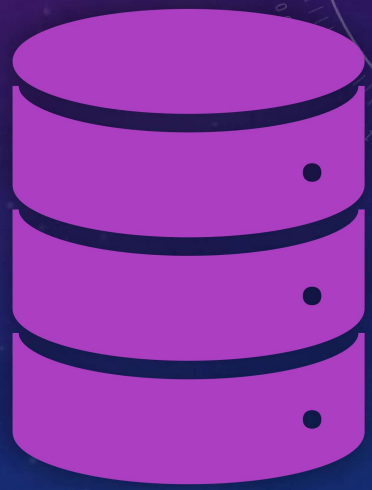
STORING LOTS OF DATA

- Imagine you are writing a program that will take input from 100 different people.
- They will each input their age, for example
- How would we store these 100 ints?



SHOULD WE STORE THE DATA LIKE THIS?

int a = 20

int b = 21

int c = 19

int d = 27

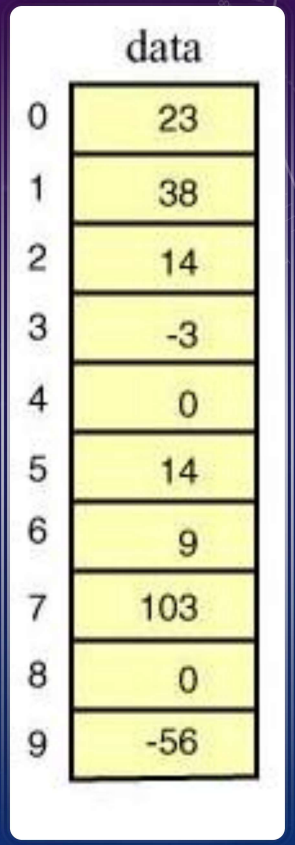
int e = 22

int f = 45

…… etc?

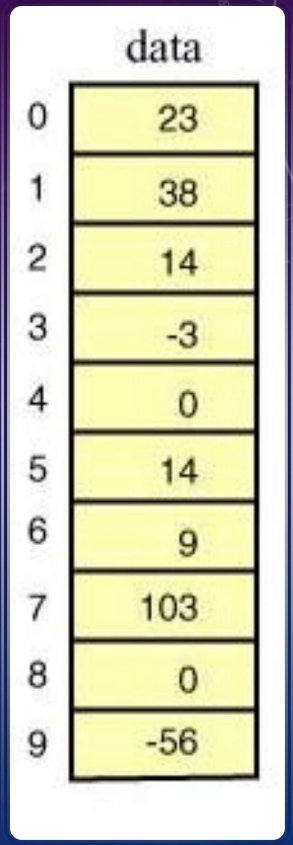
ARRAYS!

- Arrays are a way to group large numbers of variables together
- 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“
- What do we mean by contiguous?



ARRAYS!

- Each slot holds a value, and all the values are of the same type. In the example array here, each slot holds an int
- Arrays have names, for example this one is called 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)



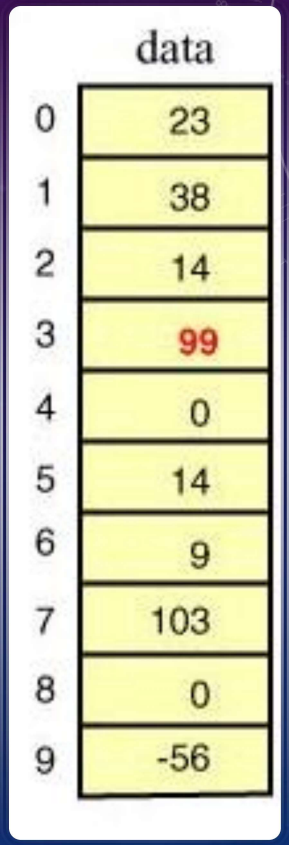

```
int[] data = {23, 37, 14, -3, 0, 14, 9, 103, 0, -56};
```

ARRAYS

- **Important:**
- The slots are numbered sequentially starting at **zero**.
- If there are N slots in an array, the indexes will be 0 through N-1
- If you write a for loop cycling through all of the slots in an array, make sure it stops at N-1

