Kareem Saleh

README: Browser Navigation System

Introduction

This document provides an overview of the Browser Navigation System, specifying its components, walkthrough and functions.

Features

- Visit a URL and store it in the browsing history.

- Go back to the previous URL.

- Go forward to a previously visited URL (only available after going back).

- View search history.

- Clear browsing history.

- Save and restore the last browsing session.

Components

1. BrowserLinkedList (A class employing a Doubly Linked List)
   • Parent class of BrowserStack
   • Implements insertion, removal, and iteration.
   • Used to implement the backward and forward stacks for navigation.

2. BrowserArrayList (A class employing a Circular Array)
   • Parent class of BrowserQueue
   • Implements dynamic resizing, modular arithmetic for circular behavior, and indexing.
   • Used to implement the Queue for saving search history.

3. BrowserStack (A class employing a stack)
   • Child class of BrowserLinkedList.
   • Used for managing the back and forward navigation history.
   • Implements an Iterator to traverse the stack for saving session data

4. BrowserQueue
   • Child class of BrowserArrayList.
   • Used for managing the search history.

5. BrowserNavigation (Navigation through the menu)

- Manages visits, back, forward, and browsing history.
- Uses the custom Stack and Queue to manage navigation.
- Implements closeBrowser() method using an iterator
- Implements restoreLastSession() method to read info from file into the forward and backward stacks as well as history queue

6. StackIterator (Custom Iterator for Saving State)
   - Implements an Iterator to traverse and save stack elements when closing the browser.
   - Ensures that closeBrowser() can properly record navigation history
7. Main.java
   - Driver program that runs and tests the browser navigation system

---

Usage Instructions

Run the program and use the menu to navigate.

Menu Options (User has to input a number to carry out a function)

1 - Visit a URL

2 - Go Back

3 - Go Forward

4 - View Search History

5 - Clear Search History

6 - Save Session and Close Browser

7 - Restore Last Session

8 – Exit

File Handling

The session data is stored in 'session_data.txt'. This file is used to save the current browsing session and restore it later. When restored, it repopulates the current stacks and queues. The file structure includes sections for forward navigation, backward navigation, and history, each ending with 'End'.

Kareem Saleh

Order of Growth and Method Brief:-

BrowserLinked List Class:

- **insert(String a)**:

- insert is of order O(1) since we have last node and we are adding to the end by just incrementing last and placing element, which takes constant time and doesn't depend on the size of the list.

- **remove()**:

- remove is of order O(1) since we have last node and we are removing from the end by just decrementing last, removing element after storing it to return, which takes constant time and doesn't depend on the size of the list.

- **LLdoClear()**:

- LLdoClear() is of order O(1) since we just make first and last nodes point to null, which takes constant time and doesn't depend on the size of the list.

- **iterator()**:

- iterator is of order O(1) since we just return a new LLiterator object that doesn't require traversal of the list , which takes constant time and doesn't depend on the size of the list.

- **isEmpty()**:

- isEmpty is of order O(1) since we are just checking if first is null, which takes constant time and doesn't depend on the size of the list.

- **hasNext()** (from LLiterator):

- hasNext is O(1) since we are just checking if the current node is null, which takes constant time and doesn't depend on the size of the list.

- **next()** (from LLiterator):

- next is O(1) since we are just accessing the current node's data and moving to the next node, which takes constant time and doesn't depend on the size of the list.

BrowserArrayList Class:

- **getSize():**

- getSize() is of order O(1) since it returns the private variable size, which takes constant time and doesn't depend on the size of the list

- **isEmpty()**:

- isEmpty is of order O(1) since we are just checking if front index is -1, which takes constant time and doesn't depend on the size of the list.

- **isFullNorm()**:

- isFullNorm is of order O(1) since we are just checking if front index is 0 and back index is capacity-1, which takes constant time and doesn't depend on the size of the list.

- **isFullLoop()**:

- isFullNorm is of order O(1) since we are just checking if (back+1) % capacity) == front, which takes constant time and doesn't depend on the size of the list.

- **add(String a):**

  - add is O(n) in the case of a full array since we have multiple single for loops allowing us to resize and copy elements and multiplication constant is ignored

  - add is O(1) in the case of a none-full array since we just put the element at the desired index

  - Therefore, add is of order O(n) since O(n) dominates the O(1)

- **delete():**

  - delete is O(1) since removing at end requires no for loops, which takes constant time and doesn't depend on the size of the list.

