

# Arrays

- An array is a collection of values of the same type stored consecutively in memory.
- `int arr[] = new int[5];`

This declaration has 2 parts:

the stuff on the right of the assignment

the stuff on the left

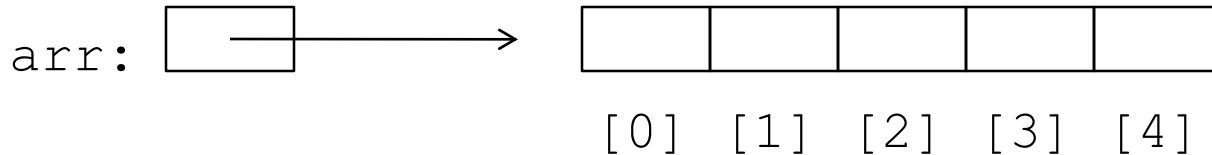
Let's start with the part on the right

keyword **new** has two properties

1) allocates a chunk of memory big enough for 5 ints

2) brings back the **address** of that chunk of memory(or throw Exception if out of memory)

- `int arr[] = new int[5];`
- Produces this:



- The **`int arr[]`** part creates a reference variable named `arr`. This ref variable contains the address of the first cell in the memory chunk. **`arr`** contains the **address** of the `[0]`'th cell. That's why there is an arrow coming out of `arr` pointing at the `[0]` cell. Thus ***`arr` points to*** or **references** the beginning of the array

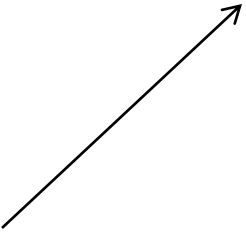
# Array Terminology

- `arr` is the reference variable
- The chunk of memory is the object
- References point to objects.
- Putting values into the array is easy
- Use `.length` to determine how many cells the array has
- ```
for (int i=0 ; i<arr.length ; ++i )  
    arr[i] = i*2;
```

now `arr: [-]---` `[0][2][4][6][8]`

# The `.length` property

- Once an array has been dimensioned you can always go back and ask the array how many cells of capacity it has.
- `int arr[] = new int[5];`
- `println( "arr has " + arr.length + " cells" );`



This expression produces the  
number 5

# The array “discipline”

- There are certain rules to follow in order to use an array correctly. We refer to these rules as the ***array discipline***.
- Once you master these rules and become an advanced programmer you may find occasion to bend or violate them. In general however there are many good reasons for always following the rules.

# The rules

- 1) When you declare an array you should declare an int named count or such to track how many values you have put into the array
- 2) initialize count to 0 and then use count to represent two things:
  - **the number of values** you have put into the array so far
  - **the index position** of where the next value should be stored

