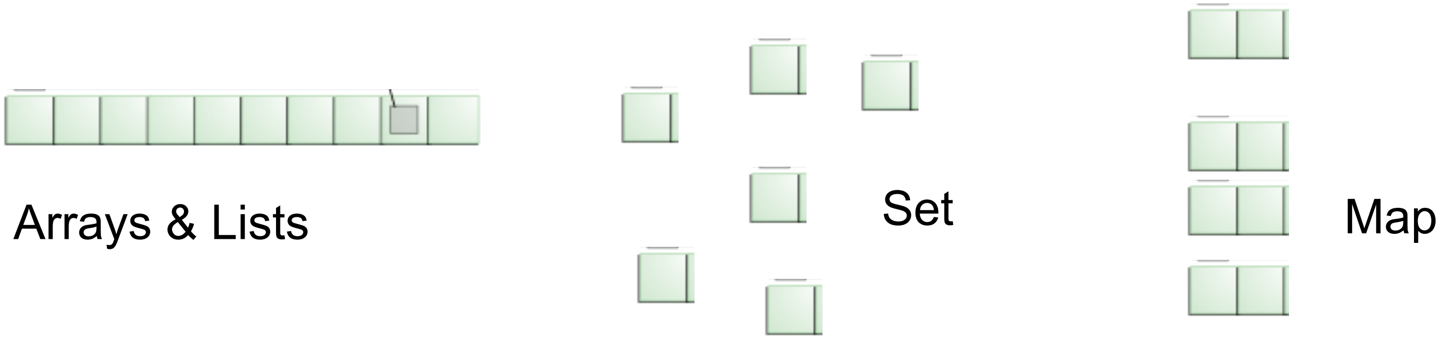
EECS 1710
Programming for Digital Media

Lecture 11 :: Arrays [1]

Collections or “Composite” types

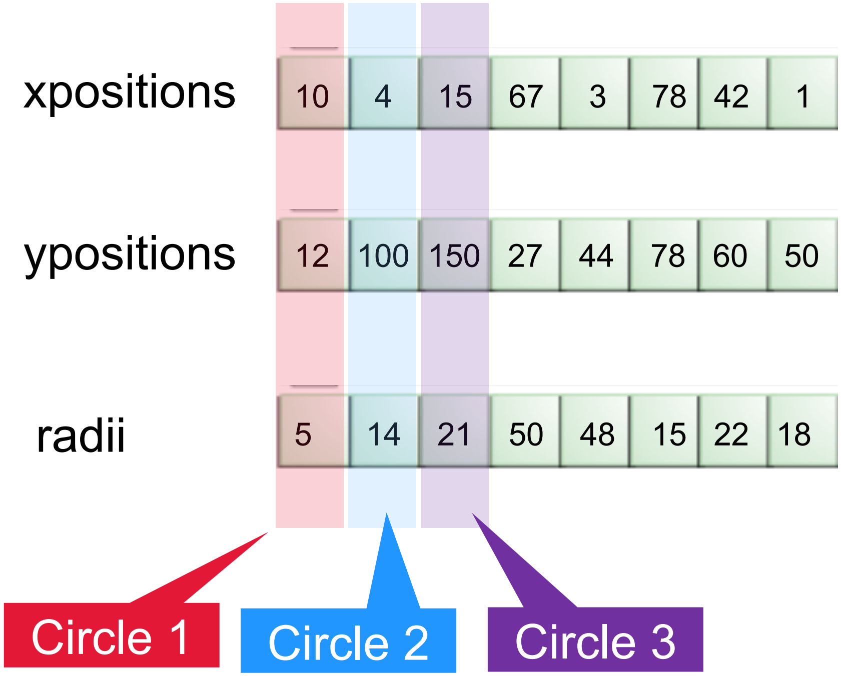
- Often, we would like to track a collection of values
  - E.g. set of ints, set of chars, set of Pixels, etc.
- There are many different types of collections provided in Java. We will explore more later.



