Kaitlin Hoffmann

Office Hours:

SH 243 MR 11:00 - 12:30 PM via appointment https://calendly.com/hoffmank4/15min

Email: hoffmank4@newpaltz.edu

For TA Office Hours and Email – Please see syllabus

NEW! Supplemental Instruction: Sign up at my.newpaltz.edu

ARRAYS

COMPUTER SCIENCE I

OBJECTIVES

- Arrays
- Passing Primitives vs Passing References/Objects
- Min/Max Algorithm

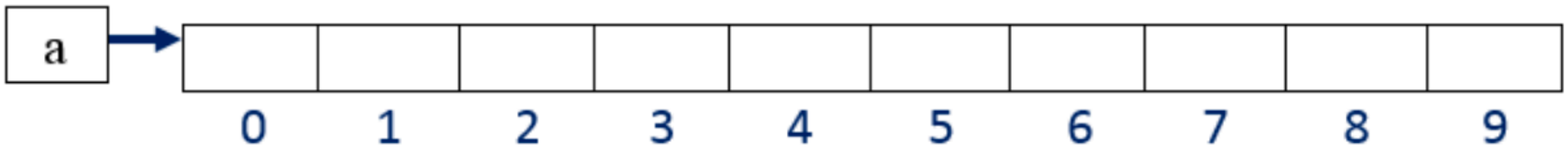
- Let’s suppose we want to have 10 int variables.
- **Question:** How could we do this in Java?

WHAT IS AN ARRAY?

- An **array** is a data structure in computer science that can hold multiple variables together.
- More specifically, an array is a data structure with a defined size that holds a collection of data each specified by an index.
- Arrays can hold ***any*** data type such as ints, doubles, characters, etc. and different types of objects (which we will get to later).

VISUALIZING AN ARRAY

Let's see how we can think of an array visually:



DECLARING AN ARRAY

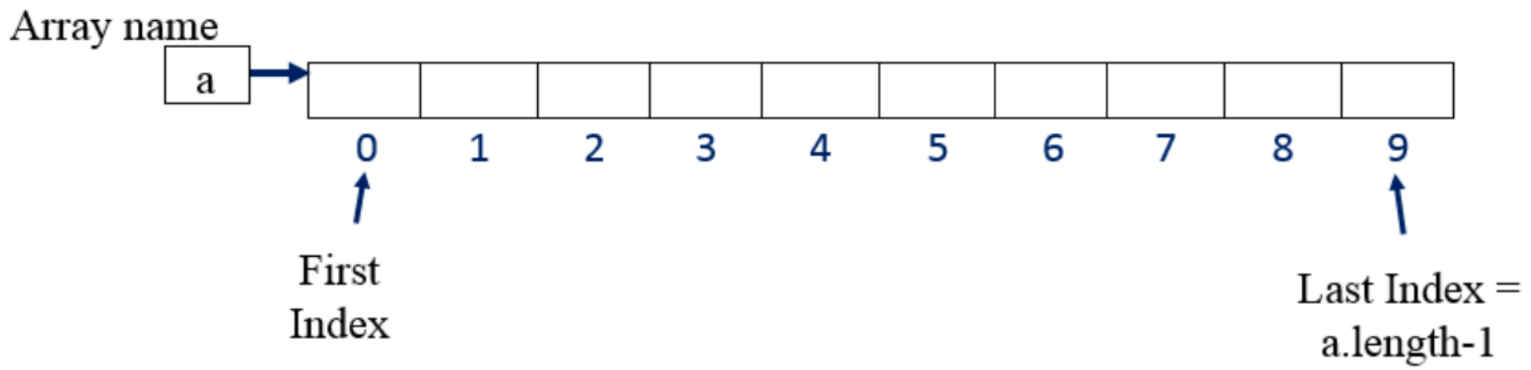
- The way we declare (create) an array in Java is:

  type[ ] arrayName = new type[size];

- **Ex.** If we want to make an array to hold 10 int values:

  int[ ] a = new int[10];

- The array has **10 slots** that can each hold an int value. Each slot is represented by an index. We ***cannot*** change the size once set. At the moment, it's empty:



INDICES OF AN ARRAY

- Every array starts with index **0**.
- Every array has a length that is represented by the *arrayName.length*

