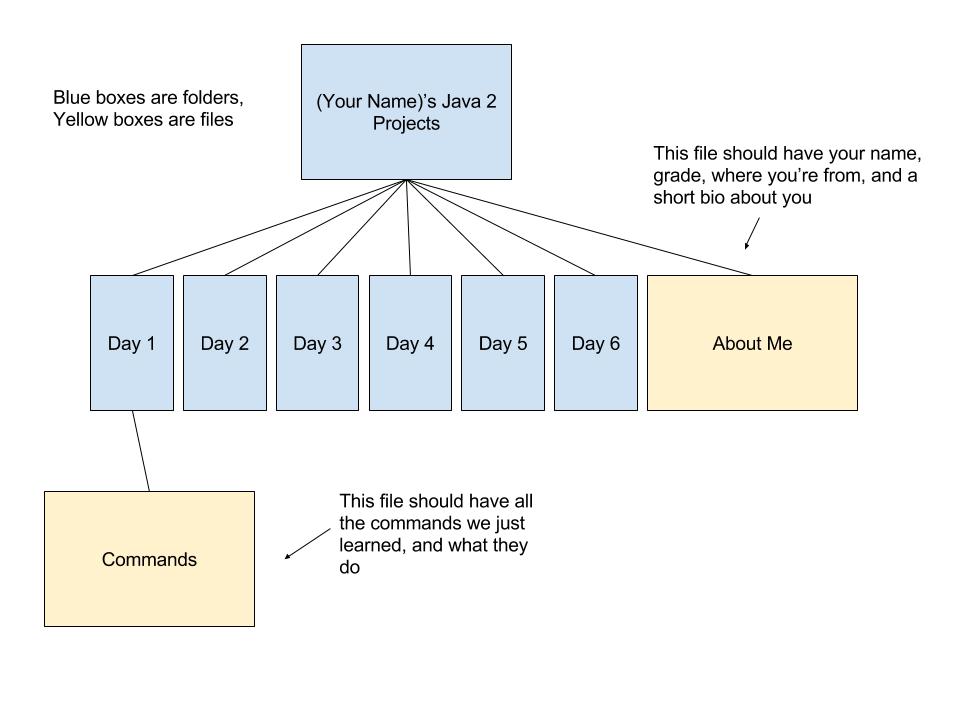
Command Line Practice



Doubly Linked List Project

You are given 2 files: dllist.java and dllistTest.java. Dllist is where you’ll implement a Doubly Linked List. It has an internal class called Node for each node of the list, which stores the item, and references to the previous and next nodes.

Fill in the following methods of the dllist class:

* setPosition: sets the current position in the list to first, next, previous, or last.
* Delete: removes the node at the current position
* Insert: inserts the given item at the given position. Should throw an IllegalArgumentException if you attempt to insert an item before the first item or following the last.

Run the tests in the dllistTest file periodically to see which features are working.

When all the unit tests are working, in the dllistTest.java file, add the following tests.

* A test called insertLastGetItemSecondTest that does inserts 2 items in the last position and then gets an item and asserts this is equal to the last item inserted.
* A test called insertFirstGetItemSecondTest that does inserts 2 items in the first position and then gets an item and asserts this is equal to the last item inserted.
* A test called deleteTest which inserts many items, then deletes them one by one, asserting each time that the new position and new current item is correct.
* A test called insertPreviousAtFirstTest which attempts to insert an item before the first item and expects an IllegalArgumentException.
* A test called insertFollowingAtLastTest which attempts to insert an item following the last item and expects an IllegalArgumentException.
* A test called setPositionLastTest that sets the position to the end of the list and then gets the item and makes sure it is the last item.
* A test called setPositionFirstTest that inserts several items, then sets position to first, then checks that the item is correct.
* A test called insertFollowingTest that inserts several items, then inserts something following the current item and makes sure that the current item remains the same.

When you are finished, your solution should have 17 tests that pass.