- One of the most fundamental data types for holding multiple things, is called an ARRAY

For example

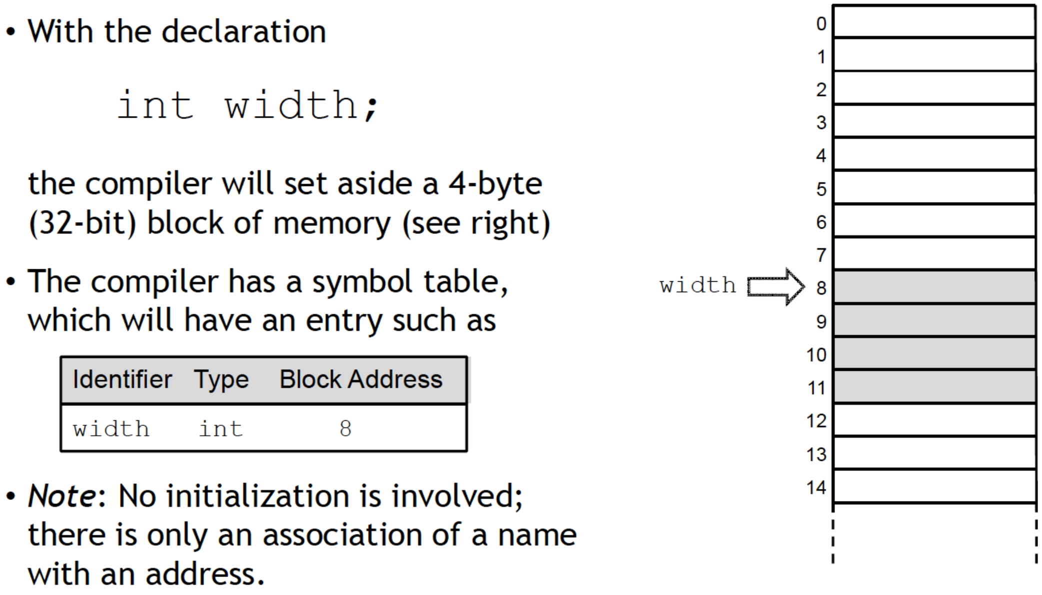
- You want to track a set of start positions (x values)
- Or a set of y values….
- Or imagine you want to track and modify many circles



xpositions	10	4	15	67	3	78	42	1
ypositions	12	100	150	27	44	78	60	50
radii	5	14	21	50	48	15	22	18

What makes composite types different from one another ?

- The way they organize + store multiple values **in memory**
- Recall: (Lect. 03)
  - memory generally broken up into bytes
  - each byte is given a location (block/memory address)
  - primitives store values in collections of bytes



- With the declaration

```
int width;
```

the compiler will set aside a 4-byte (32-bit) block of memory (see right)

