CMPT 225: Data Structures & Programming – Unit 04 – Arrays

Dr. Jack Thomas
Simon Fraser University

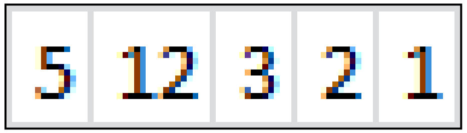
Spring 2021

Today’s Topics

- Array as ADT
- The Array in Java
- Insertion Sort
- Useful Methods
- Two-Dimensional Arrays
- Strings, chars, and Cryptography

Re-Introducing Arrays

- A very basic **Abstract Data Type**.
- A means for storing a numbered group of related data built into just about every programming language.
- The individual objects are called **elements** of the array, and should be of the same type.



Address Index: 0, 1, 2, 3, 4

- Often a building block of other, more sophisticated data structures.

Arrays in Java (The CDT)

- Declaration:

```java
int[] arrayName = {1, 3, 2, 4, 5};
//or
int[] anotherArray = new int[5];
```

- Accessing:

```java
int whatIsInFirst = arrayName[0];
int whatIsInLast = arrayName[4];
```

- Modifying:

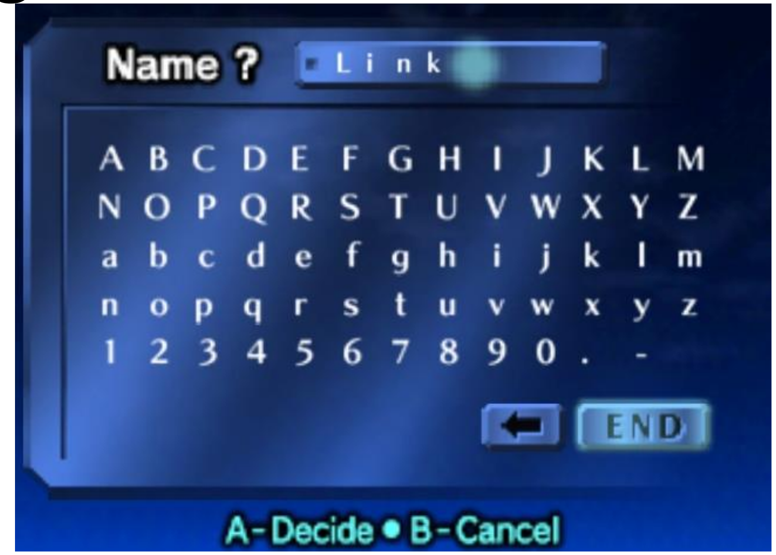
```java
arrayName[0] = 99;
arrayName[arrayName[1]] = arrayName[0];
```

- Check length:

```java
int length = arrayName.length;
```

Array Length is Fixed at Declaration

- You have to **specify how many elements** an array can store when it's declared, which **cannot be changed** afterward.
- This means every array is either full or reserving some unused memory.

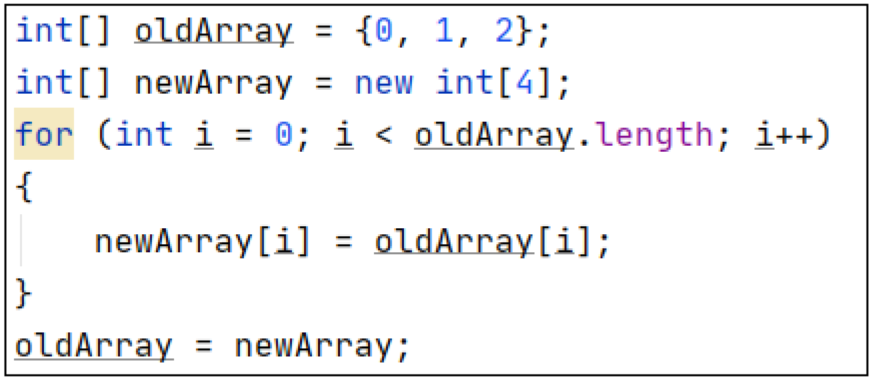


Name ?
Link
A B C D E F G H I J K L M
N O P Q R S T U V W X Y Z
a b c d e f g h i j k l m
n o p q r s t u v w x y z
1 2 3 4 5 6 7 8 9 0 . -
END
A-Decide ● B-Cancel

