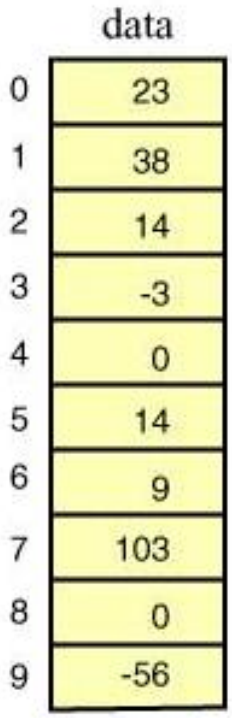
Arrays

- Say that you are writing a program that reads in 100 numbers for data.
- Would you like to declare 100 variables and write 100 input statements to read in the data?
- Even if it was 6 numbers, it's not too efficient to declare 6 separate variables.
- Instead, if the 6 variables are the same type, we can and should use arrays.

Arrays



- 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 "slots."
- Each slot holds a value, and **all the values are of the same type**. In the example array here, each slot holds an int.
- The name of this array is `data`.
- The slots are indexed 0 through 9. Each slot can be accessed by using its **index**.
- For example, `data[0]` is the slot which is indexed by zero (which contains the value 23). `data[5]` is the slot which is indexed by 5 (which contains the value 14).

Arrays

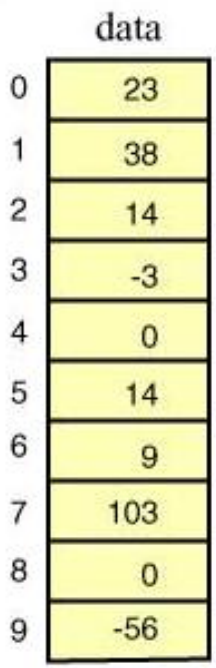
Important fact about arrays in Java:

- Slots are numbered sequentially starting at zero.

- If there are N slots in an array, the indexes will be 0 through N-1.

Using Arrays

Every slot of an array holds a value of the same type.



For example, you can have an array of `int`, an array of `double`, and so on.

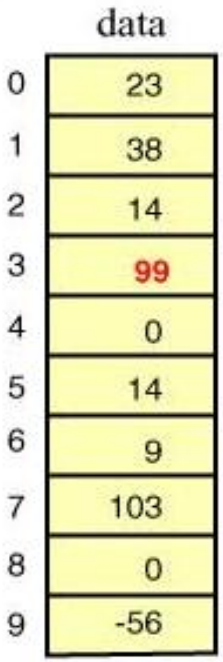
This array holds data of type `int`. Every slot may contain only an `int`.

A slot of this array can be used anywhere a variable of type `int` can be used.

For example,

```
data[3] = 99 ;
```

Using Arrays



Every slot of an array holds a value of the same type.

For example, you can have an array of `int`, an array of `double`, and so on.

This array holds data of type `int`. Every slot may contain only an `int`.

A slot of this array can be used anywhere a variable of type `int` can be used.

For example,

```
data[3] = 99 ;
```

Using Arrays

Any of the array entries (or *elements*) can be used exactly the same way as a standard variable, including arithmetic expressions.

For example, if `x` contains a 10, then

```
(x + data[2]) / 4
```

evaluates to

```
(10+14) / 4 == 6.
```

Here are some other legal statements:

```
data[0] = (x + data[2]) / 4 ;
data[2] = data[2] + 1;
x = data[3]++ ; // data in slot 3 is incremented
data[4] = data[1] / data[6];
```

Using Arrays

Array declarations look like this:

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

This names the type of data in each slot and the number of slots.

Once an array has been constructed, the number of slots it has does **not** change.

Array Boundary Checking

The **length** of an array is how many slots it has. An array of length N has slots indexed 0…(N-1)

Indexes must be an integer type. It is OK to have spaces around the index of an array. For example, `data[1]` and `data[ 1 ]` are exactly the same as far as the compiler is concerned.

It is *not legal* to refer to a slot that does not exist:
Say that an array was declared as:

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

Here are some elements of this array. Which are valid?

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

Array Boundary Checking

The **length** of an array is how many slots it has. An array of length N has slots indexed 0...(N-1)

Indexes must be an integer type. It is OK to have spaces around the index of an array. For example, `data[1]` and `data[ 1 ]` are exactly the same as far as the compiler is concerned.