- The compiler has a symbol table, which will have an entry such as

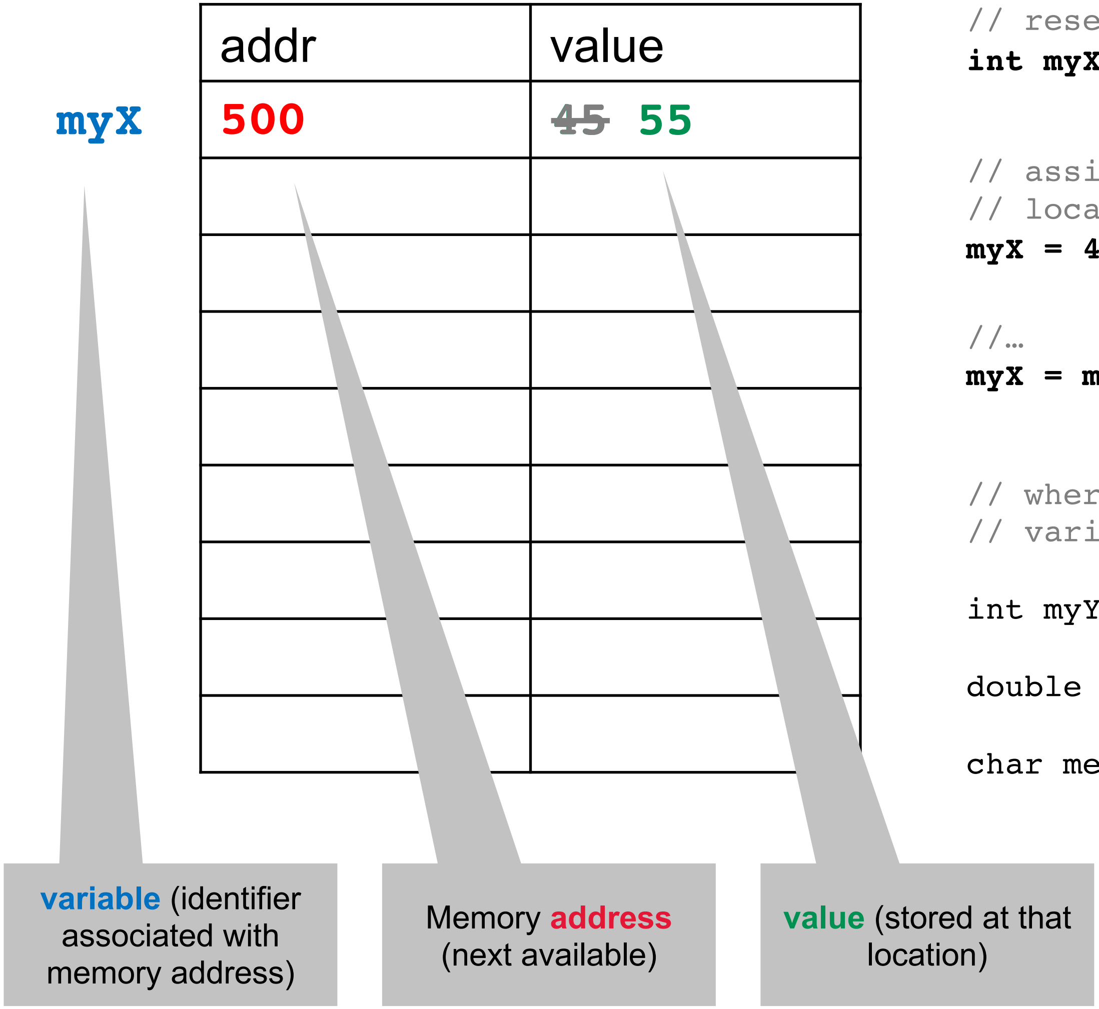
Identifier	Type	Block Address
width	int	8

- *Note*: No initialization is involved; there is only an association of a name with an address.

Recall: Memory ~ analogy of a theatre

- Theatre: memory block (storage – X number of seats)
- Seats → memory location (**address**)
- People → **values** (temp. reside in a (seat) mem. location)
- Tickets → **variables** (identifier connecting name to seat)

Primitives & Simple Memory Diagrams



	addr	value
myX	500	~~45~~ 55

variable (identifier associated with memory address)

Memory address (next available)

value (stored at that location)

```
// declaration - assume memory
// reserved at location 500
int myX;


// assignment – value stored at
// location 500 (ints take up 4 bytes)
myX = 45;

//…
myX = myX + 10;


// where does the next primitive
// variable declared go?

int myY = -3;

double weight = 97.6;

char menuKey = 'Q';
```

Primitives & Simple Memory Diagrams

	addr	value
myX	500	55
myY	504	-3

```
// declaration - assume memory
// reserved at location 500
int myX;


// assignment – value stored at
// location 500 (ints take up 4 bytes)
myX = 45;

//…
myX = myX + 10;


// where does the next primitive
// variable declared go?

int myY = -3;

double weight = 97.6;

char menuKey = ‘Q’;
```

Primitives & Simple Memory Diagrams

	addr	value
myX	500	55
weight	508	97.6
myY	504	-3

```
// declaration - assume memory
// reserved at location 500
int myX;


// assignment – value stored at
// location 500 (ints take up 4 bytes)
myX = 45;

//…
myX = myX + 10;


// where does the next primitive
// variable declared go?

int myY = -3;

double weight = 97.6;

char menuKey = ‘Q’;
```

