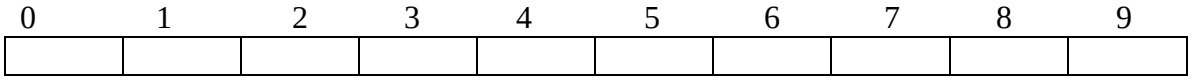
Arrays

What are they? An array is a data structure that holds a number of related variables. Thus, an array has a size which is the number of variables it can store. All of these variables must be of the same type. (In essence declaring an array allows you to use many variables of the same type without explicitly declaring all of them.) You can picture an array like below:



Each cell of the array can hold one data item. Furthermore, each cell has its own index, to distinguish it from the rest of the cells. In Java, these cells are always numbered from 0 to the size of the array minus one. The array above is indexed from 0-9 and has size 10, since it can hold 10 elements.

Here is the generic syntax for an array declaration:

type[] <var_name>;

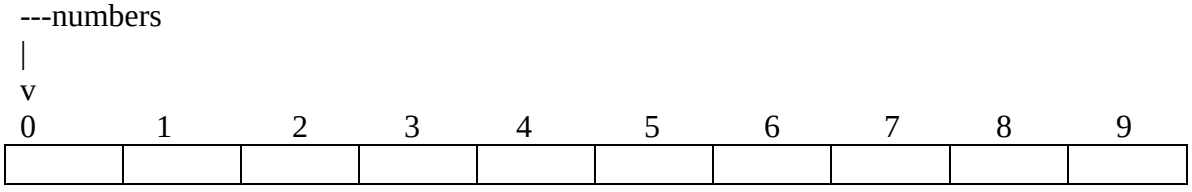
Here's an example:

```
int[] numbers;
```

The above line of code will declare an array called numbers, which will hold integer values. Its size is still undefined at this point. Since an array isn't a primitive, technically, numbers is just a reference. At the current time, no SPACE in the computer's memory has been allocated to store a set of integers. The next step is to allocate this space. In order to do this, we need to know how much space we want to allocate. If we wanted to allocate space for 10 variables, we would do the following:

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

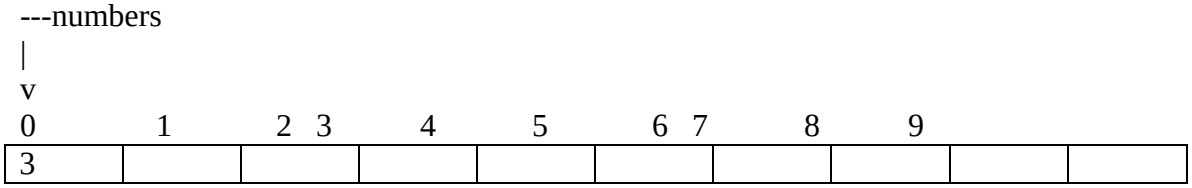
Once we've done this, the picture looks like the following:



To refer to a variable in a single cell or element of an array, you use the name of the array, followed by a bracket, the index you want to access, and then another bracket. Thus,

```
numbers[0] = 3;
```

would set the int variable in the 0 index of the numbers array to 3.



A sample program

The following program will read in five test scores and then print these out in reverse order:

```
public static void main(String[] args) {

     Scanner stdin = new Scanner(System.in);

     int index;
     int[] test_scores = new int[5];

     // Read in scores into the array.
     System.out.println("Please enter 5 test scores.");
     for (index=0; index < 5; index++)
          test_scores[index] = stdin.nextInt();

      // Print them out by going through the array in
backwards.
      printf("Here are the scores in reverse order: ");
      for (index=4; index >= 0; index--)
          System.out.print(test_scores[index]+" ");
      System.out.println();

}
```

Common mistakes made with arrays

1. Out of Bounds error: This is when you index into an array with an invalid index. For example, if you declare a `test_scores` of size 5 and tried to access `test_scores[5]` or `test_scores[-1]`, you would be trying to access a variable that does not exist, since the valid indices of `test_scores` range from 0 through 4.

   Often times, these errors are masked by the fact that the index to an array in an algorithm can be a complex expression. It is difficult to tell whether a particular expression will equal an invalid index at some point during the execution of the algorithm.

2. Not initializing all values of an array.

3. Trying to assign an entire array a value such as:

```
test_scores = 7;
```

   This will not assign 7 to each element of the array test-scores. In fact, it is not syntactically correct. This is because the left-hand side is not an integer variable, while the right hand side is an integer expression.

4. Not differentiating between an array index and the value stored in an array at a particular index. (We will talk about this more as we see sample algorithms.)

What's relatively simple about arrays?

Once you understand the difference between an array, array element, and an index to an array, it is fairly simple to follow array code and to write syntactically correct code dealing with arrays.

What is difficult about arrays?

The actual manipulation of array elements and indeces can get quite tricky. Let's look at an example, similar to the one we just did.

Let's say you are given an array named numbers, indexed from 0 to 99. You are to reverse the contents of the array. It looks like this solution might work:

```
int index, temp;
int[] numbers = new int[100];
   // ...Fill in values for numbers

for (index=0; index<100; index++) {
     temp = numbers[index];
     numbers[index] = numbers[99 – index];
     numbers[99 – index] = temp;
}
```

But, it does not. What happens here?

