Strings

Make sure you know:

1. the difference between a null String and an empty String
2. how the concatenation operator works (+)
3. why == doesn't work for comparing the equality of Strings

Important String Methods

- int compareTo(String str)
- boolean equals(String str)
- int length()
- String substring(int startIndex)
- String substring(int startIndex, int endIndex)
- int indexOf(String str)

Integer and Double (Wrapper Classes)

Make sure you know:

1. when you have to use these (ArrayLists)
2. the terms autoboxing and auto-unboxing
3. how these interact with primitve ints and doubles

Important Wrapper Class Methods:

- int compareTo(Integer other)
- boolean equals(Object obj)
- String toString()

Math Class

Absolute value:	Math.abs(x)
Exponents:	Math.pow(base, exp)
Square Root:	Math.sqrt(x)
Random Number:	Math.random()

Note: make sure you can generate random integers and rational numbers in any arbitrary range. For example, to generate a random integer from 1 to 6, you would do (int) (Math.random()*6) + 1

Arrays and ArrayLists

Make sure you can:

1. instantiate Arrays, 2D Arrays, and ArrayLists
2. use { } to initialize an array from an initializer list
3. use a loop to traverse an Array or ArrayList without going out of bounds
4. use a nested loop to traverse a 2D Array without going out of bounds
5. use indexes to retrieve elements from Arrays, 2D Arrays, and ArrayLists
6. safely remove elements from an ArrayList inside a loop without skipping any elements
7. understand how arrays and lists are sent as parameters to methods.
8. understand how 2d arrays are really arrays inside of arrays

Important List methods:

- `boolean add(E obj)`
- `int size()`
- `E get(int index)`
- `E set(int index, E element)`
- `void add(int index, E element)`
- `E remove(int index)`

What is E in all of these examples? If you made an ArrayList like so:

```
ArrayList<Integer> nums = new ArrayList<Integer>();
```

Then E would be Integer. It's whatever type of Element that the ArrayList was created to hold.

Sorting Algorithms

Selection Sort
"Select" the smallest element from the unsorted portion of the data and put it at the end of the sorted portion of the data.

- For sorting n elements it takes n-1 selections.
- After the kth pass, the first k items are in their final positions.
- Equal best/worst cases.
- Classified as n^2 efficiency for large data sets

Insertion Sort
Insert" the current element into the space of already sorted numbers

- For sorting n elements, the array is sorted after n-1 passes.

- After the kth pass, the first k items are sorted, but not necessarily in their final positions
- Best case: If the data is already sorted or nearly sorted. (very efficient)
- Worst case: If the data is sorted in reverse order.
- Classified as n^2 efficiency for large data sets

Mergesort

Break the data into 2 halves. Mergesort the left, Mergesort the right, then merge the two halves together into a sorted array.

- The merge step makes use of temporary arrays, which for a large data set could use up a lot of memory.
- equal best/worst cases.
- Classified as n(log n) efficiency for large data sets (very efficient)
- For small data sets, effectively no more efficient than Selection or Insertion Sort.

Searching Algorithms

Sequential Search

- Review how this algorithm works
- Does not require the searchable area to be sorted
- Relatively inefficient

Binary Search

- Review how this algorithms works
- Requires the searchable area to be sorted
- Extremely efficient