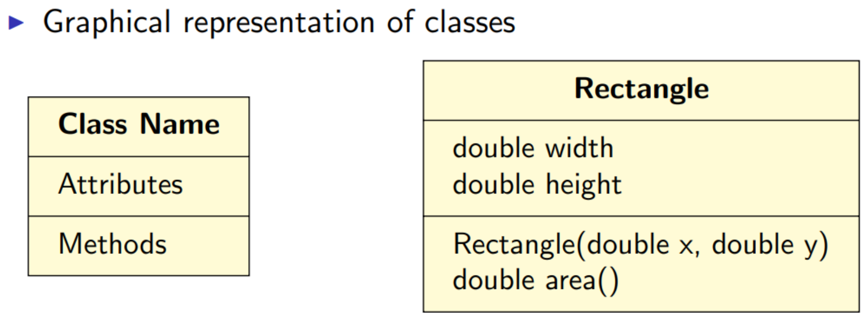
Humna Sultan

# Essentials of CS284 (Spring 2023, Prof. Akcam)

**Data Structures in Java!**

## 1/18/23

- Object Oriented Programming
  - A set of entities that collaborate with each other in order to perform some specific task
    - These entities are called objects
- Java is a collection of classes
  - Class - a named description for a group of entities that have the same characteristics
    - Entities: objects
    - Characteristics: attributes (DATA FIELDS) for each object and the operations (METHODS) that can be performed on these objects
- UML Diagrams



► Graphical representation of classes

**Class Name**
Attributes
Methods

**Rectangle**
double width
double height
Rectangle(double x, double y)
double area()

  - 
    - Rectangle(double x, double y) → Constructor (matches class name!)
- Access modifiers: public and private access
  - Prefer private access majority of the time in industry - you do not want all methods to have access
  - For this class - most of the time, methods will be public, attributes will be private
- Constructor
  - Does not have return type
- Java uses curly braces :)

## 1/20/23

- Java has a variety of primitive data types
    - byte, short, int, long, float, double, char boolean
    - Class names are also types!
- Special support is provided for Strings through the java.lang.String class
- How many bits from -128 to 127?
    - 8 bits, 1 byte
- How many bits in a short (-32768 to 32767)
    - 16 bits, 2 bytes
- Static method references / is used for several objects
    - Indicates that a method is a class method
    - You can have multiple static methods, but you cannot have two methods with the same name if one is static
        - You can have instance methods of the same name in Java
    - Called using dot notation (Rectangle.getNumberOfRectangles(); )
    - Static methods cannot call instance methods
- Constructors initialize data fields
    - You can make multiple constructors
- ++ shortcut is add one (increment), - - shortcut is minus one (decrement)
- Instance methods = non-static methods
- Static methods = class methods
- Objects are instantiated using the “new” keyword
    - You can make as many instances as needed
    - If you do not use a new keyword, you make an instance but you do not allocate space for it

★ Rectangle Project used on 1/20/23!! (https://github.com/humnasul/cs284notes/tree/main/class%20code/oop/Rectangle)

## 1/23/23

★ Person Project used on 1/23/22!!
(https://github.com/humnasul/cs284notes/tree/main/class%20code/oop/Person)

- If the attribute consists of two words, use camel case
- /** xxx */
- private static final int SENIOR_AGE = 65;
  - final keyword means value cannot be changed
  - Capitalize final variables for readability
- Constructor initializes variables for use
- You need to initialize an array of a certain size or with values
  - Arrays start at 0
- Arrays are initialized by default in Java
  - int[] scores = new int[5] initializes [0,0,0,0,0]
- names.length → gives you length of array names
- Enhanced for loop

```
for (int i : scores) {
        System.out.println(i);
}
```

- 2D arrays have rows and columns
  - double matrix[][] = new double[ROWS][COLS]

## 1/25/23

int x = 4
double y = x

- This code works

double x = 8.9
int y = x

- This code gives an error

```
String greeting = "Welcome";
String welcome = greeting;
System.out.println(welcome);
greeting = "hello";
System.out.println(welcome);
```

- Both print statements print “Welcome”
    - Value of welcome does not change when variable greeting changes

```
int[] data1 = {1,2,3,4,5};
int[] data2 = data1;
System.out.println(data1[0]);
data2[0] = 8;
System.out.println(data1[0]);
```

- First print statement prints 1, second print statement prints 8

- Strings are immutable, values within them cannot be changed
- You must make an object to call non-static methods from main (because main is static)
    - Do not make everything static to fix the problem!

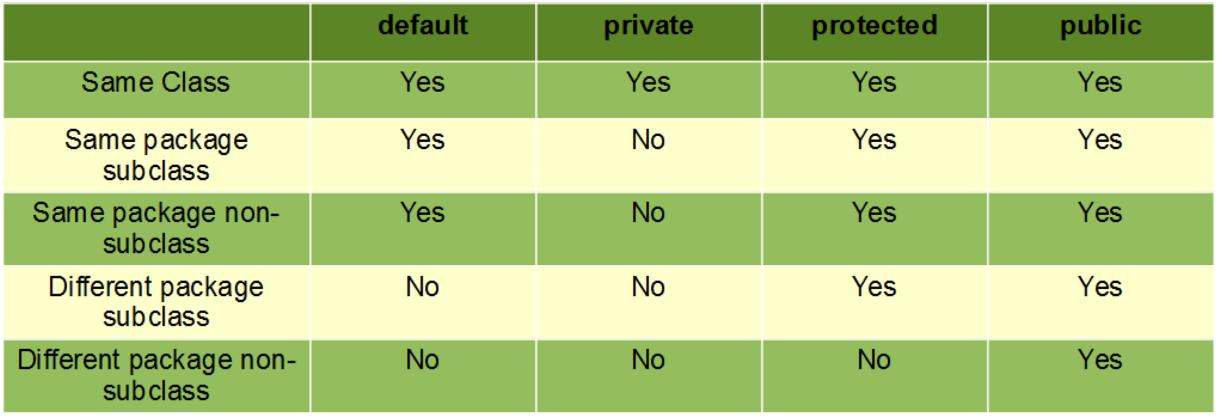
- Java math class has math functions for use

