AP Computer Science Name _____ Per ____
Chapter 6 Notes
Arrays, `ArrayList`, Searching, Sorthing and 2D Arrays

6.1 Arrays

Array Elements

An **array** is an object that is used to store a list of values. It is made out of a contiguous block of memory that is divided into a number of *cells*. Each cell holds a value, and all the values are of the same type.

```
int[ ] data = new int[10];   // creates the array "data"
```

	data
0	23
1	38
2	14
3	-3
4	0
5	14
6	9
7	103
8	0
9	-56

The name of this array is `data`. Each cell in the `data` array can hold an `int`.

The cells are indexed 0 through 9. Each cell can be accessed by using its index.

`data[0]` contains the value 23. `data[5]` contains the value 14.

Remember:

- The cells are numbered sequentially starting at 0.
- If there are N cells in an array, the indexes will be 0 through N-1.

Sometimes the index is called a *subscript*. The expression `data[5]` is usually pronounced "data-sub-five" as if it were an expression from mathematics: $data_5$.

The value stored in a cell of an array is sometimes called an *element* of the array. An array has a fixed number of cells and cannot grow or shrink.

The array `data` holds data of type `int`. Every cell contains an `int`. A cell of this array can be used anywhere a variable of type `int` can be used. For example,

`data[3] = 99 ;` works just like an assignment to an `int` variable.

- Change the *elements* of the `data` array to reflect the following statements:

```
data[9] = data[2] + data[6];

int x = data[3]++ ;

data[0] = (x + data[2]) / 4 ;

int i = 2;

data[i] = data[i] + 1; // variables can be used for indexes

data[4] = data[1] / data[6];

data[0] = data[6] + 8;
```

	data
0	23
1	38
2	14
3	-3
4	0
5	14
6	9
7	103
8	0
9	-56

Arrays are Objects

All arrays are objects and have an *instance variable* `length` which stores the number of elements in the array.

Common error: `length` is an instance variable and is sometimes confused with the `length()` method of the `String` class, which returns the number of characters in a string.

Array objects *do not* have any methods. Array declarations look like this:

```
type[] arrayName;
```

This tells the compiler that *arrayName* is the name of an array containing *type*. However, the actual array is not constructed by this declaration. This declaration merely declares a reference variable *arrayName* which, sometime in the future, is expected to refer to an array object.

Often an array is declared and constructed in one statement, like this:

```
type[] arrayName = new type[ length ];
```

This statement does two things: (1) It tells the compiler that *arrayName* will refer to an array of *type*. (2) It constructs an array object containing *length* number of cells.

An array is an *object*. The array constructor uses different syntax than other object constructors:

```
new type[ length ]              // [ ] are used instead of ( )
```

This names the type of data in each cell and the number of cells. Once an array has been constructed, the number of cells it has *cannot* change. Here is an example:

```
int[] data = new int[10];
```

Are the index references to the right illegal or legal?

`data[ -1 ]`	
`data[ 10 ]`	
`data[ 1.5 ]`	
`data[ 0 ]`	
`data[ 9 ]`	

As a Java program is running, each time an array index is used it is checked to be sure that it is OK. This is called *bounds checking*. If the program refers to a cell that does not exist, an `ArrayIndexOutOfBoundsException` exception is thrown, and the program is terminated.

Arrays as Parameters

An entire array can be passed as a parameter to a method. Because an array is an *object*, when the entire array is passed as a parameter, a *copy* of the reference/address to the original array is passed. **Be careful**: a method that gets an array as a parameter can permanently change an element of the array.

```
int[] data = new int[5];
.
.
obj.processGrades(data);
.
.
```

```
public void processGrades (int[] d)
{
   d[3] = 101;
}
```

	data
0	23
1	38
2	14
3	-3
4	0

Array Initialization

Each cell of a numeric array is initialized to **0.** Each cell of an array of object references is initialized to `null`.

```
public class Example1
{
  public static void main ( String[] args )
  {
    int[] stuff = new int[4];

    stuff[0] = 23;
    stuff[1] = 38;
    stuff[2] = 7*2;

    System.out.println("stuff[0] has " + stuff[0] );
    System.out.println("stuff[1] has " + stuff[1] );
    System.out.println("stuff[2] has " + stuff[2] );
    System.out.println("stuff[3] has " + stuff[3] );
  }
}
```

