ILLINOIS INSTITUTE OF TECHNOLOGY
DEPARTMENT OF COMPUTER SCIENCE

Introduction to Advanced Studies I
Course CS 401 01

CS401 SW Project for Spring 2024

Professor:
Michael Yonshik Choi, PhD

T.A.:
Tarun Sai Yakkala

Student:
Harlee Ramos

Due Date:
04/21/2024

Spring 2024

Table of Contents

Table of Figures

Table of Diagrams

Introduction

The field of inventory management in the fashion retail industry is dynamic and complex, demanding precise organization and quick access to information. A powerful Java-based program has been developed to improve operational efficiency in clothing businesses. This application, which features a user-friendly console interface, is intended to address the multiple issues of inventory management. These issues include keeping track of fluctuating inventory, quickly accessing item details, easily changing item descriptions, and efficiently sorting and arranging stock. Furthermore, the application includes a crucial data recovery feature, allowing retailers to quickly restore inventory information from backups when necessary.

This document provides a detailed overview of the program, including its functions, underlying software requirements, design diagrams, operational procedures, testing protocols, and debugging notes. It is a useful resource for developers, managers, and end users to better understand and interact with the program.

A. Problem Specification:

The application has the objective to offer a solution for inventory management, of a clothing store.

What problems are being solved?

The program addresses numerous inventory management issues clothing shops face:

- Tracking Inventory: Managing a dynamic inventory with added, updated, and removed items.
- Retrieving item details quickly using multiple methods of search.
- Data Management: Easily updating item descriptions, growing stock, and deleting unsold items.
- Sorting and organizing: Sorting inventory by order simplifies stock checks and updates.
- Data Recovery: Restoring lost inventory data from backups.

B. Software Specification: What functions are there?

The java application includes the following functionalities, facilitated through a console-based menu:

- Display Inventory: Shows the current stock by listing all clothing items.
- Add Data: Allows the addition of new clothing items to the inventory.
- Delete Data: Enables the removal of existing items from the stock.
- Update Data: Permits updating the details of an existing item.
- Sorting: Provides options to sort the inventory using different sorting algorithms for comparison purposes.
- Searching: Implements both linear and binary search to find specific items.
- Restore Data: Can reload inventory data from a file to restore previous states.
- Analyze Data: Provides basic analysis like counting the total number of items in stock.

C. Design Diagram Document

UML diagram of all classes

Diagram 1. UML Diagram of all classes of the project. Source. UML Diagram IntelliJ IDEA 2024.1.

Class CS401prj

UML diagram

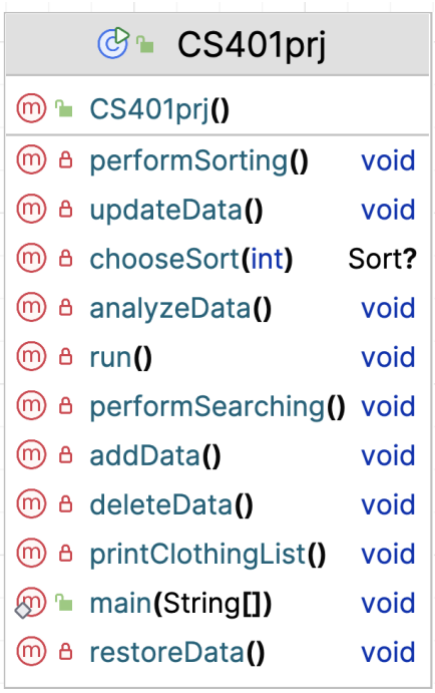


Diagram 2. UML diagram of the Class CS401prj. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

```
Class CS401prj
    Variables:
        - clothingStack: Stack for clothing items, maximum size 300

    Method main(args: Array of Strings)
        Create an instance of Main class
        Call the run method on this instance

    Method run()
        Create a Scanner object for input
        Initialize file and scanner for reading clothing data
        Try:
            Read data line by line from file, split and parse it, and push to clothingStack
        Catch FileNotFound, NumberFormat exceptions and handle them
        Initialize main interaction loop for user commands:
            Display menu and prompt for user input
            Based on choice:
                1. Display clothing stock
                2. Perform sorting
```

3. Perform searching
4. Add new data
5. Delete existing data
6. Update existing data
7. Restore data from file
8. Analyze data
9. Exit program

Method performSorting()
  Prompt user to choose two sorting algorithms
  Initialize array from clothingStack
  For each chosen algorithm:
    Sort the array using the chosen algorithm
    Measure and print execution time and memory usage
  Repopulate stack with sorted data

Method chooseSort(choice: Integer)
  Return an instance of the sorting class based on user choice

Method printClothingList()
  If stack is not empty:
    Loop through stack, pop each item, print, and push to a temporary stack
  Restore items to the main stack from the temporary stack

Method performSearching()
  Prompt user for ID to search
  Initialize array from clothingStack
  Perform linear search, binary search, and hash table search
  Print results including performance metrics

Method addData()
  Prompt user for clothing ID and description
  Create new clothing item and push it to the stack

Method deleteData()
  Prompt user for clothing ID
  Temporarily transfer items to another stack, excluding the item to delete
  Restore items to the main stack, print deletion result

Method updateData()
  Prompt user for clothing ID and new description

Temporarily transfer items to another stack, update item if found

Restore items to the main stack, print update result

Method restoreData()

Clear the stack

Re-read data from file and repopulate the stack

Print the restored list of clothing items

Method analyzeData()

Print the count of clothing items in the stack

End Class

Interface: StackInterface

UML diagram

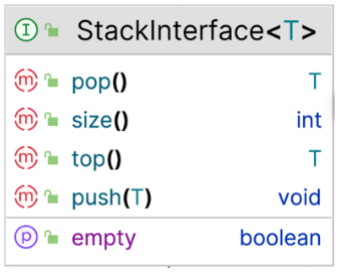


Diagram 3. UML diagram of the Interface: StackInterface. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

```
Interface: StackInterface
  Generic Type: T

  Operations:
    // Pop the top element from the stack and return it.
    // Throws StackUnderflowException if the stack is empty.
    Method pop:
      Return Type: T
      Steps:
        Check if stack is empty:
          If true, throw StackUnderflowException
          Otherwise, remove and return the top element

    // Return the top element of the stack without removing it.
    // Throws StackUnderflowException if the stack is empty.
    Method top:
      Return Type: T
      Steps:
        Check if stack is empty:
          If true, throw StackUnderflowException
          Otherwise, return the top element

    // Check if the stack is empty.
    // Return true if empty, false otherwise.
    Method isEmpty:
      Return Type: boolean
      Steps:
        Return true if stack is empty
        Otherwise, return false
```

// Add an element to the top of the stack.

Method push:

  Input: element (T)

  Steps:

    Place the element at the top of the stack

// Return the number of elements in the stack.

Method size:

  Return Type: int

  Steps:

    Return the count of elements in the stack

Class ArrayStack

UML diagram

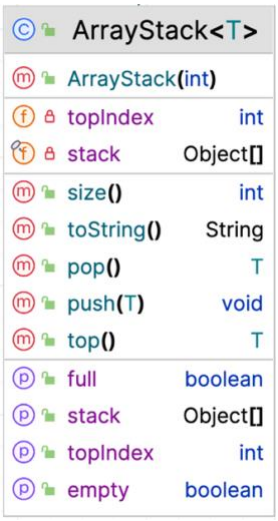


Diagram 4. UML diagram of the class ArrayStack. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode: ArrayStack Class

```
// Class declaration for ArrayStack implementing a StackInterface.
Class ArrayStack<T>
    // Private fields
    stack: Array of Object
    topIndex: Integer initialized to -1

    // Constructor to initialize the stack with a specific maximum size.
    Constructor(maxSize: Integer)
        stack = new Array of Object with size maxSize

    // Method to add an item to the top of the stack.
    Method push(item: T)
        If stack is not full
            Increment topIndex
            stack[topIndex] = item
        Else
            Throw new StackOverflowException with message "Push attempted on a full stack."

    // Method to remove and return the top item of the stack.
    Method pop: T
```

```
    If stack is not empty
        item: T = stack[topIndex] (cast to T)
        stack[topIndex] = null  // Clear the reference
        Decrement topIndex
        Return item
    Else
        Throw new StackUnderflowException with message "Pop attempted on an empty stack."

// Method to get the top item of the stack without removing it.
Method top: T
    If stack is not empty
        Return stack[topIndex] (cast to T)
    Else
        Throw new StackUnderflowException with message "Top attempted on an empty stack."

// Method to check if the stack is empty.
Method isEmpty: Boolean
    Return topIndex == -1  // True if topIndex is -1, indicating empty stack

// Method to return the number of items in the stack.
Method size: Integer
    Return topIndex + 1  // Size is topIndex + 1 because topIndex is zero-based.

// Helper method to check if the stack is full.
Method isFull: Boolean
    Return topIndex == stack.length - 1  // True if topIndex is at the last position of the array

// Method to get the underlying array of the stack. For internal or testing use.
Method getStack: Array of Object
    Return stack

// Getter for topIndex.
Method getTopIndex: Integer
    Return topIndex

// Setter for topIndex. This allows changing the topIndex, usually for testing or internal adjustments.
```

```
Method setTopIndex(topIndex: Integer)
    This.topIndex = topIndex

// Overridden toString method for a string representation of the stack's state.
Method toString: String
    Return "ArrayStack{" + "stack=" + Arrays.toString(stack) + ", topIndex=" + topIndex + '}'
```

StackOverflowException

UML diagram

Diagram 5. UML diagram of the class StackOverflowException. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: StackOverflowException extends RuntimeException

Constructors:

Constructor: StackOverflowException

Description: Initializes a new StackOverflowException object without a specific message.