## 1/27/23

- You can have multiple constructors with different parameters
    - They must have different parameters - otherwise error
- If a class extends another class (inheritance), you need to call super() from the constructors to reference the superclass

```
public Rectangle(){
  this(0,0);
}
```

- Even if the two variables referenced by this() are doubles, the 0 and 0 will be automatically converted



| | default | private | protected | public |
|---|---|---|---|---|
| Same Class | Yes | Yes | Yes | Yes |
| Same package subclass | Yes | No | Yes | Yes |
| Same package non-subclass | Yes | No | Yes | Yes |
| Different package subclass | No | No | Yes | Yes |
| Different package non-subclass | No | No | No | Yes |

- https://www.geeksforgeeks.org/access-modifiers-java/
- **For this class**, use private for data fields and use public / default for methods

# 1/30/23

- Method overloading
  - Runtime specifies which toString() to utilize
  - You can have multiple methods of the same name, but different parameter lists
    - A different return type does not count as overloading

| Overriding | Overloading |
|---|---|
| - Methods have<br>  - Same name<br>  - Same parameter list<br>  - Same return type<br>- Final methods cannot be overridden<br>- The method will be overridden in subclass (multiple classes involved)<br>- Static methods cannot be overridden<br>- Runtime polymorphism | - Methods have<br>  - Same name<br>  - Different parameter list<br>- Methods are defined in the same class<br>- Compile-time polymorphism<br>- Static methods can be overloaded<br>- Cannot overload by return type |

| Interface | Abstract Class |
|---|---|
| - Only can have abstract methods<br>- Only have static & final variables<br>- Cannot provide an implementation of abstract class<br>- Uses the keyword “implements”<br>- Can extend another Java interface only<br>- Members must be public | - Can have abstract & non-abstract (concrete) methods<br>- Can have non-final variables, final, static, non-static variables<br>- Can provide an implementation of an interface<br>- Uses the keyword “extends”<br>- Can extend another Java class, implement multiple interfaces<br>- Members can be private, protected, default |

- Abstract classes
  - You cannot create an object declaration inside
  - Use an abstract class when you need a base class for two or more subclasses that share some attributes
  - An abstract class can implement an interface, but an interface cannot extend an abstract class
-
  ```
  public abstract void drawHere();
  ```
  - In an abstract class, the subclasses of the abstract class must have this abstract method
    - Almost like delegating tasks to subclasses
  - Can have constructors
    - Abstract classes have variables, you may want to initialize variables

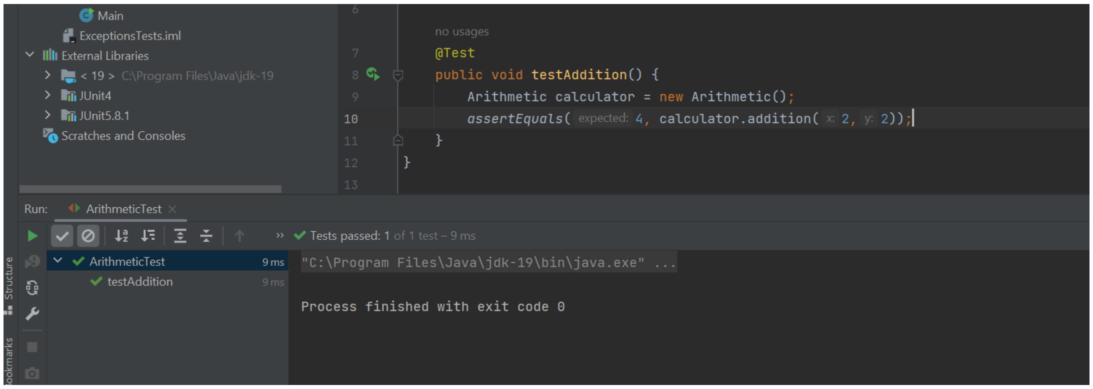
        - You cannot make objects that can use the constructors, but subclasses will have super() that can access the abstract class’ constructor
    - An abstract class can implement an interface but does not have to define all the methods of the interface
        - Implementation of methods is responsibility of subclasses
- You cannot have more than one base class / superclass for a subclass
- Abstract classes and Interfaces both cannot have objects / be instantiated

- ★ Used IntelliJ Rectangle project (https://github.com/humnasul/cs284notes/tree/main/class%20code/oop/Rectangle)

## 2/1/23

- Methods to call are determined at runtime when you use polymorphism
- Object.equals() method checks if two objects are equivalent
- getClass() returns the runtime class of the object "this".
- Downcasting - converting superclass type to subclass type

