Week 2 Notes

Prof Bill - Apr 2018

Week 2 notes covering:

A. Arrays

B. Linked Lists

C. Stacks and Queues

thanks… yow, bill

A. Arrays

**** Muganda Ch 7 Array, ArrayList**

7.1 Intro

Some Java syntax:

```
int [] numbers;   // array declaration
numbers = new int[6];   // array created with new

int [] moreNumbers = new int[6];   // both in one stmt
```

Once created, you can't change array size.
/* p 408 - keyboard Scanner idiom is strong! */
/* Java checks array bounds, gives nice error messages… C does not */

7.2 Processing

Public field in every array: numbers.length
For-each loop is easy to use and a strong idiom.

```
for( int each_num: numbers) {
      // each_num is numbers[0… length] in each iteration
}
```

Size of array can be set at run-time.

```
numbers = new int[ numTests];
```

7.3 Arguments

Command line arguments are sent to main() via an array of strings.

```
public static void main( String[] args) {
      …
}
```

7.4 Useful Algorithms

Comparing items in arrays: don't use ==; loop and inspect each items
Also: average, min, max

7.5 Returning arrays

7.6 Arrays of Strings

7.7 Array of Objects

String is an Object (just sayin)

Java sets initial array item values to 0/null.

```
Student[] myClass = new Student[21];
// question: how many Student objects are created in the stmt above?
```

7.8 Sequential Search

Performance is O(n)

7.9 2D Arrays

Google the syntax, or use your IDE
p 453 - nice figure showing organization of 2D array
ragged array - 2D array with different row sizes

7.10 3D Arrays

7.11 Selection Sort, Binary Search

Selection Sort performance = O(n^2)
Binary search = O(log n)
/* we'll dig deeper into many search and sort algorithms later */

7.12 Command-line args

Idiom:

```
public static void main( String[] args) {
      // args is idiom!
}
```

Borrowed from C.

```
int main( int argc, char *argv[] )  {
      …
}
```

7.13 ArrayList

Array that automatically resizes as necessary
Best of both worlds (array, linked list): O(1) get, no max size
Very convenient and popular!
Generic.

```
ArrayList<String> names = new ArrayList();
```

**** Morin Ch 2**

Get is O(1)

Removing an element in array requires shifting of other elements in the array

2.1 ArrayStack

He does, basically, ArrayList.

Amortized analysis - amortize the Big-O cost over multiple operations (resize example)

/* we'll talk more about this later when we dig deeper into algorithm analysis */

B. Linked Lists

**** Muganda Ch 20 Linked List**

20.1 Intro

Array is consecutive cells in memory; linked list is not
Node - one link in the list
self-referential - node has pointer a node, next
p 1242 - Node is an inner class, not accessible to the user of the linked list
Draw your boxes => code

20.2 Operations

Simple, compact interface:
isEmpty, size, add, add(position), remove(position), remove(value)
/* missing get - it's O(n), right? */
head and tail pointers
p 1246 - a simple linked list implementation, we will do this

20.3 Doubly-linked list

Each node has next and prev pointers
Makes remove easier

20.4 Recursion

Meh. Recursion costs more than iteration /* question for 220 students: why? */

**** Morin Ch 3 Linked Lists**

Primary linked list disadvantage vs array: get is O(n)
Another one - Linked list is LOTS of work and $$$ for garbage collection! Fragmented memory.

3.1 Singly-Linked List

See the boxes!
Queue - add to tail, remove from head

3.2 Doubly-Linked List

dummy node - empty node at head of list, so that null checks are avoided

circular list - next of tail node points to the head, again… more null avoidance
/* both of these are style choices more than anything else */

C. Stack and Queues

**** Muganda Ch 21 Stacks and Queues**

Important: this visualization website is a great way to learn these structures!
www.cs.usfca.edu/~galles/visualization/Algorithms.html

21.1 Stacks

LIFO = Last in, first out
Examples: Pez, plates, pancakes, and function calls!

Stack operations:

- ➢ void push(item) - add item to top of stack
- ➢ item pop() - remove top item on stack and return it
- ➢ item top() - return the top item on stack; no change (sometimes called peek())
- ➢ boolean isEmpty() - returns true if stack has no items

Stack is part of Java Collections Framework (JCF)

```
Stack<Pancake> breakfast = new Stack<>();
```

21.1 Array implementation

Array of items plus an integer to track the top of stack
/* must know how to roll your own stacks and queues! */

Array is most popular.

- fixed size; means stack full error possible
+ O(1)
+ best for garbage collection; you don't track a zillion nodes like linked list

Example and code: push, pop

21.3 Linked list implementation

Push/pop each item to/from the head of the linked list
Example and code: push, pop

21.4 Queues

FIFO = First in, first out

Examples: In line at the Jew-el, printer jobs

Queue operations:

- ➔ void enqueue(item) - add item to the end of the queue
- ➔ item dequeue() - remove item from the front of the queue
- ➔ item peek() - return the item at the front of the queue; no change
- ➔ boolean isEmpty() - returns true if the queue holds no items

Queue is an interface in the JCF, usually implemented with JCF LinkedList:

```
Queue<Customer> checkoutLine = new LinkedList<>();
```

The JCF version is a little gross, actually. Operations are: offer, poll, peek (blech)
docs.oracle.com/javase/tutorial/collections/interfaces/queue.html

21.5 Array Implementation

A little tougher than Stack…
Array of items plus two integers: front and rear of queue; circular effect in array

Example and code: enqueue, dequeue
/* Muganda p 1314 - JavaFX gui example */

21.6 Linked List Implementation

Use head and tail of linked list to track front and rear of queue
Example and code: enqueue, dequeue

21.7 Generics

Similar to the JCF

21.8 Queues and Breadth-first Search

We'll do this… later!

Also...

- ❖ **Deque** (pronounced "deck") is a double-ended queue… add/remove from front/rear; which is just a general list
- ❖ **Priority queue** - very important structure; very different as items are ordered by priority, not when they're added; more later!