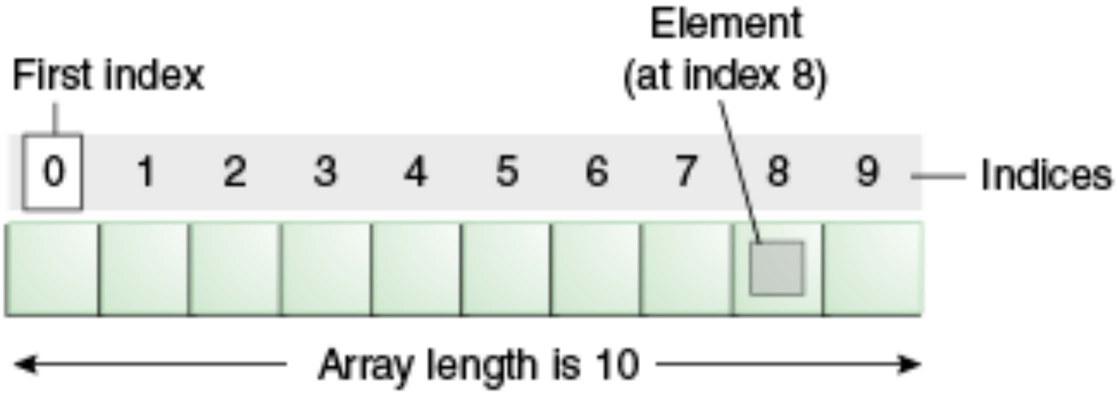
Primitives & Simple Memory Diagrams

	addr	value
myX	500	55
myY	504	-3
weight	508	97.6
letter	516	‘Q’

```
// declaration – assume memory
// reserved at location 500
int myX;


// assignment — value stored at
// location 500 (ints take up 4 bytes)
myX = 45;


//…
myX = myX + 10;


// where does the next primitive
// variable declared go?

int myY = -3;

double weight = 97.6;

char letter = ‘Q’;
```

Array → simple data structure for a sequential collection of uniform data values

- in Java an array is a container object that holds a **fixed** number of values of a single type
  - the length of an array is established when array is created

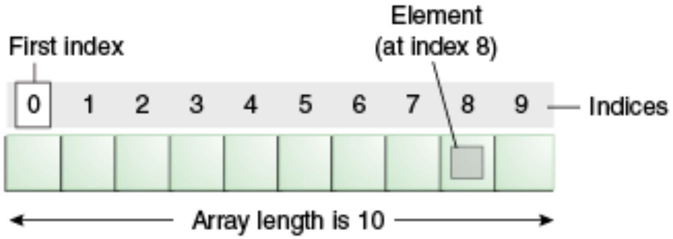


https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays (declaration in 2 stages)

- to declare an array variable use the element type followed by an empty pair of square brackets
- to declare the array itself, use **new** operator followed by element type followed by length of array (in square brackets)

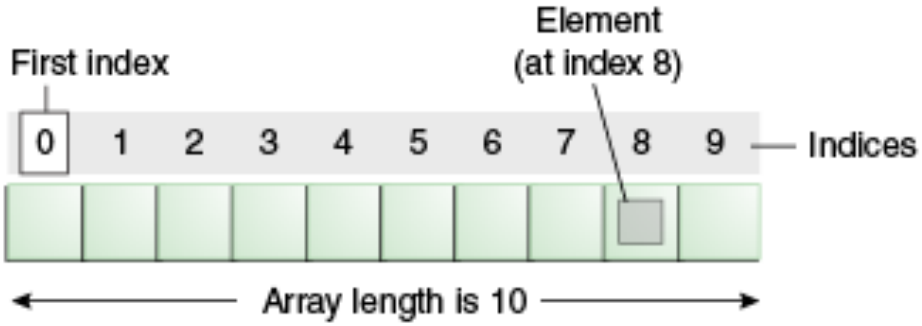
```
double oneElement;

double[] collection;
// collection is an array of double
// values

collection = new double[10];
// collection is an array of 10 double
// values
```



