What if we need 10 different ints for storage?

```java
int num1;
int num2;
int num3;
int num4;
int num5;
int num6;
int num7;
int num8;
int num9;
int num10;
...
```

What is an Array?

- It’s an easy way to declare lots of variables that all have the same type.
- E.g. declare an array of integers
  ```java
  int[] data = new int[5]; //total ints = 5
  ```
- When an array of particular primitive type is created, Java initializes the elements to the types default value. E.g. Array of ints – default value is zero
  ![Array of int default values](image)

What if we want to store lots of things...

- But we don’t want to declare a separate variable for each one?
- That’s what arrays are good for
### Array Elements and Indices

- **To initialize integer array**
  - data[0] = 6;
  - data[1] = 10;
  - data[2] = 12;

- The number within square brackets is called an **index**

- The valid **indices** are 0 thru (array length - 1)
  - 0: first element of the array
  - n-1: last element of the array

### An Array is an Object

<table>
<thead>
<tr>
<th>int[] data;</th>
<th>data is a reference variable whose type is int[], meaning “array of ints”. At this point its value is null.</th>
</tr>
</thead>
<tbody>
<tr>
<td>data = new int[5];</td>
<td>The new operator causes a chunk of memory big enough for 5 ints to be allocated on the heap. Here, data is a assigned a reference to the heap address.</td>
</tr>
<tr>
<td>data[0] = 6;</td>
<td>Initially, all five ints are 0. Here, three of them are assigned other values.</td>
</tr>
<tr>
<td>data[1] = 10;</td>
<td></td>
</tr>
<tr>
<td>data[2] = 12;</td>
<td></td>
</tr>
<tr>
<td>int[] info = {6, 10, 12, 0, 0};</td>
<td></td>
</tr>
<tr>
<td>int[] info = new int[]{6, 10, 12, 0, -1};</td>
<td></td>
</tr>
</tbody>
</table>

### Array of Primitives

- int[] data;

- data = new int[3];

- value 500 is indicating address where the array object is stored (heap address)

- data[0] = 5;
- data[1] = 10;

### Using Array Elements in Expressions

- An **element** of an array of ints can be used virtually anywhere an expression of type int is valid.
- Likewise for arrays of other types

- int[] data = new int[] {6, 10, 12, 0, 0};
- int x = data[0];
- data[3] = data[2];
- System.out.println(“data[0] is “ + data[0]);
Accessing an Array's Length

- `ArrayName.length` gives size of the array

```java
int[] data;
data = new int[5];  // data.length is 5
data[0] = 6;
data[1] = 10;
data[2] = 12;
```

//How to Sum the contents of an array
int result = 0;
for (int i = 0; i < data.length; i++) {
    result = result + data[i];
}

---

Passing arrays

- Methods can take an array as input
  
  ```java
  return-type methodName(int [] data) {...}
  ```

- This gives the function to access each element of the array
- It also gives it ability to change the array
  - Hence we say arrays are passed by reference unlike variables that are passed by values

---

Returning Array

- Similarly a method can also return an array
  ```java
  int [] methodName(..) {
      ...
      return arrayName;
  }
  ```

- Uses:
  - Allows any type of method to create an array
  - This may provide access to array that may be declared private

---

Complete the sum(..) method

```java
public class ArrayToolkit{
    /*
    * Takes an array of ints as an argument.
    * returns the sum of all the integers in the array.
    */
    public static int sum (int [] data) {
        int result = 0;
        for (int i = 0; i < data.length; i++) {
            result = result + data[i];
        }
        return arrayName;
    }
}
```

---

Welcome to DrJava
> int[] data = new int[] {6, 10, 12, 0, 0};
> ArrayToolkit.sum(data)
28
Array Out of Bounds Exceptions

```
public class ArrayToolkit{
    public static int sum(int[] data){
        int sum = 0;
        for (int i = 0; i <= data.length; i++){
            sum = sum + data[i];
        }
        return sum;
    }
}
```

```
> int[] data = new int[] {6, 10, 12, 0, 0};
> ArrayToolkit.sum(data)
ArrayIndexOutOfBoundsException
```

---

Declaring & Initializing Arrays of Primitive Type

```
int[] info1 = {2000, 100, 40, 60};
int[] info2 = new int[]{2000, 100, 40, 60};

char[] choices1 = {'p', 's', 'q'};
char[] choices2 = new char[]{p, 's', 'q'};

double[] temps1 = {75.6, 99.4, 86.7};
double[] temps2 = new double[]{75.6, 99.4, 86.7};
```

Note: The advantage of using the "new type[]" syntax is that it can be used in an assignment statement that is not a variable declaration statement.

---

Complete this method

```
public class ArrayTool{
    /* Returns true if all integers in the data array are positive, false otherwise. */
    public static boolean allPositive(int[] data){
    }
}
```

---

Finding Max

```
> int[] data = new int[] {6, 10, 12, 0, 0};
> ArrayTool.max(data)
12
```

Complete max method, that finds the maximum value in the array.
Assume that input passed is valid, no need for error checking.

//Dr Java Interactions Pane
> int[] data = new int[] {6, 10, 12, 0, 0};
> ArrayTool.max(data)
12
Finding Max

- Suppose you want to find the largest value in an array called `scores`:

```java
int largestScore = 0;
for (int i = 0; i < scores.length; i++) {
    if (scores[i] > largestScore) {
        largestScore = scores[i];
    }
}
```

- What is wrong with this approach?

Improved Solution

- To find the largest value in an array `scores` of (possibly negative) integers:

```java
int largestScore = scores[0];
for (int i = 1; i < scores.length; i++) {
    if (scores[i] > largestScore) {
        largestScore = scores[i];
    }
}
```

Finding Location of Max Value

- Suppose you want to find the location in which you find the largest value in an array `scores`:

```java
int largestScore = scores[0];
int index = 0;
for (int i = 1; i < scores.length; i++) {
    if (scores[i] > largestScore) {
        largestScore = scores[i];
        index = i;
    }
}
```

Error Checking

- `int[] a = new int[value]` or `Int[] a = b; // where b is of type int[]`
- If value is <=0 or b is null then we have run time errors
  - If value = 0
    - Creates an array object of size zero
    - But there are no elements
  - If value is negative, then NegativeArraySize Exception is raised
    - No check value before creating array of size value.
- Check before accessing array elements
  - if(a == null || a.length == 0)