Image credit: https://zelda.fandom.com/wiki/Name_Registration

Expanding or Shrinking an Array

- To **change an array's size**, you need to declare a new array of the desired size, copy over the data, and then reset the array's name to point to the new array.



```
int[] oldArray = {0, 1, 2};
int[] newArray = new int[4];
for (int i = 0; i < oldArray.length; i++)
{
    newArray[i] = oldArray[i];
}
oldArray = newArray;
```

- This is why arrays are often just a **part of more complicated data structures**, since they need supporting methods to do things like change their size or keep track of how many entries are in use.

Arrays are Objects

- Unlike the **primitive data types** (int, boolean, double, short, etc), arrays are objects.
- The array's name is a **reference** to this object.
- If you try to copy an array with "int[] newArray = oldArray;", you'll just end up with two names pointing at the same place in memory – changing one changes both.
- To make a completely separate copy, use the .clone() function of the array, "int[] newArray = oldArray.clone();".
- You can also store objects of **different classes** in the same array **if they share a superclass**.

Sorting an Array with Insertion Sort

- A whole lot of programming problems are essentially a question of **sorting** – putting a set of data in a desired order, usually while trying to be efficient about time and storage.
- **Insertion Sort** is a simple approach to sorting, one implementation of which can work for an array of comparable values – like, say, a set of numbers.

The Insertion Sort Algorithm

```
Algorithm InsertionSort($A$):
    Input: An array $A$ of $n$ comparable elements
    Output: The array $A$ with elements rearranged in
    nondescreasing order
    for $i$ <- 1 to $n - 1$ do
        Insert $A[i]$ at its proper location in $A[0]$,
        $A[1]$, … , $A[i - 1]$.
```

What Does That Mean Though?

- Basically, you move through the array one number at a time, from start to finish.
- For each number, you start going backward through the numbers you’ve sorted so far, comparing them one at a time, until you find the spot where your new number belongs.
- Since you know the numbers you’ve already sorted are in order, you only need to keep looking back until you find the first one that’s smaller than the one you’re currently trying to sort, and then stop – that’s where your number belongs!

