CS1101 Programming and Problem Solving

Dr. Gina Bai

Spring 2023

Logistics

- **ZY-7A** on zyBook > Assignments
  - Due: **Wednesday, March 29**, at 11:59pm
- **PA09 – A, B** on zyBook > Chap 11
  - Due: **Thursday, March 30**, at 11:59pm
- **ZY-7B** and **ZY-8A**on zyBook > Assignments
  - Due: **Wednesday, April 5**, at 11:59pm

Start Early!!!

Logistics

- Midterm Exam 2
  - Grades are posted on Gradescope (with an email notification)
  - Regrade requests:
    - MUST be submitted **within TWO weeks** (by **April 11**)
    - Email your instructor in the format of:

      Question#X-Y: be very specific on subproblems

      Deduction: which deduction should be reconsidered

      Rationale: why do you believe the points should be given back

for–each Loop (Enhanced for Loop)

zyBook Chap 7.9

for-each Loop (Enhanced for Loop)

- Simplifies certain array loops
- Provides the ability to examine each element of an array
  - **ONLY** allows elements to be **accessed** forward **from** the **first** element **to** the **last** element

for Loop

```
for(int i = 0; i < <arrayName>.length; ++i) {
    <type> <varName> = <arrayName>[i];
    <statement(s) using varName>;
}
```

for-each Loop

```
for(<type> <varName> : <arrayName>) {
    <statement(s) using varName>;
}
```

Equivalent Implementations

```java
// Using for loop to print out each element in the array
for (int i = 0; i < arr.length; ++i) {
    int element = arr[i];
    System.out.println(element);
}

// Using for-each loop to print out each element in the array
for (int element : arr) {
    System.out.println(element);
}
```

Limitations

- Cannot be used to **modify** array

```
// Only changes num, not the array.
for (int num : numbers) {
    num = num * 2;
}

// Equivalent implementation
for (int i = 0; i < numbers.length; ++i) {
    int num = numbers[i] * 2;
}
```

Limitations

- Cannot be used to **access index**

```
// Example: Return the index of target value in the array
for (int num : numbers) {
    if (num == target) {
        return ???; // Cannot get the index
    }
}

for (int i = 0; i < numbers.length; ++i) {
    int num = numbers[i];
    if (num == target) {
        return i; // Can get the index
    }
}
```

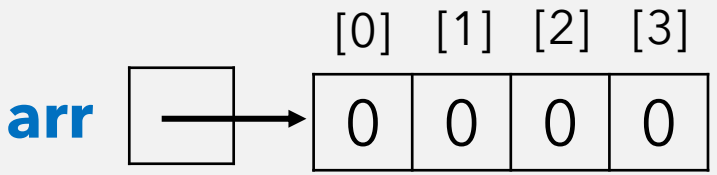
Passing Arrays as Parameters

Recap – Passing Parameters

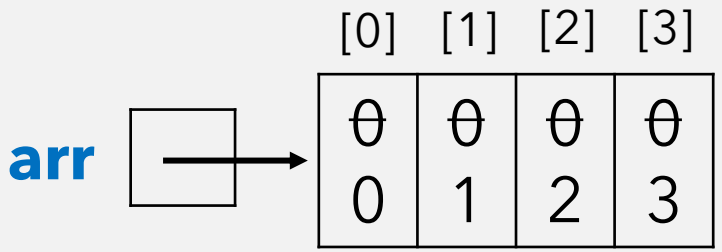
- When a **primitive type** is passed as a parameter, the **value** is copied

- When an **object** is passed as a parameter, the **reference** is copied
  - For example, Scanners, Strings, and Arrays.