Steps:

Call the superclass (RuntimeException) constructor without any arguments.

Constructor: StackOverflowException

Input: message (string)

Description: Initializes a new StackOverflowException object with a custom message describing the exception.

Steps:

Call the superclass (RuntimeException) constructor with the message argument.

StackUnderflowException

UML diagram

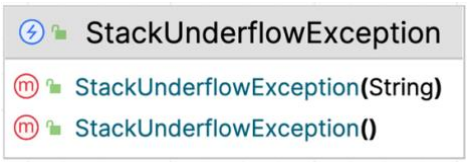


Diagram 6. UML diagram of the class StackUnderflowException. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: StackUnderflowException extends RuntimeException

Constructors:

Constructor: StackUnderflowException

Description: Initializes a new StackUnderflowException object without a specific message.

Steps:

Call the superclass (RuntimeException) constructor without any arguments.

Constructor: StackUnderflowException

Input: message (string)

Description: Initializes a new StackUnderflowException object with a custom message describing the exception.

Steps:

Call the superclass (RuntimeException) constructor with the message argument.

Class Sort

UML diagram

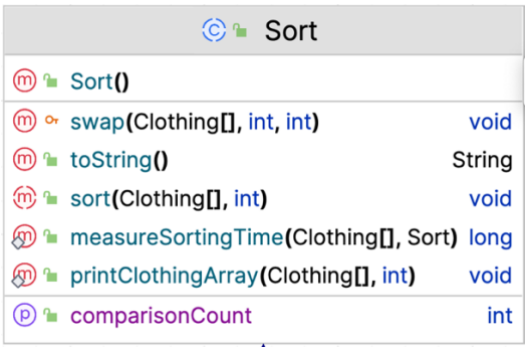


Diagram 7. UML diagram of the class Sort. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Abstract Class: Sort

Abstract Method: sort

Input: clothingArray (array of Clothing objects), clothingCount (integer)

Description: Sorts the clothingArray up to the specified clothingCount. Implementation details are dependent on the subclass.

Abstract Method: getComparisonCount

Output: Returns the number of comparisons made during the sorting process.

Method: swap

Input: array (array of Clothing objects), index1 (integer), index2 (integer)

Description: Swaps two elements at index1 and index2 in the array.

Steps:

Store the element at index1 in a temporary variable.

Set the element at index1 to the element at index2.

Set the element at index2 to the temporary variable.

Static Method: measureSortingTime

Input: clothingArray (array of Clothing objects), sortAlgorithm (Sort object)

Output: Execution time of the sorting algorithm in nanoseconds.

Description: Measures the execution time of a sorting algorithm using a copy of clothingArray.

Steps:

Copy the clothingArray to prevent modification of the original array.

Record the current time in nanoseconds.

Call the sort method of sortAlgorithm using the copy of the array.

Record the current time in nanoseconds after sorting is complete.

Calculate and return the difference in time (end time - start time).

Static Method: printClothingArray

Input: array (array of Clothing objects), count (integer)

Description: Prints each element of the array up to the specified count.

Steps:

Loop through the array up to the specified count.

If the current element is not null, print the element.

Method: toString

Output: Returns a string representation of the Sort object.

Description: Returns a generic string "Sort{}" for any Sort object.

Class BubbleSort

UML diagram

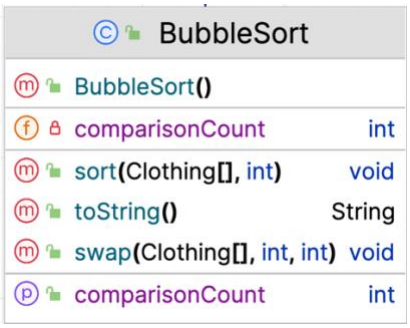


Diagram 8. UML diagram of the class BubbleSort. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: BubbleSort extends Sort

Variable:

comparisonCount (integer) initialized to 0 // stores the number of comparisons made during the sort

Method: sort

Input: clothingArray (array of Clothing objects), clothingCount (integer)

Description: Sorts the clothingArray using the bubble sort algorithm up to the specified clothingCount.

Steps:

Initialize swapped (boolean) to false

Repeat:

Set swapped to false

For i from 1 to clothingCount - 1

Increment comparisonCount

If the ID of clothingArray[i-1] > the ID of clothingArray[i]

Swap elements at indices i-1 and i in clothingArray

Set swapped to true

Until swapped is false

Method: swap

Input: array (array of Clothing objects), index1 (integer), index2 (integer)

Description: Swaps two elements at index1 and index2 in the array.

Steps:

Store the element at index1 in a temporary variable

Set the element at index1 to the element at index2

Set the element at index2 to the temporary variable

Method: getComparisonCount

Output: Returns the value of comparisonCount

Description: Retrieves the total number of comparisons made during the sort.

Method: setComparisonCount

Input: comparisonCount (integer)

Description: Sets the comparisonCount to the provided value.

Method: toString

Output: Returns a string representation of the BubbleSort object.

Description: Returns a string indicating the type of sort and the current comparison count.

Class HeapSort

UML diagram

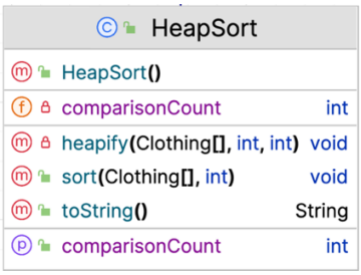


Diagram 9. UML diagram of the class HeapSort. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

```
HeapSort Class
// Class declaration for HeapSort extending a base class Sort.
Class HeapSort extends Sort
    // Private instance variable for tracking the number of comparisons.
    comparisonCount: Integer initialized to 0

    // Override the sort method from Sort class to provide heap sort implementation.
    Method sort(clothingArray: Array of Clothing, clothingCount: Integer)
        // Build the initial heap.
        For i from (clothingCount / 2 - 1) down to 0
            heapify(clothingArray, clothingCount, i)

        // Extract elements from the heap and rebuild the heap each time.
        For i from clothingCount - 1 down to 1
            // Swap the root of the heap with the last element of the unsorted section.
            swap(clothingArray, 0, i)
            // Rebuild the heap ignoring the last i elements.
            heapify(clothingArray, i, 0)

    // Private method to ensure the subtree rooted at 'i' is a heap.
    Method heapify(clothingArray: Array of Clothing, heapSize: Integer, i: Integer)
        largest: Integer = i  // Assume root is the largest.
        left: Integer = 2 * i + 1  // Calculate left child index.
        right: Integer = 2 * i + 2  // Calculate right child index.

        // Check if the left child exists and is larger than the root.
        If left < heapSize
            Increment comparisonCount
            If clothingArray[left].getId() > clothingArray[largest].getId()
                largest = left

        // Check if the right child exists and is larger than the current largest.
```

```
    If right < heapSize
        Increment comparisonCount
        If clothingArray[right].getId() > clothingArray[largest].getId()
            largest = right

    // If largest is not the root, swap with root and continue heapifying.
    If largest != i
        swap(clothingArray, i, largest)
        heapify(clothingArray, heapSize, largest)

// Method to return the number of comparisons made during sorting.
Method getComparisonCount: Integer
    Return comparisonCount

// Setter method for the comparison count.
Method setComparisonCount(comparisonCount: Integer)
    this.comparisonCount = comparisonCount

// Override the toString method for string representation of the object state.
Method toString: String
    Return "HeapSort{" + "comparisonCount=" + comparisonCount + '}'
```

Class InsertionSort

UML diagram

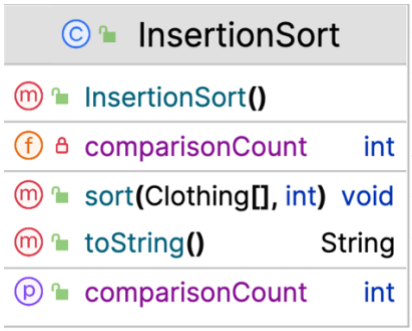


Diagram 10. UML diagram of the class InsertionSort. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: InsertionSort extends Sort

```
Variable:
  comparisonCount (integer) initialized to 0  // stores the number of comparisons made during the sort

Method: sort
  Input: clothingArray (array of Clothing objects), clothingCount (integer)
  Description: Sorts the clothingArray using the insertion sort algorithm up to the specified clothingCount.
  Steps:
    For each i from 1 to clothingCount - 1
      Set key to clothingArray[i]
      Initialize j to i - 1
      While j is greater than or equal to 0
        Increment comparisonCount
        If ID of clothingArray[j] is greater than ID of key
          Move clothingArray[j] one position to the right to clothingArray[j + 1]
          Decrement j
        Else
          Break the loop
      Set clothingArray[j + 1] to key

Method: getComparisonCount
  Output: Returns the number of comparisons made during the sort
  Description: Retrieves the total number of comparisons made during the sort.

Method: setComparisonCount
  Input: comparisonCount (integer)
  Description: Sets the comparisonCount to the provided value.
```

Method: toString

Output: Returns a string representation of the InsertionSort object.

Description: Returns a string indicating the type of sort and the current comparison count.