Check This Visualization Out

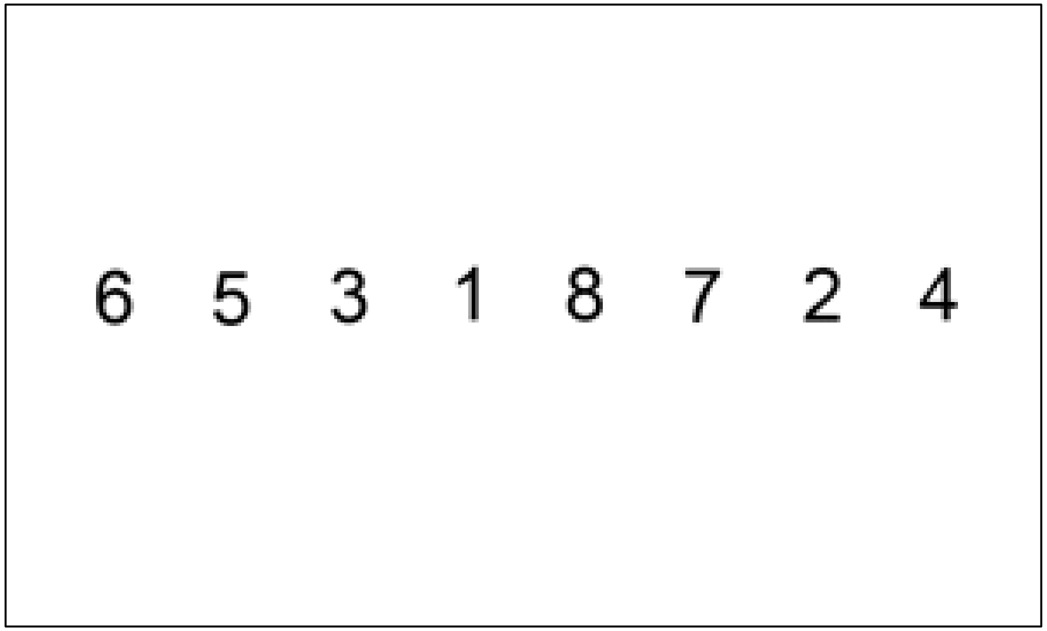
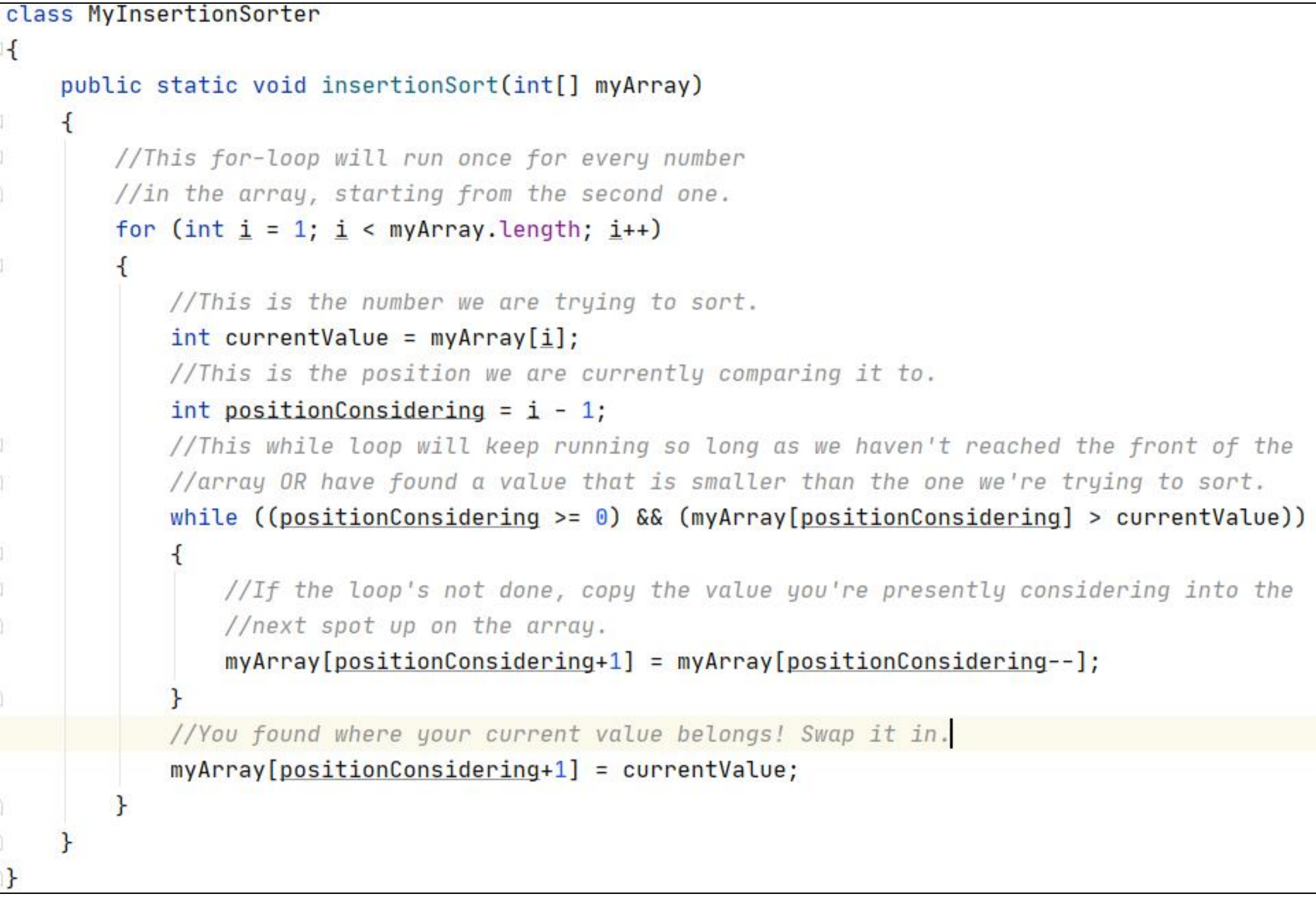


