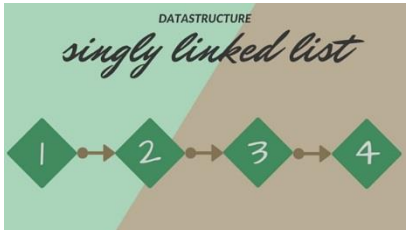
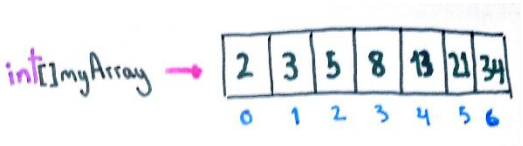
Data Structures

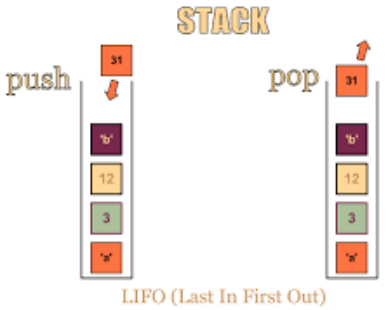
"Get your data structures correct first, and the rest of the program will write itself."
- *David Jones*

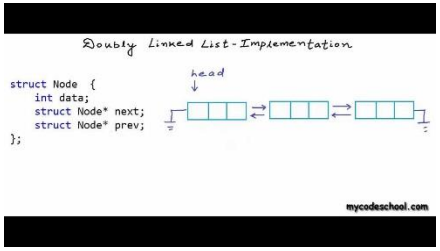
Data Structures

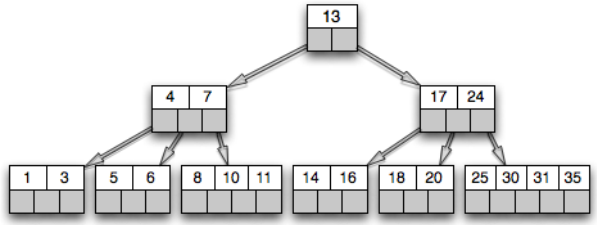
- A *Data Structure* is:
  - "An organization of information, usually in computer memory", for better algorithm efficiency."

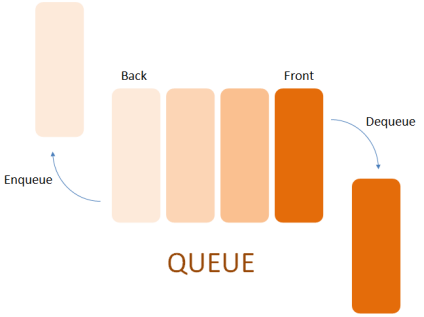












Data Structure Concepts

- Data Structures are containers:
  - they hold other data
  - arrays are a data structure
  - ... so are linked lists

- Other types of data structures:
  - stack, queue, tree,
    binary search tree, hash table,
    dictionary or map, set, and on and on
  - en.wikipedia.org/wiki/List_of_data_structures
- Different types of data structures are optimized for certain types of operations

Core Operations

- Data Structures have three core operations
  - a way to add things
  - a way to remove things
  - a way to access things (without modifying the data)
- Details depend on the data structure
  - For instance, a *List* may need to:
    - add at the end
    - access by location (index)
    - remove by location (index)
- More operations added depending on what data structure is designed to do

Implementation-Dependent Data Structures

- Arrays
  - Collection of objects stored contiguously in memory
  - Accessed through an index
- Linked Structures
  - Collection of node objects
    - Store data and reference to one or more other nodes
  - Linked Lists
    - Linear collection of nodes
    - Single-linked List – nodes contain references to next node in list
    - Double-Linked List – nodes contain reference to next and previous nodes in list
  - Trees
    - Hierarchical structure
    - Nodes reference two or more “children”

Implementation-Independent Data Structures

- Abstract Data Types (ADTs)
  - Descriptions of how a data type will work without implementation details
  - Description can be a formal, mathematical description
  - Java interfaces are a form of ADTs
- Examples:
  - Bag, Set, Stack, Queue, List

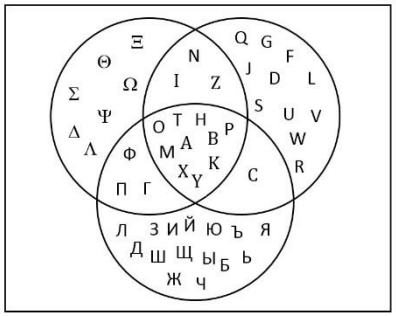
Implementing ADTs

- The operations and behaviors are already specified:
  - For instance, every *Stack* has `push`, `pop` and `peek` methods
  - Think Java interface
- But given an interface describing an ADT, how implement it?
  - Must decide which internal storage container to use to hold the items in the ADT
  - Currently, few choices: arrays anyone?
  - Later add linked structures

