Unit 7 → ArrayLists

ArrayList

- A class
- A mutable list of object references
- Syntax
    - `import java.util.ArrayList;`
    - `ArrayList <E> list = new ArrayList<E>();`
        - Where `<E>` can only be an object
        - Can also be declared as…
            - `ArrayList list = new ArrayList();`
    - `E` → represents the object being used
        - Excludes `int` and `double`
        - Use `Integer` and `Double`
- Array vs ArrayList
    - ArrayLists *are* changeable (mutable)
        - Can use `Integer`, `Double`, and `String`
    - Arrays are *not* changeable
        - Can use `int`, `double`, and `String`

ArrayList Methods

- Adding a number before the desire input places it in the numbered index
    - When added at a specific position, it shifts everything in the ArrayList right once
- `.size()` is used to find length of ArrayList
- Removing a number shifts everything in the ArrayList left once

Traversing ArrayLists

- `IndexOutOfBounds` → results from accessing index outside of range
- Can be accomplished with `while`, `for`, or `for each` loop
- Use `i--;` so it doesn't skip over an indice
- DO NOT USE AN ENHANCED FOR LOOP FOR ADDING AND REMOVING
    - Results in `ConcurrentModificationException`

Developing Algorithms Using ArrayLists

- `==` to check if lowercase or uppercase; `.equals()` to check for same exact
- `MyProgram.java: Line x: Out of memory! Please try again`
    - An error caused by an infinite loop

Linear Search

- We can use transversals to search for individual elements in an Array
- If it does not contain the target element, return -1
- Steps
    - Traverse through the ArrayList
    - Assignment the ith element to a variable
    - Set an If statement to see if variable is equal to target
        - If they are equal, return the index
    - Outside the for loop, return -1
- Linear (sequential) search checks each element until the target is reached
    - Could be used on `Integer`s, `String`s, or `Array`s
    - The longer the data size, the longer the process takes
- Don't use for each loop because they complete the loop regardless
    - Won't return the index

Sorting

- Organizing data can make it easier to search through

Selection Sort

- Sorts an array by repeatedly finding the minimum value and moving it to the front of the array
    - Implementation
        - For loop starting at 1 after the index
        - Traverse each `index` to the second to last element
        - Find minimum
        - Swap the `index` and `minIndex`

Insertion Sort

- Sorts an array by sorting each element compared to the elements already sorted to the left
- Implementation
    - Traverse each element starting at index 1
    - Shift sorted elements to place current elements
    - Start with a `for` loop where `index` (i) = `1`
    - CHECK EACH INDEX UNTIL CHOSEN VALUE HAS A LOWER VALUE THAN THE CURRENT INDEX
    - Create a `while` loop
        - `while (sortedIndex > -1 && array[sortedIndex] > array[currentIndex])`
        - `{`
            - `array{sortedIndex+1]=array[sortedIndex];`
            - `sortedIndex--;`
        - `}`
        - `array[sortedIndex+1]=currentIndexValue;`
        - `sortedIndex` should be assigned to a value of `currentIndex-1`
        - `currentIndexValue` should be assigned to `array[index]`
    - **LAYOUT**

```
for (int i = 1; i < arr.length; i++)
{
      int curNumber = arr[i];
      int curIndex = i-1;
      while ( curIndex >= 0 && arr[curIndex] > curNumber)
      {
            arr[curIndex+1] = arr[curIndex];
            curIndex--;
      }
      arr[curIndex + 1] = curNumber;
}
```

Insertion vs. Selection

- Depends on how sorted the list is
- If somewhat sorted → *selection*
- If not at all → *insertion*