Image credit: https://upload.wikimedia.org/wikipedia/commons/0/0f/Insertion-sort-example-300px.gif


```java
class MyInsertionSorter
{
    public static void insertionSort(int[] myArray)
    {
        //This for-loop will run once for every number
        //in the array, starting from the second one.
        for (int i = 1; i < myArray.length; i++)
        {
            //This is the number we are trying to sort.
            int currentValue = myArray[i];
            //This is the position we are currently comparing it to.
            int positionConsidering = i - 1;
            //This while loop will keep running so long as we haven't reached the front of the
            //array OR have found a value that is smaller than the one we're trying to sort.
            while ((positionConsidering >= 0) && (myArray[positionConsidering] > currentValue))
            {
                //If the loop's not done, copy the value you're presently considering into the
                //next spot up on the array.
                myArray[positionConsidering+1] = myArray[positionConsidering--];
            }
            //You found where your current value belongs! Swap it in.
            myArray[positionConsidering+1] = currentValue;
        }
    }
}
```

```java
int[] someNumbers = {3, 1, 2};
MyInsertionSorter.insertionSort(someNumbers);
System.out.println(someNumbers[0] + " " + someNumbers[1] + " "
    + someNumbers[2]);
```

```
1 2 3
```

Useful Methods in Java for Arrays

- **.equals(A, B):** Returns true if two arrays are exactly equal, with all the same elements.
- **.fill(A, x):** Fills array A entirely with elements of variable x. Good for setting a default value.
- **.copyOf(A, n):** Creates an array that's a copy of the first n values of array A. If n > A.length, the remaining slots are filled with default values (0s for numbers, nulls for objects, etc).

Useful Methods Continued

- **.copyOfRange(A, s, t):** Like copyOf, except s and t define the start and end indexes for where to copy from in A.
- **.toString(A):** Returns a string of comma-separated .toString()’s for all the array’s elements. Good for printing a whole array.
- **.sort(A):** Applies the quick-sort algorithm.

Two-Dimensional Arrays

- What if we want an array... OF arrays?

Image credit: https://www.youtube.com/watch?v=G2jUhnCU9iA&ab_channel=brytons

Thinking in Two Dimensions

- Whereas a regular array is like a **row**, a two-dimensional array is like a **grid**.
- For example, “int[][] example = new int[3][5]” would give “example” three arrays of five integers each, essentially a 3x5 grid.
- Also known as a **Matrix**.

Image credit: https://www.warnerbros.com/movies/matrix

No That’s Too Many Movie Parodies

- Accessing a specific number now requires **specifying both arrays**, like “example[0][2]”, which would give you the value in the first array’s third position.
- You can also interact with **any individual array**, like getting the second array’s length with “example[1].length”.
- Remember that **each individual array is still a unique object**, so for example, the .equals(A, B) function would not consider two otherwise-identical arrays to be equal here because of their different memory addresses. For this, java.util.Arrays includes a .deepEquals(A, B) function.

Strings, char, and Caesar

- In Java, Strings are an object, not a primitive data type – essentially an array of characters.
- We can make an array of characters into a String; with “String word = new String(char array)”, and the reverse with “char[] array = word.toCharArray();”
- This is all you’d need to know to implement a Caesar Cypher, encrypting messages by swapping every letter with the one three letters along in the alphabet. Give it a try to test your knowledge!

Recap – End of the Array of Slides

- Reintroduced the Array as an **Abstract Data Type**, and given the **Concrete Data Type** description for Java.
- Introduced and implemented our first algorithm, **Insertion Sort**.
- Saw how **two-dimensional arrays** form a grid of data.
- Discussed the implications of arrays as **objects**, such as the difference between a String and an array of chars.