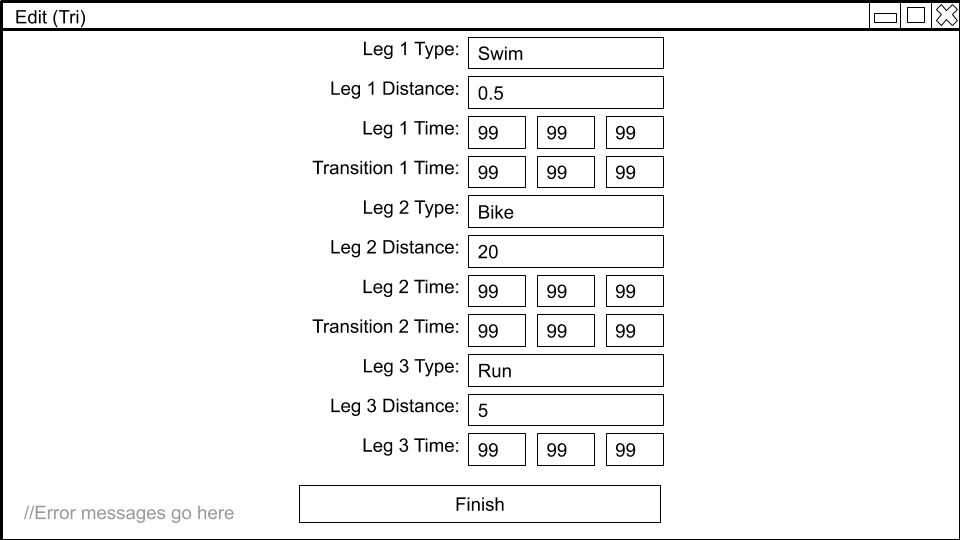
**Figure 6: View Tri Screen**

This screen is used to view the details of a specific tri race and perform actions on it. The user is brought to this screen when they click “View” next to the race they want to view, and the race is a tri.



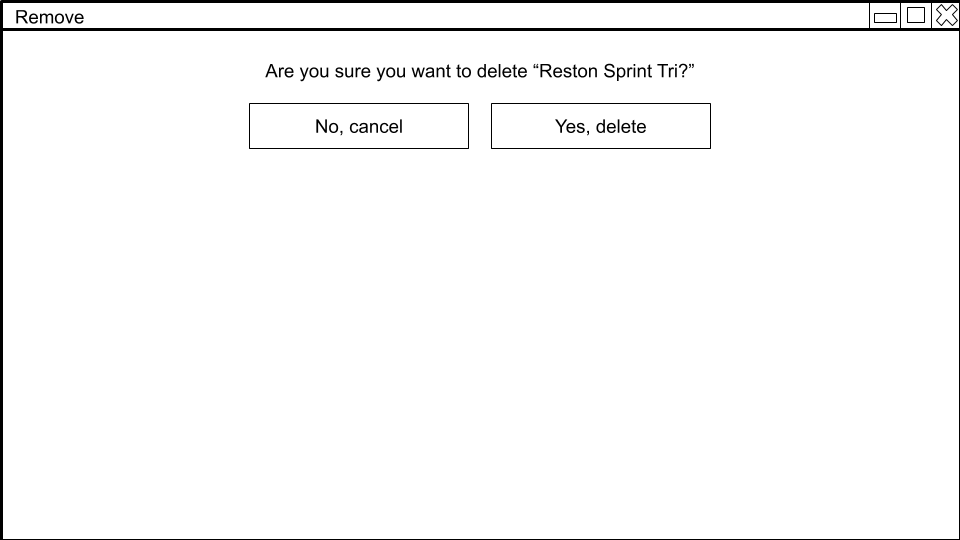
**Figure 7: Edit Screen**

This screen is used to edit information in a race object. The user is brought to this screen if they click “Edit” in the “View” screen.



**Figure 8: Edit Tri Screen**

This screen is used to edit information in a tri. The user is brought to this screen if they click “Finish” on the “Edit” screen, and the race is a tri.



**Figure 9: Remove Screen**

This screen is used to confirm that the user wants to remove a race object. The user is brought to this screen when they click “Remove” on the “View” screen.