  In this example: a.length

  The value of a.length is 10

  int len = a.length;

  this line stores the value of the length in a variable
- Every array has the last index of *length-1*

  In this example: a.length-1 which is 9

STORING VALUES IN ARRAYS

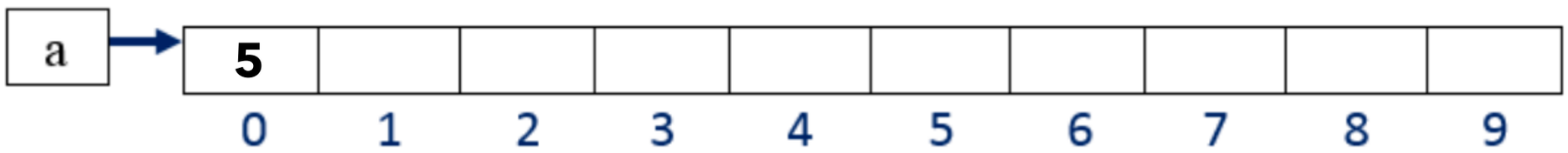
- *a[index]* is how we refer to a space or slot in an array
- if we want to refer to the **first slot**, we say **a[0]**
- if we want to refer to the **last slot**, we say **a[a.length-1]**
- the **second slot** is **a[1]**
- the **second to last** slot is **a[a.length-2]**

EXAMPLE

- **Question:** How can we assign **5** to the first slot?

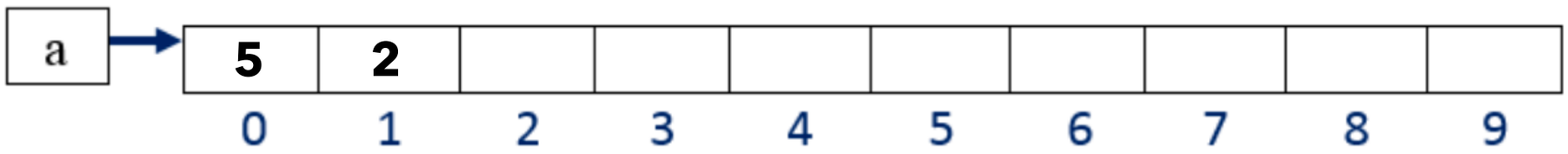
  a[0] = 5;

  This works very similar to int x=5;



EXAMPLE

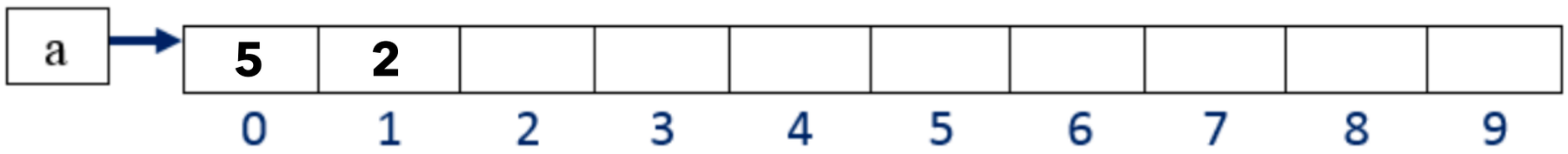
- **Question:** How can we assign **2** to the second slot?



EXAMPLE

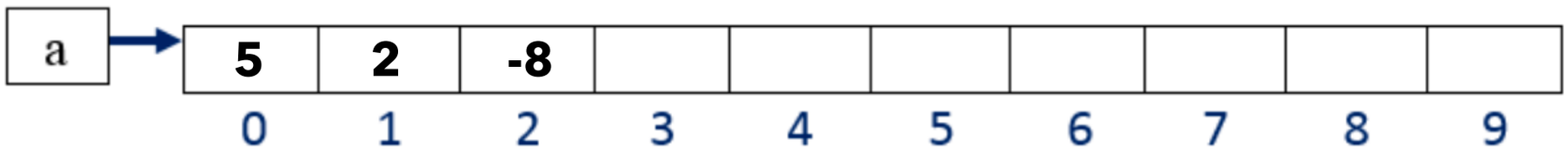
- **Question:** How can we assign **2** to the second slot?

```
a[1] = 2;
```