- **ArraydoClear()**:

  - ArraydoClear() is of order O(1) since we just make front and back indices be -1 and creating(+assigning) new empty array, which takes constant time and doesn't depend on the size of the list.

BrowserStack Class:

- **push(String url)**:
  - push is O(1) since we use LL insert, which is O(1) because we have the last node and we are adding to the top of the stack by just incrementing last and placing the element, which takes constant time and doesn't depend on the size of the list.
- **pop():**
  - pop is O(1) since we use LL remove, which is O(1) because we have the last node and we are removing from the top of the stack by just decrementing last, removing the element after storing it to return, which takes constant time and doesn't depend on the size of the list.
- **peek():**
  - peek is O(1) since we are just checking the data of the last node (top of the stack) without modifying the list, , which takes constant time and doesn't depend on the size of the list.

BrowserQueue Class:

- **enqueue(String url):**
  - enqueue uses the Arraylist add which is O(n) in the case of a full array since we have multiple single for loops allowing us to resize and copy elements and multiplication constant is ignored

  - enqueue uses the Arraylist add which is O(1) in the case of a none-full array since we just put the element at the desired index, which takes constant time and doesn't depend on the size of the list.

  - Therefore, enqueue uses the Arraylist add which is of order O(n) since O(n) dominates the O(1)
- **dequeue():**
  - dequeue is using ArrayList's remove which is O(1) since we have front index and we are removing from the front of the queue by just incrementing front, removing element after storing it to return. This takes constant time and doesn't depend on the size of the list.
- **peek():**
  - peek is O(1) since we are just checking the data of the last node (top of the stack) without modifying the list, , which takes constant time and doesn't depend on the size of the list.

Kareem Saleh

StackIterator Class:

- **forwardIterator()**:
  - forwardIterator() is O(n) since the while loop iterates over each of the elements in the forward stack using the iterator.

- **backwardIterator()**:
  - backwardIterator() is O(n) since the while loop iterates over each of the elements in the backward stack using the iterator.

BrowserNavigation Class:

- **goBack()**:
  - goBack() is O(1) since no loops and just popping a single element from the top of the backward stack then pushing it into the forward stack. This takes constant time and doesn't depend on the size of the list.

- **goForward()**:
  - goForward() is O(1) since no loops and just popping a single element from the top of the forward stack then pushing it into the backward stack. This takes constant time and doesn't depend on the size of the list.

- **showHistory()**:
  showHistory() is O(n) since it uses a for loop to iterate over the history queue's elements and print them one by one.

- **clearHistory()**:
  - clearHistory() is O(1) since it calls Arrayclear() and ArraydoClear(), which just make the front and back indices be -1 and assign a new empty array. This takes constant time and doesn't depend on the size of the list.

- **HistorySaver()**:
  - HistorySaver() is O(n) since it uses a for loop to iterate over the history queue elements and append them to a string.

- **closeBrowser()**:
  - closeBrowser() is O(n) since it contains forwardIterator() and backwardIterator() with single while loops and HistorySaver() with a for loop.

- **restoreLastSession()**:
  - restoreLastSession() is O(n) since it contains multiple while loops to read and restore elements from the file into the stacks and queue.

Kareem Saleh

Main Class:

- **visitWebsite(String url)**
  - visitWebsite(String url) is O(n) since the enqueue operation on Hist (BrowserQueue) takes O(n) due to the queue's underlying array-based structure, which dominates the O(1) time complexity of the LLclear() method in Bstack (BrowserStack), as well as the O(1) push operation for Fstack.

Test Cases:-

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
2
No previous page available
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
3
No forward page available
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
amazon
Now at amazon

Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
apple
Now at apple
```

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
2
Now at apple
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
3
Now at apple
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
4
amazon
apple

Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
cocacola
Now at cocacola
```

Kareem Saleh

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
vodafone
Now at vodafone

Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
nike
Now at nike
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
addidas
Now at addidas

Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
ups
Now at ups
```

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
4
amazon
apple
cocacola
vodafone
nike
addidas
ups

Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
nba
Now at nba
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
2
Now at nba
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
2
Now at ups
```

Kareem Saleh

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
pepsi
Now at pepsi

Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
wikipedia
Now at wikipedia
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
khanacademy
Now at khanacademy

Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
1
Enter your url:
utdallas.edu
Now at utdallas.edu
```

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
4
amazon
apple
cocacola
vodafone
nike
addidas
ups
nba
pepsi
wikipedia
khanacademy
utdallas.edu
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
6
Message written to file successfully.
Browsing session saved.
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
7
Message written to file successfully.
Last session restored successfully.
```

Kareem Saleh

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
2
Now at utdallas.edu
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
3
Now at utdallas.edu
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
5
Browsing history cleared.
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
4
No browsing history available
```

```
Type the number that corresponds with your choice.
Menu:
1- Visit a URL
2- Undo
3- Redo
4- View Search History
5- Clear Search History
6- Close Browser
7- Restore Last Session
8- Exit
8
Exiting...