**Data Table**

|  |  |  |  |
| --- | --- | --- | --- |
| **Name** | **Description** | **Data Type** | **Sample Data** |
| Race | Stores data for a specific race | Custom Class   |  | | --- | | **Race** | | String: name  Date: date  String: Type  double: distance  Time: time  Tags[]: tags  Tri: tri | | getName()  getDate()  getType()  getDistance()  getTime()  getTags()  getTri()  setName()  setDate()  getType()  setDistance()  setTime()  setTags()  setTri() | | Name: Reston Sprint Tri  Date: 1/1/2020  Type: Tri  Distance: 16  Time: 01:00:00  Tags: Sunny, Reston  Tri: Tri |
| Race[] | Stores all race data | Array of type Race | N/A |
| Tri | Because not every race is a tri, Tri stores all extra data reserved for tri races. | Custom Class   |  | | --- | | **Tri** | | String: typeOne  Double: distanceOne  Time: timeOne  Time: transitionOne  String: typeTwo  Double: distanceTwo  Time: timeTwo  Time: transitionTwo  String: typeThree  Double: distanceThree  Time: timeThree | | getTypeOne()  getDistanceOne()  getTimeOne()  getTransitionOne()  getTypeTwo()  getDistanceTwo()  getTimeTwo()  getTransitionTwo()  getTypeThree()  getDistanceThree()  getTimeThree()  setTypeOne()  setDistanceOne()  setTimeOne()  setTransitionOne()  setTypeTwo()  setDistanceTwo()  setTimeTwo()  setTransitionTwo()  setTypeThree()  setDistanceThree()  setTimeThree() | | TypeOne: Swim  DistanceOne: .5  TimeOne: 00:05:00  TransitionOne: 00:03:00  TypeTwo: Bike  DistanceTwo: 20  TimeTwo: 00:30:00  TransitionTwo 00:02:00  TypeThree: Run  DistanceThree: 5  TimeThree: 00:30:00 |
| Time | Stores times | Custom class   |  | | --- | | **Time** | | int hours  int minutes  int seconds | | getHours()  getMin()  getSec()  setHours()  setMin()  setSec() | | hours: 1  minutes: 23  seconds: 49 |
| Date | Stores dates | Custom class   |  | | --- | | **Date** | | int month  int day  int year | | getMonth()  getDay()  getYear()  setMonth()  setDay()  setYear() | | month: 12  day: 2  year: 2019 |
| races.txt | File to store race information | Text file | N/A |

**Algorithms**

**Text file to array**

try

open race file

catch

print “could not find file”

new array race[]

int raceCounter = 0

loop while file has next

race[i] = info from file

raceCounter++

end loop

close file

**Sort**  
loop race[] for raceCounter

int smallest = i  
 loop race[] for raceCounter, j = i + 1

if race[j]get(title, date, distance, time) < race[smallest].get(title, date, distance, time)

smallest = j

end if

end loop

(String, Date, int, Time) temp = race[i].get(title, date, distance, time)

race[i].get(title, date, distance, time) = race[smallest].get(title, date, distance, time)

race[smallest].get(title, date, distance, time) = temp

end loop

**Add**

try

race[raceCounter + 1] = info from text fields

if race = tri

open tri screen

try

race[raceCounter + 1].setTri = info from text fields

catch

print “error adding race”

else

race[raceCounter + 1].setTri = null

catch

print “error adding race”

**Search**

searchTerm = info from text field

boolean found = false

loop race[] for raceCounter

if race[i] = searchTerm

print race[i]

found = true

end if

end loop

if found = false

print “no races found matching search”

end if

**Edit**

try

race[current] = info from text fields

if type = tri

open tri screen

try

race[current].setTri = info from text fields

catch

print “error editing race”

end if

catch

print “error editing race”

