AP Computer Science A
Problem Set 5

Dr. Balchunas

Instructions

In this problem set, submit a zipped folder of your .java files. Add comments to the file if you believe the class name is not enough to tell me which problem it is.

Stack

Stack API

int size()	Returns the number of elements in the stack
void push(T obj)	Pushes obj to the top of the stack
T pop()	Removes and returns the top element of the stack
T peek()	Shows the top element of the stack
String toString()	Shows the elements in the stack

Problem 1

Write a generic data type for a Stack data structure using a resizing array implementation.

Problem 2

Write a generic data type for a Stack data structure using a linked list implementation.

Queue

Queue API

int size()	Returns the number of elements in the queue
void enqueue(T obj)	Adds obj to the queue
T dequeue()	Removes the next element of the queue
T peek()	Shows the next element of the queue to be removed
String toString()	Shows the elements in the queue

Problem 3

Write a generic data type for a Queue data structure using a resizing ring-buffer array implementation.

Problem 4

Write a generic data type for a Queue data structure using a linked list implementation.

Deque

Deque API

int size()	Returns the number of elemnets in the dequeue
void addFirst(T obj)	Adds obj to the front
void addLast(T obj)	Adds obj to the end
T removeFirst()	Remove and return the element from the front
T removeLast()	Remove and return the element from the end
String toString()	Shows the elements in order

Problem 5

Write a generic data type for a double-ended queue, Deque (pronounced "deck"), data structure using a resizing ring-buffer array. A ring-buffer array is one that uses two pointers to keep track of the contiguous dequeue. e.g. at some point after adding 5 elements and removing 2 from the front the underlying array might look like `[0 0 3 4 5 0 0 0 0]`. By using a ring-buffer array, you do not need to shift elements when removing them from the front of the array.

Problem 6

Write a generic data type for a double-ended queue, Deque (pronounced "deck"), data structure using a doubly-linked list implementation where each element points to the element in front of and behind it.

ArrayList

ArrayList API

int size()	Returns the number of elements in the ArrayList
void add(T obj)	Adds obj to the end of the ArrayList
void add(int index, T obj)	Adds obj at the specified index and shifts all other elements
T get(int index)	Return the element at position index
T set(int index, T obj)	Set the element at the specified index to obj and return the element previously at index
T remove(int index)	Remove and return the element at index. Reindex the remaining elements
String toString()	Shows the elements in order

Problem 7

Write a generic data type for an ArrayList data structure using a resizing array implementation.

Problem 8

Write a generic data type for an ArrayList data structure using a linked list implementation.

Randomized Queue

RandomizedQueue API

int size()	Returns the number of items in the RandomizedQueue
void enqueue(T obj)	Adds obj to the queue
T dequeue()	Remove and return a random item
T sample()	Return a random item, but do not remove it
String toString()	Shows the elements in random order

Problem 9

A randomized queue is similar to a stack or queue, except that the item removed is chosen uniformly at random from items in the data structure. Create a generic data type for a RandomizedQueue data structure using a resizing array implementation.

Problem 10

Create a generic data type for a RandomizedQueue data structure using a linked list implementation. Think about why this would be more efficient than using a circular array implementation.

Problem 11

Write a method `removeDuplicates()` which removes duplicate elements from an ArrayList. You may use Java's ArrayList for this or your own ArrayList that uses a resizing array implementation.

```
removeDuplicates([1, 2, 3, 2, 1, 5]) // [1, 2, 3, 5]
removeDuplicates([1, 1, 1, 1, 2, 3]) // [1, 2, 3]
```

Stacks and Queues

Problem 12

Most modern text editors have bracket highlighting where it will tell you if your parentheses or brackets are mismatched. Given a string containing text, or code, return `true` if the parentheses `()`, curly braces `{}`, and square brackets `[]` are balanced correctly, and `false` otherwise. Hint: this is a stack problem.

Problem 13

The goal of this problem is to write a method `evaluate(String expression)` that takes a "fully parenthesized arithmetic expression" and returns the evaluation as a `double`. A fully parenthesized arithmetic expression looks like this $(1 + ((2 + 3) * (4 * 5)))$. The parentheses are balanced. Here each character has a space between it, but you might want to ignore those spaces in your own program after using the string's `replaceAll()` method. It might seem like a daunting task to write a program that does this, but E.J. Dijkstra came up with a simple idea in the 1960s to perform such an operation.