# Passing Arrays

- **When you pass an array, you must pass it to a method that is written to receive an array.**

```
public static void main( String[] args )
{
    int arrCnt = 0;
    int arr[] = new int[5]
    for (arrCnt=0 ; arrCnt<arr.length ; arrCnt++)
        arr[arrCnt] = arrCnt * 2;

    printArray( arr, arrCnt);
}
private static void printArray( int[] array, int cnt )
{
    for (int i=0 ; i < cnt ; ++i
        System.out.println( array[i] );

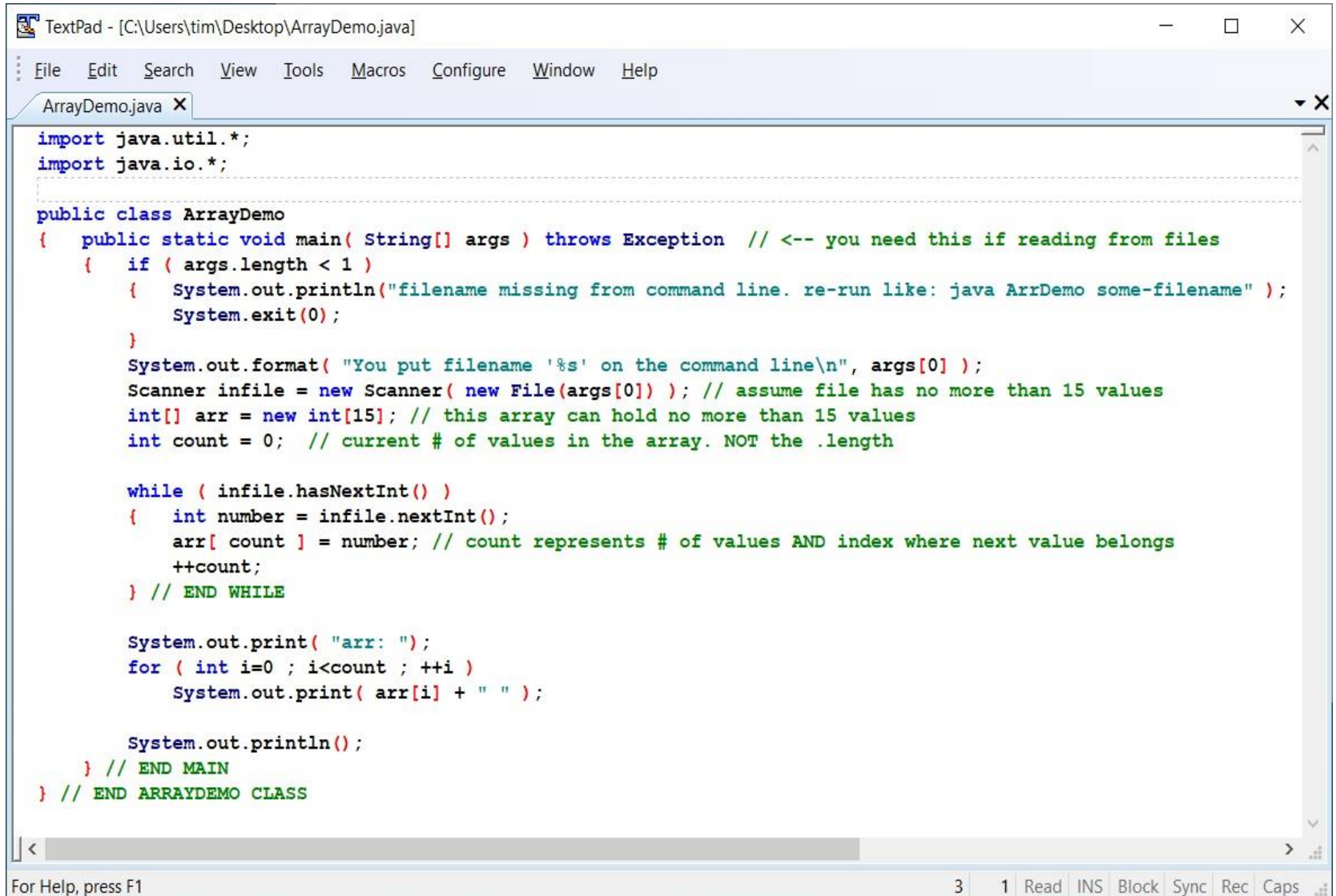
    System.out.println();
}
```



# Passing Arrays II

- When you pass an array you are just passing a copy of the address where the array starts. You are not passing a copy of the data values.
- It would be very memory inefficient to make a copy of the actual array and send that to the method.
- It is much more efficient to just pass a copy of the address (reference)