## 2/3/23

- “Throwable” class is superclass to all classes
- Checked exceptions
  - Normally not due to programmer error
  - Beyond control of programmer
  - Ex: IOException, FileNotFoundException
- Unchecked exceptions
  - Programmer error
- Process finished with exit code 0 : successful (System.exit(0); )
- Process finished with exit code 1: interrupts execution of program (System.exit(1); )
  - There is an error in the code and execution stopped
- If trying to add JUnit to a project, do @Test in the file and do alt + enter
  - JUnit 5.8.1
  - 

    ```
    no usages
    @Test
    public void testAddition() {
        Arithmetic calculator = new Arithmetic();
        assertEquals( expected: 4, calculator.addition( x: 2, y: 2));
    }
    }
    ```

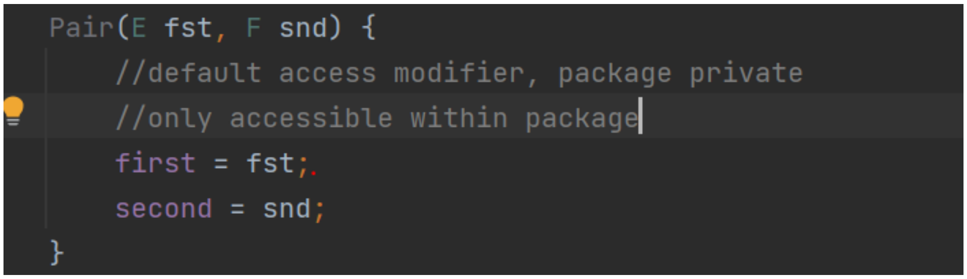
    ```
    Tests passed: 1 of 1 test – 9 ms
    ArithmeticTest 9 ms
      testAddition 9 ms
    ```

    ```
    "C:\Program Files\Java\jdk-19\bin\java.exe" ...

    Process finished with exit code 0
    ```

    - Make sure you have multiple test cases: 2 + 2 = 4 AND 2 * 2 = 4
- Testing functions in JUnit : assertEquals, assertNotEquals, assertNull, assertTrue, assertFalse

## 2/6/23

- Lists
- ArrayList
  - Arrays have a fixed size
  - Constant time access to elements
  - Removal is linear
  - Insertion is linear
  - .add() adds elements chronologically
- LinkedList
  - Grow and shrink - no fixed size
  - Linear time access
  - Linear time insertion and removal (except if previous element is supplied, then constant)