https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Arrays

- the values in an array are called elements
- the elements can be accessed using a zero-based index



```
// set all elements
// to equal 100.0

collection[0] = 100.0;
collection[1] = 100.0;
collection[2] = 100.0;
collection[3] = 100.0;
collection[4] = 100.0;
collection[5] = 100.0;
collection[6] = 100.0;
collection[7] = 100.0;
collection[8] = 100.0;
collection[9] = 100.0;
```

```
int n = collection.length;
// all arrays automatically have a special property “length” = size of
array (this is accessed using the “.” or “dot” syntax (as above)
```

https://docs.oracle.com/javase/tutorial/java/nutsandbolts/arrays.html

Declaring an Array

Lets consider arrays of primitive types first

```
// integer array:
int [] arrayOfInts;

// double array:
double [] arrayOfDoubles;

// char array:
char [] arrayOfChars;
```

Arrays of primitive types

```
byte[]    anArrayOfBytes;
short[]   anArrayOfShorts;
long[]    anArrayOfLongs;
float[]   anArrayOfFloats;
double[]  anArrayOfDoubles;
boolean[] anArrayOfBooleans;
char[]    anArrayOfChars;
```

Declaration & Initialization/Assignment

```
int [] anArray = new int[10];
                    // creating the array
                    // i.e. reserves block of memory

anArray[0] = 100;   // initialize first element
anArray[1] = 200;   // initialize second element
anArray[2] = 300;   // and so forth
…
```

```
int [] anArray = {  // create and init together
    100, 200, 300,
    400, 500, 600,
    700, 800, 900,
    1000
};
```

NOTE: this approach can only be used while declaring the array (it cannot already exist)

an Array → in memory?

	addr	value
myX	500	55
myY	504	-3
weight	508	97.6
letter	516	‘Q’
anArray	518	
		…

```
int myX;
myX = 45;
myX = myX + 10;

int myY = -3;
double weight = 97.6;
char letter = ‘Q’;
```

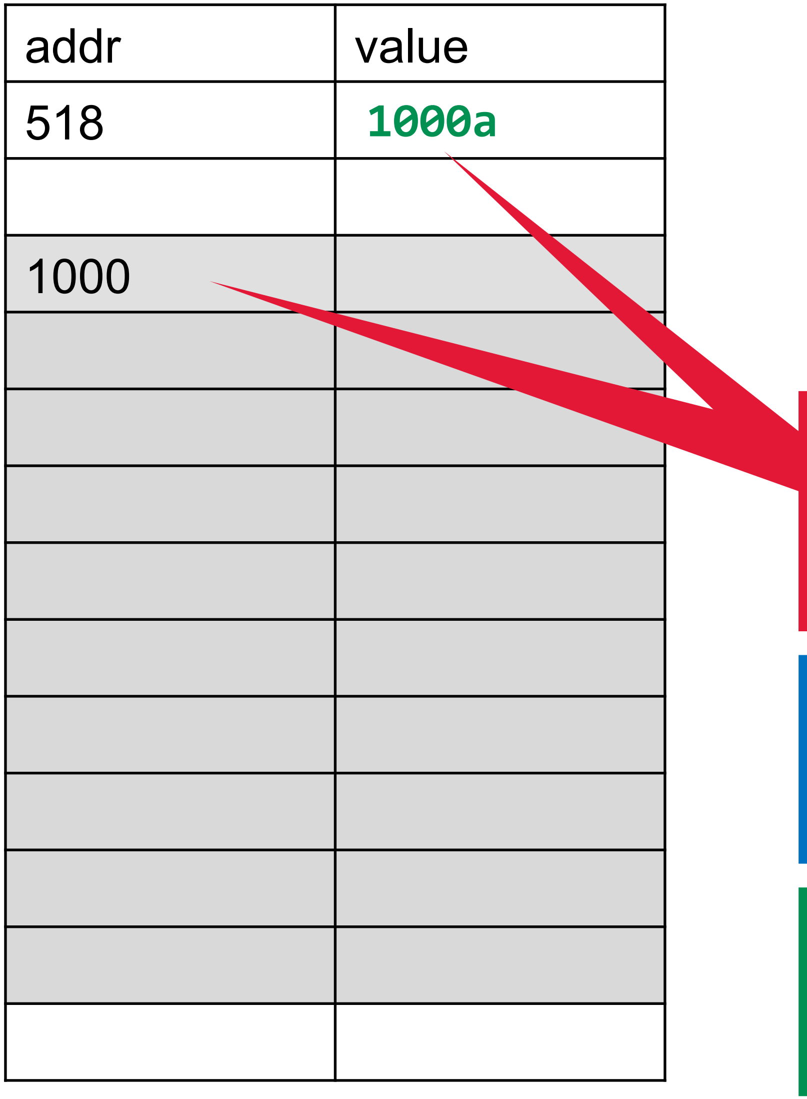
Fixed memory (pre-reserved at compile time)