EXAMPLE

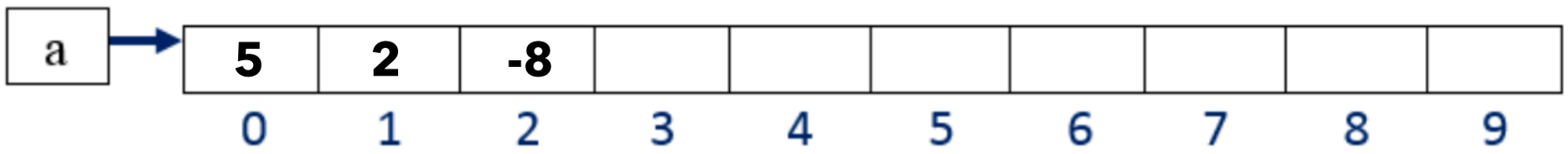
- **Question:** How can we assign **-8** to the third slot?



EXAMPLE

- **Question:** How can we assign **-8** to the third slot?

```
a[2] = -8;
```



EXAMPLE DONE IN JAVA

```java
public class Main {

    public static void main(String[] args) {

        int[] a = new int[10];

        a[0] = 5;
        a[1] = 2;
        a[2] = -8;

        System.out.println(a[0]);
        System.out.println(a[1]);
        System.out.println(a[2]);
        System.out.println(a.length);

    }
}
```

Output

```
5
2
-8
10
```

STORING VALUES IN ARRAYS

- Instead of going line by line to insert a value into an array, can you guess what we could use to store values?

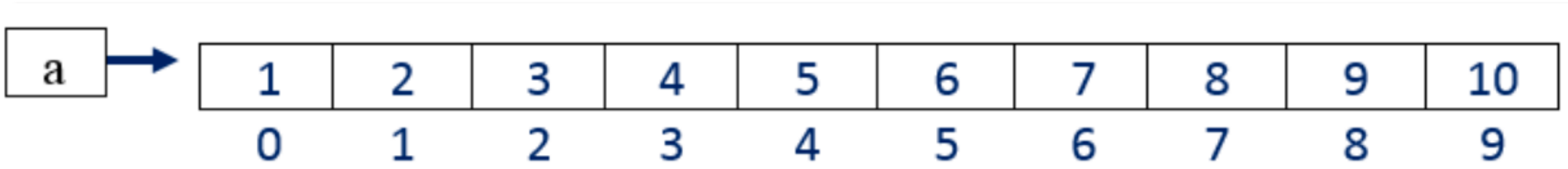
STORING VALUES IN ARRAYS

- Instead of going line by line to insert a value into an array, can you guess what we could use to store values? **A loop!**
- A loop allows us to visit each index of an array one after the other.

```
/* We make sure i is less than a.length since
   the length is always 1 more longer than
   the last index */
for(int i = 0; i < a.length; i++) {

}

//or

for(int i = 0; i <= a.length-1; i++) {

}
```

EXAMPLE 1

- Let's code the array below. We'll declare an int array with a size of 10, then initialize it with the following values:



EXAMPLE 1

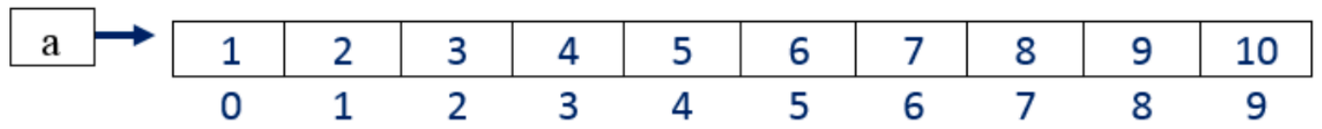
- Let's code the array below:

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

for(int i = 0; i<a.length; i++) {
    a[i] = i+1;
    System.out.print(a[i] + " ");
}

System.out.println();
//OR
int j = 1;
for(int i = 0; i<a.length; i++) {
    a[i] = j;
    System.out.print(a[i] + " ");
    j++;
}
```

Output