He suggested you use two stacks, one for operators and one for values. Traverse the expression from left to right with the following operations:

- Push operands to the operand stack
- Push operators to the operators stack
- Ignore left parentheses
- On encountering right parentheses, pop an operator, pop the resiquite number of operands, and push the resulting value to the operand stack.

After the final right parentheses is processed, the operand stack will have a single value on it: the result of the expression.

Your evaluator should handle the operations +, -, *, /, and square root as `sqrt()`.

You may only consider expressions with full parentheses but think about how you might add precedence.

```
evaluate("( 1 + ( ( 2 + 3 ) * ( 4 * 5 ) ) )"); // 26
evaluate("sqrt(((7 * 3) + (6 - 2)))");          // 5
evaluate("(2 + 1.5)");                          // 3.5
```

Problem 14

Given a strings representing an expression in Reverse Polish Notation, evaluate it. You can read about Reverse Polish Notation here https://en.wikipedia.org/wiki/Reverse_Polish_notation. Hint: this is a stack problem.

```
evaluateRPN(String expression)
evaluateRPN("3 4 -")          // -1
evaluateRPN("3 4 + 5 6 + *") // 77
evaluateRPN("13 12 + sqrt")  // 5
evaluateRPN("13 12 + 7 +")   // 32
```

Linked List Problems

Problem 15

Write a method `getMiddle()` that returns the data corresponding to the middle of a linked list by using two pointers. A `LinkedList` has only an attribute which is a reference to the first node. This means you can find the middle of a linked list by traversing the linked list only one time. Hint: use a "fast" pointer that traverses two nodes at a time and a "slow" pointer that traverses one node at a time.

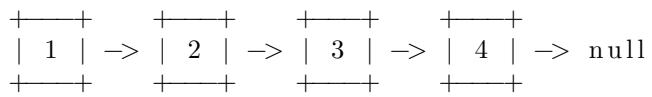
Problem 16

Given a linked list, determine if the linked list has a cycle in it by filling in the method `hasCycle()`. A `LinkedList` has only an attribute which is a reference to the first node.

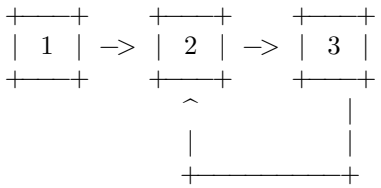
There is a cycle in a linked list if there is some node in the list that can be reached again by continuously following the next pointer.

Return `true` if there is a cycle in the linked list. Otherwise, return `false`. Hint: use two pointers.

No Cycle:



Cycle:



Problem 17

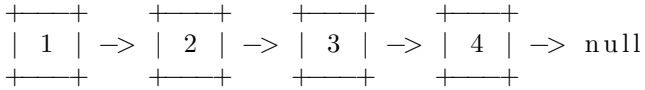
Write a method `reverse()` that reverses a `LinkedList` (the head becomes the tail and the tail becomes the head). A `LinkedList` has only an attribute which is a reference to the first node. Your algorithm should work in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space and should reverse the list in place. Hint: think of this like a swapping algorithm that uses a temporary variable.

Problem 18

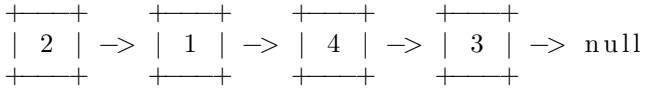
Write a method `removeAll(Item key)` that removes all elements from a `LinkedList` with a particular value. A `LinkedList` has only an attribute which is a reference to the first node. Your algorithm should work in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space and should run in place.

Problem 19

Write a method `swapInPairs()` that swaps every two adjacent nodes. You must solve the problem without modifying the values in the list's nodes (i.e., only nodes themselves may be changed.) Your algorithm should work in $\mathcal{O}(n)$ time and $\mathcal{O}(1)$ space and should run in place.



becomes



Problem 20

Write a static method `merge(LinkedList<Integer> li1, LinkedList<Integer> li2)` that merges two sorted linked lists into one sorted `LinkedList<Integer>`. Return the sorted `LinkedList<Integer>`.

```
list1 = 2 -> 4 -> 10 -> 11 -> 12
list2 = 1 -> 3 -> 5

return: 1 -> 2 -> 3 -> 4 -> 5 -> 10 -> 11 -> 12
```