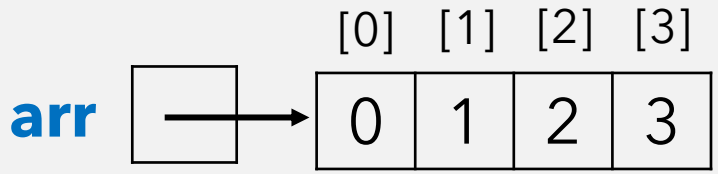
```java
import java.util.Arrays;

public class ArraysParameterDemo {
    public static void main(String []args) {
        int[] arr = new int[4];









}
```



```java
import java.util.Arrays;

public class ArraysParameterDemo {
    public static void main(String []args) {
        int[] arr = new int[4];

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









}
```

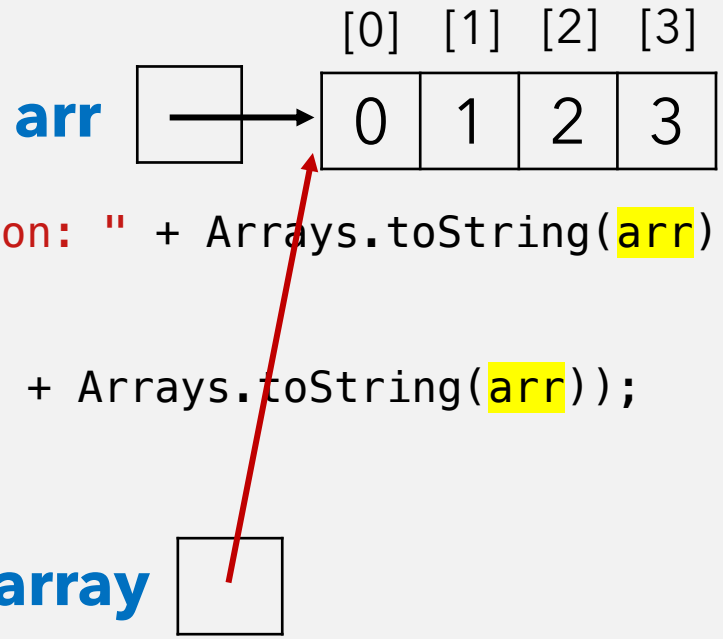


```java
import java.util.Arrays;

public class ArraysParameterDemo {
    public static void main(String []args) {
        int[] arr = new int[4];

        for (int i = 0; i < arr.length; ++i) {
            arr[i] = i;
        }
        System.out.println("Array after initialization: " + Arrays.toString(arr));

        incrementAll(arr);
        System.out.println("Array after increment: " + Arrays.toString(arr));
    }




}
```



```java
import java.util.Arrays;

public class ArraysParameterDemo {
    public static void main(String []args) {
        int[] arr = new int[4];

        for (int i = 0; i < arr.length; ++i) {
            arr[i] = i;
        }
        System.out.println("Array after initialization: " + Arrays.toString(arr));

        incrementAll(arr);
        System.out.println("Array after increment: " + Arrays.toString(arr));
    }

    public static void incrementAll(int[] array) {



    }
}
```

In method call, use the name arr only. NO [ ]

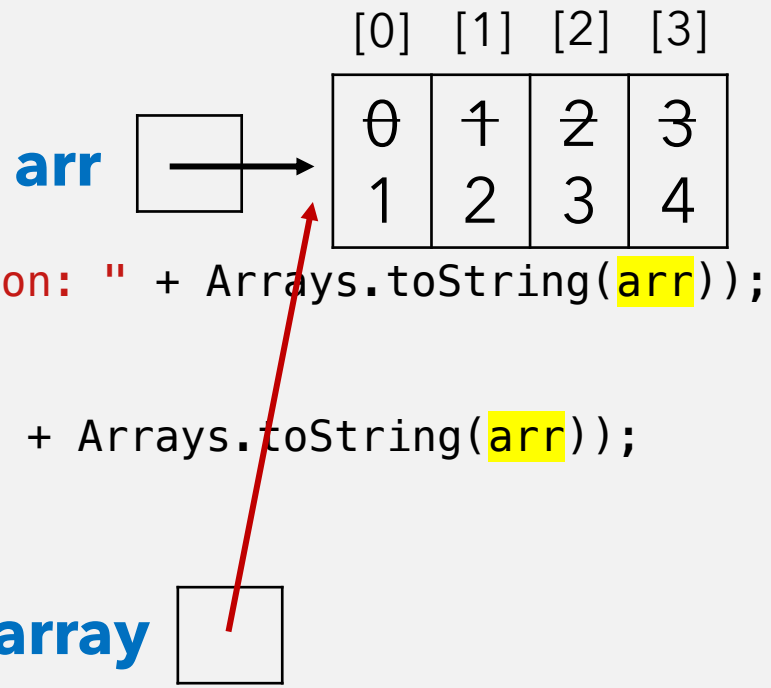


<type>[]