```
1 2 3 4 5 6 7 8 9 10
1 2 3 4 5 6 7 8 9 10
```

PRINTING ARRAYS



- Notice I **didn't** print out the array like:

  System.out.println(a);

- If I were to do that, we would get a crazy looking result like:

  [I@3796751b

- This is the **memory address** of where the array is stored.
- Instead, we must print out the contents of an array using a loop like shown in the previous example.

PRIMITIVE VS REFERENCE VARIABLES

- Why is the memory address printed?

  The array variable is actually a **reference variable** that points to the address of the first index.

- Variables in Java are classified as either **primitive** or **reference** variables.
  - The primitive data types (int, double, char, etc.) are primitive variables.
- A primitive variable's information is stored as the ***value*** of that variable, whereas a reference variable holds a ***reference to information*** related to that variable (in this case, the memory address).
  - Reference variables are almost always objects in Java.

PRIMITIVE VS REFERENCE VARIABLES

- The method call **System.out.println** prints the value of the variable.
- The value of a primitive variable is **concrete**, whereas the value of a reference variable is a **reference**.
- When we attempt to print the value of a reference variable, such as an array, the output contains the ***type*** of the variable and an ***identifier*** created for it by Java.
- When we print out the array, a, from before, we get:

  [I@3796751b

- This tells us that the given variable is of type array Integer, and its identifier is 3796751b.

RETRIEVING VS STORING – DON'T GET CONFUSED!

- **Question:** How can we store the element **x** at index 3 in an array?

```
a[3] = x;
```

- **Question:** How can we store the element at index 3 in a variable x?

```
int x = a[3];
```

- When the a[i] is on the ***left side*** of a statement, it means **assign** a value to index i of the array.
- When the a[i] is on the ***right side*** of a statement, it means **get** the array element at index i.

ANOTHER WAY TO DECLARE AN ARRAY

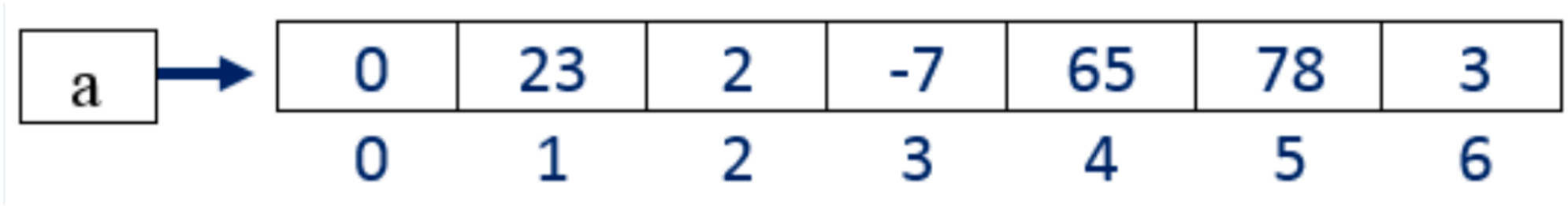
```
//declare the array with a type:
int[] a;

//assign a fixed size for the array:
a = new int[10];
```

ANOTHER WAY TO INITIALIZE AN ARRAY

```
//if you already know the values you want to place in an array:
int[] a = {1, 2, 3, 4, 5, 6, 7, 8, 9};
```

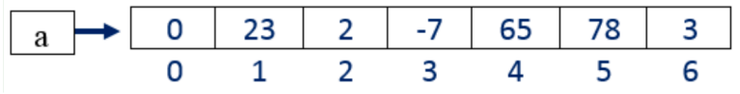
EXAMPLE 2

- Let's code the array below. How could we increase the value of each element of the array by 1?
- Let's create a method that will print our array.



EXAMPLE 2

- Let's code the array below. How could we increase the value of each element of the array by 1?
- Let's create a method that will print our array.



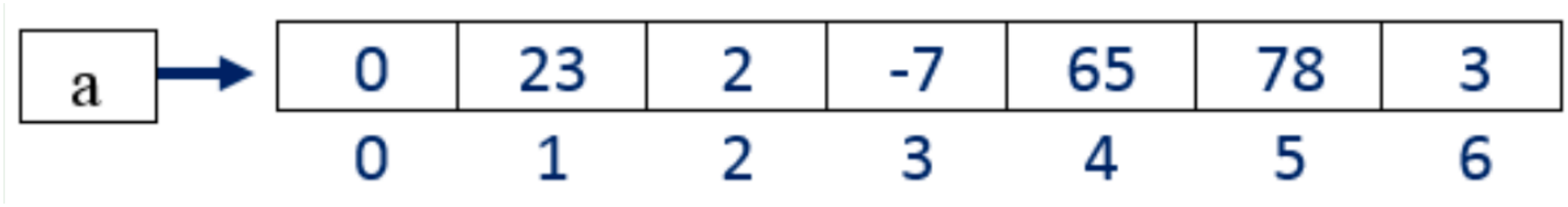