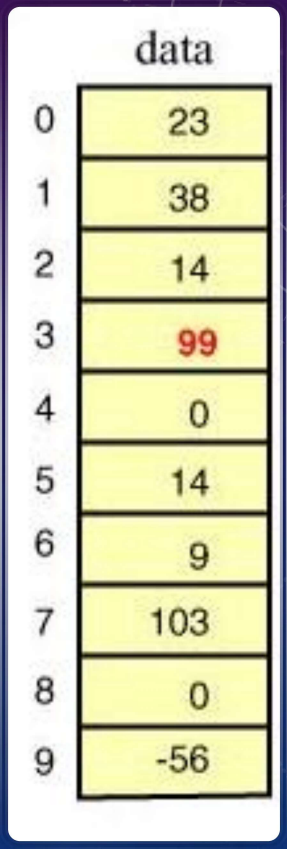
USING ARRAYS

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



USING ARRAYS

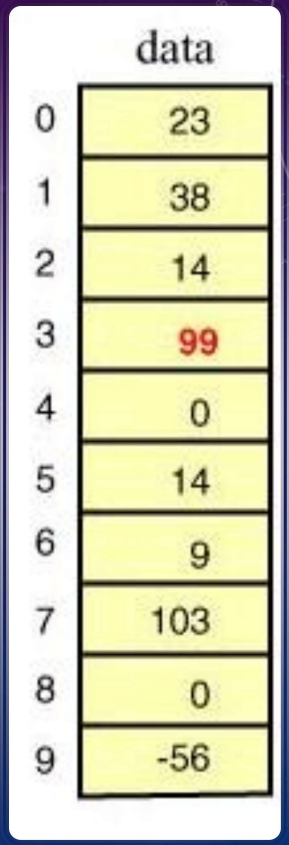
- Any of the array entries (or *elements*) can be used exactly the same way as a standard variable, including arithmetic expressions.
- For example, if x contains a 10, then
  - `(x + data[2]) / 4`
    - evaluates to
  - `(10+14) / 4 == 6`



USING ARRAYS

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];`



DECLARING 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 will **not** change.
  - Why do you think that is?

DECLARING ARRAYS

Examples:

- int[] myArray = new int[20];
- double[] theArray = new double[5];
- String[] words = new String[17];
- char[] charArray = new char[256];

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 *not legal* to refer to a slot that does not exist

ARRAY BOUNDARY CHECKING

- Say that an array was declared:
- `int[] data = new int[10];`
- Here are some elements of this array, are they valid?
- `data[ -1 ]`
- `data[ 10 ]`
- `data[ 1.5 ]`
- `data[ 0 ]`
- `data[ 9 ]`

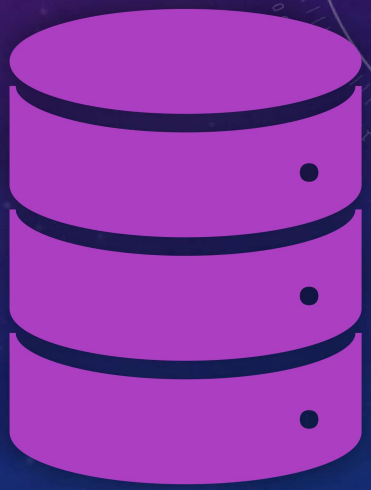
```
Error line 17: ArrayIndexOutOfBoundsExceptionError
```

VARIABLES AS INDEX VALUES

The index of an array is always an integer type

This means 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 are further examples:

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

INITIAL VALUES

- When array is created, all values are initialized depending on array type:
  - Numbers: 0
  - Boolean: `false`
  - Object References: `null`

ARRAY INITIALISATION AS A LIST

- You can declare, construct, and initializese 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.
- So in this example, `data[0]` gets the 23

COPYING ARRAYS

- Say we have two arrays:

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

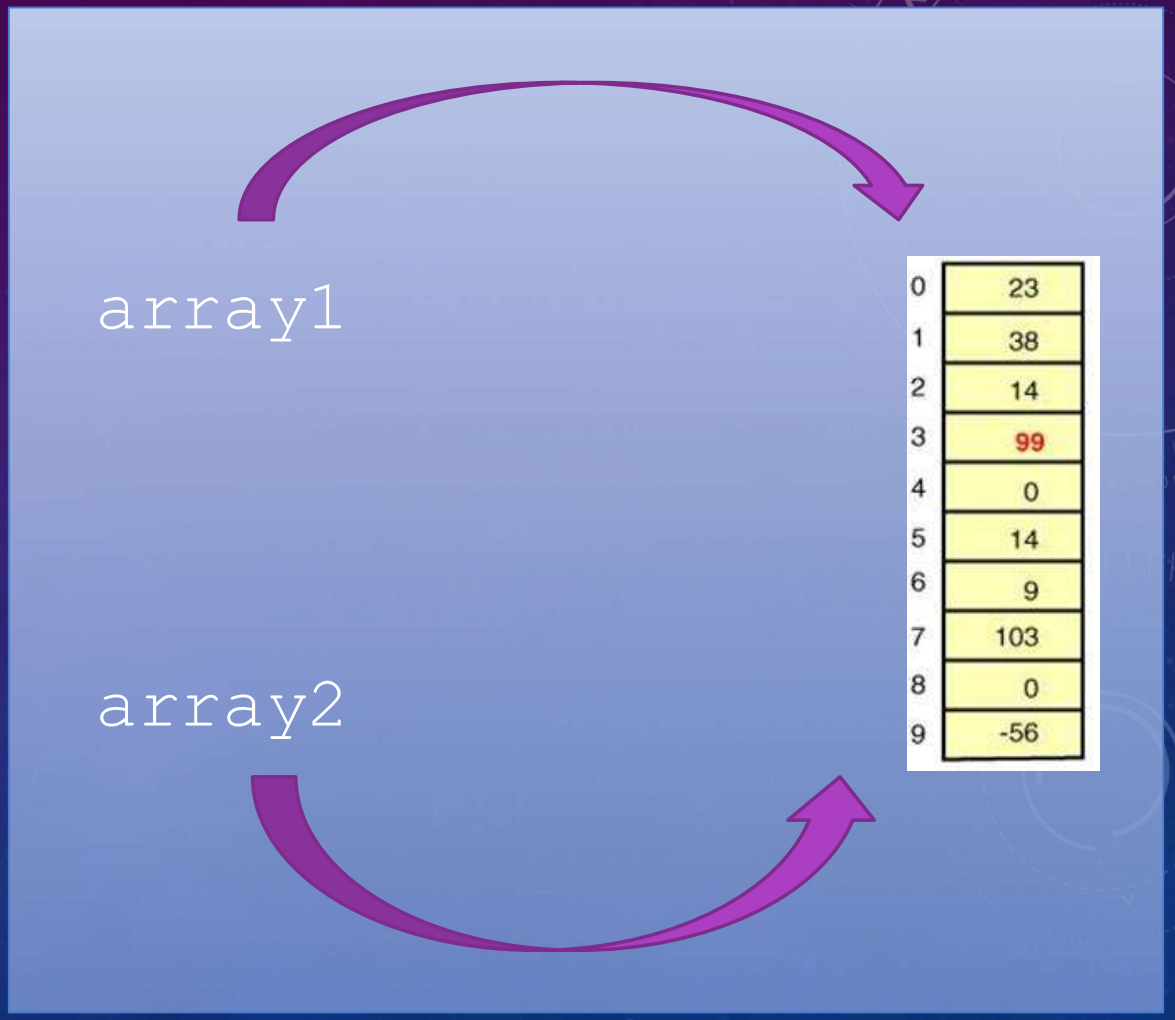
- How do we copy the contents of array1 into array2?

- Can we just do this?

- `array2 = array1;`

COPYING ARRAYS

- We just get two references to the same array!



COPYING ARRAYS

- This does not work for copying arrays!

```
array2 = array1;
```

- This doesn't cause an error, so you must remember this
- This will not copy the array – it will only copy the array **location**

- Arrays must be dealt with on an element by element basis

COPYING ARRAYS

- You must copy all the elements one by one
- 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, isn't it?
- We can produce the same effect using a loop

COPYING ARRAYS

- Arrays must be of the same type…

```
double[] array1 = {9,8,7,6,5,4,3,2,1,0};
double[] array2 = new double[10];

for(int i = 0; i < array1.length; i++){

    array2[i] = array1[i];

}
```

ARRAYS MUST BE OF THE SAME TYPE...

```
33
34          double[] arrayx = {9,8,7,6,5,4,3,2,1,0};
35          String[] arrayy = new String[10];
36
37          for(int i = 0; i < arrayx.length; i++){
38
39              arrayy[i] = arrayx[i];
40
41          }
42
```

Problems @ Javadoc Declaration Console

```
<terminated> helloWorld [Java Application] /Users/drnat/.p2/pool/plugins/org.eclipse.justj.openjdk.hotspot.jre.full.macosx.x86_64_22.0.2.v202
Exception in thread "main" java.lang.Error: Unresolved compilation problem:
        Type mismatch: cannot convert from double to String

        at helloWorld/helloWorld.helloWorld.main(helloWorld.java:39)
```

PRINTING ARRAYS

```
for(int j = 0; j < array.length; j++)
{
    System.out.println(array[j]);

}
```

ARRAYS AND LOOPS

- THINK OF FOR LOOPS!