Bags and Sets

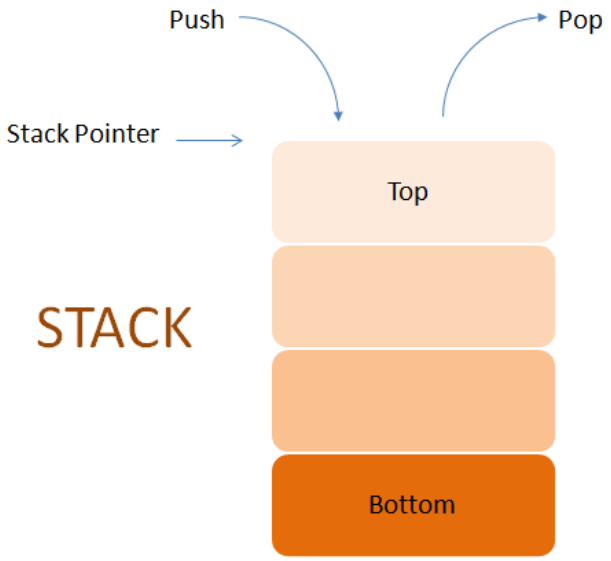
- Simplest ADT is a Bag
  - items can be added, removed, accessed
  - no implied order to the items
  - duplicates allowed

- Set
  - same as a bag, except duplicate elements not allowed
  - union, intersection, difference, subset



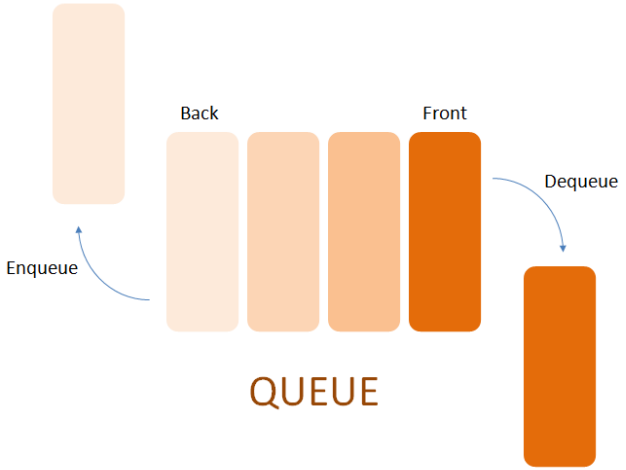
Stacks

- Only access last item inserted
  - Expected behavior: Last in, First out (LIFO)
    - push (put object on top)
    - pop (remove object from top)
    - peek / top (look at object on top)
  - Other useful operations
    - make empty
    - size
    - is empty?



Queues

- Only access item that has been there the longest
  - Expected behavior: First in, First out (FIFO)
    - enqueue (insert at the back)
    - dequeue (remove from the front)
    - front (look at the object at the front)
  - Other useful operations
    - make empty
    - size
    - is empty?

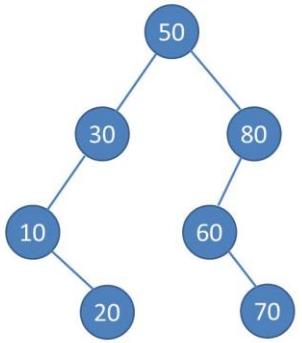


Lists

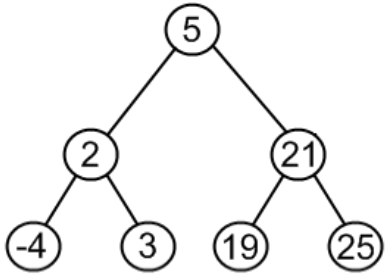
- Linear collection of items
  - Ordered List
    - Items in list are arranged in a pre-determined ordering
      - For instance, alphabetically or by ascending numerical value
  - Indexed List
    - Items are accessed by position in list
    - Additions / deletions can also be done by position
  - Unordered List
    - Items are stored without an implicit ordering

Types of Trees

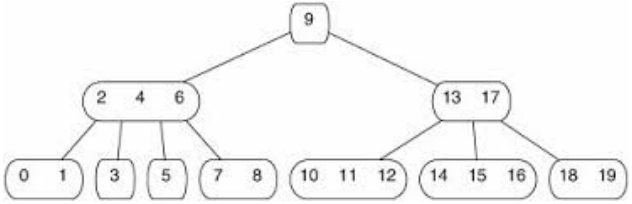
- Binary Search Trees (BSTs)
  - Items stored in sorted order



- Heaps
  - Items stored according to the “Heap Property”
- AVL and Red-Black Trees
  - BSTs that stay balanced



- Splay Trees
  - BST with most recently items at top
- B-Trees
  - Another variation of BST
  - Nodes can have more than two children



Other ADTs

- Hash Tables
  - Hash function:
    - computes an *index* into an array of *buckets* or *slots*
    - look in the bucket for the desired value
- Maps
  - Collection of items with a key and associated values
  - Similar to hash tables
- Graphs
  - Nodes with unlimited connections between other nodes
- Sparse vectors and sparse matrices

Generic Containers

- All Collection classes, including ADTs, should be generic
  - only write them once,
  - hold lots or all types of data
  - Java achieves genericity through inheritance and polymorphism
- ADTs have an *internal storage container*
  - What is storing the stuff?
  - implementation vs. abstraction
  - in Java, usually holds Objects. Why?

ADTs and Data Structures in Programming Languages

- Modern programming languages usually have a library of data structures
  - Java collections framework
  - C++ standard template library
  - .Net framework (small portion of VERY large library)
  - Python lists and tuples
  - Lisp lists

Data Structures in Java

- Part of the Java Standard Library is the *Collections Framework*
- A library of data structures
- Built on two interfaces
  - Collection
  - Iterator
- http://java.sun.com/j2se/1.5.0/docs/guide/collections/index.html

Stacks and Queues in the Java Collection API

- No Queue class in the Java collections API
  - Can implement Queue interface using Java LinkedList class
- Stack extends Vector (which is almost exactly like ArrayList)
  - Hmmm?
- One reason the Java Collections Library is often said to be broken