```
public class Main {

    public static void main(String[] args) {

        int[] a = {0, 23, 2, -7, 65, 78, 3};

        for(int i = 0; i<a.length; i++) {
            a[i]+=1;
        }

        printArray(a);
    }
    public static void printArray(int[] array) {
        for(int i = 0; i<array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
}
```

Output

```
1 24 3 -6 66 79 4
```

EXAMPLE 2

- How could we take the previous example and place it in a method?
- This is where **void** comes into play when changing values of an array (or all references/objects)



PASSING PRIMITIVES AND REFERENCES IN METHODS

- When we use java primitive types in our parameters, we get a temporary variable that has **the scope of the method only**.
  - This is why we must return the new value of our parameter if we are modifying it and re-assign it:

```
public class Main {
    public static void main(String[] args) {

        int num = 5;
        num = changeValue(num);
        System.out.println("Num = " + num);

    }
    public static int changeValue(int x) {
        return x+2;
    }
}
```

Output

```
Num = 7
```

PASSING PRIMITIVES AND REFERENCES IN METHODS

- When we pass an array we are actually passing the **memory address**(reference variable) of where the array is.
  - When we change the array, the value is actually being changed since we are accessing the memory address of the array.
  - So when we modify an array in a method, we don't need to return the array.
- The only time we need to return an array is when we create a new array inside the method.
- Let's see this from example 2:

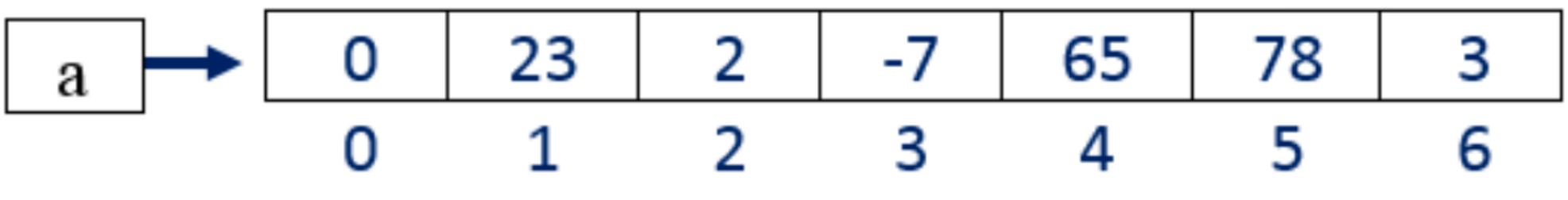
```java
public class Main {
    public static void main(String[] args) {

        int[] a = {0, 23, 2, -7, 65, 78, 3};
        System.out.print("Original Array = ");
        printArray(a);

        increaseByOne(a);
        System.out.print("\nIncreased by 1 = ");
        printArray(a);

    }
    //increase by 1
    public static void increaseByOne(int[] a) {
        for(int i = 0; i<a.length; i++) {
            a[i]++;
        }
    }
    //print array
    public static void printArray(int[] a) {
        for(int i = 0; i<a.length; i++) {
            System.out.print(a[i] + " ");
        }
    }
}
```

Output