Class MergeSort

UML diagram

Diagram 11. UML diagram of the class MergeSort. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: MergeSort extends Sort

Variable:

comparisonCount (integer) initialized to 0 // stores the number of comparisons made during the sort

Method: sort

Input: clothingArray (array of Clothing objects), clothingCount (integer)

Description: Sorts the clothingArray using the merge sort algorithm up to the specified clothingCount.

Steps:

If clothingCount is less than 2

Return (no need to sort)

Calculate mid as clothingCount / 2

Create two subarrays, left (size mid) and right (size clothingCount - mid)

Copy the first half of clothingArray to left

Copy the second half of clothingArray to right

Recursively sort the left array

Recursively sort the right array

Merge the two sorted arrays back into clothingArray

Method: merge

Input: clothingArray (array of Clothing objects), left (array of Clothing objects), right (array of Clothing objects), leftSize (integer), rightSize (integer)

Description: Merges two sorted subarrays, left and right, into clothingArray.

Steps:

Initialize indices i, j, k to 0 for left, right, and clothingArray respectively

While both i < leftSize and j < rightSize

Increment comparisonCount

If ID of left[i] is less than or equal to ID of right[j]

Set clothingArray[k] to left[i] and increment i and k

Else

Set clothingArray[k] to right[j] and increment j and k

Copy remaining elements from left to clothingArray if any

Copy remaining elements from right to clothingArray if any

Method: getComparisonCount

Output: Returns the number of comparisons made during the sort

Description: Retrieves the total number of comparisons made during the sort.

Method: setComparisonCount

Input: comparisonCount (integer)

Description: Sets the comparisonCount to the provided value.

Method: toString

Output: Returns a string representation of the MergeSort object.

Description: Returns a string indicating the type of sort and the current comparison count.

Class QuickSort

UML diagram

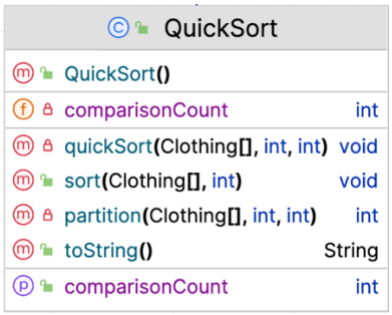


Diagram 12. UML diagram of the class QuickSort. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: QuickSort extends Sort

Variable:

comparisonCount (integer) initialized to 0 // stores the number of comparisons made during the sort

Method: sort

Input: clothingArray (array of Clothing objects), clothingCount (integer)

Description: Initiates the quick sort algorithm on the clothingArray.

Steps:

Call quickSort method with parameters (clothingArray, 0, clothingCount - 1)

Method: quickSort

Input: array (array of Clothing objects), low (integer), high (integer)

Description: Recursively sorts the array using the quick sort method.

Steps:

If low is less than high

Determine the pivotIndex using the partition method

Recursively call quickSort on the subarray from low to pivotIndex - 1

Recursively call quickSort on the subarray from pivotIndex + 1 to high

Method: partition

Input: array (array of Clothing objects), low (integer), high (integer)

Description: Partitions the array into two parts; those less than the pivot and those greater than the pivot, and swaps elements to ensure this order.

Output: The index of the pivot element after partitioning

Steps:

Set pivot to the element at index high

Initialize i to (low - 1)

For each j from low to high - 1

Increment comparisonCount

If ID of array[j] is less than or equal to ID of pivot

Increment i

Swap elements at indices i and j in array

Swap elements at indices i + 1 and high in array

Return i + 1

Method: getComparisonCount

Output: Returns the number of comparisons made during the sort

Description: Retrieves the total number of comparisons made during the sort.

Method: setComparisonCount

Input: comparisonCount (integer)

Description: Sets the comparisonCount to the provided value.

Method: toString

Output: Returns a string representation of the QuickSort object.

Description: Returns a string indicating the type of sort and the current comparison count.

Class SelectionSort

UML diagram

Diagram 13. UML diagram of the class SelectionSort. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: SelectionSort extends Sort

Variable:

comparisonCount (integer) initialized to 0 // stores the number of comparisons made during the sort

Method: sort

Input: clothingArray (array of Clothing objects), clothingCount (integer)

Description: Sorts the clothingArray using the selection sort algorithm up to the specified clothingCount.

Steps:

For i from 0 to clothingCount - 2

Set minIndex to i

For j from i + 1 to clothingCount - 1

Increment comparisonCount

If ID of clothingArray[j] is less than ID of clothingArray[minIndex]

Update minIndex to j

Swap elements at indices minIndex and i in clothingArray

Method: getComparisonCount

Output: Returns the number of comparisons made during the sort

Description: Retrieves the total number of comparisons made during the sort.

Method: setComparisonCount

Input: comparisonCount (integer)

Description: Sets the comparisonCount to the provided value.

Method: toString

Output: Returns a string representation of the SelectionSort object.

Description: Returns a string indicating the type of sort and the current comparison count.

Class LinearSearch

UML diagram

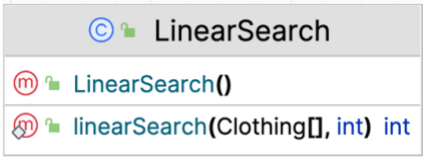


Diagram 14. UML diagram of the class LinearSearch. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: LinearSearch

Method: linearSearch

Input: array (array of Clothing objects), id (integer)

Description: Searches for a Clothing object in the provided array by ID and returns the index of the first occurrence.

Output: integer index of the found item, or -1 if the item is not found

Steps:

For each index i from 0 to the length of array - 1

If the Clothing object at index i is not null and its ID matches the given id

Return i (index of the found item)

Return -1 (indicates item not found)

Class HMap

UML diagram

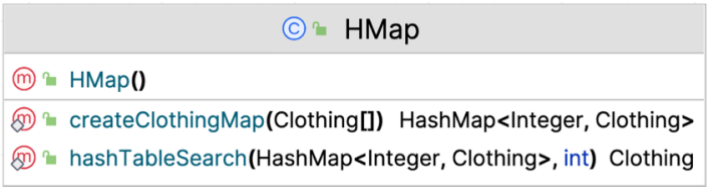


Diagram 15. UML diagram of the class HMap. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

```
CLASS HMap

    // Define a static method createClothingMap that takes an array of Clothing objects and returns a HashMap
    METHOD createClothingMap(array of Clothing)
        // Initialize an empty HashMap for storing Clothing objects with Integer keys
        DECLARE map as HashMap of Integer keys and Clothing values

        // Loop through each Clothing object in the array
        FOR EACH clothing IN array
            // Check if the clothing object is not null
            IF clothing IS NOT NULL THEN
                // Add the clothing object to the map using its ID as the key
                map.PUT(clothing.getId(), clothing)
            END IF
        END FOR

        // Return the populated HashMap
        RETURN map
    END METHOD

    // Define a static method hashTableSearch that searches for a Clothing object by ID in the provided HashMap and returns the number of iterations
    METHOD hashTableSearch(map as HashMap of Integer to Clothing, id as Integer)
        // Initialize the iteration count as 1
        DECLARE iterations as 1

        // Attempt to retrieve the Clothing object by its ID from the map
        DECLARE found as Clothing
        found = map.GET(id)

        // Check if the Clothing object was found
```

```
        IF found IS NOT NULL THEN
            // If found, return the number of iterations
            RETURN iterations
        ELSE
            // If not found, return -1 indicating a failed search
            RETURN -1
        END IF
    END METHOD

END CLASS
```

Class BinarySearch

UML diagram

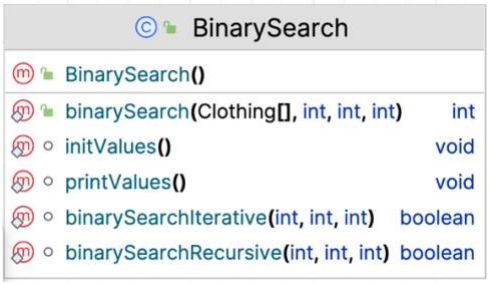


Diagram 16. UML diagram of the class BinarySearch. Source. UML Diagram IntelliJ IDEA 2024.1.

Pseudocode

Class: BinarySearch

Constants:

SIZE = 100 // fixed size of the array

Variables:

values = new array of integers[SIZE] // array to hold integers

Method: initValues

Description: Initialize the array 'values' with non-decreasing random integers starting from 0.

Steps:

Initialize a random number generator

Set values[0] = 0

For index from 1 to SIZE-1

Increment values[index] by a random number (0 to 3) added to values[index-1]

Method: printValues

Description: Prints all the integers in 'values' array, formatted to three digits, in rows of 10.

Steps:

Initialize a decimal format for three digits

Print "The values array is:"

For each index from 0 to SIZE-1

Get value at index

Format and print value

If (index + 1) % 10 == 0 then print newline

Method: binarySearchIterative

Input: target (integer), first (integer), last (integer)

Description: Perform a binary search iteratively to find 'target' between indices 'first' and 'last'.