**Remove**

if confirm = true

loop for raceCounter

i = current

race[i] = race[i + 1]

current = i + 1

end loop

raceCounter- -

end if

**Array to text file**

loop for raceCounter

outfile print race[i]

end loop

close file

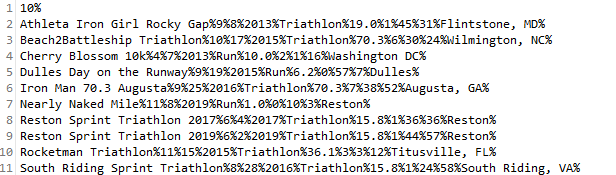
**Test Plan**

|  |  |  |
| --- | --- | --- |
| **Step Description** | **Input** | **Expected Outcome** |
| User opens the program | N/A | File is read into array, home screen appears |
| User opens the program, file is not found | N/A | Program closes |
| User clicks “Sort” dropdown and selects “Title” | Dropdown selection | Array is sorted by title, home screen refreshes |
| User clicks “Sort” dropdown and selects “Date” | Dropdown selection | Array is sorted by date, home screen refreshes |
| User clicks “Sort” dropdown and selects “Distance” | Dropdown selection | Array is sorted by distance, home screen refreshes |
| User clicks “Sort” dropdown and selects “Time” | Dropdown selection | Array is sorted by time, home screen refreshes |
| User clicks “Ascending” dropdown and selects “Ascending” | Dropdown selection | Home screen refreshes, home screen loads array forward, dropdown text is set to “Ascending” |
| User clicks “Ascending” dropdown and selects “Descending” | Dropdown selection | Home screen refreshes, home screen loads array backward, dropdown text is set to “Descending” |
| User clicks “Add Race” button | Button Press | Add screen appears |
| User enters valid data and clicks “Finish” button | Data entry, button press | New object is created, returns to home screen |
| User enters valid data, Type = Tri, and clicks “Finish” button | Data entry, button press | Add Tri screen appears |
| User enters valid data on tri screen and clicks “Finish” button | Data entry, button press | New object is created, returns to home screen |
| User enters invalid data and clicks “Finish” button | Data entry, button press | Error message appears |
| User enters invalid data, Type = Tri, and clicks “Finish” button | Data entry, button press | Error message appears |
| User enters invalid data on tri screen and clicks “Finish” button | Data entry, button press | Error message appears |
| User clicks “Search” button | Button press | Search screen appears |
| User enters valid data and clicks “Search” | Data entry, button press | Home screen loads only items in the array that match the entered data, returns to home screen |
| User enters valid data and clicks “Search,” but data matches no item in the array | Data entry, button press | Error message appears |
| User enters invalid data and clicks “Search” | Data entry, button press | Error message appears |
| User clicks “View” button | Button press | View screen appears |
| User clicks “Edit” button | Button press | Edit screen appears, text boxes filled with existing information |
| User enters valid data and clicks “Finish” | Data entry, button press | Object is edited according to data entered, returns to view screen |
| User enters valid data, Type = Tri, and clicks “Finish” button | Data entry, button press | Edit Tri screen appears, text boxes filled with existing information |
| User enters valid data on tri screen and clicks “Finish” button | Data entry, button press | Object is edited according to data entered, returns to view screen |
| User enters invalid data and clicks “Finish” button | Data entry, button press | Error message appears |
| User enters invalid data, Type = Tri, and clicks “Finish” button | Data entry, button press | Error message appears |
| User enters invalid data on tri screen and clicks “Finish” button | Data entry, button press | Error message appears |
| User clicks “Remove” button | Button press | User is prompted to confirm their decision |
| User clicks “No, cancel” | Button press | Returns to view screen |
| User clicks “Yes, delete” | Button press | Object is removed from array, returns to view screen |
| User clicks “Exit” | Button press | Returns to home screen |
| User clicks “Exit” | Button press | Array is written to file, program closes |

Section C: Development

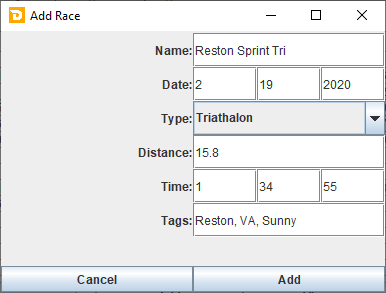
**Input**

The data that this program uses is imported from a text file initially, and can then be added on to from user input.



**Figure 10: Initial text file**

The algorithm that is used to import data from the text file is included under Section C: Algorithms (Figure 12).