Bracket Matching Worksheet

For each of the following, use a stack to determine whether the brackets are matched correctly.

1. ((({[]({})]})))

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |

1. [{}](([{}[[()]()]])())

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |  |  |  |

1. ([{{[()[]]{}}])

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  |  |  |  |  |  |  |

Reverse Polish Notation Worksheet

Convert the following from prefix to postfix notation, and solve them:

|  |  |  |  |
| --- | --- | --- | --- |
|  | Prefix Notation | Postfix Notation | Solution |
| 1 | 3 + 4 |  |  |
| 2 | 8 x 9 |  |  |
| 3 | (6 - 4) x 3 |  |  |
| 4 | 5 + ((1 + 2) × 4) − 3 |  |  |
| 5 | (162 / (2 + 1 )) + 4 |  |  |

Solve the following postfix notation problems:

|  |  |  |
| --- | --- | --- |
|  | Postfix Notation | Solution |
| 1 | 3 11 + |  |
| 2 | 3 11 + 5 - |  |
| 3 | 18 2 - 4 / |  |
| 4 | 2 3 11 + 5 - x |  |
| 5 | 2 1 12 3 / - + |  |

Stack Calculator Project

You are given 2 files: Calc.java and CalcTest.java. Calc is a class that will work as a reverse Polish notation/postfix calculator. It has an internal class, Stack, that will be used by Calc to hold the numbers in the calculator. The actual operations (addition, subtraction, multiplication, and division) are methods of the Calc class.

The Stack implementation uses an array of doubles to store the numbers in the stock and an int to store the current size of the stack. Fill in the following methods in class Stack:

* push: adds the double d onto the stack, increments the size
* pop: removes the top number from the stack and returns it
* peek: returns the top number of the stack without modifying it
* size: returns the current size of the stack

Then, fill in the following methods in class Calc. Make sure that each method first checks to make sure that there are at least 2 numbers in the stack. Let the “add” method serve as a template for these methods.

* subtract: subtracts the top 2 numbers on the stack, pushes the result back onto the stack
* multiply: multiplies the top 2 numbers on the stack, pushes the result back onto the stack
* divide: divides the top 2 numbers on the stack, pushes the result back onto the stack

Run the tests in the CalcTest file periodically to see which features are working.

When all the unit tests are working, in the CalcTest.java file, add the following tests.

* A test called pushPopDepthTest that does several pushes and pops and asserts the depth stays correct
* 3 tests based on the underflowAddTest called underflowSubtractTest, underflowMultiplyTest, and underflowDivideTest. These tests should make sure an exception is thrown when an attempt is made to operate on less than 2 numbers
* 3 tests based on singleSubtractTest called singleAddTest, singleMultiplyTest, and singleDivideTest. These tests assert correct values for each of the operations
* 3 tests based on doubleMultiplyTest called doubleAddTest, doubleSubtractTest, and doubleDivideTest. These tests should assert correct values for each of the operations performed twice in a row.
* A test called compoundTest which performs a long postfix calculation and makes sure the answer is correct

When you are finished, your solution should have 28 tests that pass.

Sorting Time Comparison Project

You’re given the file SortingTimes.java to start (note it is the same example we just went over in class). Right now, the file contains functions for 2 sorting algorithms, Bubble Sort and Selection Sort.

First, create methods that perform bubble sort and selection sort on Strings. Use the compareTo method and return the arrays in alphabetic order.

Create 2 more functions in which you implement the Insertion Sort algorithm for arrays of ints and arrays of Strings.

In the main, create a third copy of the array with the random integers of it. Run your insertion sort algorithm on that, and display how much time that takes.

Create an array of 15-20 random Strings. A line of song lyrics is a nice way to get a bunch of words in non-alphabetic order. Perform all 3 sorting methods on this array as well and display how much time they take.

Which algorithm is the slowest, and why?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

On which types of data would one of these algorithms perform better than others?

