STUDY GUIDE

Stacks

* Stacks can be viewed as a pushdown method or a last-in-first-out (LIFO) data structure.
* This type of structure operates on the same principle as a Pez Dispenser – used for storing candy. To fill a Pez, you have to insert the candy from the top (push). As the candy is being pushed down, the last piece of candy inserted will be the first one removed (popped), hence the LIFO structure.
* This structure of stacks is integrated with the doubly linked list method. While adding new data, we simply rename the head of the linked list to be at the top. When removing data we access the head, or last data added, to retrieve the following data in the stack.
* In relation to the pop and push method is the peek method. This allows the user to retrieve the data information without having to remove a link or change the top value.

Queues

* In contrast, queues use a systematic structure of the first-in-first-out (FIFO) principle.
* This type of structure operates on the same principle as fast-food restaurants. When food is ordered it is placed at the end of the list (enqueue). As the orders are made complete, the remaining orders become closer to the front of the list and popped from the list (dequeue), hence the FIFO structure.
* The head and tail value change when data is added to a queue-structured linked list. If data is placed in enqueue, the new data becomes the value of tail. Oppositely, if data is placed in dequeue, data next in queue becomes the value of head.

Priority Queues

* Priority Queues are similar to queues but offer a managing feature that implements items of priority.
* This structure maintains the same principle of FIFO. Although, data is sorted by highest priority when added as the tail. This feature allows new data to be executed based on its importance, rather than the order of arrival.

The Heap

* The heap is a structure that allows priority queues to be maintained more efficiently. For example, the Heap uses a top-down effect that orders data in a branching method. This method of branching, similar to a tree, allows faster access to data access. This increased efficiency is possible because the heap doesn’t need to access all links, only the links of higher priority.

|  |  |
| --- | --- |
| STACKS | QUEUES |
| sTACKING DISHES ON TOP OF EACH OTHER | A TICKET LINE |
| tHE JAVA VIRTUAL MACHINE | AN ESCALATOR |
| THE UNDO BUTTON IN MICROSOFT WORD | A CAR WASH |
| CONTINUING A SAVED VIDEO GAME | PRINTER QUEUE |
| TENNIS BALL CONTAINER | KEYBOARD STROKES |

Constructor: A special method that instantiates the instance variables of the class

UML diagram: A graphical display of the variables, methods, parameters, return values, and public or private accessibility of an object or class in a software package.

Flat file: A file containing records that have no structured interrelationship.

Parse: To break a sequence of text characters into syntactic tokens based on separator and to obtain a semantic.

Random access: the ability to directly access any element in a data structure.

Index: a number used to track changes in the logical storage of data rather than the physical storage of data.

Array: a sequence of elements of the same data type.

JVM: Java Virtual Machine

Word to instantiate an object: new

Accessor method: get

Keyword to use Java classes: import

User enters a value to signal end of input: sentinel value

Private int value: private is the accessor modifier, int is the data type, and value is the name of the instance variable.

The first element in the array: when computing the maximum value in an array of integers.

args is an array of Strings: public static void main (String [] args)

Dequeue: A linear list data structure in which data can be added to or removed from the list at either end but only at the two ends.

False: In a stack represented by a linked list, we delete, or pop, based on the value of the item stored in a node.

True: In a sorted linked list, we insert in such a way that the list is still sorted after the insertion.

True: In a queue represented by a linked list, we delete, or dequeue, at the beginning of the list.

True: A doubly-linked list allows us to go either forward or backwards from a given node.

False: In a queue represented by a linked list, we delete, or dequeue based on the value of the item stored in a node.

True: When we successfully delete an item from a list; the number of items in a list decreases by 1.

True: In a stack represented by a linked list, we insert, or push, at the beginning of the list.

False: In a queue represented by a linked list, we insert, or enqueuer, at the beginning of the list.

False: In a stack represented by a linked list, we delete, or pop, at the end of the list.

When inserting an element in the middle of a doubly-linked list, how many pointers need to be reset? 4

When deleting an element in the middle of a doubly-linked list, how many pointers need to be reset? 2

4 Parts of a basic program

Data Payload (lowest part of program) – Holds the actual data, and uses methods to manage stored data.

Data Structure – This class holds instance variables and methods to manage the data structure

Application – This class contains code for implementing features of an application.

Driver (Highest part of program) – The driver contains the Main class. The driver program runs the program, normally doing three things: opening files for input/output, creating instances of the “application” class, and finishing the program by cleaning-up and terminating the program.

Phone Book Application (4 Parts)

Data Payload – Record class

Data Structure – ArrayList class

Application – FlatFile class

Driver – Main class

**Why use encapsulation?**

The purpose of encapsulation gives the programmer the ability to write portable code. Meaning, if we swapped the payload (Record) with another payload, the program would need little to no change for a successful operation. Also, encapsulation divides blocks of code into a “need to know” basis. This is efficient because it avoids data to be fatally altered.

**Why use Generics?**

The generic construct is used for flexibility, efficiency, and portable code. It’s flexible because it allows programs to be written with raw data types and allows the users to specify the data type. Also, it is very efficient because it catches errors during compile time and not runtime. This allows for easy debugging.

**Priority Queue?**

A priority queue functions the same as a queue, adding items from the tail and dequeing them from the head. Furthermore, the priority queue uses an insertion-sort to move a node higher in the queue if it’s priority is of higher value. This check will be done at the beginning of every newly queued piece of data. Depending on the priority set, a node may be added at the end of the list and then immediately after be placed at the front of the queue, if its priority is number one. An example of a priority queue can be witnessed at airports. Aircraft that approach landing strips are given a priority queue based on importance, fuel levels, weather, and many more factors. Depending on the priority each aircraft waits for their turn to land in the runway.

XML STACK

|  |  |
| --- | --- |
| <phonebook>  <name>  John Smith  </name>  <number>  555.1212  </number>  </phonebook> | 8: (tag, phonebook)  7: (tag, number)  6: (data, 555.1212)  5: (tag, number)  4: (tag, name)  3: (data, John Smith)  2: (tag, name)  1: (tag, phonebook) |

**BINARY TREES**

Nodes with children are called Parents.

Children with no nodes are called Leaves or External Nodes.

Nodes with the same parent are called Siblings.

Nodes which are not leaves are called Internal Nodes.

The top Node is the Root.

The depth of a node is the number of edges from the root to the node.

The height of a node is the number of edges from the node to the deepest leaf.

The height of the tree is a height of the root.

A full binary tree is a binary tree in which each node has exactly zero or two children.

A complete binary tree is completely filled, from left to right.

Advantages of Trees

Trees reflect structural relationships in the data

Trees are used to represent hierarchies

Trees provide an efficient insertion and searching

Trees are flexible with data. Allows moving of subtrees with minimum effort

Traversals

Depth-first: Pre-Order, In-Order, Post-Order

Breadth-first: Level-Order

|  |  |  |
| --- | --- | --- |
| Pre-Order  Visit Node/Parent  Visit Left  Visit Right | In-Order  Visit Left  Visit Node/Parent  Visit Right | Post-Order  Visit Left  Visit Right  Visit Node/Parent |