**1. Title:**

Project Title: Sole Table

Program Name: shoe\_tracker

**2. Authors:**

Grace Joyce (gjoyce@wisc.edu)

Alex Pletta (apletta@wisc.edu)

Lu Duan (lduan23@wisc.edu)

Joziah Mays (jmays2@wisc.edu)

Liang Shang (lshang6@wisc.edu)

**3. Description:**

Our program will solve the problem of storing inventory of shoes. Much like a corporate store, inventory must be kept in an efficient manner so that products can be found when asked. Specifically, when it comes to shoe stores, the store has to not only know what shoes they have, but also how many and how many in each size. Using a hash table, I think I will be able to store shoes based on ID or product numbers.

**4. Stakeholder:**

*Corporate Shoe Store*

The primary user will be the corporate shoe store. As said above, the program will use a hash table with a bucket collision resolution to store product numbers (key) and store how much in each size (value) is still in stock. The bucket collision resolution will be useful because it will allow the same product to be hashed to the same index, with the only difference being the size of each shoe. If different products hash to the same index, their values will be compared. Since it is a bucket, I will be able to traverse the list to get a count of the number of shoes in a specific size. This will simply give the stakeholder an efficient way to find products and know how much of each product they have.

*Personal Use*

A secondary user will be individuals. Using the Sole Table program, people can keep track of their own shoes as they buy/sell them. The program will help organize their shoes and make it easier to look them up. This could be advantageous if they were away from their collection but still wanted to know what they had in stock. Additional functionality could be built in per user feedback to include different data fields and/or add new features.

**5. Output**

* All output contained in GUI windows
  + Name of shoe
  + Product number
  + Amount in stock
  + Error message if shoe not in stock
  + Picture of product
  + List of sizes (hidden behind a button)

*Format*

What is the primary result (output) that the program will produce or provide to the user?

* Output in GUI window
* List of sizes comes from hashtable bucket
* → make example pic from powerpoint or similar

*Example*

Give an example (using "stub data" -- just make up the "data") to show a format of your program's output.

* See GUI windows in Diagrams section

**6. Input**

* Product number
* Potentially picture of object (for image recognition)
* Selection to look up or add a new object
* For adding a new object, need other data fields
  + Picture
  + Product Number
  + Size of Product
  + Shoe name
* If wanting to load more than one product at a time, use a csv with the data in columns and read in the rows to get the above data fields, load each item individually by row for all rows
* Image scan could potentially pre-load fields

→ See GUI windows in Diagrams section for illustration

*Required Data*

What input sources are required to produce the primary output?

* Product number

*Example*

Give an example of the format for any required input sources.

* Product number: Integer with max 12 digits
* Product name: String with max 40 characters
* Size: Range and format limited by category (US, EU, UK, and M, F)
* Quantity: Integer, no max