Name does not matter, reference matters.

```java
import java.util.Arrays;

public class ArraysParameterDemo {
    public static void main(String []args) {
        int[] arr = new int[4];

        for (int i = 0; i < arr.length; ++i) {
            arr[i] = i;
        }
        System.out.println("Array after initialization: " + Arrays.toString(arr));

        incrementAll(arr);
        System.out.println("Array after increment: " + Arrays.toString(arr));
    }

    public static void incrementAll(int[] array) {
        for (int i = 0; i < array.length; ++i) {
            array[i]++;
        }
    }
}
```

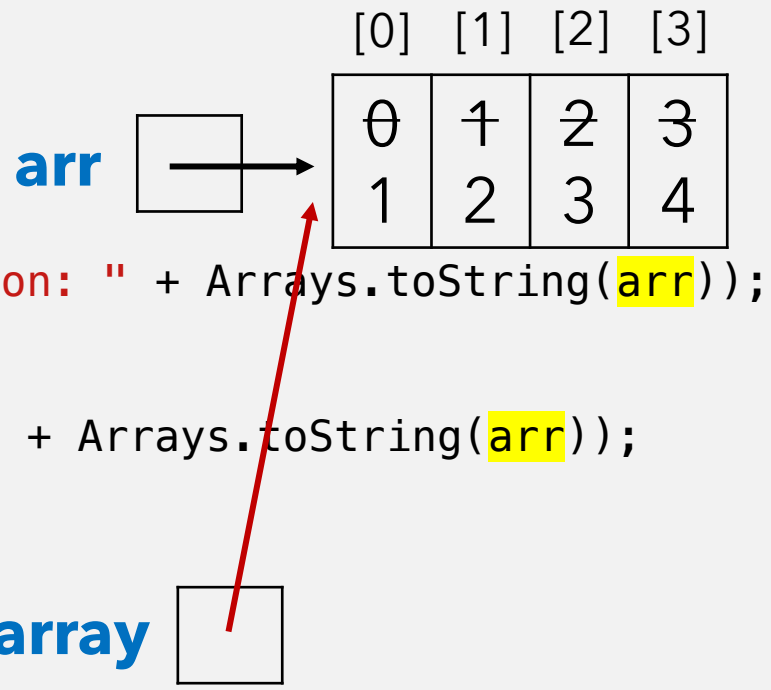
In method call, use the name arr only. NO [ ]



<type>[]

Name does not matter, reference matters.

```java
import java.util.Arrays;

public class ArraysParameterDemo {
    public static void main(String []args) {
        int[] arr = new int[4];

        for (int i = 0; i < arr.length; ++i) {
            arr[i] = i;
        }
        System.out.println("Array after initialization: " + Arrays.toString(arr));

        incrementAll(arr);
        System.out.println("Array after increment: " + Arrays.toString(arr));
    }

    public static void incrementAll(int[] array) {
        for (int i = 0; i < array.length; ++i) {
            array[i]++;
        }
    }
}
```




```
$ javac ArraysParameterDemo.java
$ java ArraysParameterDemo
Array after initialization: [0, 1, 2, 3]
Array after increment: [1, 2, 3, 4]
```