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

New memory
(asks for a block big enough to fit 10 ints at run time)

New memory block

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

anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
```



	addr	value
anArray	518	1000a
	1000	

```
// declaration only
int [] anArray;
```

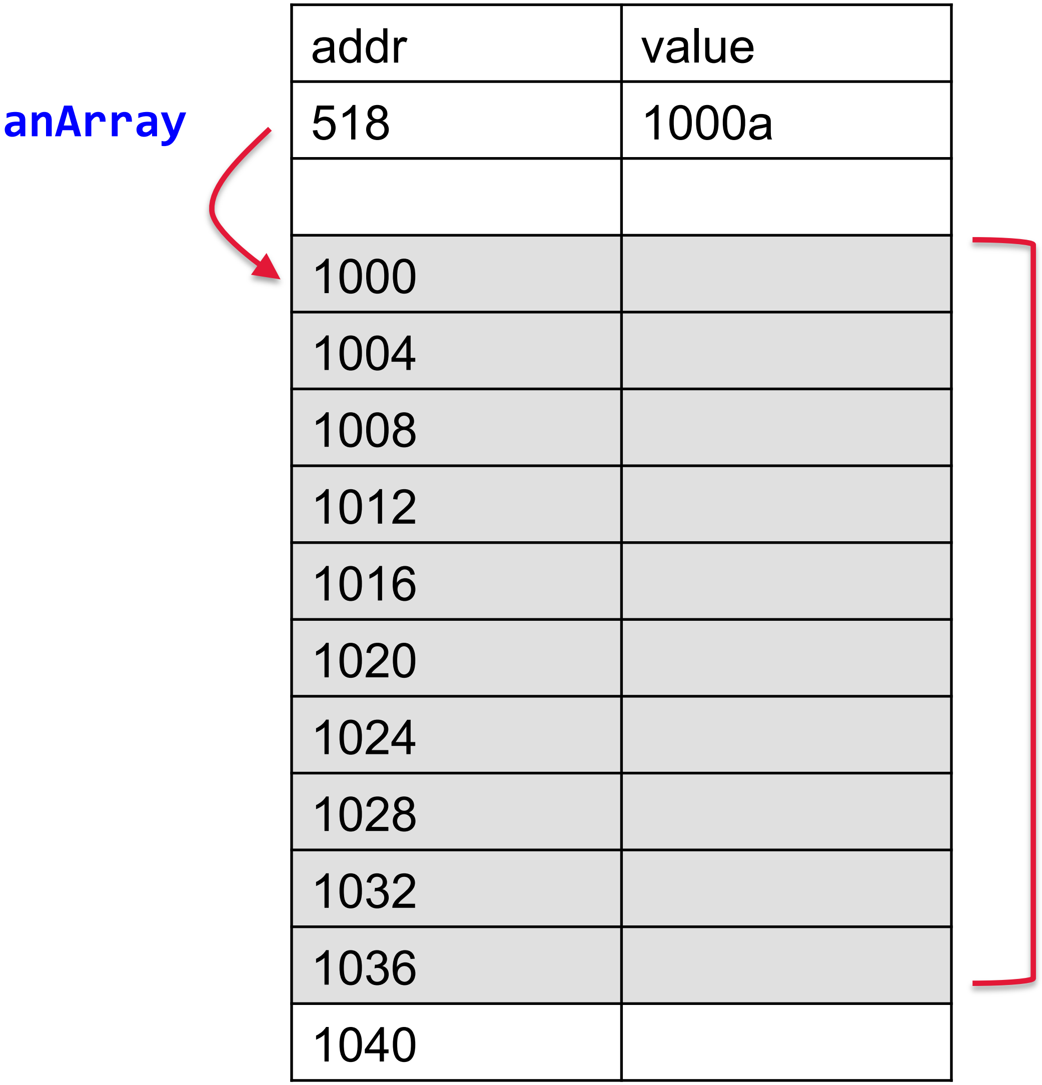
A valid location is determined at runtime (array variables only store the address that is determined):
1000a = address location 1000

The address acts as a “reference” to where the actual array exists in memory (where the array stores its collection of values)

variables that reserve memory at runtime (and store an address) are called “REFERENCE TYPES” (as opposed to primitive types)

In memory ??

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

anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
```