Output: true if target is found, otherwise false

Steps:
  While first <= last
    Calculate midpoint as (first + last) / 2
    If target equals values[midpoint]
      Return true
    Else if target > values[midpoint]
      Update first to midpoint + 1
    Else
      Update last to midpoint - 1
  Return false

Method: binarySearchRecursive
  Input: target (integer), first (integer), last (integer)
  Description: Perform a binary search recursively to find 'target' between indices 'first' and 'last'.
  Output: true if target is found, otherwise false
  Steps:
    If first > last
      Return false
    Calculate midpoint as (first + last) / 2
    If target equals values[midpoint]
      Return true
    Else if target > values[midpoint]
      Recursively search from midpoint + 1 to last
    Else
      Recursively search from first to midpoint - 1

Method: binarySearch (for Clothing array)
  Input: clothingArray (array of Clothing), i (integer), j (integer), id (integer)
  Description: Placeholder for binary search in a Clothing array (implementation not provided)
  Output: 0 (as placeholder)

D. Operational document

Running the Java application

This terminal image shows the Java jar file running. To run it, place the jar file in the same location as the text file on the disk and add lines.

```
cd /Users/harleeliz/Desktop/

java -jar CS401prj_HR.jar
```

Figure 1. Main menu of the java application. Source Mac OS Terminal Version 2.14 (453).

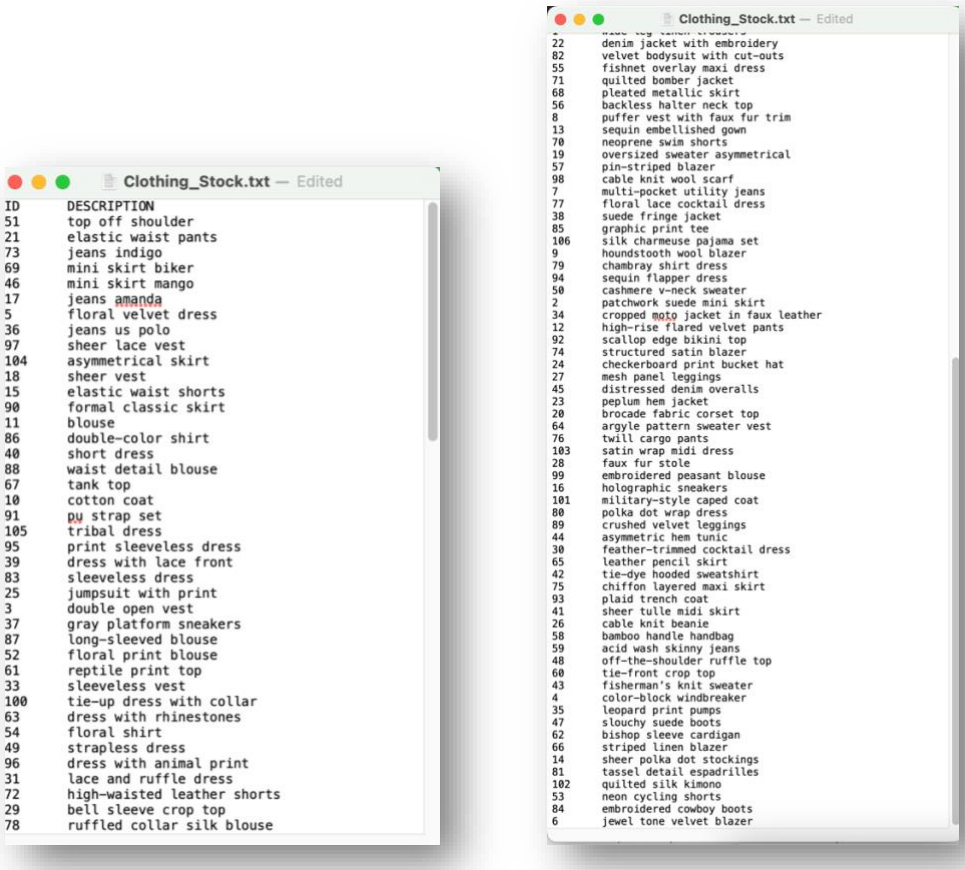


Figure 2. Contents of the text file Clothing_Stock.txt. Source Mac OS Terminal Version 2.14 (453).

Selection 1: Display clothing stock.

In this selection, the application will print on the screen the current contents of the Clothing_Stock.txt file. The user must input an integer number.

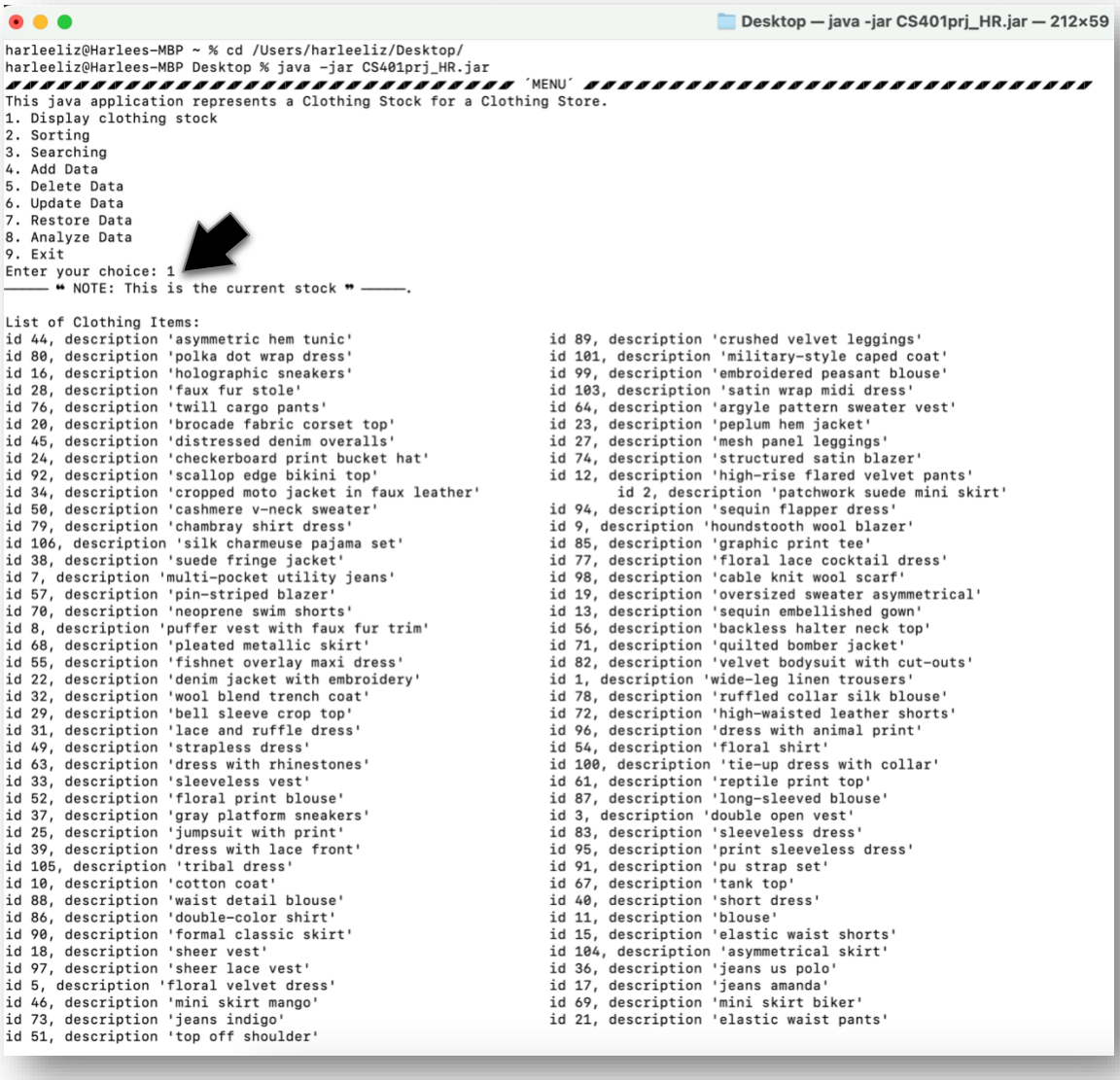


Figure 3. Selection 1: Display clothing stock. Source Mac OS Terminal Version 2.14 (453).

Selection 2. Sorting,

This option will perform six types of searching; the user will be presented with a menu to choose the type of sorting algorithm to be applied to the data.

1: Bubble Sort, 2: Heap Sort, 3: Insertion Sort, 4: Merge Sort, 5: Quick Sort, 6: Selection Sort
The user must input an integer number.

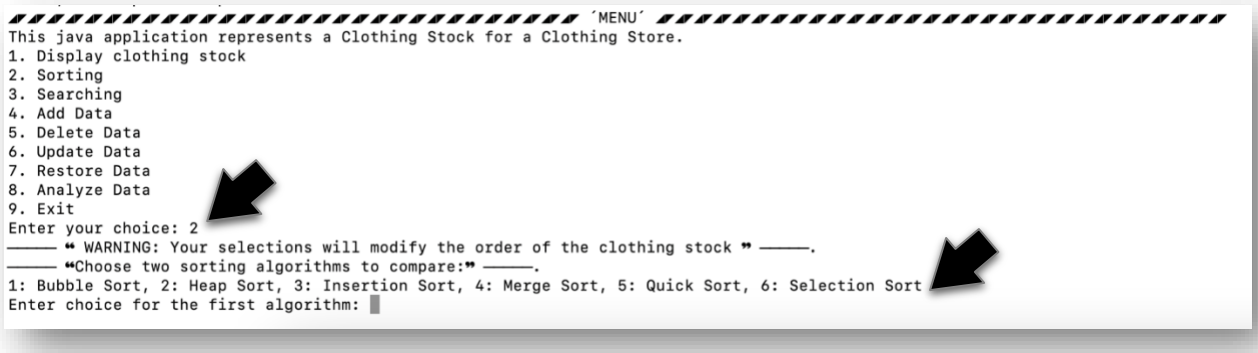


Figure 4. Selection 2. Sorting. Source Mac OS Terminal Version 2.14 (453).