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Queue Project

You’re given the Queue.java class to start. This is the starting code for a Linked List implementation of a Queue. We are going to use our Queue class to keep track of buying and selling shares in a mini stock market.

First, you need to fill in the methods in the Queue class for insert, remove, and peek. Make sure your insert method creates a new node and inserts it correctly. You need to check if the list is empty when you are heading because you will want to change both the head and the tail in that case.

Create a tester class for your Queue called QueueTest. Create a new Queue, try insert, remove, and peek several times each with print statements to ensure that your Queue class works as expected.

Now, here’s how the mini-market will work. The user enters how many shares they are buying or selling, and the price. We will use the queue to keep a running total of the shares we have and their prices. Then, as we sell, we calculate the gain (or loss) and add that to the total gain. For example, if we buy 5 shares at $3 each, then 5 shares at $2 each, then sell 7 shares at $4 each, we have made money. The first 5 shares were bought at $3 and sold at $4, so we made $1/share there, so $5 total. Then the last 2 shares were bought at $2 each and sold at $4, so we made $2/share there, so $4 total. So all 7 shares got us $9 in captital gains.

Next, create a new class called Market. The Market class should have a Queue variable and a variable to keep track of the total gains (capital gains) made so far, both of which are started in a constructor. The Market class should have the following methods:

* A void method buyShares, which takes in an int representing the number of shares and a double representing the price of the shares. This method should add the shares at that price to the queue.
* A void method sellShares, which takes in an int representing the number of shares and a double representing the price of the shares. This method should remove the shares from the queue, and keep track of the profit made in the gains variable.
* A double-returning method getGains, which returns the gains variable.

Next, create a new main class called App. In the main, create a new Market. Then, prompt the user to either buy, sell, see gains, or quit. Use the scanner to get user input and conditionals to decide which of the Market methods to call. Keep prompting the user to buy, sell, see gains, or quit until they quit.

Merge Sort Project

You are to create a program that reads in a document which contains information about businesses, sorts them into alphabetic order using merge sort, and then allows the user to search them.

First, create a class called BusinessRecord. This should contain String variables for the name and phone number of each business, and the following methods:

* A constructor which sets the name and the phone number
* Getters and setters for name and phone number
* A method called stringOut which returns a string consisting of the name and phone number separated by a comma
* A method called printOut that prints out the name and phone number.

Next, create a class called BusinessList. This is the class for keeping track of a list of BusinessRecords. It should have an int called numberOfBusinesses, an array of BusinessRecords, an int to keep track of the number of searches performed, and an int to keep track of the number of unsuccessful searches. You will create the following methods:

* A constructor which takes in a the name of the file, and reads it in line by line. The file will look like this:

2

Mike’s Bikes, (510) 888-8888

Bob’s Burgers, (888) 654-3210

The first line is the number of business in the file. Read that in first. For each of the rest of the lines, you will separate it by the comma, create a new BusinessRecord with the name and phone number, add the BusinessRecord to the array, and increment the numberOfBusinesses.

* A void method called mergeSort, which has no input variables. This should create the workspace for the mergeSort algorithm, and call the recMergeSort method.
* A void method called recMergeSort, which takes in a workspace array, a lower bound, and an upper bound. This is a recursive function to break down the array in smaller and smaller halves and then merge them back up so they are sorted. First, it should check the base case: if the lower and upper bounds are equal, then we are done. Otherwise, it should find the midpoint, call itself on both the upper and lower halves, and then call the merge method.
* A void method called merge, which takes in a workspace array, a low pointer, a high pointer, and an upper bound. This method is the actual mergesort method. It takes the 2 arrays represented by the bounds (with the high pointer starting at the midpoint+1) and knits them together in order in the workspace, then writes them back into the original array in order.
* A method called find which returns a String and takes in a String, the business name. This should perform a binary search on the sorted array of BusinessRecords, and if the business name is found, it should return the phone number. Otherwise, it should return the String “NOT FOUND”.
* A void method called search with no input variables. This should create a scanner, and keep prompting the user to search for a business until they enter a blank line. For each search, call the find method and print the result. Keep track of the number of searches and the number of unsuccessful searches and print them after they have entered the blank line.