Array itself is modified

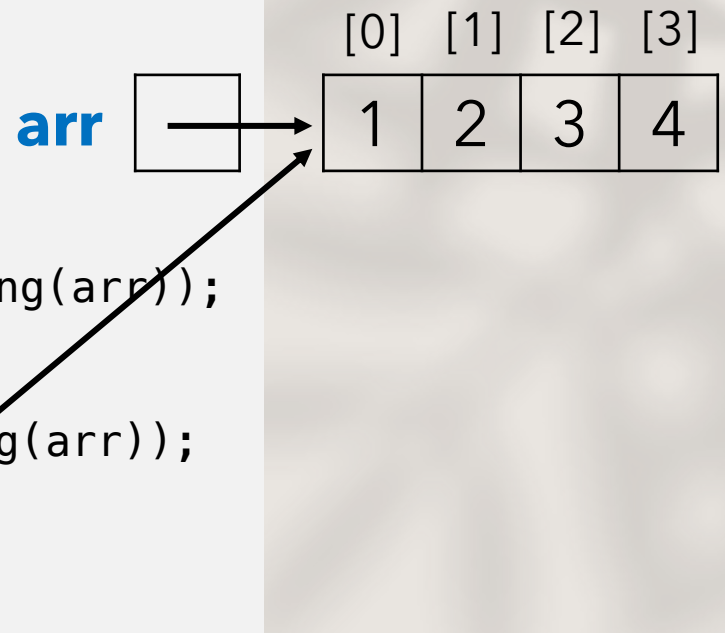
Returning Arrays

Returning Arrays

- The **return type** for a method can be an **array**.

- Returning an array typically occurs **when a new array is created within a method** rather than modifying an array parameter.

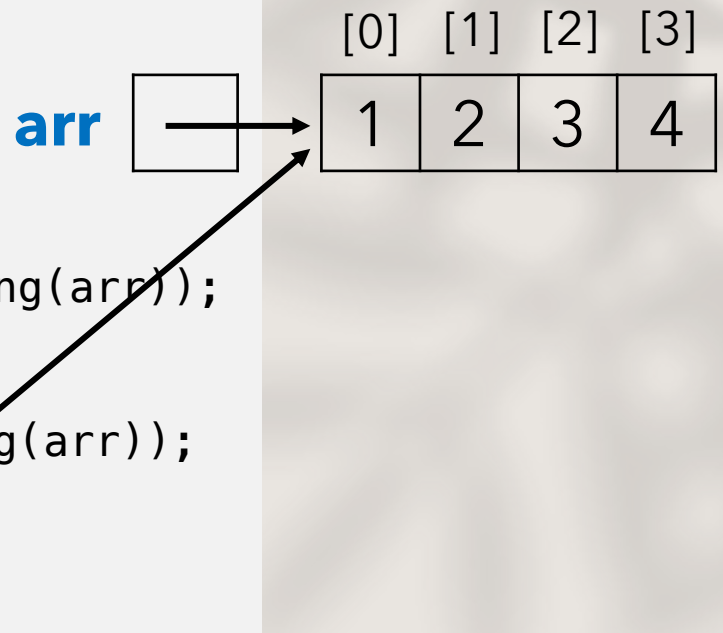
```java
import java.util.Arrays;

public class ExpandArray {
    public static void main (String[] args) {
        int[] arr = { 1, 2, 3, 4 };
        System.out.println("Before double size: " + Arrays.toString(arr));

        arr = doubleSize(arr);
        System.out.println("After double size: " + Arrays.toString(arr));
    }

    public static int[] doubleSize (int[] arr) {

    }
}
```