Output:

Initializer Lists

You can declare, construct, and initialize the array all in one statement:

```
int[] data = {23, 38, 14, -3, 0, 14, 9, 103, 0, -56 };
```

This declares an array of `int` which is named `data`. Then it constructs an `int` array of 10 cells (indexed 0-9). Finally it puts/assigns the designated values into the cells. The first value in the initializer list corresponds to index 0, the second value corresponds to index 1, and so on. (So in this example, `data[0]` gets/is assigned the value 23.)

You do not have to specify how many cells the array has. The compiler will count the values in the initializer list and make that many cells. Once an array has been constructed, the number of cells does not change. Initializer lists are usually used only for *small arrays*.

- Write a declaration for an array of `double` named `dvals` that is initialized to contain 0.0, 0.5, 1.5, 2.0, and 2.5.

- What value does the instance variable `dvals.length` contain? _____

Copying Contents Of Arrays

A program can have any number of arrays in it. Often values are copied back and forth between the various arrays. Here is an example program that uses two arrays.

Complete the program so that the values in `valA` are copied into the corresponding cells of `valB`.

```
public class CopyArray
{
  public static void main ( String[] args )
  {
    int[] valA = { 12, 23, 45, 56 };

    int[] valB = new int[4];









   }
}
```

Super Bug Alert: The following statement does not do the same thing:

```
valB = valA ;
```

Remember that arrays are objects. The statement above will merely copy the object reference in `valA` into the object reference variable `valB`, resulting in two ways to access the single array object: The object that `valB` previously referenced is now lost (it has become garbage.)

6.2 Looping Through Arrays

`for` Loop

- Rewrite the copying of `valA` into `valB` using a `for` loop from the example above:

- Reversing the elements of an array involves swapping the corresponding elements of the array: the first with the last, the second with the next to the last, and so on, all the way to the middle of the array. Given an array `a` and two other `int` variables , `k` and `temp` , write a loop that reverses the elements of the array. Do not use any other variables besides `a` , `k` , and `temp` .

Enhanced `for` Loop

To loop through every cell in an array, we can use an enhanced `for` loop, also called a *for each* loop because it processes "each" cell in the array in turn. Do not use an enhanced for loop when you need to keep track of the index of a cell.

```
int[] grades = {89,  97,  56,  76,  77,  89,  20};


for (int x : grades)
   System.out.print (x + " ");
```

- Write a method, `hasNegative` which is passed an array of `int`s and returns `true` if the array contains a negative number, `false` otherwise. Use an *enhanced for* loop.

6.3 Arrays of Objects

Any type of data can be held in the elements of an array:

- Primitive types, such as `int`, `double` or `boolean`.
- Object references. (*Address* in memory)

All the elements of an array must be of the same type. So far in these notes, the elements have been primitive types. However, an array can be made with elements of any data type, including object references.

What does this statement do?

```
String str;
```

In the following example, a reference variable is declared, and then an object is constructed. A reference to the object is then put in the reference variable:

```
String str;              // declare a reference variable
str = "Hello World" ;    // construct the object and
                         // save its reference
```

- What could you do if your program needed to keep track of 1000 different strings?

You can declare an array of `String` references:

```
String[] strArray;                      // 1.
```

This declares a variable `strArray` which in the future may refer to an array object. Each cell of the array may be a reference to a `String` object (but so far there is no array).

To create an array of 8 `String` references do this:

```
strArray = new String[8] ;              // 2.
```

Now `strArray` refers to an array object. The array object has 8 cells. However none of the cells refer to an object (yet). The cells of an array of object references are automatically initialized to `null`, the special value that means "no object".

Now, store the following `String` references in cells 0-3 of the array:

```
strArray[0] = "Hello" ;                 // 3.
strArray[1] = "APCS" ;
strArray[2] = "Hello" ;
strArray[3] = strArray[0] + strArray[1];
```

- What can you say about `strArray[0]` and `strArray[2]`? _____

- What would the following statement output?

```
System.out.println(strArray[0]);
```

- Write the *boolean expression* to determine if the `String` objects stored in cells 0 and 1 are equal:

Recall these facts about all arrays:

1. Each element of an array is of the same type.
2. The length of an array (the number of cells) is *fixed* when the array is created.