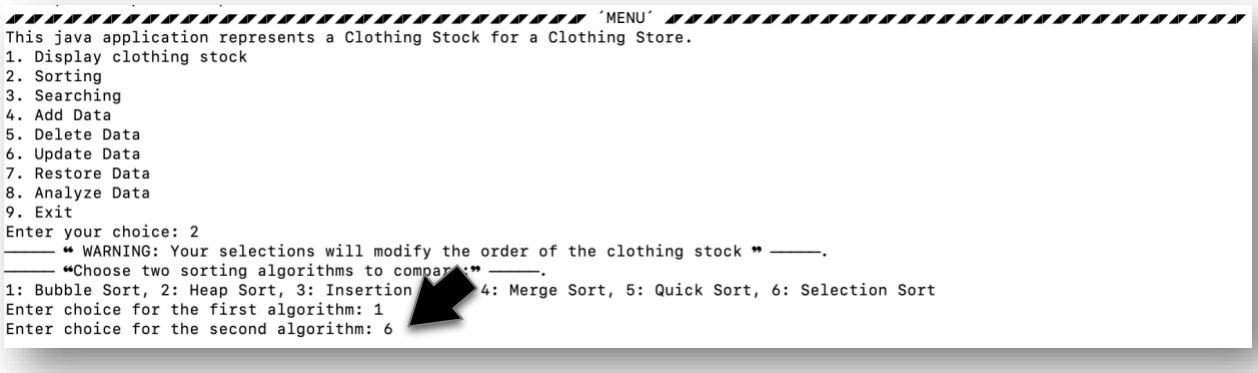


Figure 5. Selection 2. Sorting, user input for the two sorting options

Print of the results:

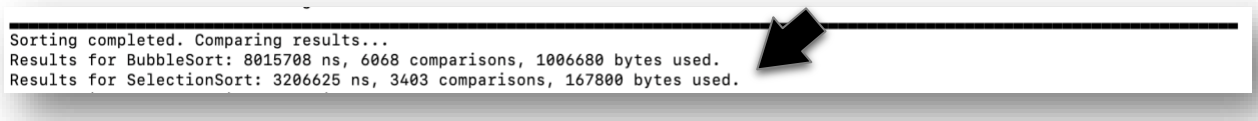


Figure 6. Selection 2. Sorting, print of the results. Source Mac OS Terminal Version 2.14 (453).

Sorted list for the first algorithm.

```
Sorted list from the first algorithm:
id 1, description 'wide-leg linen trousers'          id 2, description 'patchwork suede mini skirt'
id 3, description 'double open vest'                 id 5, description 'floral velvet dress'
id 7, description 'multi-pocket utility jeans'       id 8, description 'puffer vest with faux fur trim'
id 9, description 'houndstooth wool blazer'          id 10, description 'cotton coat'
id 11, description 'blouse'                          id 12, description 'high-rise flared velvet pants'
id 13, description 'sequin embellished gown'         id 15, description 'elastic waist shorts'
id 16, description 'holographic sneakers'            id 17, description 'jeans amanda'
id 18, description 'sheer vest'                      id 19, description 'oversized sweater asymmetrical'
id 20, description 'brocade fabric corset top'       id 21, description 'elastic waist pants'
id 22, description 'denim jacket with embroidery'    id 23, description 'peplum hem jacket'
id 24, description 'checkerboard print bucket hat'   id 25, description 'jumpsuit with print'
id 27, description 'mesh panel leggings'             id 28, description 'faux fur stole'
id 29, description 'bell sleeve crop top'            id 31, description 'lace and ruffle dress'
id 32, description 'wool blend trench coat'          id 33, description 'sleeveless vest'
id 34, description 'cropped moto jacket in faux leather' id 36, description 'jeans us polo'
id 37, description 'gray platform sneakers'          id 38, description 'suede fringe jacket'
id 39, description 'dress with lace front'           id 40, description 'short dress'
id 44, description 'asymmetric hem tunic'            id 45, description 'distressed denim overalls'
id 46, description 'mini skirt mango'                id 49, description 'strapless dress'
id 50, description 'cashmere v-neck sweater'         id 51, description 'top off shoulder'
id 52, description 'floral print blouse'             id 54, description 'floral shirt'
id 55, description 'fishnet overlay maxi dress'      id 56, description 'backless halter neck top'
id 57, description 'pin-striped blazer'              id 61, description 'reptile print top'
id 63, description 'dress with rhinestones'          id 64, description 'argyle pattern sweater vest'
id 67, description 'tank top'                        id 68, description 'pleated metallic skirt'
id 69, description 'mini skirt biker'                id 70, description 'neoprene swim shorts'
id 71, description 'quilted bomber jacket'           id 72, description 'high-waisted leather shorts'
id 73, description 'jeans indigo'                    id 74, description 'structured satin blazer'
id 76, description 'twill cargo pants'               id 77, description 'floral lace cocktail dress'
id 78, description 'ruffled collar silk blouse'      id 79, description 'chambray shirt dress'
id 80, description 'polka dot wrap dress'            id 82, description 'velvet bodysuit with cut-outs'
id 83, description 'sleeveless dress'                id 85, description 'graphic print tee'
id 86, description 'double-color shirt'              id 87, description 'long-sleeved blouse'
id 88, description 'waist detail blouse'             id 89, description 'crushed velvet leggings'
id 90, description 'formal classic skirt'            id 91, description 'pu strap set'
id 92, description 'scallop edge bikini top'         id 94, description 'sequin flapper dress'
id 95, description 'print sleeveless dress'          id 96, description 'dress with animal print'
id 97, description 'sheer lace vest'                 id 98, description 'cable knit wool scarf'
id 99, description 'embroidered peasant blouse'      id 100, description 'tie-up dress with collar'
id 101, description 'military-style caped coat'      id 103, description 'satin wrap midi dress'
id 104, description 'asymmetrical skirt'             id 105, description 'tribal dress'
id 106, description 'silk charmeuse pajama set'
```

Figure 7. Sorted list for the first algorithm. Source Mac OS Terminal Version 2.14 (453).

Sorted list for the second algorithm.

```
Sorted list from the second algorithm:
id 1, description 'wide-leg linen trousers'          id 2, description 'patchwork suede mini skirt'
id 3, description 'double open vest'                 id 5, description 'floral velvet dress'
id 7, description 'multi-pocket utility jeans'       id 8, description 'puffer vest with faux fur trim'
id 9, description 'houndstooth wool blazer'          id 10, description 'cotton coat'
id 11, description 'blouse'                          id 12, description 'high-rise flared velvet pants'
id 13, description 'sequin embellished gown'         id 15, description 'elastic waist shorts'
id 16, description 'holographic sneakers'            id 17, description 'jeans amanda'
id 18, description 'sheer vest'                      id 19, description 'oversized sweater asymmetrical'
id 20, description 'brocade fabric corset top'       id 21, description 'elastic waist pants'
id 22, description 'denim jacket with embroidery'    id 23, description 'peplum hem jacket'
id 24, description 'checkerboard print bucket hat'   id 25, description 'jumpsuit with print'
id 27, description 'mesh panel leggings'             id 28, description 'faux fur stole'
id 29, description 'bell sleeve crop top'            id 31, description 'lace and ruffle dress'
id 32, description 'wool blend trench coat'          id 33, description 'sleeveless vest'
id 34, description 'cropped moto jacket in faux leather' id 36, description 'jeans us polo'
id 37, description 'gray platform sneakers'          id 38, description 'suede fringe jacket'
id 39, description 'dress with lace front'           id 40, description 'short dress'
id 44, description 'asymmetric hem tunic'            id 45, description 'distressed denim overalls'
id 46, description 'mini skirt mango'                id 49, description 'strapless dress'
id 50, description 'cashmere v-neck sweater'         id 51, description 'top off shoulder'
id 52, description 'floral print blouse'             id 54, description 'floral shirt'
id 55, description 'fishnet overlay maxi dress'      id 56, description 'backless halter neck top'
id 57, description 'pin-striped blazer'              id 61, description 'reptile print top'
id 63, description 'dress with rhinestones'          id 64, description 'argyle pattern sweater vest'
id 67, description 'tank top'                        id 68, description 'pleated metallic skirt'
id 69, description 'mini skirt biker'                id 70, description 'neoprene swim shorts'
id 71, description 'quilted bomber jacket'           id 72, description 'high-waisted leather shorts'
id 73, description 'jeans indigo'                    id 74, description 'structured satin blazer'
id 76, description 'twill cargo pants'               id 77, description 'floral lace cocktail dress'
id 78, description 'ruffled collar silk blouse'      id 79, description 'chambray shirt dress'
id 80, description 'polka dot wrap dress'            id 82, description 'velvet bodysuit with cut-outs'
id 83, description 'sleeveless dress'                id 85, description 'graphic print tee'
id 86, description 'double-color shirt'              id 87, description 'long-sleeved blouse'
id 88, description 'waist detail blouse'             id 89, description 'crushed velvet leggings'
id 90, description 'formal classic skirt'            id 91, description 'pu strap set'
id 92, description 'scallop edge bikini top'         id 94, description 'sequin flapper dress'
id 95, description 'print sleeveless dress'          id 96, description 'dress with animal print'
id 97, description 'sheer lace vest'                 id 98, description 'cable knit wool scarf'
id 99, description 'embroidered peasant blouse'      id 100, description 'tie-up dress with collar'
id 101, description 'military-style caped coat'      id 103, description 'satin wrap midi dress'
id 104, description 'asymmetrical skirt'             id 105, description 'tribal dress'
id 106, description 'silk charmeuse pajama set'
```

Figure 8. Sorted list for the second algorithm. Source Mac OS Terminal Version 2.14 (453).