Next, we create another class called BusinessSearch. This should have a main. Simply create a BusinessList using the filename “directory.txt”, call mergeSort on that list (for testing purposes, print out the sorted list so you can ensure your mergesort works correctly), and then call search on that list.

Here is a test file to use, feel free to add more businesses:

5

Comal, (111) 111-1111

Little Plearn, (222) 222-2222

Cactus Tacqueria, (333) 333-3333

Picante, (444) 444-4444

La Note, (555) 555-5555

Tree Project

You are given the Tree and TreeTest classes. Your job is to change each node so that instead of storing a single integer value to represent the line number, it stores a Queue of integer values which represent all the line numbers the word is on. Use your Queue class from the earlier assignment.

Here is a new words.txt file to use for testing:

apple banana cheese

dog elephant human

apple

fairy godmother cheese banana

banana banana

goblin fairy

half human elf

Heap Project

You are given the code for all of heapsort on the next page. You need to perform it by hand, step by little step. Assume we have called the sort function on this array:

488 667 634 380 944 594 783 584 550 665 721 819 285 344 503 807 491 623 845 300

If you finish early, try to adapt the code for use on Strings. Which lines would we have to change?

1. import java.util.Scanner;
2. /\* Class HeapSort \*/
3. public class HeapSort {
4. private static int N;
5. /\* Sort Function \*/
6. public static void sort(int arr[]) {
7. heapify(arr);
8. for (int i = N; i > 0; i--)
9. {
10. swap(arr,0, i);
11. N = N-1;
12. maxheap(arr, 0);
13. }
14. }
15. /\* Function to build a heap \*/
16. public static void heapify(int arr[]) {
17. N = arr.length-1;
18. for (int i = N/2; i >= 0; i--)
19. maxheap(arr, i);
20. }
21. /\* Function to swap largest element in heap \*/
22. public static void maxheap(int arr[], int i) {
23. int left = 2\*i ;
24. int right = 2\*i + 1;
25. int max = i;
26. if (left <= N && arr[left] > arr[i])
27. max = left;
28. if (right <= N && arr[right] > arr[max])
29. max = right;
30. if (max != i) {
31. swap(arr, i, max);
32. maxheap(arr, max);
33. }
34. }
35. /\* Function to swap two numbers in an array \*/
36. public static void swap(int arr[], int i, int j) {
37. int tmp = arr[i];
38. arr[i] = arr[j];
39. arr[j] = tmp; }

Choose Your Own Adventure: Final Project

You are to create a Choose Your Own Adventure game for the final project this week.

This game will read in a file that contains all the possible “rooms” in the adventure game. Each room has the letter r followed by the room’s name, one or more lines with the letter d followed by the description, and a list of options (preceeded by the letter o) and the tags for the rooms they lead to (preceeded by the letter t). Each room can have up to 12 options, letters a-l.

To play, the user types one of the following:

* One of the letters that are options for the current room, which should bring them to that room.
* ‘r’ to restart the game
* ‘q’ to quit the whole program
* ‘y’ to print out all the information for all of the rooms
* ‘z’ to go back a room

It is up to you to design how all of this is implemented. You should spend at least 15-30 minutes designing how your project will work: what classes, variables, and methods will be needed and how they will interact. You will turn in a text file describing your final design for this project.

Here is a very simple example file with only 2 rooms to test with. After you ensure that your program works on these rooms, create our own file with a set of many rooms to play with.

r room1

d This is room one.

d You are awake.

o Go to sleep.

t room2

o Stay awake.

t room1

r room2

d You are sleeping.

o Stay asleep.

t room2

o Wake up.

t room1