Frequent Bug: It is easy to confuse the length of an array with the number of cells that contain a reference. In the example, only four cells contain references to `Strings`, but `strArray.length` will be 8.

- Write the code to print out the contents of `strArray` using an *enhanced for* loop. What will the output be when the loop is executed?

- Re-write your code to print out only those cells which contain an address to a `String` object.

- Use an *enhanced for* loop to loop through an array of `String` objects named `words` and sum and print the number of words which begin with the letter S (upper or lowercase):

6.4 `ArrayList` Class

The `ArrayList` class is part of the `java.util` package. It works like an array in that it stores a list of values and references them by index. All elements in an `ArrayList` must be *objects*. We can specify the element type for each `ArrayList` object we create by writing the class name in angle brackets < >. Use wrapper objects for primitive types (`Integer` for `int`, `Double` for `double`, etc.).

`ArrayList` Syntax

```
import java.util.ArrayList;

ArrayList<objectType> arrayListName = new ArrayList<objectType>();
```

- Create an `ArrayList` of `String` objects named `names`.

The `ArrayList` class includes methods to allow addition, insertion, removal and searching for a specific element.

`ArrayList()`
Constructor: creates an empty list.

`int size()`
Returns the number of elements currently in the list.

`boolean add (Object obj)`
Adds the object to the end of this list. Always returns true.

`void add (int index, Object obj)`
Inserts object at the specified index in the list. If the insertion is not at the end of the list, shifts the element currently at that position and all elements following it one unit to the right (or down) (i.e., adds 1 to their indexes). Adjusts the size of the list.

`Object get (int index)`
Returns the object at the index.

`Object set (int index, Object obj)`
Replaces the element at position `index` with object `obj`. Returns the object formerly at the specified position/index.

`Object remove (int index)`
Removes the object at the index from this list and returns it.

- Given that an `ArrayList` of `Integer` objects named `a` with 30 elements has been declared , assign 5 to its last element.

Array

- An array remains a fixed size throughout its existence.
- The number of cells is accessed in the `length` instance variable.
  `arrayName.length`
- The number of elements in the array has to be kept track of by the programmer. The number of elements will be different than the length of the array if the array is not full.
- Last position in array has index of
  `arrayName.length - 1`
- Use when you know the exact number of elements in advance.
- Use brackets [ ] to access a cell.
  `arrayName[ 0 ]`

`ArrayList`

- An `ArrayList` object grows and shrinks as needed.
- The number of cells is also the number of elements in the list and is returned from method:
  `arrayListName.size();`
- Last position in array has index of
  `arrayListName.size() - 1`
- Use the `get()` method to access a cell **NEVER** brackets `[]`.
  `arrayListName.get(0)`

- Given an `ArrayList` of `Integer` objects, named `scores`, write the code to find the minimum and maximum values, then delete all occurrences of the minimum and maximum values. After the values have been deleted, print a message with the average of values left in `ArrayList scores`.

6.5 Searching Arrays

A common problem with arrays is searching for a particular element in the array.

Linear or Sequential Search

A *linear* or *sequential search* loops through array, looking at each element to see if it is the one we want.

```
String[] names = {"Sue", "John", "Henry", "Joan"};
String myName = new String("Henry");
boolean found = false;
```

- Use an enhanced `for` loop to write a linear search to determine if `myName` is in array `names`:

Question: How do you decide whether to use a regular `for` loop or an enhanced `for` loop to process the contents of an array?