```
Pair(E fst, F snd) {
    //default access modifier, package private
    //only accessible within package
    first = fst;
    second = snd;
}
```

★ Used Lists project ([https://github.com/humnasul/cs284notes/tree/main/class%20code/lists%2C%20sets%2C%20etc/Lists](https://github.com/humnasul/cs284notes/tree/main/class%20code/lists%2C%20sets%2C%20etc/Lists))

## 2/8/23 + 2/10/23

- ★ 2/8/23 - coded in Lists project in IntelliJ (https://github.com/humnasul/cs284notes/tree/main/class%20code/lists%2C%20sets%2C%20etc/Lists)

- Algorithm efficiency
    - Getting a precise measure of the performance of an algorithm
    - Using Big-O notation expresses the performance of an algorithm as a function of the number of times to be processed
    - Processing time increases in proportion to the number of inputs n
    - Target
        - If target is present, for loop will execute (on average) (n+1) / 2 times
        - If target is not present, for loop will execute n times
    - O(n) is referred to as Big O
        - Linear growth rate
    - n * m occurs when there are nested loops - each iteration of of the outer loop, n, causes iterations of the inner loop, denoted as m (inner loop must be constant time, like an if)
        -
    - O(n^2) → quadratic growth rate
        - Ex: for loop inside for loop
            - If there if an if inside both nested for loops, we don't consider the time spent processing the for loop because it is trivial in comparison
    - If you have a nested if and you have two conditions compared with &&, replacing the && with || reduces the time because only one of the conditions has to be checked / be true
        - Max number of times is 2
    - Logarithmic growth rate (asked on tests)
        - Pay attention to where for loop starts, where for loop ends, and what happens within the loop
        - O(logbase2 n)

<u>Notations : in order from most efficient to least</u>

1. O(1) - constant
2. O(log n) - logarithmic
3. O(n) - linear
4. O(n logn) - log linear
5. Popular for sorting algorithms
6. O(n^2) - quadratic
7. O(n^3) - cubic
8. O(2^n) - exponential
9. O(n!) - factorial

- 2n = O(n) → the constant 2 is irrelevant

- f(n) = O(g(n))
    - We can say that g(n) is the the max growth rate
- O(g(n)) is a set of functions
- The c-value provides the upper bound
    - Creates a tighter bound
- Comparing growth rates: 100n == 7n + 50
    - Both are dependent on n, BUT the constants (7 and 100) don’t make a big difference)
    - Both are linear time, both have a time of Big O(n)

## 2/13/23

- 0 <= f(n) <= c g(n)
    - т(n) = n^2 + 5n + 25
    - 0 <= n^2 + 5n + 25 <= c n^2
        - 1 + 5/n + 25/n^2 <= c
        - no = 5, c = 3; 1 + 5/5 + 25/25
    - 1 + 1 + 1 <= c
    - 3 <= c

★ Reviewed Big O example 2 in slides

## 2/15/23

- exponential and factorial algorithms have a much longer runtime
- Single linked list (singly)
    - Link only goes in one direction
- Double linked list
    - next = node → associates with data

★ “Linked” package inside “Lists” project in intelliJ ([https://github.com/humnasul/cs284notes/tree/main/class%20code/lists%2C%20sets%2C%20etc/Lists/src/Linked](https://github.com/humnasul/cs284notes/tree/main/class%20code/lists%2C%20sets%2C%20etc/Lists/src/Linked))

## 2/22/23

- Linked lists
    - When head == null, tail also == null
        - You need to assign both
    - Establish links for the new nodes first, then do previous and next of the previously existing nodes
    - How does a compiler know where to place a node?
        - Provide the location or iterate through starting with the head or tail for placement

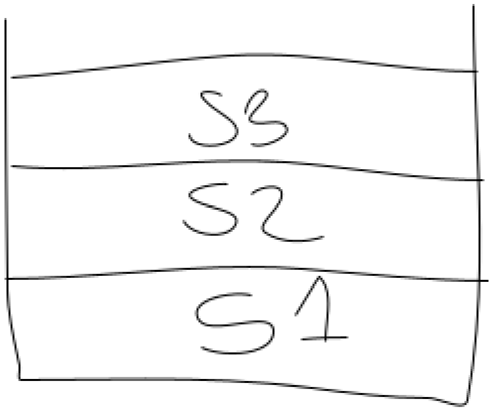
## 2/24/23

- Big O - Constants get dropped
- Quiz answers

1. O(n)
2. n/2 * n = O(n^2)
3. C = 2, n0 = 4
    - a. If you have 2 positive numbers choose the larger value for n0 relating to the solution
        - i. You want to achieve the higher intersection point

- Linked lists
    - EX: 12 - 15 - 20 - 25
        - If you want to remove head (12) …
            - head = head.next;
                - Points to next value
            - head.prev = null;
                - Ensures that the value before the newly assigned head is null
        - If you want to remove tail (25) …
            - if (tail.data == number)
                - tail = tail.previous;
                - tail.next = null;
        - Remove internal node (20)
            - current = head;
            - loop until (current.data = number)
            - current.next.previous = current.previous;
            - current.previous.next = current.next;
    - Make sure head and tail aren't null for you to start at
        - Elements before head should be null and elements after tail should be null
    - Insertion at head or tail is O(n), insertion anywhere else is O(n)
- Intro to stacks
    - Last-in, first-out = LIFO
    - Operations:
        - empty() - test for an empty stack
        - peek() - inspect the top element
        - pop() - retrieve the top element
            - Removes top element and changes the top element (head)
        - Put a new element on the stack (E push (E obj) )
            - Adds element to the top (changing top element aka head)
    - Palindromes - reading from bottom or top will be the same word
        - tail != liat

★ Both pop() and push() alter the head



- Last one you put if the first one you take out

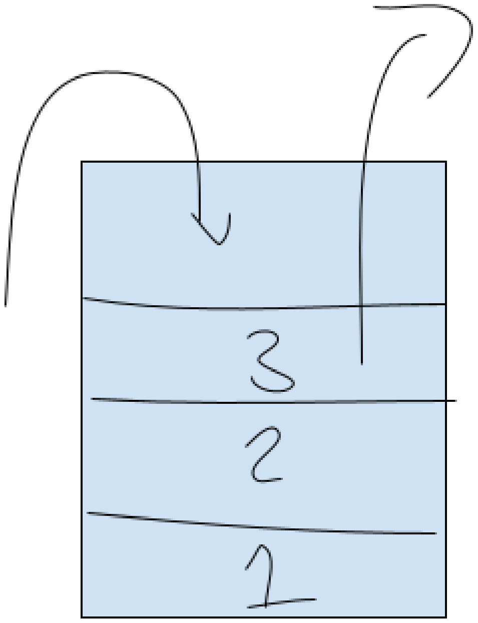
- Test questions
    - For this provided link list, implement an insert method

## 2/27/23

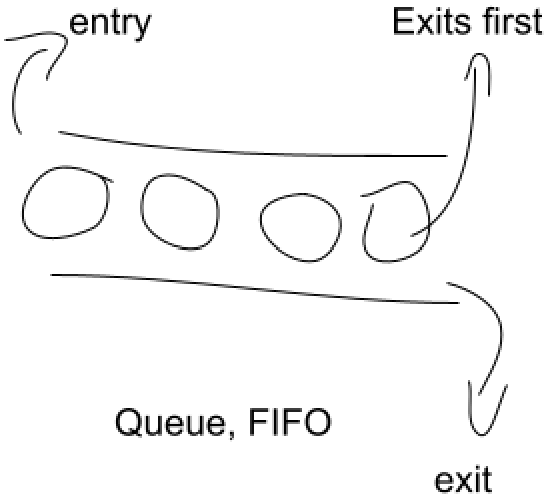
- StringBuilder adds functions to String to allow you to easy access diff parts and manipulate
- Palindromes
    - Ignore case
    - buildReverse() → returns reversed String
- Stacks can be used as a solution to counting balanced parenthesis
    - Have a stack with parenthesis and if the size is even then parenthesis are done properly
    - Have two stacks - one with open parenthesis and one with closed parentheses; make sure the sizes are equal
- Vectors
    - Can grow and shrink in size
    - We use vectors to implement stacks in java
        - Vector applications can be applied to stacks as a result
- Stack program
    - Embed node class into linkedstack file so that the linkedlist and linkedstack can use nodes directly as elements

## 3/1/23

- Queues
  - Widely used data structure
  - FIFO list : first-in, first-out
    - Opposite of stack
  - Examples of use
    - Operating systems
      - Tasks and resources
    - Printing queues
  - Queues ensure that tasks are done in the order they were generated

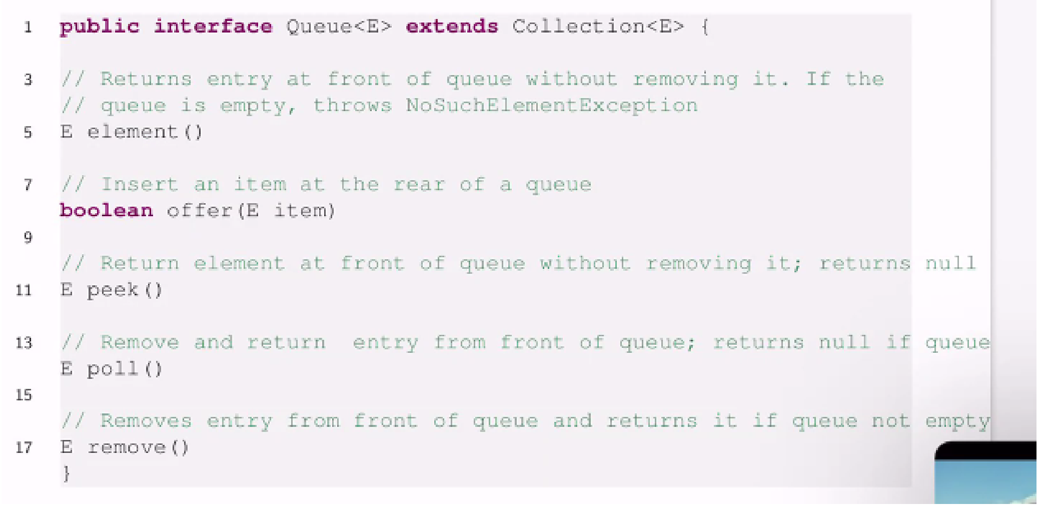


Stack, LIFO



Queue, FIFO

- Queue functions
  - queue.offer(x)
  - queue.peek()
- Queue interface

    - 

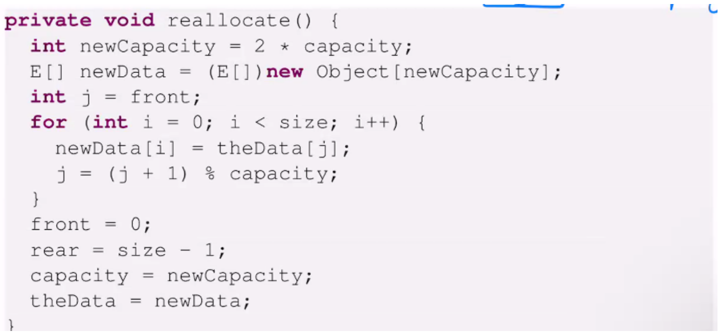
```
public interface Queue<E> extends Collection<E> {