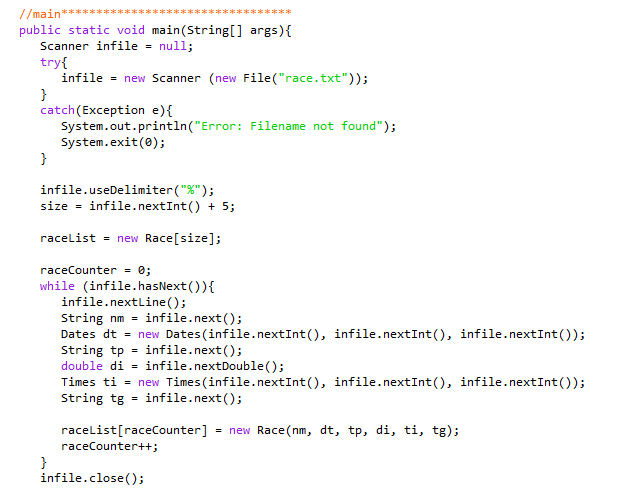
**Figure 11: User Input**

This is the screen where the user inputs data (sample data included). The algorithm that is used to input new data in this way is included under Section C: Algorithms (Figure 20).

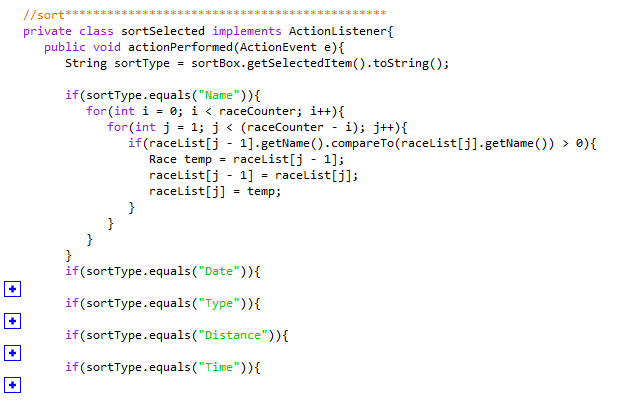
**Processing**

|  |  |  |  |
| --- | --- | --- | --- |
| **Algorithm** | **Method/Class Name** | **Purpose** | **Comments** |
| Input initial data (Figure 12) | main() | Inputs initial race data into array | Data is read from text file |
| Sort (Figure 13) | sortSelected | Sorts the array | Sort type is selected by the user from a dropdown menu |
| Compare dates (Figure 14) | compareDate() | Compares two dates | Used while sorting array by date |
| Toggle sort (Figure 15) | adSelected | Toggles sort by ascending or descending | Ascending/descending is selected by the user from a dropdown menu. This selection does not rearrange array, only the order that the data is listed on the main screen |
| View (Figure 16) | viewPressed | Searches for a race | Searches by name inputted by the user to open a window with information about a specific race. |
| Search (Figure 17) | searchPressed | Searches for multiple races | Searches by a search term is inputted by user to list races matching the search term are listed on the main screen |
| Edit (Figure 18) | savePressed | Rewrites an object in the array | New data is inputted by user, object index is determined by previous “view” search |
| Delete (Figure 19) | yesPressed | Removes an object from the array | Object index is determined by previous “view” search |
| Add (Figure 20) | addPressed | Adds a new object to the array | Data is inputted by user |
| Exit program (Figure 21) | exitPressed | Saves data | Data is saved to text file and the program is closed |