anArray

addr	value
518	1000a
1000	
1004	
1008	
1012	
1016	
1020	
1024	
1028	
1032	
1036	
1040	

new int[10] means:

10 x consecutive integer sized blocks of memory allocated starting at the address specified by anArray

anArray = 1000a

(i.e. a block of 10 ints = 10 * 4 bytes) is “reserved” for the array “anArray”

In memory ??

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

anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
```

	addr	value
anArray	518	1000a
	1000	100
	1004	
	1008	
	1012	
	1016	
	1020	
	1024	
	1028	
	1032	
	1036	
	1040	

“anArray[0]” means …
value at address: anArray + 0 ints

`anArray+0 = (1000+0)a = 1000a`

```
anArray[0] = 100
i.e. assign integer 100 to
value at address:
“1000a + 0” = 1000a
```

In memory ??

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

anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
```

	addr	value
anArray	518	1000a
	1000	100
	1004	200
	1008	
	1012	
	1016	
	1020	
	1024	
	1028	
	1032	
	1036	
	1040	

“anArray[1]” means …
value at address: anArray + 1 int

`anArray+1*4 = (1000+4)a = 1004a`

```
anArray[1] = 200
i.e. assign integer 200 to
value at address:
“1000a + 4” = 1004a
```

In memory ??

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

anArray[0] = 100;
anArray[1] = 200;
anArray[2] = 300;
```

	addr	value
anArray	518	1000a
	1000	100
	1004	200
	1008	300
	1012	
	1016	
	1020	
	1024	
	1028	
	1032	
	1036	
	1040	

“anArray[2]” means …
value at address: anArray + 2 ints

```
anArray+2*4 = (1000+8)a = 1008a
```

```
anArray[2] = 300
i.e. assign integer 300 to
value at address:
“1000a + 8” = 1008a
```

Creating & Initializing an Array of doubles

```
double [] anArray = new double[10];

anArray[0] = 100.0; // initialize first element
anArray[1] = 24.57; // initialize second element
anArray[2] = 300.4; // and so forth
```

```
double [] anArray = { // create and init together
      100.0, 24.57, 300.4
};
```

Creating & Initializing an Array of chars

```
char [] anArray = new char[3];

anArray[0] = ‘e’; // initialize first element
anArray[1] = ‘g’; // initialize second element
anArray[2] = ‘g’; // and so forth
…
```

```
char [] anArray = { // create and init together
       ‘e’,’g’,’g’
};
```

Creating & Initializing an Array of booleans

```
boolean [] anArray = new boolean[3];

anArray[0] = true; // initialize first element
anArray[1] = false; // initialize second element
anArray[2] = false; // and so forth
```