**7. Diagrams**

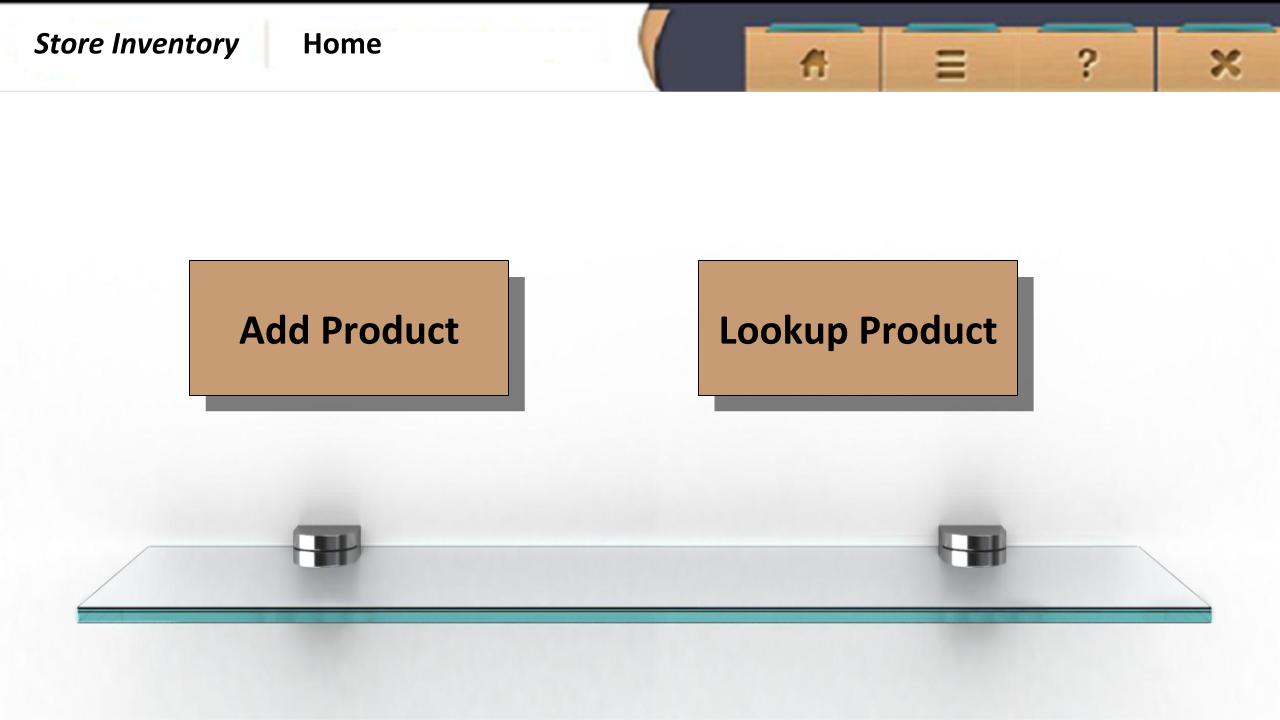
**Home screen (required)**

1. Re-consider Nielsen's [10 Heuristics of User Interface Design (Links to an external site.)Links to an external site.](https://www.nngroup.com/articles/ten-usability-heuristics/) and address as many as possible within your pages.