# Filling an Array from a file



```
TextPad - [C:\Users\tim\Desktop\ArrayDemo.java]
File Edit Search View Tools Macros Configure Window Help
ArrayDemo.java X
import java.util.*;
import java.io.*;

public class ArrayDemo
{
    public static void main( String[] args ) throws Exception // <-- you need this if reading from files
    {
        if ( args.length < 1 )
        {
            System.out.println("filename missing from command line. re-run like: java ArrDemo some-filename" );
            System.exit(0);
        }
        System.out.format( "You put filename '%s' on the command line\n", args[0] );
        Scanner infile = new Scanner( new File(args[0]) ); // assume file has no more than 15 values
        int[] arr = new int[15]; // this array can hold no more than 15 values
        int count = 0; // current # of values in the array. NOT the .length

        while ( infile.hasNextInt() )
        {
            int number = infile.nextInt();
            arr[ count ] = number; // count represents # of values AND index where next value belongs
            ++count;
        } // END WHILE

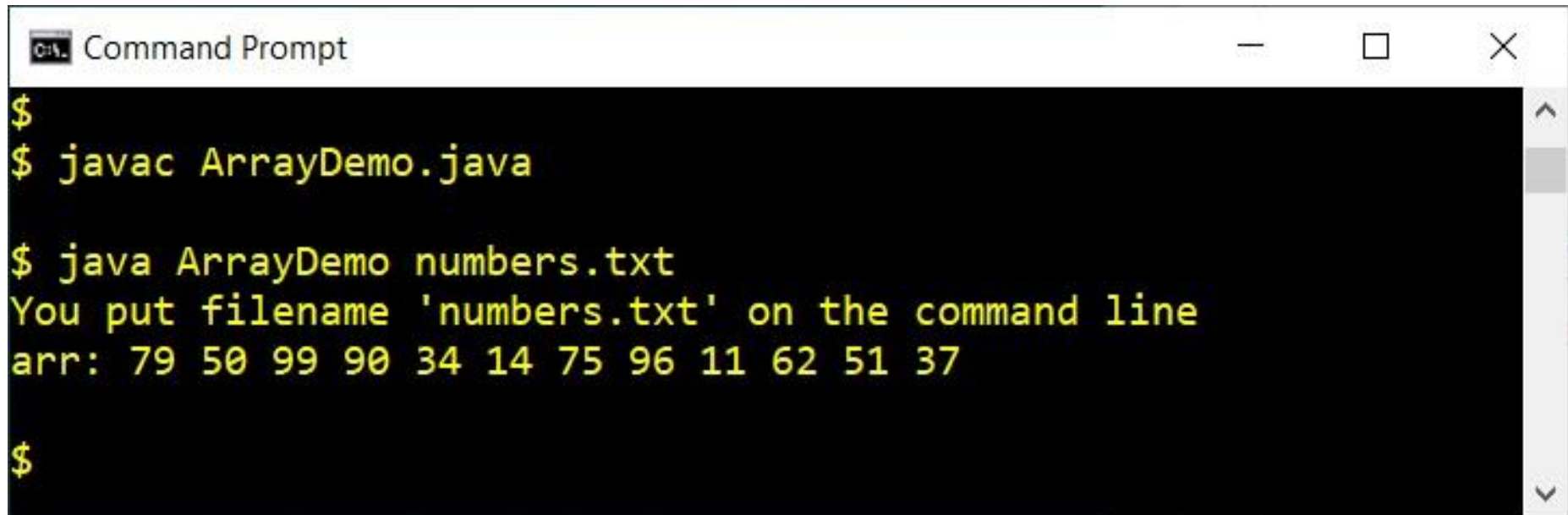
        System.out.print( "arr: " );
        for ( int i=0 ; i<count ; ++i )
            System.out.print( arr[i] + " " );

        System.out.println();
    } // END MAIN
} // END ARRAYDEMO CLASS

For Help, press F1
3 1 Read INS Block Sync Rec Caps
```

numbers.txt contains: 79 50 99 90 34 14 75 96 11 62 51 37

This is what the execution looks like



```
$  
$ javac ArrayDemo.java  
  
$ java ArrayDemo numbers.txt  
You put filename 'numbers.txt' on the command line  
arr: 79 50 99 90 34 14 75 96 11 62 51 37  
  
$
```

Watch how we gradually refine the loop that reads the numbers into the array

- `while ( infile.hasNextInt() )`
  - `{ int number = infile.nextInt();`
  - `arr[ count++ ] = number;`
  - `} // END WHILE`
- 
- Notice we increment the count inside the []s

And now another refinement of the loop

- `while ( infile.hasNextInt() )`
  - `{ int number = infile.nextInt();`
  - `arr[count++] = infile.nextInt();`
  - `} // END WHILE`
- 
- There was no need to read the number into a variable. We can read it directly into the array