```
Original Array = 0 23 2 -7 65 78 3
Increased by 1 = 1 24 3 -6 66 79 4
```

EXAMPLE 3

- Let’s sum the values of the array from example 2.



- Let's sum the values of the array from example 2.
- Let's then put this in a method.

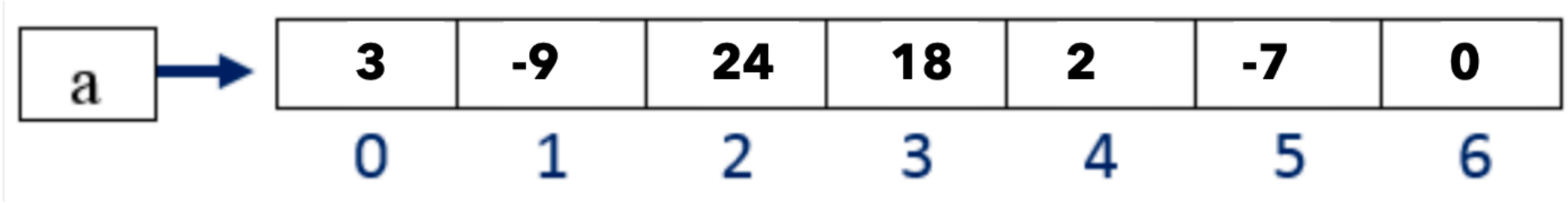
```java
public class Main {
    public static void main(String[] args) {

        int[] a = {0, 23, 2, -7, 65, 78, 3};
        increaseByOne(a);

        int sum = sumArray(a);
        System.out.println("Sum = " + sum);

    }
    //sum values from array
    public static int sumArray(int[] a) {
        int sum = 0;
        for(int i = 0; i<a.length; i++) {
            sum+=a[i];
        }
        return sum;
    }
    //increase by 1
    public static void increaseByOne(int[] a) {
        for(int i = 0; i<a.length; i++) {
            a[i]++;
        }
    }
    //print array
    public static void printArray(int[] a) {
        for(int i = 0; i<a.length; i++) {
            System.out.print(a[i] + " ");
        }
    }
}
```

Output

```
Sum = 171
```

EXERCISE 1

- Initialize the array below and print the array. Then try printing it backwards. Try using methods.



EXERCISE 1

- Initialize the array below and print the array. Then try printing it backwards. Try using methods.

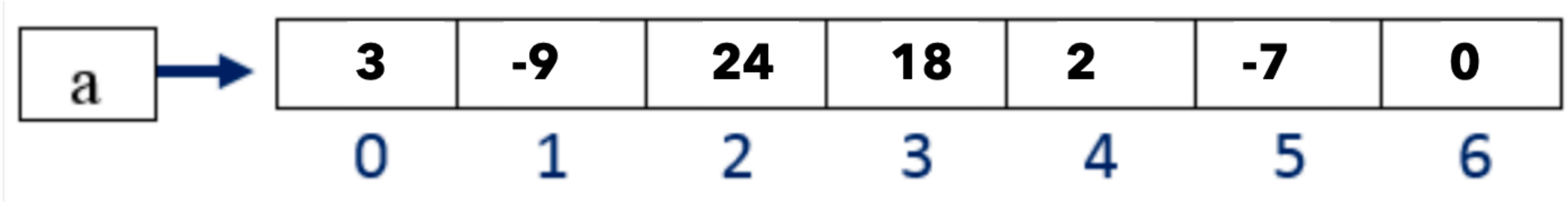
```
public class Main {

    public static void main(String[] args) {

        int[] a = {3, -9, 24, 18, 2, -7, 0};
        printArray(a);
        System.out.println();
        printBackwards(a);

    }
    public static void printArray(int[] array) {
        for(int i = 0; i<array.length; i++) {
            System.out.print(array[i] + " ");
        }
    }
    //print array backwards
    public static void printBackwards(int[] array) {
        for(int i = array.length-1; i>=0; i--) {
            System.out.print(array[i] + " ");
        }
    }
}
```

Output

```
3 -9 24 18 2 -7 0
0 -7 2 18 24 -9 3
```

EXAMPLE 4

- Let's find the average of the values in the array



EXAMPLE 4

- Let’s find the average of the values in the array

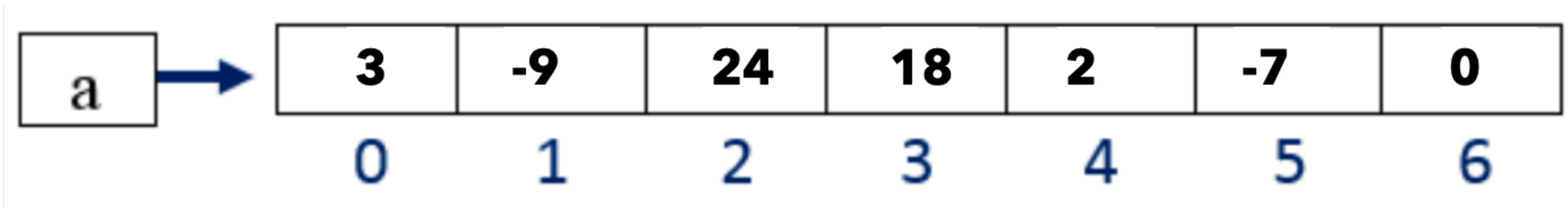
```
public class Main {
    public static void main(String[] args) {