Selection 3. Searching

In his option, the application will ask the user for an articular ID; the user must input an ID that already exists on the clothing stock. After the user inputs the ID, the application will proceed to search for it, and the results will display the number of iterations, the time in nanoseconds, and the memory used in the process.

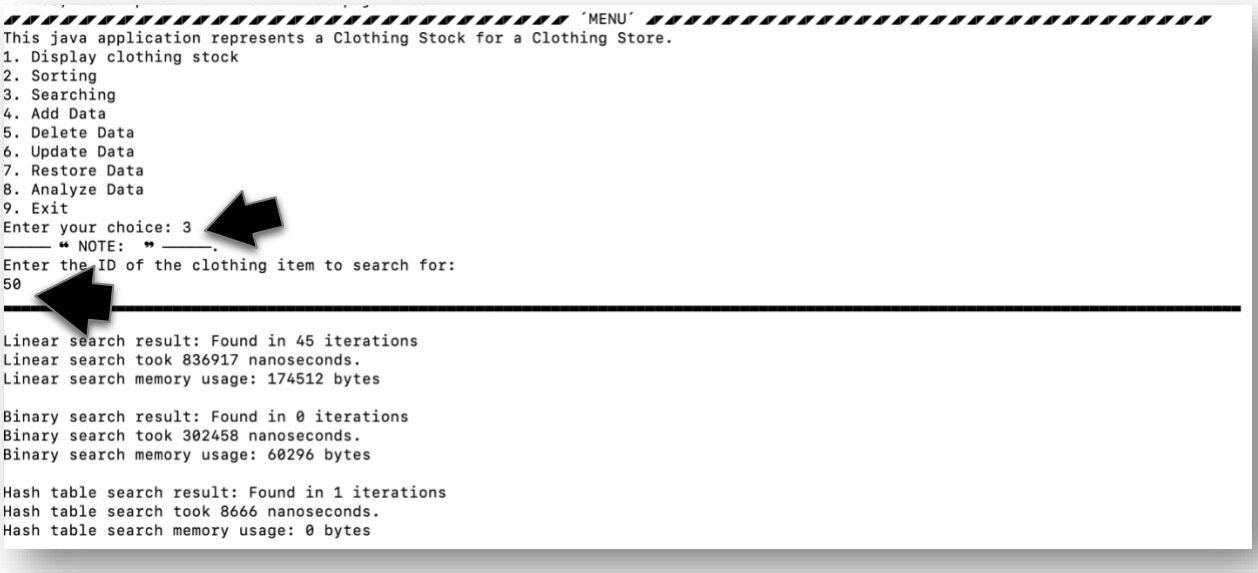


Figure 9. Selection 3. Searching. Source Mac OS Terminal Version 2.14 (453).

Selection 4. Add Data

In this selection, the user will be able to create a new article in the stock. The application will ask the user to input an ID (integer number).
The user must input an integer number.
Note: This option also allows ID duplicates.
If the user returns to option 1, the changes will be displayed on the clothing stock.

Figure 10. Selection 4. Add Data. Source Mac OS Terminal Version 2.14 (453).

Instance in when the user input an ID that already exists. The application will accept the user input and create the new ID.

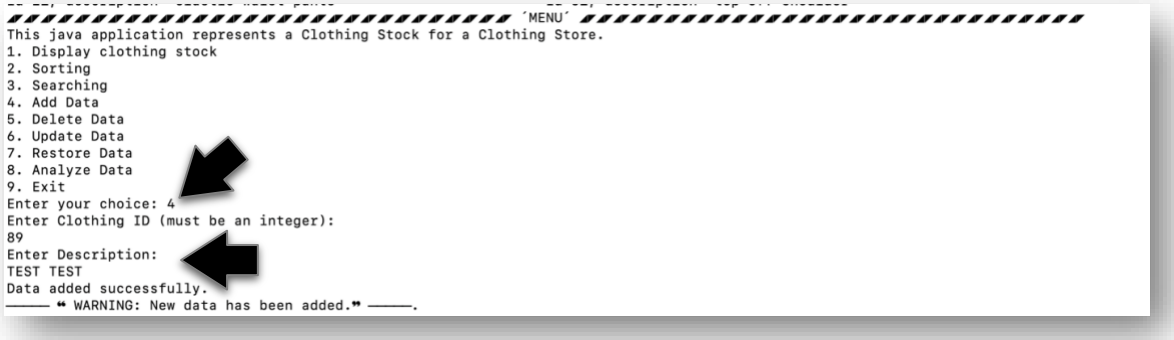


Figure 11. Selection 4. Add Data, example of adding an existing ID. Source Mac OS Terminal Version 2.14 (453).

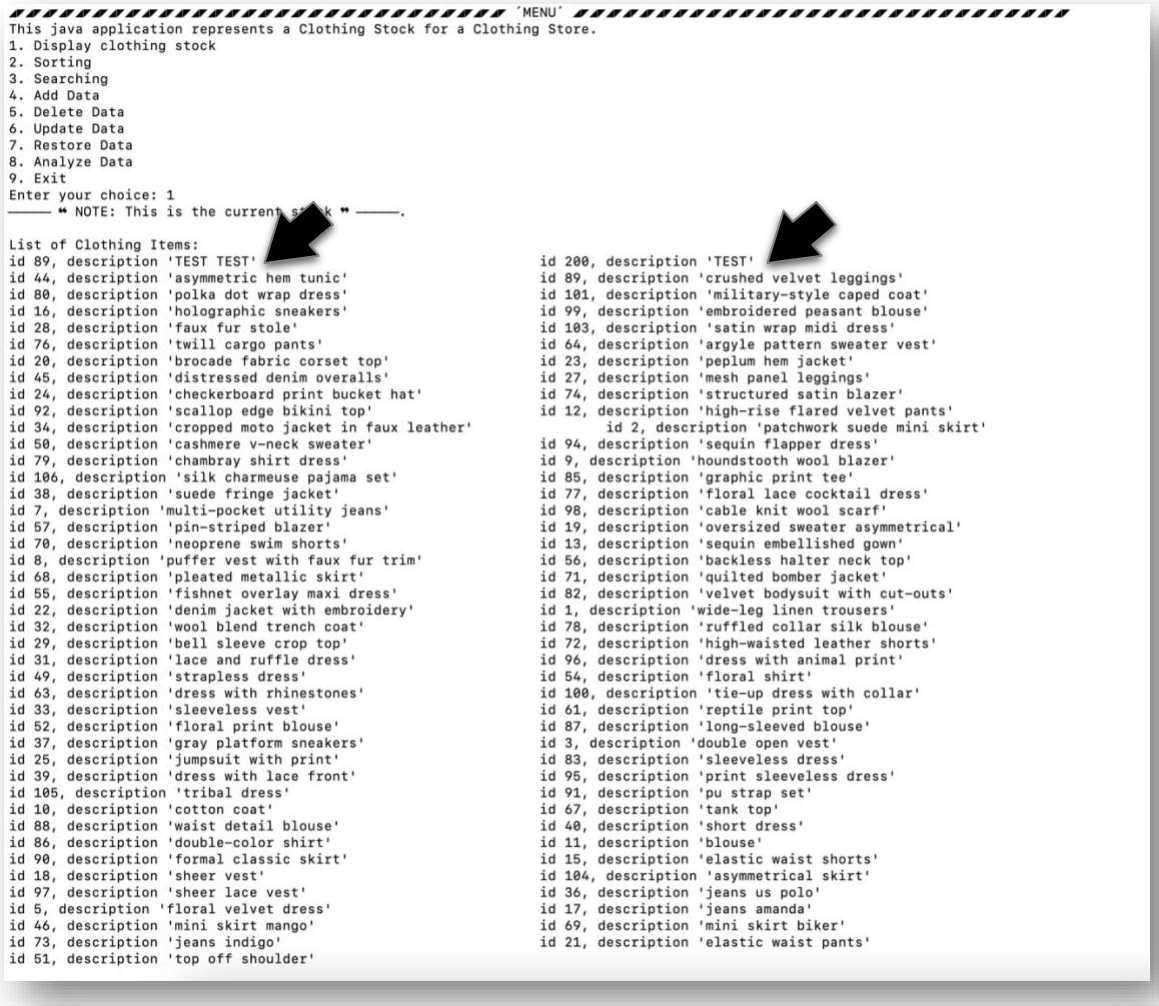


Figure 12. Selection 4. Add Data, example of duplicated ID. Source Mac OS Terminal Version 2.14 (453).

Selection 5: Delete Data.

This feature lets users delete an article. The application requires an ID and integer number to delete. After this, the app will request a description. Enter a string description at this prompt.
This user erased test IDs 89 (duplicated) and 200.

It also checks if the ID is in stock.

Figure 13. Selection 5: Delete Data. Source Mac OS Terminal Version 2.14 (453).

To review the changes the user must return to the 1st option.