// Returns entry at front of queue without removing it. If the
// queue is empty, throws NoSuchElementException
E element()

// Insert an item at the rear of a queue
boolean offer(E item)

// Return element at front of queue without removing it; returns null
E peek()

// Remove and return  entry from front of queue; returns null if queue
E poll()

// Removes entry from front of queue and returns it if queue not empty
E remove()
}
```

- https://docs.oracle.com/javase/7/docs/api/java/util/Queue.html#:~:text=Interface%20Queue&text=A%20collection%20designed%20for%20holding,%2C%20extraction%2C%20and%20inspection%20operations.

- Simulations + Queues
    - Simulations are used to study the performance of a physical system by using a physical, mathematical, or computer model of the system
        - Queueing theory
    - EX: used to simulate structure of bridges etc.
- Simulation Example: Blue Sky Airlines
    - Creating 2 waiting lines
    - Strategies:
        - Take turns serving passengers from both lines so that the average time for all is not too long
        - Serve the passenger waiting the longest
        - Serve any frequency flyers before any regular passengers
            - Will annoy regular passengers
    - Consider potential events in time intervals

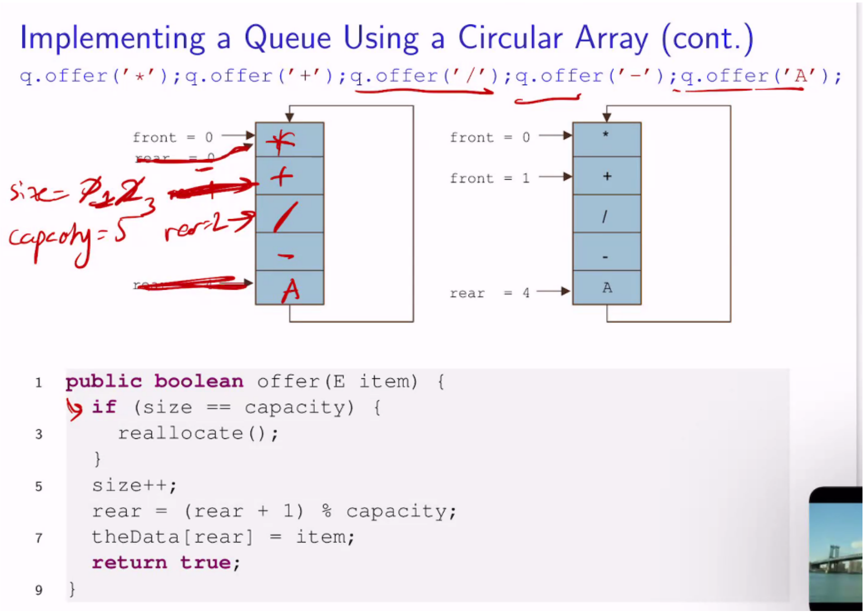
★ IntelliJ - Look at StackInt project → Queues package (https://github.com/humnasul/cs284notes/tree/main/class%20code/data%20structures%20in%20java/StackInt/src/Queues)

## 3/3/23

- Peek - returns data from the front
- Using a single-linked list to implement a queue
    - public class ListQueue<E> implements Queue<E> { }
- Implementing a queue using a circular array
    - The time efficiency of using a single or double linked list to implement a queue is acceptable
    - Storage space is increased when using a linked list due to references stored in the nodes
    - TIME EFFICIENCIES AND STORAGE ISSUES CAN BE PREVENTED USING… circular arrays!!
- Circular arrays
    - Rear is connected to the front of the array
    - If the first element is empty but the rest are filled (5 spots total), the rear is 4+1 % 5 which is 0, going back to the front!
- Reallocate:
    - 