It is *not legal* to refer to a slot that does not exist:
Say that an array was declared as:

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

Here are some elements of this array, are they valid?

```
data[ -1 ] always illegal
data[ 10 ] illegal (given the above declaration)
data[ 1.5 ] always illegal
data[ 0 ] always OK
data[ 9 ] OK (given the above declaration)
```

Array Boundary Checking

Now, when you get

```
Error line 17:
ArrayIndexOutOfBoundsExceptionError
```

It means you've overstepped the boundaries of the array, either with an index less than 0, or greater than N-1, where N is the length of the array.

In English,

```
Array Index Out Of Bounds Exception Error
```

Variables as Index Values

The index of an array is always an integer type.

It does not have to be a literal.

It can be any expression that evaluates to an integer. For example, the following are legal:

```
int values[] = new int[7];
int index = 0;
values[ index ] = 71; // put 71 into slot 0
index = 5;
values[ index ] = 23; // put 23 into slot 5
index = 3;
values[ 2+2 ] = values[ index-3 ]; // same as
//values[ 4 ] = values[ 0 ];
```

Variables as Index Values

Using an expression for an array index is a very powerful tool.

Often a problem is solved by organizing the data into arrays, and then processing that data in a systematic way using variables as indexes. Here is a further example:

```
double[] val = new double[4];
val[0] = 0.12;
val[1] = 1.43;
val[2] = 2.98;
int j = 3;
System.out.println("slot 3:" + val[j] );
System.out.println("slot 2:" + val[j-1] );
j = j-2;
System.out.println("slot 1:" + val[j] );
```

Array initialisation as a list

You can declare, construct, and initialise 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 slots (indexed 0…9).

Finally it puts the designated values into the slots. The first value in the **list** corresponds to index 0, the second value corresponds to index 1, and so on. (So in this example, data[0] gets the 23.)

The compiler will count the values in the list and make that many slots.

Lists are usually used only for small arrays.

Copying Arrays

Say we have two arrays:

```
int array1[] = {17,12,32,103,5};
int array2[] = {22,57,13,203,15};
```

How do we copy the contents of array1 into array2?

Can we just do this?

```
array2 = array1;
```

Copying Arrays

Say we have two arrays:

```
int array1[] = {17,12,32,103,5};
int array2[] = {22,57,13,203,15};
```

How do we copy the contents of array1 into array2?

Can we just do this?

~~`array2 = array1;`~~

We should **never** do this! Worst of all, it does not cause an error.

Arrays should always be dealt with on an element by element basis.

Copying Arrays

How should we do it then?

How about…

```
array2[0] = array1[0];
array2[1] = array1[1];
array2[2] = array1[2];
array2[3] = array1[3];
array2[4] = array1[4];
```

This will work, but it's a little inefficient

Remember, we can use a variable of type `int`, instead of just a literal…

Copying Arrays

This is the generally accepted way.
Remember, the two arrays must must be of the same type.

```
double array[] = {17.0,23.4,57.678…};
double array2[] = {22.57,67.2,…};
```

```
for(int i = 0; i < array.length; i++)
{
      array[i] = array2[i];
}
```

Copying Arrays

This is the generally accepted way.
Remember, the two arrays must must be of the same type.

```
double array[] = {17.0,23.4,57.678…};
double array2[] = {22.57,67.2,…};
```

```
for(int i = 0; i < array.length; i++){
      array[i] = array2[i];
}
```

```
//To print any array, it's just the same…
for(int j = 0; j < array2.length; j++){
      System.out.println(array2[j]);
}
```

For arrays in general…

THINK OF `FOR` LOOPS!

Why? Because `for` loops execute for an exact number of times, no more, no less. This is tailor made for arrays which are always of a definite size.

Exercise 1

```
import java.util.Scanner;

public class Exercise01{
   public static void main(String[] args) {
      Scanner kbinput = new Scanner(System.in);

      System.out.println("Please enter array size");
      int size = kbinput.nextInt();

      int array[] = new int[size];

      fillArray(array, size);
      printArray(array, size);
   }
}
```

Defining the Method

```
public static void fillArray(int a[], int aSize)
{
   <your code here>

}//end method
```