...Program finished with exit code 0
Press ENTER to exit console.
```

Test Case 1: Undo Without History Steps:

1. Viewing the menu.
2. Select option `2` (Undo).

Expected Output:

- No previous page available

Test Case 2: Redo Without History Steps:

1. Viewing the Menu.
2. Select option `3` (Redo).

Expected Output:

- No forward page available

Test Case 3: Visit Multiple URLs and Check Navigation Steps:

1. Start the program.
2. Select option `1` (Visit a URL) and enter `amazon`.
3. Select option `1` again and enter `apple`.
4. Select option `2` (Undo).
5. Select option `3` (Redo).

Expected Output:

- Now at amazon
- Now at apple
- Now at apple
- Now at apple

Test Case 4: View Search History Steps:

1. Select option `4` (View Search History).

Expected Output:

- `amazon`

  `apple`

Test Case 5: Visit Additional URLs and Search History Steps:

1. Visit `cocacola`, `vodafone`, `nike`, `addidas`, `ups`. (By Selecting option `1` repeatedly)
2. Select option `4` (View Search History).

Expected Output:

- Now at cocacola
- Now at vodafone
- Now at nike
- Now at addidas
- Now at ups

- `amazon`

  `apple`

  `cocacola`

  `vodafone`

  `nike`

  `addidas`

  `ups`

Test Case 6: Visiting then Undoing Multiple Times Steps:

1. Visit `nba`. (By Selecting option `1`)
2. Select option `2` (Undo) twice.

Expected Output:

- Now at nba
- Now at nba
- Now at ups

Test Case 7: Visit Additional URLs and Search History Steps:

1. Visit `pepsi`, `wikipedia`, `khanacademy`, `utdallas.edu`. (By Selecting option `1` repeatedly)

2. Select option `4` (View Search History).

Expected Output:

- Now at pepsi
- Now at wikipedia
- Now at khanacademy
- Now at utdallas.edu

- ```
  amazon

  apple

  cocacola

   vodafone

   nike

   addidas

   ups

  nba

  pepsi

  wikipedia

  khanacademy
  ```
  ```

```
utdallas.edu
```

Test Case 8: Restore Last Session Steps:

1. Close the browser using option `6`.
2. Restore session using option `7`.

Expected Output:

- Message written to file successfully

  Browsing session saved.

- Last session restored successfully.

Test Case 9: Undo:

1. Viewing the menu.
2. Select option `2` (Undo).

Expected Output:

- Now at utdallas.edu

Test Case 10: Redo:

3. Viewing the menu.
4. Select option `3` (Redo).

Expected Output:

- Now at utdallas.edu

Test Case 11: Clear Search History Steps:

1. Select option `5` (Clear Search History).
2. Select option `4` (View Search History).

Expected Output:

- Browsing history cleared.
- No browsing history available.

Test Case 12: Exiting Program:

1. Select option `8` (Exit).
2. **Expected Output:**

- Exiting…

Implementations:-

1. **Visiting a Website (`visitWebsite(String url)`)**

- **Pushing an element in the backward stack**
  a) Accesses the push in the BrowserStack class
  b) Push inherits the insert method from the BrowserLinkedList class
  c) It creates a new Node with the inputted url
  d) Checking if empty list : first node being null indicates empty linked list
  e) If empty, we make first and last nodes point to new node
  f) If not, we make the next pointer in the element pointed to by last node point to the new Node. Then, we make newNode's previous point to the element pointed to by last node. Then, we make last node point to the newNode.
  g) Incrementing modCount to keep track of modifications which will be crucial in iterator.
- **Clearing the forward stack**
  a) Accessing the inherited LLclear() from BrowserLinkedList
  b) The public LLclear() accesses the private method LLdoClear() to avoid tampering with array
  c) LLdoClear() makes both the first and last nodes point to null
  d) Incrementing modCount to keep track of modifications which will be crucial in iterator.
- **Enqueue the history queue with the new url**
  a) Accessing the enqueue method from the BrowserQueue class
  b) This enqueue method calls the add method in BrowserArrayList class
  c) add method first checks if array is normally full or circularly full
  d) If front index is at 0 and back is at capacity-1, then array is normally full

e) If (back+1) % capacity is equal to front, then the back index is in the index just before front index. To allow the back index to chase the front index without going out of bounds, the mod division is employed. Assume last reaches capacity – 1 and front is at index 1. Then (back+1) % capacity = (capacity-1+1) % capacity = capacity%capacity = 0. Therefore, the back index wraps around (without out of bounds exception) and is in the index right before front. This is the case of the wrapped around full array