```
private void reallocate() {
  int newCapacity = 2 * capacity;
  E[] newData = (E[])new Object[newCapacity];
  int j = front;
  for (int i = 0; i < size; i++) {
    newData[i] = theData[j];
    j = (j + 1) % capacity;
  }
  front = 0;
  rear = size - 1;
  capacity = newCapacity;
  theData = newData;
}
```

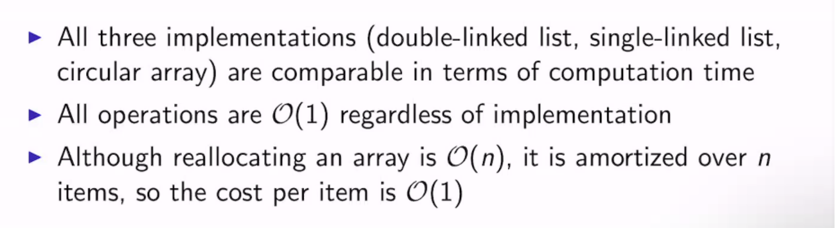
- Links:
    - [https://www.baeldung.com/java-generic-array](https://www.baeldung.com/java-generic-array)
    - [https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html](https://docs.oracle.com/javase/8/docs/api/java/util/Iterator.html)



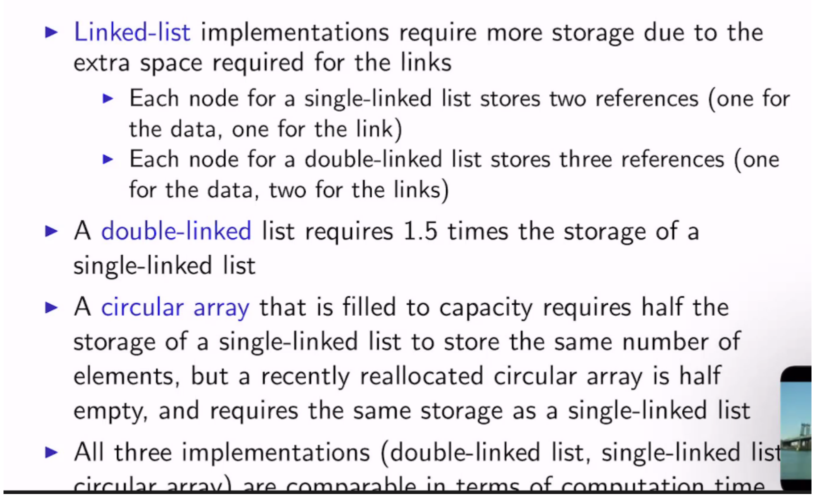
## Implementing a Queue Using a Circular Array (cont.)

q.offer('*');q.offer('+');q.offer('/');q.offer('-');q.offer('A');