        int[] a = {3, -9, 24, 18, 2, -7, 0};
        double avg = getAverage(a);
        System.out.println("Average = " + avg);

    }
    //method to get average of array
    public static double getAverage(int[] a) {
        int sum = getSum(a);
        return (double)sum/a.length;
    }
    //method to get sum of array
    public static int getSum(int[] a){
        int sum = 0;
        for(int i = 0; i< a.length; i++) {
            sum+=a[i];
        }
        return sum;
    }
}
```

Output

```
Average = 4.4285714285714295
```

EXERCISE 2

- Create a method that counts how many values are greater than the average value of an array.
- Whether on paper or computer, you can use the method from the previous exercise to get the average. The method was: ***getAverage(int[] a)***



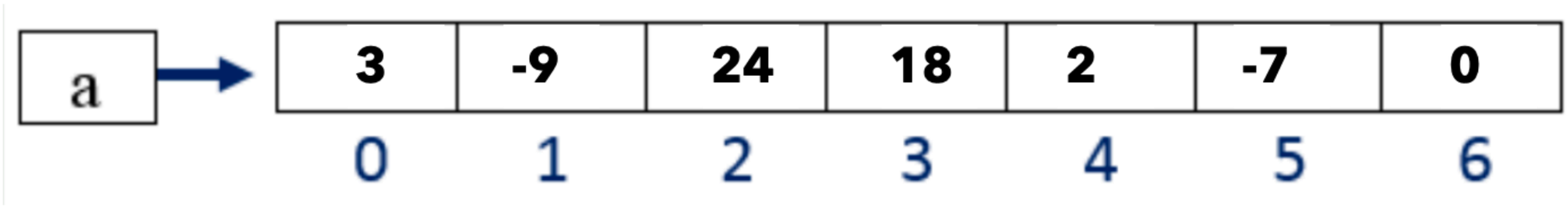
```java
public class Main {
    public static void main(String[] args) {

        int[] a = {3, -9, 24, 18, 2, -7, 0};
        int count = getCount(a);
        System.out.println("Count = " + count);


    }
    //method to count values that are greater than average
    public static int getCount(int[] a) {
        double avg = getAverage(a);
        int count = 0;
        for(int i = 0; i<a.length; i++) {
            if(a[i]>avg) {
                count++;
            }
        }
        return count;
    }
    //method to get average of array
    public static double getAverage(int[] a) {
        int sum = getSum(a); //calling our method from below
        return (double)sum / a.length;
    }
    //method to get sum of array
    public static int getSum(int[] a) {
        int sum = 0;
        for(int i = 0; i < a.length; i++) {
            sum+=i;
        }
        return sum;
    }
}
```

EXERCISE 2

- Create a method that counts how many values are greater than the average value of an array.
- Whether on paper or computer, you can use the method from the previous exercise to get the average. The method was: ***getAverage(int[] a)***

EXAMPLE 5

- Let's make an array of all the values from another array that are divisible by 2.



EXAMPLE 5

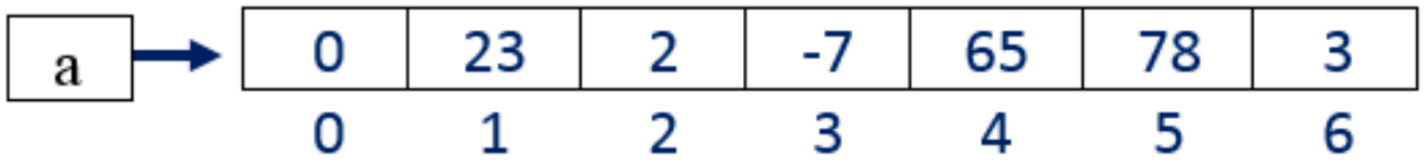