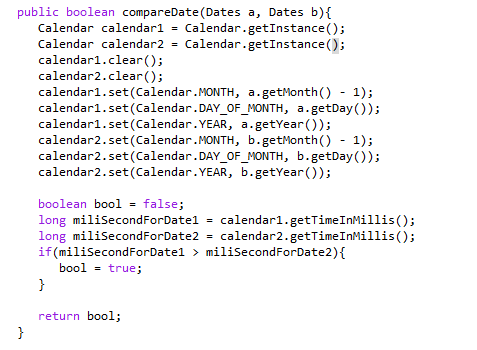
**Algorithms**



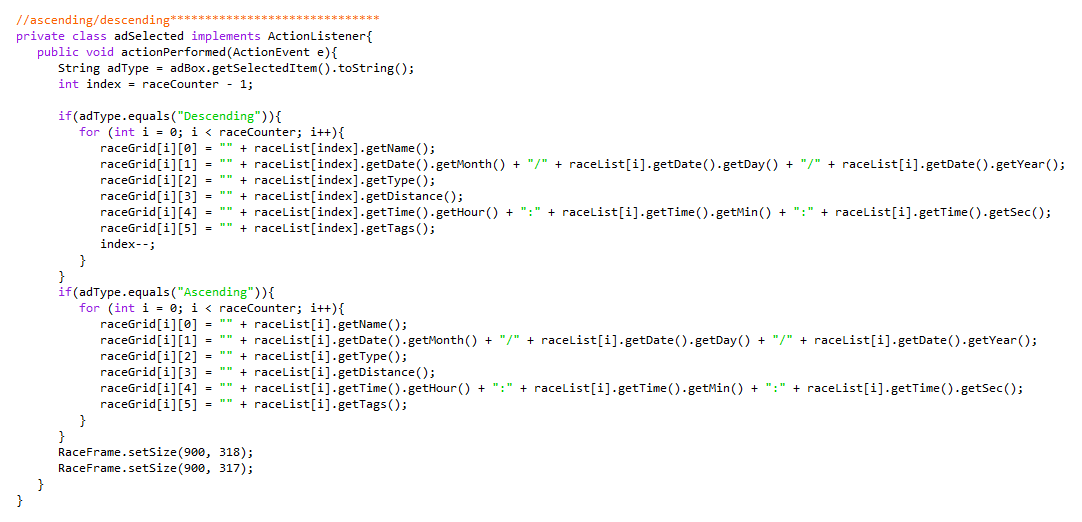
**Figure 12: Input initial data**



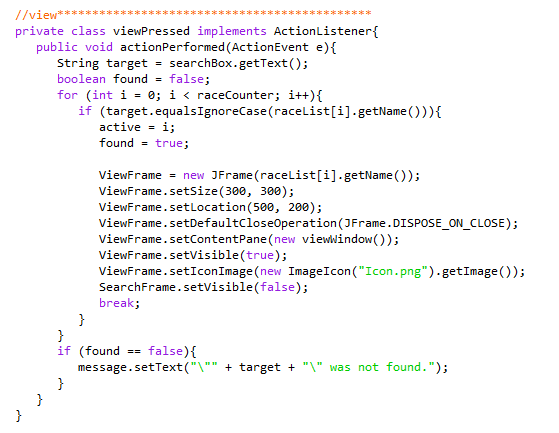
**Figure 13: Sort**



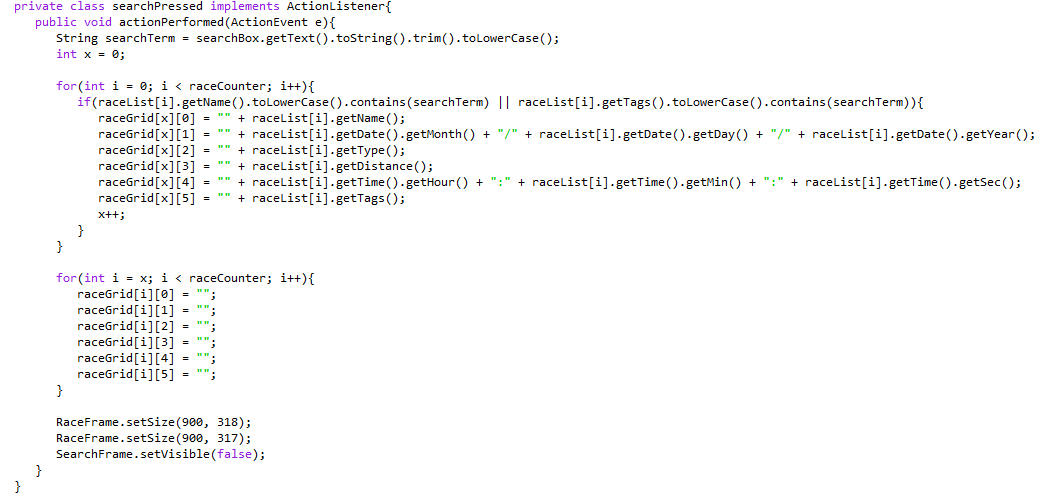
**Figure 14: Compare dates**



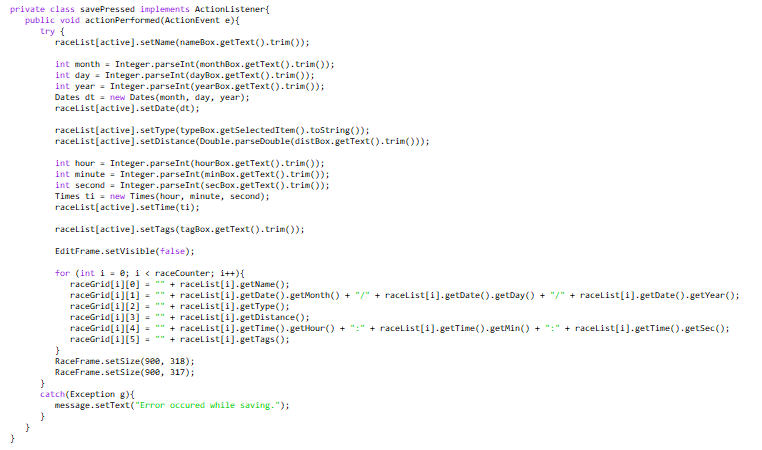
**Figure 15: Toggle sort**



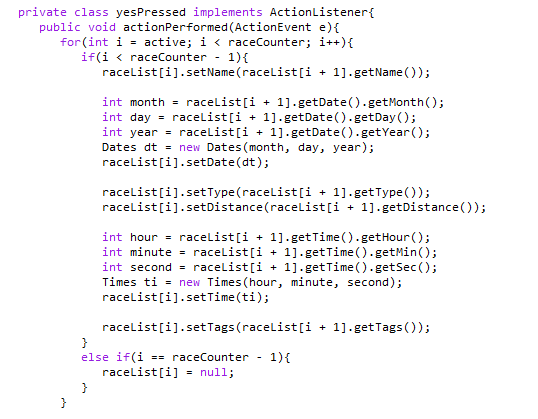
**Figure 16: View**



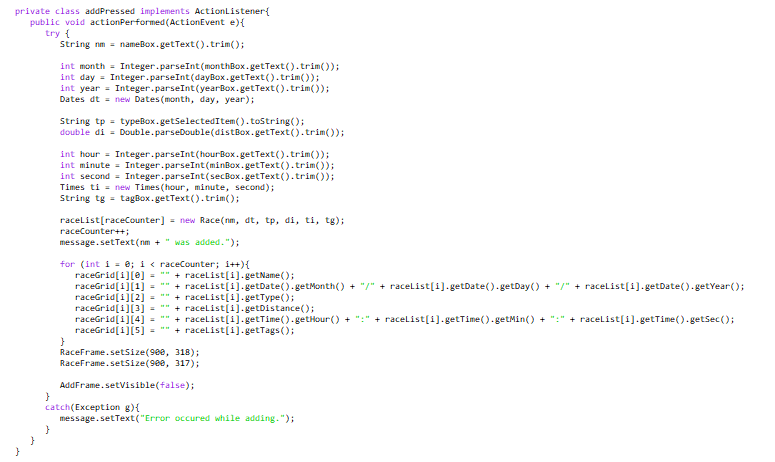
**Figure 17: Search**



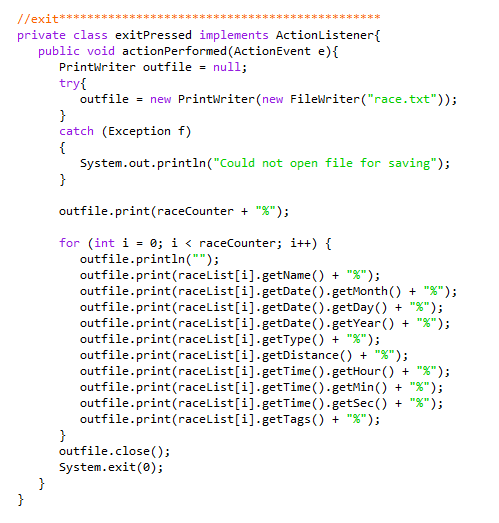
**Figure 18: Edit**



**Figure 19: Delete**



**Figure 20: Add**



**Figure 21: Exit**

**Data Structures**

|  |  |  |  |
| --- | --- | --- | --- |
| **Data Structure** | **Purpose** | **UML** | **Comments** |
| Custom class - Race | Stores all data related to a specific race | |  | | --- | | **Race** | | String: name  Date: date  String: Type  double: distance  Time: time  Tags[]: tags | | getName()  getDate()  getType()  getDistance()  getTime()  getTags()  setName()  setDate()  getType()  setDistance()  setTime()  setTags() | | Custom class |
| Array of races - Race[] raceList | Stores all race data. Allows for iteration and indexing |  | Custom class used as a data type |
| Custom class - Times | Stores integers for hours, minutes, and seconds | |  | | --- | | **Time** | | int hours  int minutes  int seconds | | getHours()  getMin()  getSec()  setHours()  setMin()  setSec() | | Custom class |