```
public boolean offer(E item) {
  if (size == capacity) {
    reallocate();
  }
  size++;
  rear = (rear + 1) % capacity;
  theData[rear] = item;
  return true;
}
```

- 

  - All three implementations (double-linked list, single-linked list, circular array) are comparable in terms of computation time
  - All operations are $\mathcal{O}(1)$ regardless of implementation
  - Although reallocating an array is $\mathcal{O}(n)$, it is amortized over $n$ items, so the cost per item is $\mathcal{O}(1)$

- 

  - Linked-list implementations require more storage due to the extra space required for the links
    - Each node for a single-linked list stores two references (one for the data, one for the link)
    - Each node for a double-linked list stores three references (one for the data, two for the links)
  - A double-linked list requires 1.5 times the storage of a single-linked list
  - A circular array that is filled to capacity requires half the storage of a single-linked list to store the same number of elements, but a recently reallocated circular array is half empty, and requires the same storage as a single-linked list
  - All three implementations (double-linked list, single-linked list circular array) are comparable in terms of computation time

★ Look at queues iterator implementation on Canvas

- ○ Stacks do not implement iterators because it is implied that (https://github.com/humnasul/cs284notes/blob/main/class%20code/data%20structures%20in%20java/ListQueue-Iterator.java)

- ➢ TEST TOPICS
  - One question runtime analysis
  - One question lists
  - One question java basics
  - One questions stacks or queues
  - UML diagram
  - Visibilities for methods etc.
  - 6 questions total - one is bonus (5 required)
  - TOPICS: All topics on canvas + queues
    - To review: abstract classes, interfaces, etc.
- ➢ NOTES FROM TEST REVIEW
  - Abstract classes can have constructors
    - You can call super();
  - Interfaces can only have static and final attributes
  - Know difference between overloading / overriding
    - Overloading is within same class
    - Overriding is subclass overriding a superclass
  - Animal - bird (inheritance relationship)
    - Animal - toString(), getName()
    - Bird - toString(), getSeeds()
    - Animal aBird = new Animal();
      - aBird.toString();
        - Will call animal's toString()
    - Animal aBird = new Bird();
      - aBird.toString();
        - Will call bird's toString()
        - toString method is overridden, so the bird class's' method will be called
          - At runtime, it binds to include the overridden bird methods
      - aBird.getSeeds();
        - Not a valid call!! - syntax error
      - ((Bird) aBird).getSeeds();
        - This will cast aBird as a bird so you can access the method
      - Bird.getName();
        - Will call getName from Bird class (overridden)

ADDITIONAL NOTES

- Public can be accessed by any class, private can only be accessed by the class it is defined in
- Polymorphism allows for several implementations of the same interfaces
- You can only extend one class
- Arraylists - removal and insertion is linear time; not fixed size
  - You need to use the non-primitive type when making an array list

        - EX: ArrayList<Integer>, ArrayList<Boolean>
- SLL’s do not have a previous field

## 3/10/23

- Recursion - method calls itself in order to make problem smaller until it reaches a base case
- Stacks are utilized for recursion!!
- When a method goes recursively infinitely, there is a StackOverflowError
- Tail recursion: calculation is embedded, better for performance
    - Second / new parameter used in tail recursion = accommodator
- In general, loops have a smaller runtime than recursion
    - Recursive solutions are used for readability - 100 lines can become 3 lines

## 3/20/23

- Trees are non-linear and hierarchical
- Tree nodes can have multiple successors BUT one predecessor
    - More than one predecessor = graph
- Trees are recursive data structures because they can be defined recursively
- Binary trees
    - 1 predecessor, at most 2 successors
- Binary tree level calculations vary depending on the textbook used - include what you are counting in quizzes and tests
    - Declarations and definitions are needed
- Levels = 1 + the level of its parent
- Height = number of nodes in the longest path the root to a leaf
    - You can also count the branches - just make sure you say what you are counting on a quiz / test
- Tree expressions: Node(i,l,r)

## 3/22/23 + 3/24/23

- Huffman can be used for file compression
- BST runtimes: [https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/](https://www.geeksforgeeks.org/complexity-different-operations-binary-tree-binary-search-tree-avl-tree/)
- Reviewed Tree Walks: preorder, inorder, postorder
    - [https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/](https://www.geeksforgeeks.org/tree-traversals-inorder-preorder-and-postorder/)
- Search operation:
    - Compare values (greater than = go right, less than - go left)

## 3/27/23 + 3/29/23

- Coding BinaryTree.java
    - Size = number of nodes
- Added insert code into BSTree.java - 3/29/23

https://github.com/humnasul/cs284notes/tree/main/class%20code/data%20structures%20in%20java/BinaryTrees

## 4/3/23

- Full, perfect, complete binary trees
    - Different qualities of trees
- Heap and priority queues
    - Heap
        - Complete binary tree
        - The value of the root is the smallest item in the tree
        - Every subtree is a heap
        - Relationship between parent and children
            - Larger number is lower
        - Insert into heap: most runtime is O(log n) when n is the number of nodes (height)
        - Because a heap is a complete binary tree, it can be implemented efficiently using an array rather than a linked data structure

## 4/5/23

- Merging heaps
    - O (n log n)
- Priority queues
    - Similar to heap implementation, takes advantage of how heaps work
- Do not use arrays.sort() and other algorithms when coding on assignments!!

## 4/10/23

- Min-heap: upperlevel nodes are smaller than lowerlevel nodes
    - Complete binary tree
    - Levels start from the left
- Selection sort: after first iteration of outer loop, you are on the minimum element
    - Once you find the minimum, you swap with the first element
    - You repeatedly find minimums and swap
    - Number of iterations = n - 1