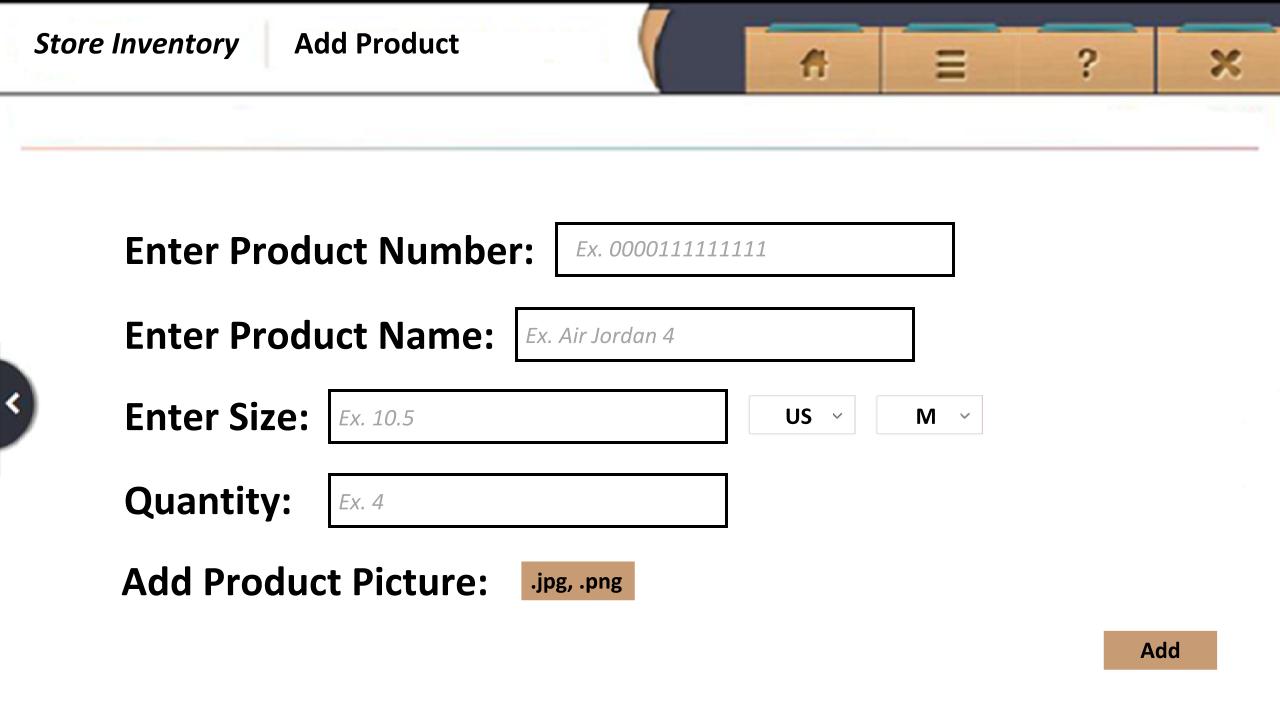
* Visibility of system status
  + Add title to each window the user is in
  + Buttons change colors when clicked
* Match between system and real world
  + Logical flow from most general search/add to most specific
  + Most recent search history for add and lookup
* User control and freedom
  + Back buttons on all windows but main page
  + Hardcode to new windows? Queue?
  + Exit app/program x on every window
  + Return to home button on all windows but main page
  + Help button on every window (specific to window)
  + Version/about/other options button
  + Delete button
    - Choose which sizes to delete and then choose how many of each size
* Consistency and standards
  + Label file types on button
  + Have greyed out example text that disappears once user clicks/taps it
* Error prevention
  + User contained between windows
  + If they try to enter wrong values, don’t let character show up and display red instruction below box
* Recognition rather than recall
  + Self-explaining button labels
  + Help button on every window explaining usage
  + Button to go back to previous window
* Flexibility and efficiency of use
  + Use buckets and efficient hash table methods
* Aesthetic and minimalist design
  + Clean design and consistent colors
  + All buttons brown
  + All type boxes transparent with grey example text
* Help users recognize, diagnose, and recover from errors
  + All errors get displayed in red below the box that was a problem
* Help and documentation
  + Help button on every window explaining usage
  + Ability to go home and close app/program from every window
  + Version/about/other options button on every window

*GUI Windows*

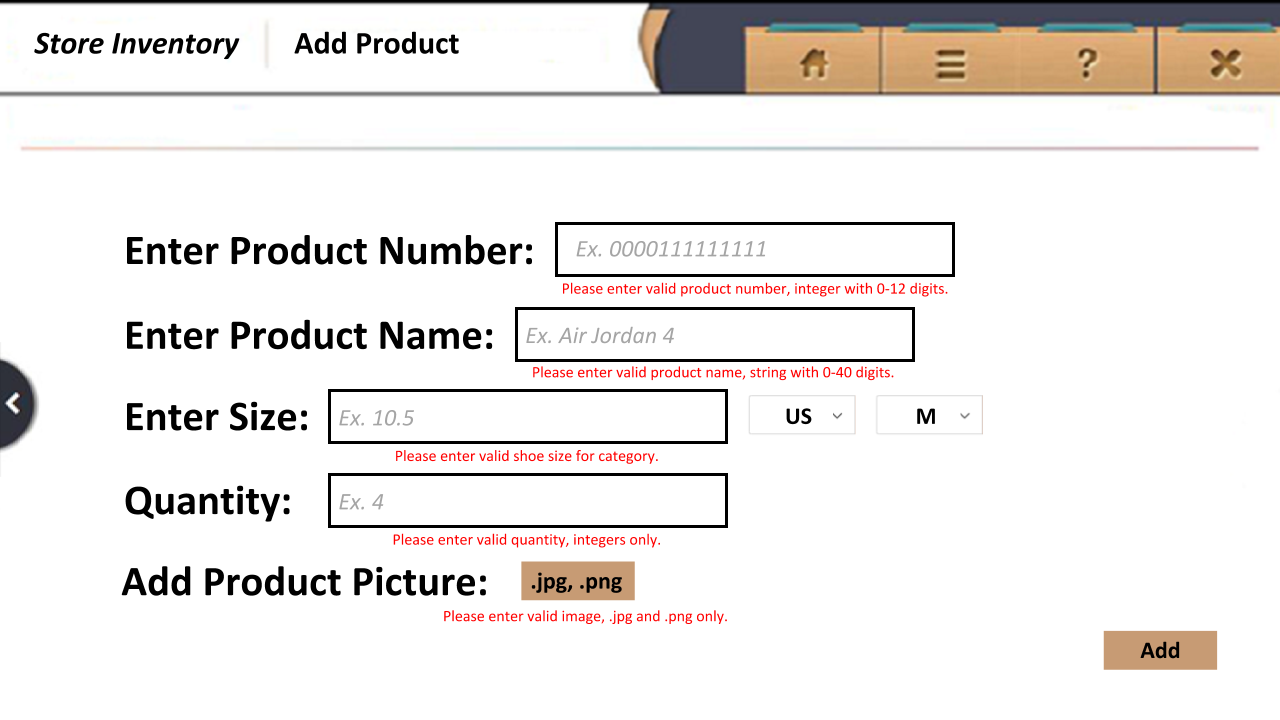
Home screen:



Add product:



Add product, need correct fields:

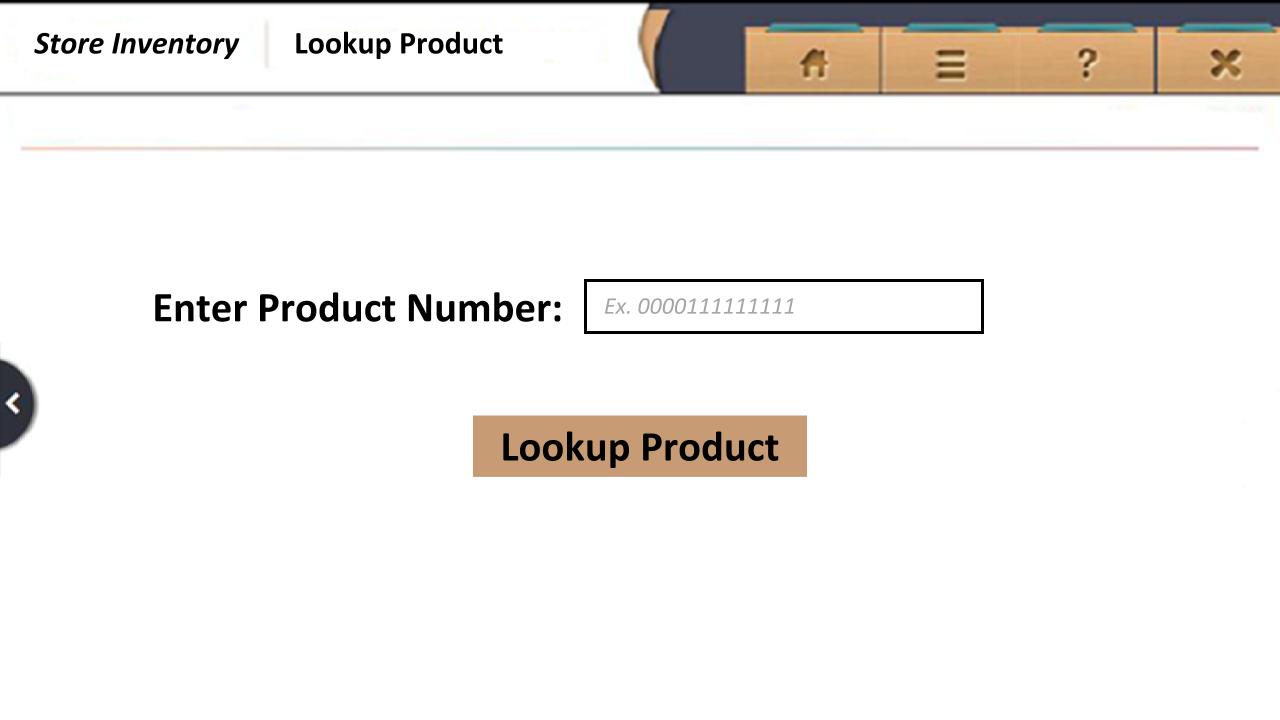


Message box, please correct fields:



Message box, product added!:



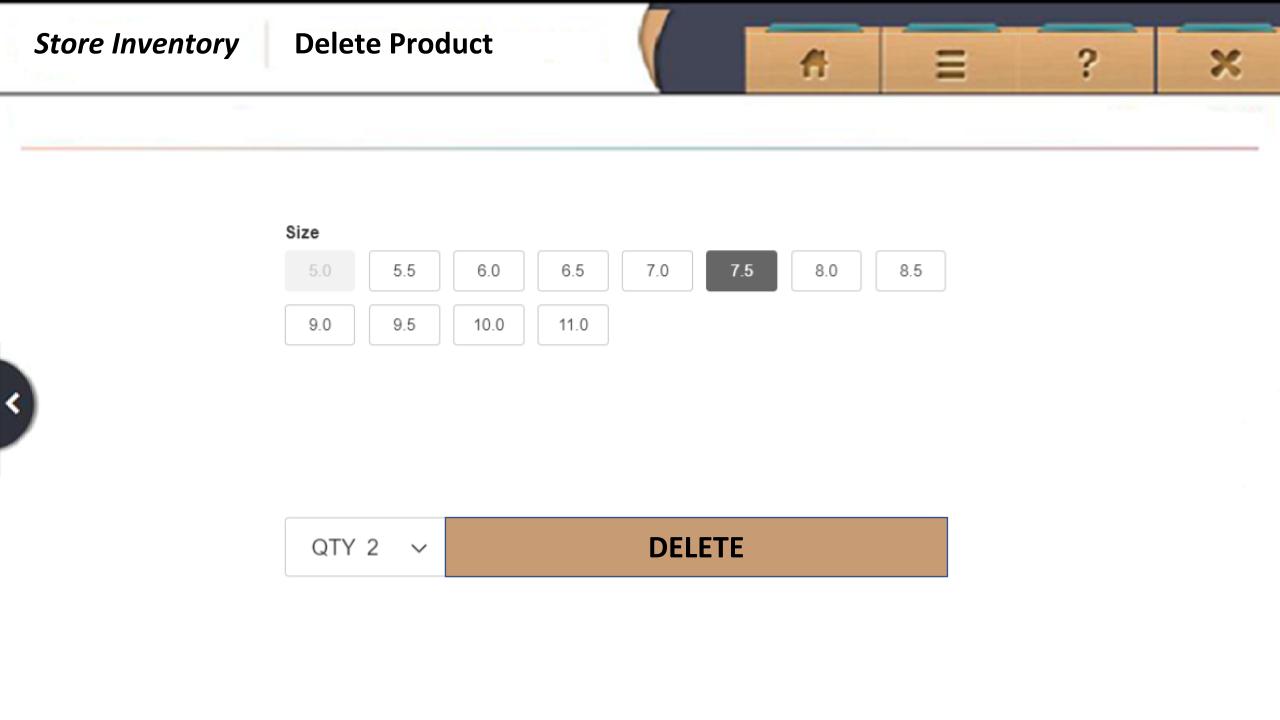
Lookup product:

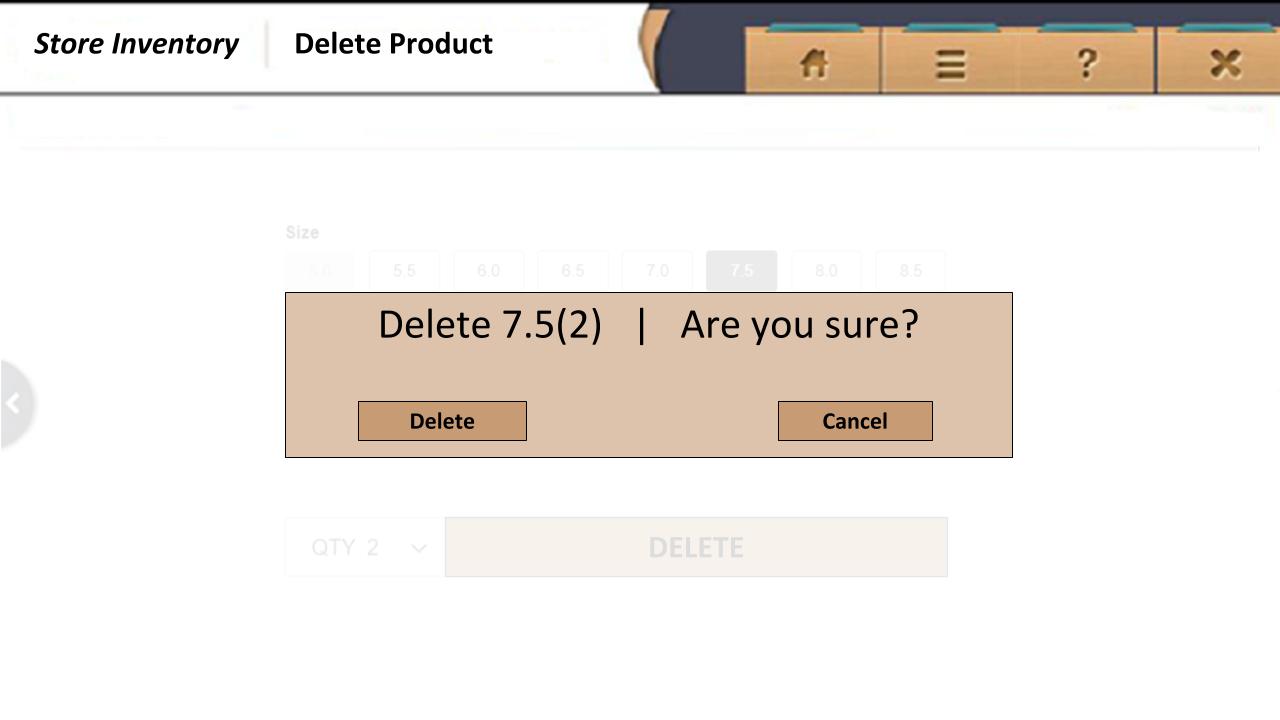
Message box, product not found, please try again:

Product info:

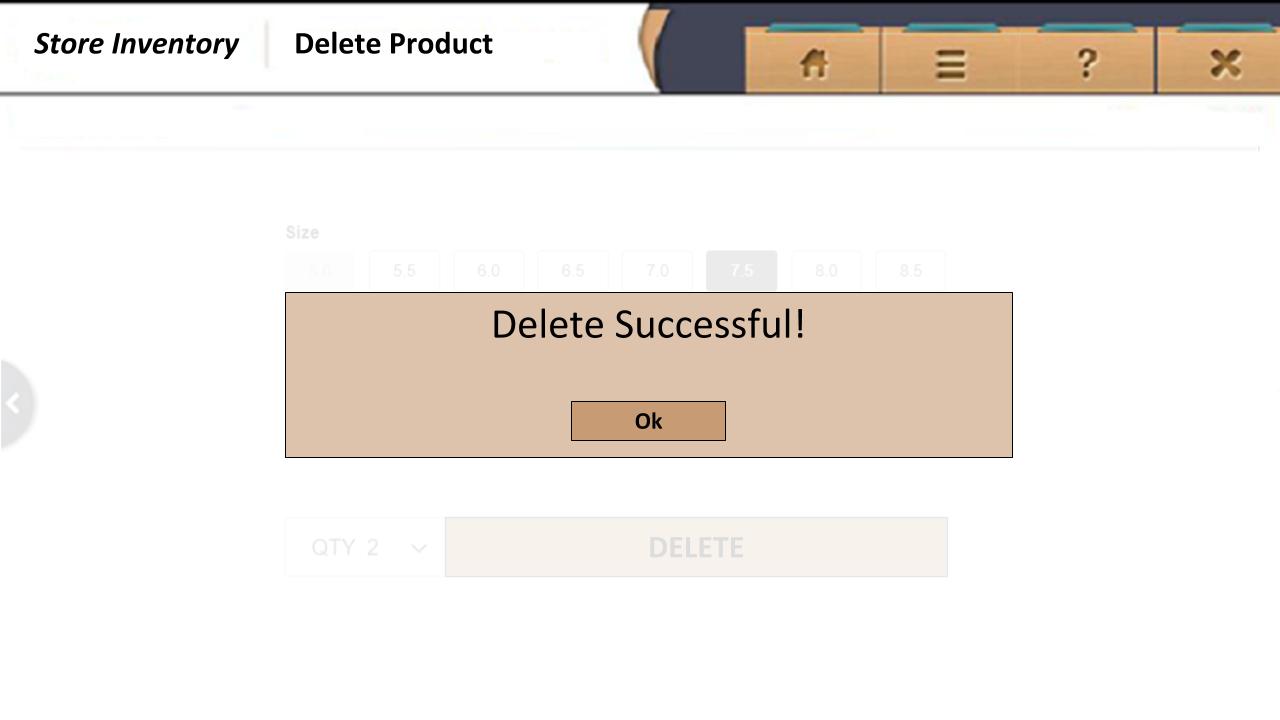


Product info (size):

Delete product:

Message box, delete (name), are you sure?

Message box, delete successful:

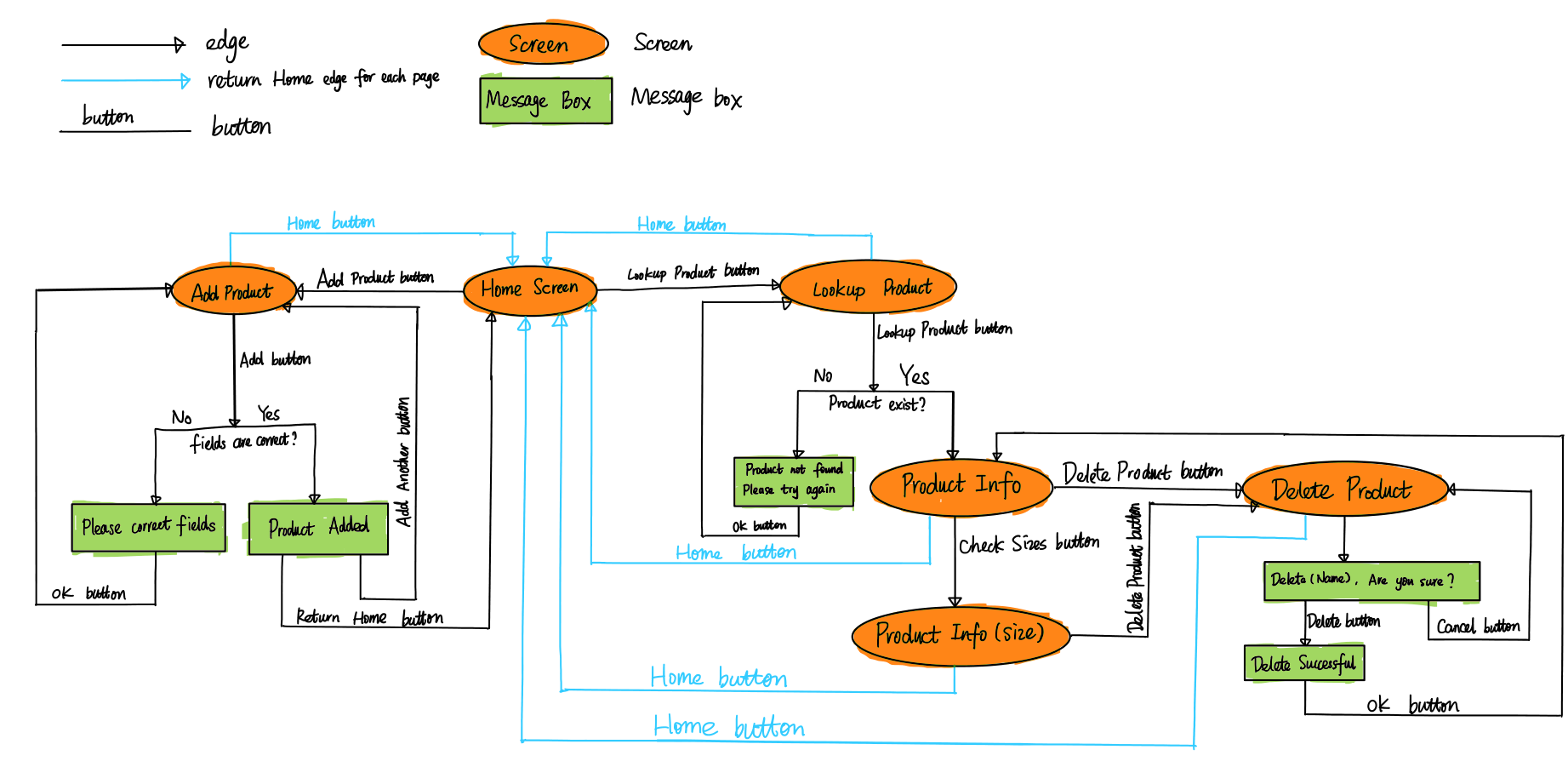


**8. Control Flow/Screen Graph**

*Control Flow:*

* Use prompted buttons for adding, looking up, or removing shoe
* Header buttons to go home, access drop-down options, access app info, and usage help always available as four buttons in upper right of screen (see example windows above)
* Option to go back on every page to previous page (\*note: would just be reverse on black button edges below)
* User contained by available options on every window
* Control Flow illustrated by Screen Graph below and GUI windows above (Diagrams section)

*Screen Graph:*



**9. Class Summary**

*AddShoe (one instance per product number input)*

- productNumber – key associated with specific shoe

- productName –name associated with product number

- size – size of shoe

- quantity – how many are currently available

- checkInputFormat - check if input data are correct format, return true if yes, otherwise false

*LookupShoe (one instance per product number input)*

- productNumber – key associated with specific shoe

- getShoe – gets information associated with key

- shoeNotFound - check if input shoe info exist in inventory, return true if yes, otherwise false

*Shoe GUI (Public Interface)*

- Shoe GUI is a Graphical User Interface which contains the signatures to the methods mousePressed, mouseReleased, isMouseOver that allows for interaction with the user.

*-mousePressed* // called each time the mouse is Pressed

*-mouseReleased()* // called each time the mouse is Released

*-isMouse over* // checks whether the mouse is over a ShoeGUI object (button)

*Button (public class that implements ShoeGUI)*

-Superclass for any button that can be added to the PApplet application. This class implements Shoe GUI interface.

*ShoeInfo (one instance per shoe)*

-hashmap – maps information associated with inputted shoe character

*MainTest (Junit 5 tests with sample “stub” data)*

- Reads shoe file and can get shoe data at expected

- Matches information from ID number to all other information

- Can input photo and retrieve name and information, vice versa.

- Other tests…

**10. Implementation Phases**

Assuming that it is not likely a team can complete entire program at one meeting, prioritize which operations must be available in each of three phases of the program development.

1. *Milestone 1* – HashTable and Shoe class implementation. Use testing methods to make sure storing information properly and input produce expected output.

- Get basic functionality working from command-Line

- Pass unit tests written with sample data.

- Confirm input and output is as expected.

2. *Milestone 2* – Create GUI interface and remaining classes.

- Implement GUI

- Add buttons to see different UI forms

- Get stakeholder approval to proceed here

3. *Milestone 3* – Manipulate features of GUI. Bug fixes.

- Add functionality with sample data.

- Test and verify expected results

4. *Future* – Create code scanner to be used as an alternate input.