- Complete the following method `linearSearch`, which accepts an array of `int`s and a target number to look for as parameters. The method will return the index of the target number if found, or -1 if the target is not found.

```
public static int linearSearch ( _____________________, ____________________ )
{
   for (int i = 0; ____________________________________________; i++)

      if ( ______________________________ == ___________________________________ )

         return i;

   return -1;
}
```

Caution: When looping through an array, be careful of *an index out of bounds error* when checking consecutive cells.

```
int[] nums = {2, 4, 3, 5, 5};
```

- You are given an `int` variable `i`, a sorted `int` array `areaCodes` that has been declared and initialized, and an `boolean` variable `duplicates`. Write some code that assigns `true` to `duplicates` if there are two adjacent elements in the array that have the same value, and that assigns `false` to duplicates otherwise. Use only `i`, `areaCodes`, and `duplicates`.

Binary Search

When we know how the data is sorted, we can use a binary search for faster processing. A binary search is more efficient than a linear search but only works if the data is stored in order in the array.

- I'm thinking of a number between 1 – 10. What is the most efficient way to guess the number I am thinking of? What steps did you use?

Complete the following method `binarySearch` which accepts a SORTED array of integers and a target number. The method will return the index if the target number is found, otherwise the method will return -1.

```
public static int binarySearch ( int[] nums, ____________________ )
{
   int low = 0;
   int high = ________________________________________;

   int middle = ______________________________________________;

   while (nums[ _____________ ] != target && low < = _______________ )
   {
      if ( ____________________ < nums[ middle ] )

         high = ___________________________________;

      else
         low = ____________________________________;

      middle = ( low + high ) / 2;
   }
   if ( ____________________________ == ____________________________ )

         return middle;

   return -1;
}
```

2	4	9	10	13	17	20	27

6.6 Sorting Arrays

Sorting is the process of arranging an array of items in order. There are many different sorting algorithms and you are expected to know two of them for this chapter: *Selection Sort* and *Insertion Sort*. You will learn about the *Merge Sort* when we get to Chapter 8: Recursion.

Selection Sort

Selection sort works by putting each value in its final position, one at a time. For each position in the array, the algorithm selects the value that should go in that position and puts it there. Know the algorithm on page 346 of textbook.

In the array below, which number should go in position [0]? _____ In position [1]? _____

12	4	16	15	22	2	23	18

Question: How many passes did it take to sort? _____

Steps for the selection sort:

1. First, scan array to find the smallest value.
2. Swap that value with the value in the first position `[0]` of the array.
3. Starting at the second position `[1]`, scan the array to find the smallest value and swap with the value in the second position `[1]`.
4. Keep doing this until we get to the last position in the array. The array is sorted.

The selection sort uses two `for` loops. The outer loop controls where the next smallest value will be stored. The inner loop finds the smallest value in the rest of array by scanning all positions greater than or equal to the index specified by the outer loop. The number of passes through the array is the array length – 1.

- Demonstrate how the following array is sorted using the selection sort. Show the array after each pass of the outer loop. How many passes through the array will there be? _____

3	2	45	34	35	20	15	14	3

Insertion Sort

Insertion sort works by inserting each value into a previously sorted subset of the array. This strategy is similar to sorting a hand of cards as it is being dealt. Each time a new card is dealt you pick it up and insert it into is proper position. Know the algorithm on page 346 of the textbook.

12	4	16	15	22	2	23	18

The insertion sort also uses two `for` loops. The outer loop controls the index in the array of the next value to be inserted. The inner loop compares the current insert value with values stored at lower indexes (the sorted subset). The number of passes through the array is the array length – 1.

- Demonstrate how the following array is sorted using the insertion sort. Show the array after each pass of the outer loop. How many passes through the array will there be? _____

3	2	45	34	35	20	15	14	3

6.7 Two-Dimensional Arrays

A two-dimensional array has values in two "dimensions", which are like the rows and columns of a table. Syntax to create a 2D array of integer values is:

```
int[][] table = new int[5][10];
```

creates a 2D array named `table` containing 5 rows and 10 columns. `table.length` contains the number of rows in array `table`. `table[0].length` contains the number of columns in the row.

0	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	19
20	20	22	23	24	25	26	27	28	29
30	31	32	33	34	35	36	37	38	39
40	41	42	43	44	45	46	47	48	49

`table.length` _____

`table[1].length` _____

`table[3][10]` _____ `table[0][5]` _____ `table[5][3]` _____ `table[2][8]` _____

- Write the code to loop through the `table` array above and increment each element by 2.