You end up reversing the list twice, so by the end of the loop, you've got the array in the exact order that you started.

Finally, we realize that we need to go through half of the list instead of the whole list, while we are swapping numbers. So, here an algorithm that correctly reverses the values in the array:

```
for (index=0; index<99/2; index++) {
     temp = numbers[index];
     numbers[index] = numbers[99 – index];
     numbers[99 – index] = temp;
}
```

Passing an array as a paramenter

Technically, an array is a reference.

```
int[] numbers;
```

numbers is a reference to the spot in memory where the beginning of the array is stored.

Since this is the case, whenever we pass an array into a function it works like a reference variable. Here is a method prototype with an array as a formal parameter:

```
public static void print(int[] board);
```

What this means is that the function print CAN change the values in the array (the actual parameter) that is passed to it. Consider the following function:

```
public static void initialize(int[] numbers,  int value) {

      for (int i=0;i < numbers.length; i++)
           numbers[i] = value;
}
```

What does this function do?

One other thing to note is that given an array, in Java we can always access the length of it. The variable length is like a public instance variable for all arrays. Though this seems silly, it can be very helpful at times.

Searching for a value in an array

One of the most common subroutines performed on an array is a search for a particular value. Let's look at a straightforward method for doing this, using a function:

```
public static boolean Find(int[] numbers, int val) {

    int index;

    for (index=0; index < numbers.length; index++) {
          if (numbers[index] == val)
                 return true;
    }
    return false;
}
```

Upon completion, the function will return either a true or false value, depending on whether or not the target value was found.Let's trace through an example, where we call this function from main as follows:

```
public static void main(String[] args) {
     int i;
     int[] num = new int[10];
     Scanner stdin = new Scanner(System.in);

     for (i=0;i<10;i++) {
          System.out.println("Enter the next number.");
          num[i] = stdin.nextInt();
      }
      if (Find(num, 3) == 1)
           System.out.println   ("You   entered   the   number
3.");
      else
           System.out.println   ("You   did   not   enter   the
number 3.");

      return 0;
}
```

Character Counting Example

Consider the following task:

Given an input of characters, count the frequencies of each alphabetic character, and print all of these out.

First, I will write a program without methods that completes this task. Then, I will show an improved design of this program with static methods. Finally, I will show an object-oriented design of this program that uses a Frequency object.

Here are some of the issues we'll have to discuss:

1) How to create 26 different counters using an array.
2) How update the proper counter once a letter is read in.
3) How to print out all the values of the counters in a reasonable manner.

Here are quick answers to these questions:

1) Use an array of size 26, and initialize each element to 0.

2) First we can test to see if the character is a letter. If so, we must determine how to find its corresponding value from 0 to 25. (Hint: Use the idea given in the last lecture!)

3) We can loop through each index (0 to 25) into the array, and for each one, print out the corresponding letter and value stored in the array at that index.

```
public static void main(String[] args) {

  int index;
  char c;
  int[] freq = new int[26];
  Scanner stdin = new Scanner(System.in);

  for (index = 0; index<26; index++)
    freq[index] = 0;

  String inp = stdin.nextLine();
  for (index=0; index<inp.length(); index++) {
    c = inp[index];
    if (c >= 'A' && c <= 'Z')
      freq[c-'A']++;
    else if (c >= 'a' && c <= 'z')
      freq[c-'a']++;
  }
```

```
  System.out.println("Letter\tFrequency\n");
  for (index = 0; index<26; index++)
    System.out.println((char)('a'+index)+"\t"+freq[index]);

}
```

```
public static void main(String[] args) {

  int index;
  int[] freq = new int[26];
  Scanner stdin = new Scanner(System.in);

  init(freq, 0);
  String inp = stdin.nextLine();
  count(freq, inp);
  printChart(freq);
}

public static void init(int[] arr, int val) {
  for (int i=0; i<arr.length; i++)
    arr[i] = val;
}

public static void count(int[] arr, String inp) {

  for (index=0; index<inp.length(); index++) {

    char c = inp.charAt(index);

    if (c >= 'A' && c <= 'Z')
      arr[c-'A']++;
    else if (c >= 'a' && c <= 'z')
      arr[c-'a']++;
  }
}

public static void printChart(int[] arr) {

  System.out.println("Letter\tFrequency\n");
  for (int index = 0; index<26; index++)
    System.out.println((char)('a'+index)+"\t"+freq[index]);

}
```

```
public class Frequency {

  private int[] arr;

  public static void main(String[] args) {

    int index;
    char c;
    Frequency fobj = new Frequency();
    Scanner stdin = new Scanner(System.in);

    fobj.init(0);
    String inp = stdin.nextLine();
    fobj.count(inp);
    fobj.printChart();
  }

  public Frequency() {
    arr = new int[26];
  }

  public  void init(int val) {
    for (int i=0; i<arr.length; i++)
      arr[i] = val;
  }

  public void count(String inp) {

    for (index=0; index<inp.length(); index++) {

      char c = inp.charAt(index);

      if (c >= 'A' && c <= 'Z')
        arr[c-'A']++;
      else if (c >= 'a' && c <= 'z')
        arr[c-'a']++;
    }
  }

public void printChart() {

  System.out.println("Letter\tFrequency\n");
  for (int index = 0; index<26; index++)
    System.out.println((char)('a'+index)+"\t"+freq[index]);

}
```