Case Full:-

f) Next, capacity is doubled then 1 is added to make the array able to carry more elements
g) We create new empty array with the new capacity

Case Back After Front:-

h) Then we copy all the elements from original array
i) Front is set to be the first index (0)
j) Back is set to become the size -1 (number of elements in original array– 1)
k) New bigger array is assigned to original array. More capacity, same elements
l) Then, back index is incremented by 1 first
m) Then the new element is placed in the updated back index
n) Size incremented by 1

Case Back Before Front:-

o) We have two indices, i looping around new array and j looping around original
p) We copy from front to the end of the first array to the first couple of indices in new array
q) Then we continue filling the array with elements from 0 to back
r) New bigger array is assigned to original array. More capacity, same elements
s) Then, back index is incremented by 1 first
t) Then the new element is placed in the updated back index
u) Size incremented by 1

Case Otherwise:-

v) If array is empty, then we change front from -1 to 0
w) Else, we make back = (back+1) % capacity to allow wrap arrounds
x) We assign the new element to the back index
y) Size incremented by 1

2. **Going Back (`goBack()`)**

- If backward stack is Empty (using isEmpty() function)
- This means that first is pointing to null
- throw new EmptyStackException(), which will print "No previous page available"

Else

- We pop the top of the backward stack, store it in a variable and push this element in the forward stack, then print the element.

---

3. **Going Forward (`goForward()`)**

- If forward stack is Empty (using isEmpty() function)
- This means that first is pointing to null
- throw new EmptyStackException(), which will print "No forward page available"

Else

- We pop the top of the forward stack, store it in a variable and push this element in the backward stack, then print the element.

---

4. **Displaying Browsing History (`showHistory()`)**

Empty

- Print "No browsing history available"

Else

- Create new index var
- Loop over history queue indices using i
- (history.front + i) % history.capacity to enable traversing the history queue with allowance of wrapping around
- In the loop, print each element we reach

---

5. **Clearing Browsing History (`clearHistory()`)**

- History Queue variable accesses the Array.clear()
- Arrayclear() is public and it accesses ArraydoClear() to avoid tampering

- Creating new empty array
- Setting front and back to -1 (indicating empty array)
- Setting size to 0

---

6. **Closing the Browser (`closeBrowser()`)**

- Creates new file called session_data.txt
- Try catch block to automatically close FileWriter
- Using FileWriter to print stacks and queue content in file

  Forward and Backward Stacks Iterator

  - Utilizes hasNext() and next() definitions from Linked List class
  - Creating current node containing first to start at beginning of the LL
  - Creating expected modCount variable to save the modCount variable we have
  - hasNext() returns Boolean response for whether current is null
  - next() checks if expected and current modCounts are equal. If yes, it checks if there are more elements in the LL. If yes, it sets a new nextItem var to hold the value of current.data, advances current, and returns the next item.

  NB: A non-identical modCount will throw a new ConcurrentModificationException() and a false return from hasNext() will throw new NoSuchElementException()

  History Queue Loop:-

  - Creating Stringbuilder to store the queue contents
  - For loop with ability to wrap around that appends the contents of the queue to the StringBuilder and adds new line
  - Converts the StringBuilder to a String and returns it

- We separate the contents of each queue or stack with End and newline to make markers that are easy to read
- Close writer
- Return the file

---

7. **Restoring the Last Session (`restoreLastSession()`)**
   - Checking if file exists
   - Try catch block to automatically close BufferedReader

- Creating line var to contain the file lines as well as a temp stack and temp queue

Stacks:-

- While loop over first set of lines till we hit an “End” marker
- Push each read line into the temporary stack
- While loop to push the popped items from the temporary stack to the original stack
- Temp stack is a necessary step to be able to repopulate the original stack in the same order not a reversed one since stacks are LIFO

Queue:-

- Temp queue unnecessary
- Enqueueing the read lines from the file to the queue in smooth manner since FIFO is used for queues

---