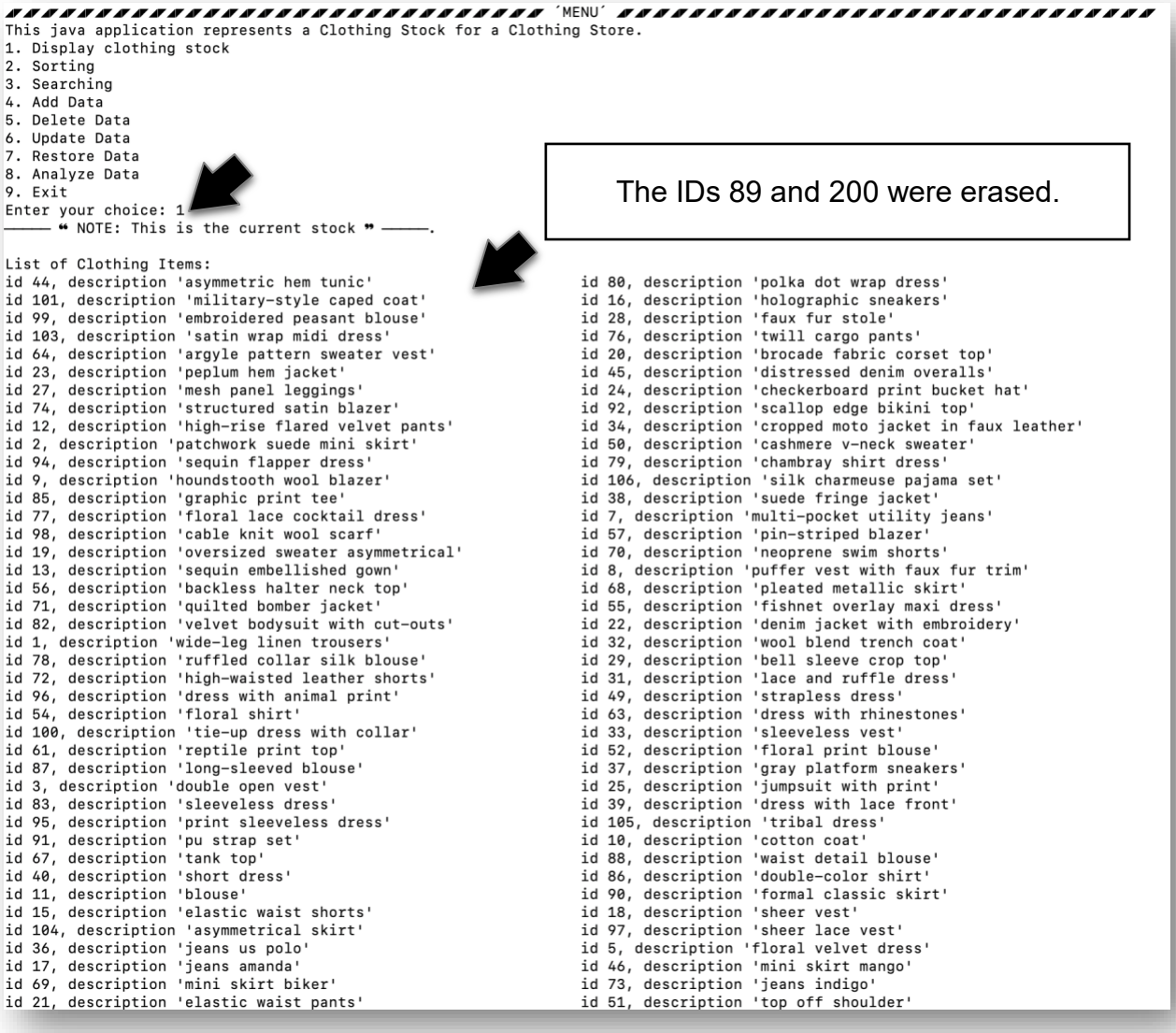


Figure 14. Review of changes after selection 5: Delete Data. Source Mac OS Terminal Version 2.14 (453).

Selection 6. Update Data

This option will allow the user to modify elements of the clothing stock. The application will ask for an ID (integer number) to modify, and then the application will proceed to ask the user for a new description (string values).
In this example the ID 44 was modified. To review the change the user must go back to option 1. The initial description was id 44, description 'asymmetric hem tunic'

And the modification is id 44, description 'floral crop top'.

Figure 15. Selection 6. Update Data. Source Mac OS Terminal Version 2.14 (453).

Selection 7: Restore Data

This selection will reset the changes in the stack to their original saved state.
The user must input an integer number.

```
////////////////////////////////////////////////// MENU //////////////////////////////////////////////////
This java application represents a Clothing Stock for a Clothing Store.
1. Display clothing stock
2. Sorting
3. Searching
4. Add Data
5. Delete Data
6. Update Data
7. Restore Data
8. Analyze Data
9. Exit
Enter your choice: 7

List of Clothing Items:
id 44, description 'asymmetric hem tunic'                    id 89, description 'crushed velvet leggings'
id 80, description 'polka dot wrap dress'                    id 101, description 'military-style caped coat'
id 16, description 'holographic sneakers'                    id 99, description 'embroidered peasant blouse'
id 28, description 'faux fur stole'                          id 103, description 'satin wrap midi dress'
id 76, description 'twill cargo pants'                       id 64, description 'argyle pattern sweater vest'
id 20, description 'brocade fabric corset top'               id 23, description 'peplum hem jacket'
id 45, description 'distressed denim overalls'               id 27, description 'mesh panel leggings'
id 24, description 'checkerboard print bucket hat'           id 74, description 'structured satin blazer'
id 92, description 'scallop edge bikini top'                 id 12, description 'high-rise flared velvet pants'
id 34, description 'cropped moto jacket in faux leather'             id 2, description 'patchwork suede mini skirt'
id 50, description 'cashmere v-neck sweater'                 id 94, description 'sequin flapper dress'
id 79, description 'chambray shirt dress'                    id 9, description 'houndstooth wool blazer'
id 106, description 'silk charmeuse pajama set'              id 85, description 'graphic print tee'
id 38, description 'suede fringe jacket'                     id 77, description 'floral lace cocktail dress'
id 7, description 'multi-pocket utility jeans'               id 98, description 'cable knit wool scarf'
id 57, description 'pin-striped blazer'                      id 19, description 'oversized sweater asymmetrical'
id 70, description 'neoprene swim shorts'                    id 13, description 'sequin embellished gown'
id 8, description 'puffer vest with faux fur trim'           id 56, description 'backless halter neck top'
id 68, description 'pleated metallic skirt'                  id 71, description 'quilted bomber jacket'
id 55, description 'fishnet overlay maxi dress'              id 82, description 'velvet bodysuit with cut-outs'
id 22, description 'denim jacket with embroidery'            id 1, description 'wide-leg linen trousers'
id 32, description 'wool blend trench coat'                  id 78, description 'ruffled collar silk blouse'
id 29, description 'bell sleeve crop top'                    id 72, description 'high-waisted leather shorts'
id 31, description 'lace and ruffle dress'                   id 96, description 'dress with animal print'
id 49, description 'strapless dress'                         id 54, description 'floral shirt'
id 63, description 'dress with rhinestones'                  id 100, description 'tie-up dress with collar'
id 33, description 'sleeveless vest'                         id 61, description 'reptile print top'
id 52, description 'floral print blouse'                     id 87, description 'long-sleeved blouse'
id 37, description 'gray platform sneakers'                  id 3, description 'double open vest'
id 25, description 'jumpsuit with print'                     id 83, description 'sleeveless dress'
id 39, description 'dress with lace front'                   id 95, description 'print sleeveless dress'
id 105, description 'tribal dress'                           id 91, description 'pu strap set'
id 10, description 'cotton coat'                             id 67, description 'tank top'
id 88, description 'waist detail blouse'                     id 40, description 'short dress'
id 86, description 'double-color shirt'                      id 11, description 'blouse'
id 90, description 'formal classic skirt'                    id 15, description 'elastic waist shorts'
id 18, description 'sheer vest'                              id 104, description 'asymmetrical skirt'
id 97, description 'sheer lace vest'                         id 36, description 'jeans us polo'
id 5, description 'floral velvet dress'                      id 17, description 'jeans amanda'
id 46, description 'mini skirt mango'                        id 69, description 'mini skirt biker'
id 73, description 'jeans indigo'                            id 21, description 'elastic waist pants'
id 51, description 'top off shoulder'
———— ❝ WARNING: The clothing stock will be restored to its original form.❞ ————.
```

Figure 16. Selection 7: Restore Data Source Mac OS Terminal Version 2.14 (453).

Selection 8. Analyze Data

This option will print on the screen the quantity of clothing elements that are currently in stock. The user must input an integer number. Note: The initial quantity of clothing elements is 106, with unique IDs.

Figure 17. Selection 8. Analyze Data. Source Mac OS Terminal Version 2.14 (453).

For example, if the user adds 2 elements, the number will increase.

Figure 18. Selection 8. Analyze Data. Source Mac OS Terminal Version 2.14 (453).

Instance where the user deletes an element.