    - Comparisons happen big O n^2 - happens several times per run
    - Swapping is linear O n - happen for each provided value, so it is linear
    - General runtime for sort: On^2
- Bubble sort:
    - Smaller values bubble up, larger values go down
    - Compares adjacent pairs and keeps moving max values to the end (can also move min values to the end in reverse)
    - After first iteration of outer loop, max value has been bubbled to the bottom
        - The next iteration does not count the last element - the last element has been sorted 🙂
    - Continues the process of iteration and sorting last elements until top is reached
    - General runtime: On^2
    - Number of comparisons runtime: On^2
    - Swaps: On^2
        - Every comparison does swap if needed, swap is n^2
- Bubble sort will do better if there are performance tests done and the list is almost sorted, otherwise it is selection sort

## 4/12/23 + 4/14/23

- Insertion sort
    - Does not find max and min, but compares adjacent elements
    - Runtime complexity: On^2
        - In best case, number of comparisons is On
- Merge sort
    - Iterating through and comparing elements
    - Steps
        - Split the array into two halves
        - Sort the left half
        - Sort the right half
        - Merge the split array
    - Runtime: O(n logn)
- Tests will not have examples with duplicate elements

    - Merge sort quiz will ask for order of operations - splitting FULLY and then merging

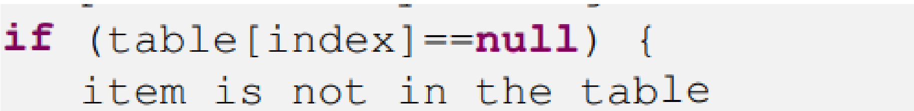
## 4/17/23

- Final exam
    - Part 1 - exam on May 4 on paper
    - Part 2 - May 11 / whatever it says online, a lot less points than part 1
        - Online, will be on canvas, can be taken anywhere
        - true/false, fill in the blank, NO CODING, etc.
    - Will not have shell sort on final
        - Merge sort is her favorite so prepare for that
- Shell sort
    - Based on gap sizes
    - [https://www.w3resource.com/ODSA/AV/Sorting/shellsortAV.html](https://www.w3resource.com/ODSA/AV/Sorting/shellsortAV.html)
    - Runtime: O(n^3/2)
- Heapsort
    - Same time complexity as mergesort (Onlogn)
        - Difference between them is that heapsort does not make separate arrays - saves memory space
            - Bigger data sets are better with heapsort because less memory is used
    - Once you finish the first iteration of the while loop, the biggest element is found and confirmed
        - Once you get the second iteration, you have found the largest 2 elements in the array
        - Once you get the third iteration, you have found the largest 3 elements in the array

## 4/19/23

- Set interface - under collection umbrella
    - Set objects…
        - Not indexed
        - Do not reveal the order of insertion of items
        - Enable efficient search and retrieval of information
        - Allow for removal of elements without moving other elements around
    - Includes different methods in interface
        - addAll - adding a collection of items, allows for union of sets
        - Iterator iterates through all elements in set
        - removeAll - removes values at intersection between sets
        - retainAll - keeps only values in the intersection
        - Does NOT have a get method because elements cannot be accessed by index - you can use an iterator or for to get the element you're looking for
- Maps and map interface
    - Set of ordered pairs whose elements are known as they key and the value
    - Includes methods in interface
        - Basics: get, put, isEmpty, remove, size
        - Others:
            - Clear
            - containsKey
            - containsValue
            - Set<Map.Entry<K,V> entrySet ()
            - Set <K> keySet()

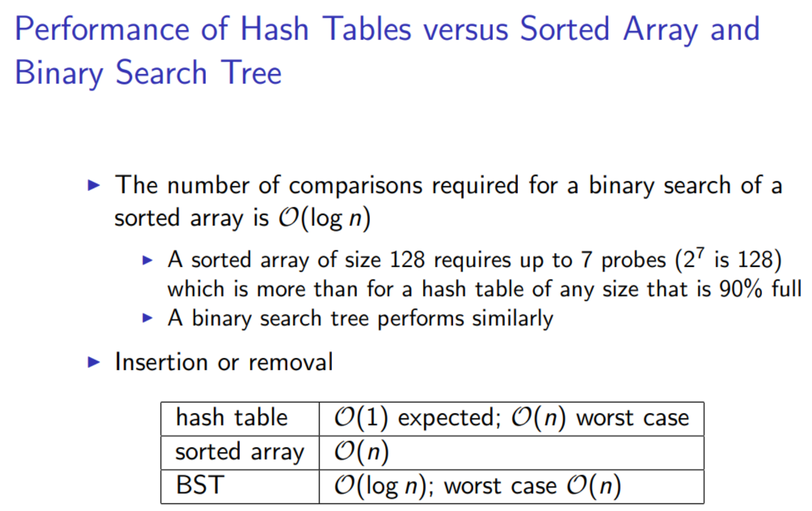
## 4/24/23

- Hash table open addressing
    - Calculate index - if index has an element already, increment the index calculated and keep moving upwards
    - If you do not have the condition that if the index is null, the item does not exist, then you need to search the whole table for the item
        - 

          ```
          if (table[index]==null) {
             item is not in the table
          ```
        - Much less efficient without that condition → having that condition makes the program more efficient and is the purpose of using a hash table
    - When deleting an item
        - You need to ensure that the deleted item is not in the table before you're inserting a new item into the deleted item's spot
- EX of hash table: insert "John", "Jill", "Ken", "Jane" into hash table in notebook
    - Probing happens when there is a collision

## 4/26/23 + 4/28/23

- Quadratic probing issue: not all table elements may be examined when looking for an insertion index - an element may not be able to be placed, even if the table is not full 🙁
- Chaining references a linked list within a hash - linked list is called bucket and this process is known as bucket hashing
    - Removing an item - you can use linked list strategies and move elements together
- Performance of hash tables
    - Load factor has the greatest impact on performance and runtime
        - Lower load factor = better performance
    - If there are no collisions, performance for search and retrieval is O(1) REGARDLESS of table size
- Number of comparisons: $c = ½ (1+ 1/(1-L))$
    - L is load factor (how full tree currently is)
- If an item is in the table, on average we must examine the table element corresponding to the item's hash code and then half the items in each list
    - Comparisons calculation: $c = 1 + L/2$
- Include these formulas on the exam formula sheet 🙂
    - She will ask you to calculate



## Performance of Hash Tables versus Sorted Array and Binary Search Tree

- The number of comparisons required for a binary search of a sorted array is $\mathcal{O}(\log n)$
  - A sorted array of size 128 requires up to 7 probes ($2^7$ is 128) which is more than for a hash table of any size that is 90% full
  - A binary search tree performs similarly
- Insertion or removal

| | |
|---|---|
| hash table | $\mathcal{O}(1)$ expected; $\mathcal{O}(n)$ worst case |
| sorted array | $\mathcal{O}(n)$ |
| BST | $\mathcal{O}(\log n)$; worst case $\mathcal{O}(n)$ |

-

- AVL Tree: the balance of every node is in the interval [-1, +1]
  - Balance = right - left
  - If you add something that changes the height of the tree, the balance can change