```java
public class Main {
    public static void main(String[] args) {

        int[] a = {3, -9, 24, 18, 2, -7, 0};
        int[] b = onlyEvensArray(a);

        System.out.print("Original Array = ");
        printArray(a);

        System.out.print("\nNew Array = ");
        printArray(b);

    }
    public static int[] onlyEvensArray(int[] a){
        int count = 0;
        for(int i = 0; i<a.length; i++) {
            if(a[i] % 2 == 0) {
                count++;
            }
        }
        int[] evensArray = new int[count];
        int j = 0;
        for(int i = 0; i<a.length; i++) {
            if(a[i] % 2 == 0) {
                evensArray[j] = a[i];
                j++;
            }
        }
        return evensArray;
    }
    //print array method
    public static void printArray(int[] a) {
        for(int i = 0; i<a.length; i++) {
            System.out.print(a[i] + " ");
        }
    }
```

Output

```
Original Array = 3 -9 24 18 2 -7 0
New Array = 24 18 2 0
```

FINDING THE MINIMUM VALUE IN AN ARRAY

- **Input:** An array



- **Question:** What is the smallest value?

Algorithm: Finding the Minimum

1. Assume the first value is the smallest.
2. Go through the rest of the array and check if the assumption is correct.
3. If the current value is $<$ the assumed smallest then update the smallest value.

EXAMPLE FINDING THE SMALLEST VALUE

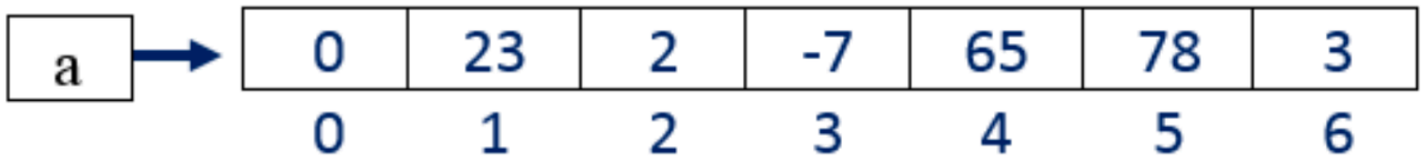

```java
public class Main {
    public static void main(String[] args) {

        int[] a = {0, 23, 2, -7, 65, 78, 3};
        int minValue = findSmallest(a);
        System.out.println("Smallest value = " + minValue);

    }

    //find the smallest integer in array
    public static int findSmallest(int[] a) {
        //1. assume the first index is smallest
        int min = a[0];
        //2. go through rest of array to check
        for (int i = 1; i < a.length; i++) {
            //3. if current value is < than current min, update min
            if (a[i] < min) {
                min = a[i];
            }
        }
        return min;
    }
}
```

Output

```
Smallest value = -7
```

FINDING WHERE THE MINIMUM IS LOCATED

- **Input:** An array



- **Question:** What is the **index** of the smallest value?

Algorithm: Finding the index of the minimum

1. Assume the first index is the smallest value.
2. Go through the rest of the array and check if the assumption is correct.
3. If the current value is $<$ the assumed smallest then update the smallest index value.

EXAMPLE FINDING THE SMALLEST VALUE


```java
public class Main {
    public static void main(String[] args) {

        int[] a = {0, 23, 2, -7, 65, 78, 3};
        int minValue = findSmallestIndex(a);
        System.out.println("Index = " + minValue);

    }

    //find the index of the smallest integer in array
    public static int findSmallestIndex(int[] a) {
        //1. assume the first index is smallest
        int minIndex = 0;
        //2. go through rest of array to check
        for (int i = 1; i < a.length; i++) {
            //3. if current value is < than current a[minIndex], update minIndex
            if (a[i] < a[minIndex]) {
                minIndex = i;
            }
        }
        return minIndex;
    }
}
```

Output

```
Index = 3
```

FINDING WHERE THE MAXIMUM IS LOCATED

- **Input:** An array
- **Question:** What is **largest** value?
- Can be done using the same method and idea but checking if the values are > than the assumed values. You'll be doing this in lab!