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

ARRAY LENGTH

- If we are uncertain about the size of an array, we can use array.length to get it
- Because arrays are a fundamental data type, we get the length using the statement

```
int length = array.length;
```

- In comparison, Strings are a class, when we get the length of a String we are calling a method and must provide brackets

```
int length = message.length();
```

EXERCISE

Write a program that:

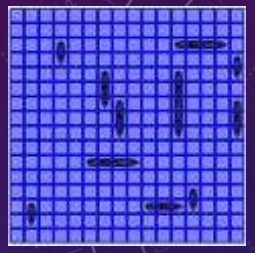
- takes the array size as input from the user,
- creates an `int` array of that size,
- populates it with values, prompting the reader for each value.

NESTED LOOPS

- This code uses nested for loops to print out each name in each slot, one character at a time
  - The outer loop selects a name in a particular slot
  - The inner loop prints out each character of that name, one at a time

```
String[] names = {"Peter", "Susan", "Keith"...};
for(int i = 0; i < names.length; i++){
  for(int j = 0; j < names[i].length(); j++){
    System.out.print(names[i].charAt(j) + " ");
  }
  System.out.println();
}
```

2D ARRAYS



- Often data comes in a two dimensional form.
- For example, maps are two dimensional, the layout of a printed page is two dimensional, a computer-generated image (such as on your computer screen) is two dimensional, and so on.

- Think Battleships, or chess in a newspaper, or reading a map. It's always just rows by columns or x by y, etc
- So, instead of one value to specify an array element or slot, we now need two

TWO DIMENSIONAL ARRAYS

- A single dimensional stores data as a list

[1 2 3 4 5 6 7 8 9]

- A two dimensional array stores data using two separate indices – like a rectangle

[1 2 3
 4 5 6
 7 8 9]

2D ARRAYS

- `int[][] myArray = new int[3][5];`
- Will result in an array the same size as if we declared it as
- `int[][] myArray = {{8,1,2,2,9},{1,9,4,0,3}, {0,3,0,0,7}};`

  - `myArray[2][4]` holds the value 7
  - `myArray[1][0]` holds the value 1

  - Remember, row first, then column

INITIALIZING 2D ARRAYS

- Usually, the number of **rows** and **columns** will be stored in variables
- Sometimes you will want to fill an array with default values
- Sometimes you will want to search through the whole array for a particular value
- It is common to use two nested loops when filling or searching a two-dimensional array:

```
for (int i = 0; i < rows; i++)
   for (int j = 0; j < columns; j++)
      board[i][j] = " ";
   }
}
```

INITIALIZING 2D ARRAYS

Let's say rows = 3 and columns = 3. Then this happens:

```
board[0][0] = " ";
board[0][1] = " ";
board[0][2] = " ";
board[1][0] = " ";
board[1][1] = " ";
board[1][2] = " ";
board[2][0] = " ";
board[2][1] = " ";
board[2][2] = " ";
```

```
for (int i = 0; i < rows; i++)
   for (int j = 0; j < columns; j++)
      board[i][j] = " ";
   }
}
```

RANDOM NUMBERS

- `Math.random()` provides a random number between 0.0 and 1.0
- `System.out.println("Here's one random number: " + Math.random());`
- `System.out.println("Here's another random number: " + Math.random());`
- The random number that is generated is of type double. If you need an `int`, you have to cast it by putting `(int)` in front

RANDOM NUMBERS

TODO:
How to generate an random int between 50 and 60?

```
//how about an int in the range of 0 to 99?
int number = (int)(Math.random()*100.0);

//How about an int in the range 0 to 76?
int number2 = ((int)(Math.random()*77);
```

FILL AN ARRAY WITH RANDOM NUMBERS

```
int[] randArray = new int [100];
for(int i = 0; i < randArray.length; i++)
{
    randArray[i] = (int)(Math.random()*100.0);
}
//Loops through 100 times and fills it in!
```