```
boolean [] anArray = { // create and init together
      true, false, false
};
```

On your own

- Question: if we create an array (size N) of a primitive type, and only (N/2) locations are assigned/initialized, what values do the rest of the locations in the array hold?
  - for an integer array `int []`
  - for a double array `double []`
  - for a char array `char []`
  - for a Boolean array `boolean []`

TRY YOURSELF (in Processing PDE)

Array Indexing

- Assume integer array: **`int [] myArray = new int[100];`**
- Zero-based indexing (positions in array start from 0, last element at (myArray.length-1)
- Let “idx” = index or position in the array:

```
int idx = 10;               // set index variable “idx”
myArray[0]                  // first element of array
myArray[1]                  // second element
myArray[idx]                // (idx+1)th element (11th in this case)
myArray[++idx]              // next element (12th element)
myArray[idx++]              // this element (11th), but then idx=12

myArray[-1] ??              // ERROR (causes an exception)
myArray[100] ??             // ERROR (in both cases, trying
                            // to read off ends of the array)
                            // recall -> myArray[99] is last element
```

Array traversal

- Usually with a loop (**for** or while)

```
int [] myArray = new int[100];

// do some assignments here

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

   // do something with an individual array element

   println(“myArray[” + index + “]=”
                                  + myArray[index]);

}
```

Array traversal

- Usually with a loop (for or **while**)

```
int [] myArray = new int[100];

// do some assignments here

int index = 0;
while (index < myArray.length) {

    // do something with an individual array element
    println(“myArray[” + index + “]=”
                                  + myArray[index]);

    index++;
}
```

Array examples

- Sum the values in an array?
- Pick a random element in the array?

Sum the values in an array?

```
final int MAX_ELEMENTS = 100;
int [] myArray = new int[MAX_ELEMENTS];


// assume array elements are set/assigned to here

int sum;

// code to sum elements




println(“The total sum = ” + sum);
```

Sum the values in an array?

```
final int MAX_ELEMENTS = 100;
int [] myArray = new int[MAX_ELEMENTS ];


// assume array elements are set/assigned to here

int sum = 0;

// code to sum elements

for (int i = 0;  i < myArray.length; i++)  {
    sum += myArray[i];
}

println(“The total sum = ” + sum);
```

Pick a random element from an array?

```
final int MAX_ELEMENTS = 16;
char[] hexDigits = {‘1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’,
                              ‘a’,’b’,’c’,’d’,’e’,’f’};


// output 3 randomly chosen hexadecimal digits

int choices = MAX_ELEMENTS;
int randomIndex;
```

Pick a random element from an array?

```
final int MAX_ELEMENTS = 16;
char[] hexDigits = {‘1’,’2’,’3’,’4’,’5’,’6’,’7’,’8’,’9’,
                              ‘a’,’b’,’c’,’d’,’e’,’f’};


// output 3 randomly chosen hexadecimal digits

int choices = MAX_ELEMENTS;
int randomIndex;

for (int i=0;  i<3; i++) {

    randomIndex = (int) floor(random(choices));

    println(hexDigits[randomIndex]);

}
```

Array examples

- Sum the values in an array?
- Pick a random element in the array?

Try yourself (for next class)

- Reverse an array?
- Find the minimum of an array of int’s?
- Find the maximum of an array of int’s?

Reverse an array?

```
final int MAX_ELEMENTS = 100;
int[] myArray = new int[MAX_ELEMENTS];
int[] myArrayReversed = new int[MAX_ELEMENTS];

// assume array elements are set/assigned to here

// now reverse the order of the elements
```

Find the maximum element in an array?

```
final int MAX_ELEMENTS = 100;
int[] myArray = new int[MAX_ELEMENTS];

// assume array elements are set/assigned to here

int indexOfMax;
int maxValue;
int currElement;                         // current element in array


// find maximum here








println(“largest value = ” + maxValue);
println(“found at i = “ + indexOfMax);
```

Find the minimum element in an array?

```
final int MAX_ELEMENTS = 100;
int[] myArray = new int[MAX_ELEMENTS ];

// assume array elements are set/assigned to here

int indexOfMin;
int minValue;
int currElement;


// find the minimum here







println(“smallest value = ” + minValue);
println(“found at i = “ + indexOfMin);
```