| Custom class - Dates | Stores integers for months, days, and years | |  | | --- | | **Date** | | int month  int day  int year | | getMonth()  getDay()  getYear()  setMonth()  setDay()  setYear() | | Custom class |

**Existing Libraries**

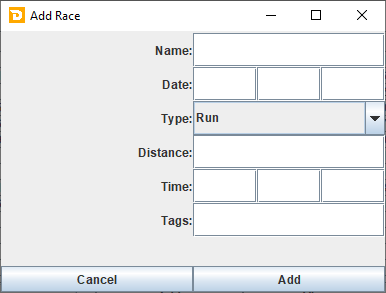
* javax.swing – Swing library used for user interface
* java.util.Calendar – Calendar library used to compare dates for sorting

**Output**



**Figure 22: Main screen**

Opened on startup



**Figure 23: Add screen**

Opened from the “Main” screen. The algorithm that is used to input new data in this way is included under Section C: Algorithms (Figure 20).

|  |  |
| --- | --- |
| **Figure 24: Search (view) screen**  Opened from the “Main” screen. The algorithm that is used to search by name is included under Section C: Algorithms (Figure 17).    **Figure 26: Edit screen**  Opened from the “View” screen. The algorithm that is used to input new data in this way is included under Section C: Algorithms (Figure 18).    **Figure 28: Search (Sort) Screen**  Opened from the “Main” screen. The algorithm that is used to search by name or tags is included under Section C: Algorithms (Figure 17). | **Figure 25: View screen**  Opened based on the search from the “Search” screen. The algorithm that is used to load this data is included under Section C: Algorithms (Figure 16).    **Figure 27: Delete screen**  Opened from the “View” screen. The algorithm that is used to delete data is included under Section C: Algorithms (Figure 19). |
|

Section D: Evaluation

**Success Criteria Status**

|  |  |
| --- | --- |
| **Success Criteria** | **Status** |
| The system can read existing race data from file into program | Success |
| The user can sort by name, date, distance, and total time | Success |
| The user can toggle sort by ascending and descending | Success |
| The user can search by name, type, and tags | Success |
| The system can save edited race information | Success |
| The user can create a new race | Success |
| The user can delete races | Success |
| The system can save array into file | Success |

**Recommendations for Further Development/Improvement**

* Move the “View” button from the button panel into the grid beside every list item, as originally planned. This provides an easy way to quickly open a race without the need to search by name.
* Add onto the “Race” class to support triathlon data (e.g. individual leg race types, times, and lengths), and implement additional screens to add and display the extra data, as originally planned.
* Additional user input error handling. Automatically include the year in parentheses next to the name of any matching races, as some recurring races have the same name every year.
* Fix date and time text formatting (e.g. “01:22:08” instead of “1:22:8,” “03/01/2015” instead of “3/1/15”).
* UI fixes. Center certain text and input fields (e.g. “Edit” screen and “Add” screen).