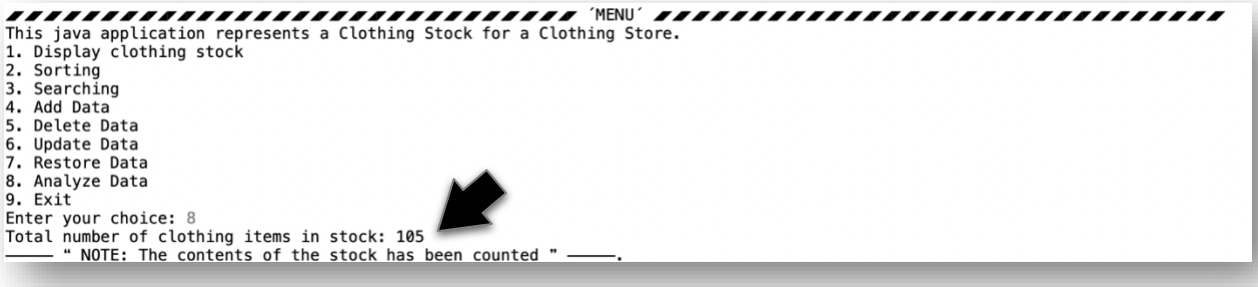


Figure 19. Selection 8. Analyze Data. Source Mac OS Terminal Version 2.14 (453).

Selection 9. Exit.

This option will stop the java application.

Figure 20. Selection 9. Exit. Source Mac OS Terminal Version 2.14 (453).

E. Testing document

Main menu testing. Input integer number. In this example when the user input an integer value between 1 and 9, the application will continue running.

Figure 21. Main menu testing. Source Mac OS Terminal Version 2.14 (453).

In the scenario in when the user input a value, which is not between 1 – 9, the application will allow the user to prompt again a number and the application will continue running.

Figure 22. Main menu testing – Integer number out of range. Source Mac OS Terminal Version 2.14 (453).

Input String text, in this example the application was terminated because the input value cannot fulfil the requirements of the conditions. The console will automatically throw an Exception message.

Figure 23. Main menu testing – String input. Source Mac OS Terminal Version 2.14 (453).

Figure 24. Main menu testing – String input. Source Mac OS Terminal Version 2.14 (453).

F. Debugging note

Debugging class CS401prj

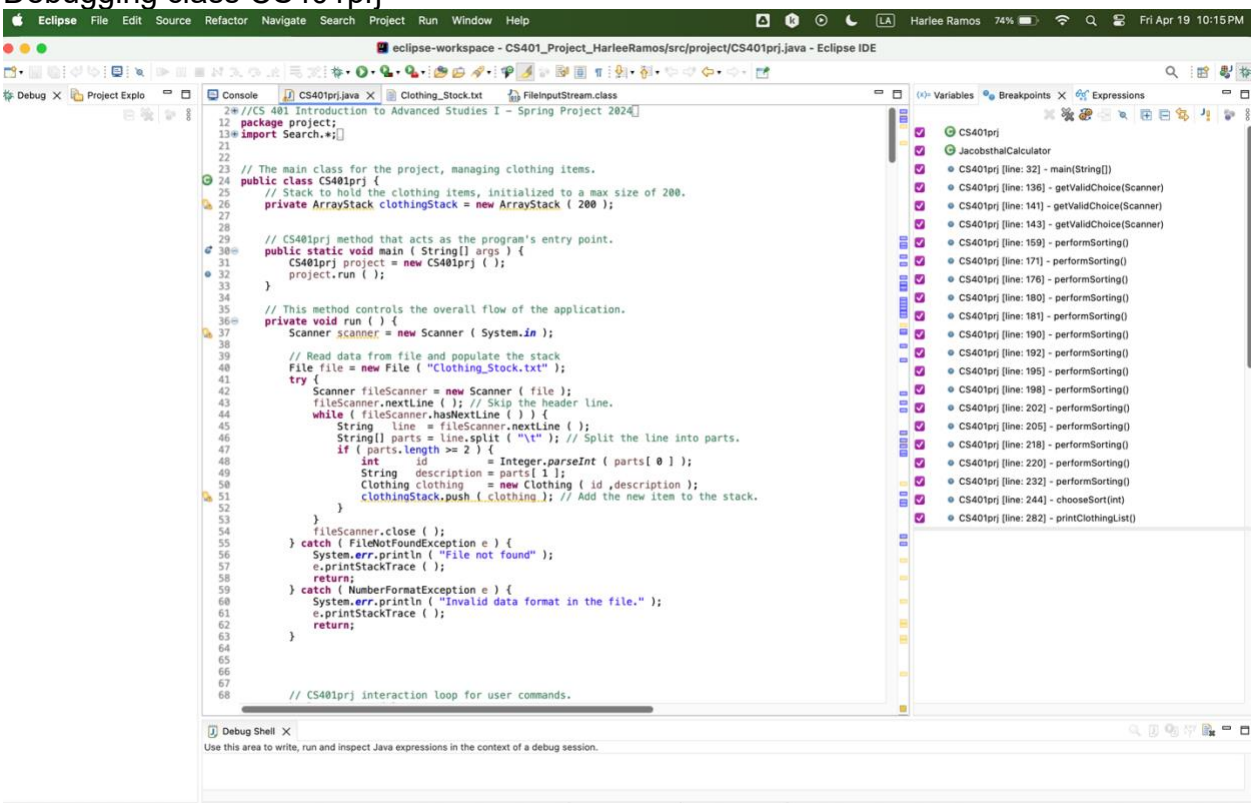


Figure 25. Debugging class CS401prj. Source: Eclipse IDE for Java Developers, Mac OS, Version: 2024-03 (4.31.0)

G. Self-evaluation notes

Method private void run ( ) {}

In the coding of this method no error handling for checking the String user prompt in the selections were considered. The error handling was considered mainly in the process of reading the file.

Method private void performSearching() {}

In the development of the while loop that checks if the ID is duplicated, the condition only considers the result of the linear search. In this section, it was difficult to incorporate the binary and hash search to fully complete the error handling. Because the array is sorted right after it is populated to prepare for binary search, And the Hash map is created from the sorted array, mapping IDs to clothing items for fast retrieval.

The performSearching method is emptying the stack into an array for searching, then pushing all elements back into the stack. This could be optimized. For instance, a search could be done directly on the stack without removing all elements, or a different data structure that allows for efficient searching could be used.

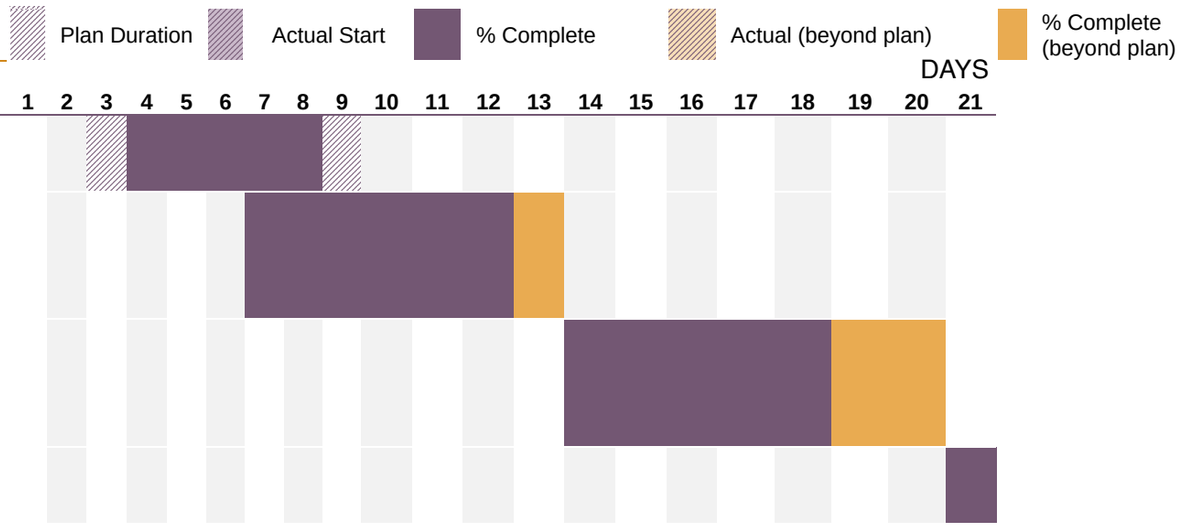
Method private void restoreData() {}

The restoreData method re-reads the file to restore the stack. Instead, of keeping an original copy of the data in memory if restoration is a common action, to avoid unnecessary file I/O operations.

H. Project management/schedule

The project timeline is depicted in the grant diagram that is presented below.

ACTIVITY	PLAN START	PLAN DURATION	ACTUAL START	ACTUAL DURATION	PERCENT COMPLETE
Problem Specification Software Specification 04/03/2024 - 04/05/2024	3	7	4	5	**100%**
Design Diagram Document Operational Document 04/08/2024 - 04/12/2024	7	6	7	7	**100%**
Test Document Debugging Self-evaluation Notes 04/15/2024 - 04/19/2024	14	5	14	7	**100%**
Project delivery 04/21/2024	21	4	21	2	**100%**



Approx. 35 hrs / p W

Conclusion

The project gave me practical experience with Java programming, problem solving, and data structure development. This initiative provided me with opportunities to develop technical skills and an analytical attitude outside of the classroom. Based on the textbook Object-Oriented Data Structures Using Java 4th Edition by Nell Dale, Daniel Joyce, and Chip Weems, the project serves as a journey from comprehending basic Java techniques to mastering complicated data manipulation and user interface design. The problems I had during the project have been turned into learning opportunities, boosting my resilience and adaptability. The event will shape my view of software development, instilling confidence, and readiness to tackle complex problems.