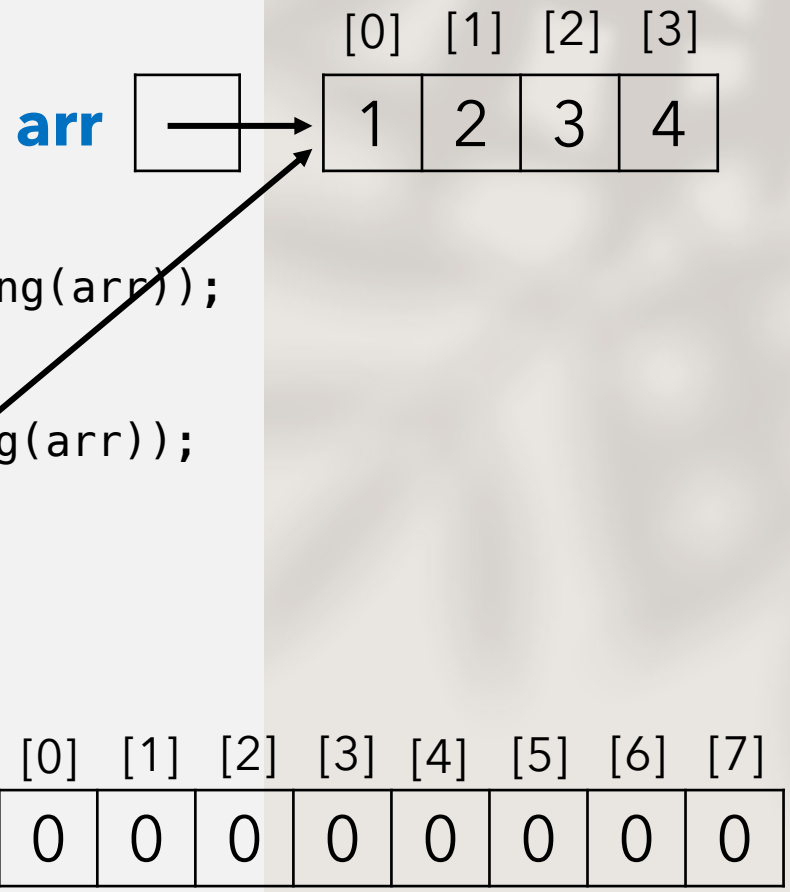
Array as parameter

```java
import java.util.Arrays;

public class ExpandArray {
    public static void main (String[] args) {
        int[] arr = { 1, 2, 3, 4 };
        System.out.println("Before double size: " + Arrays.toString(arr));

        arr = doubleSize(arr);
        System.out.println("After double size: " + Arrays.toString(arr));
    }

    public static int[] doubleSize (int[] arr) {

    }
}
```

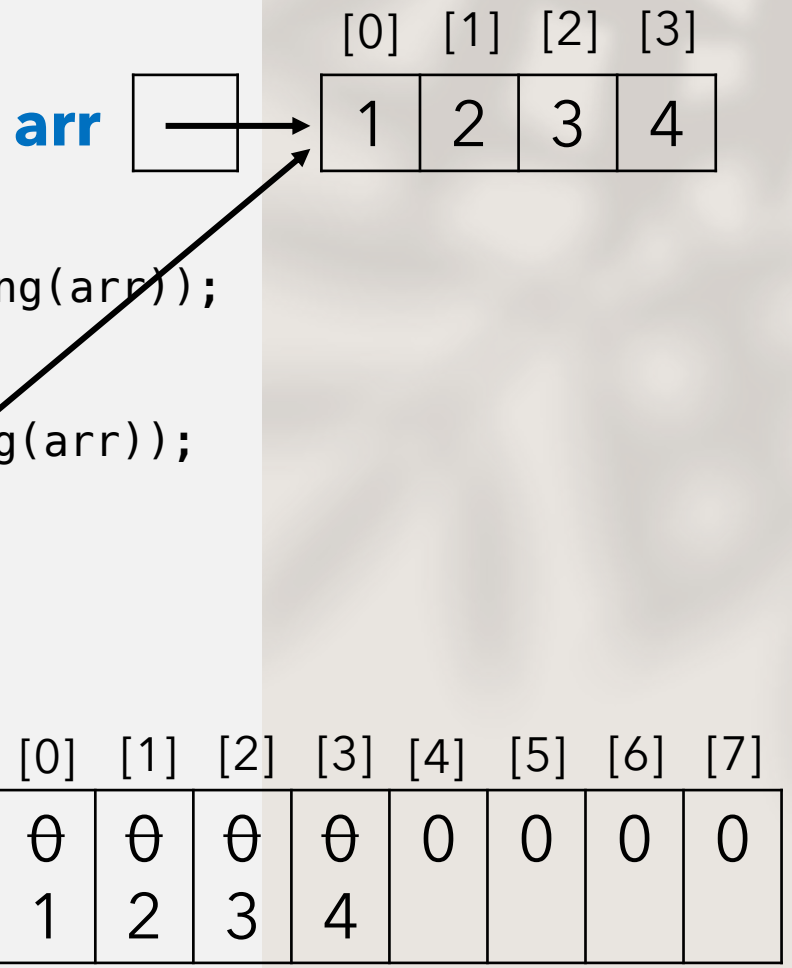
Return type



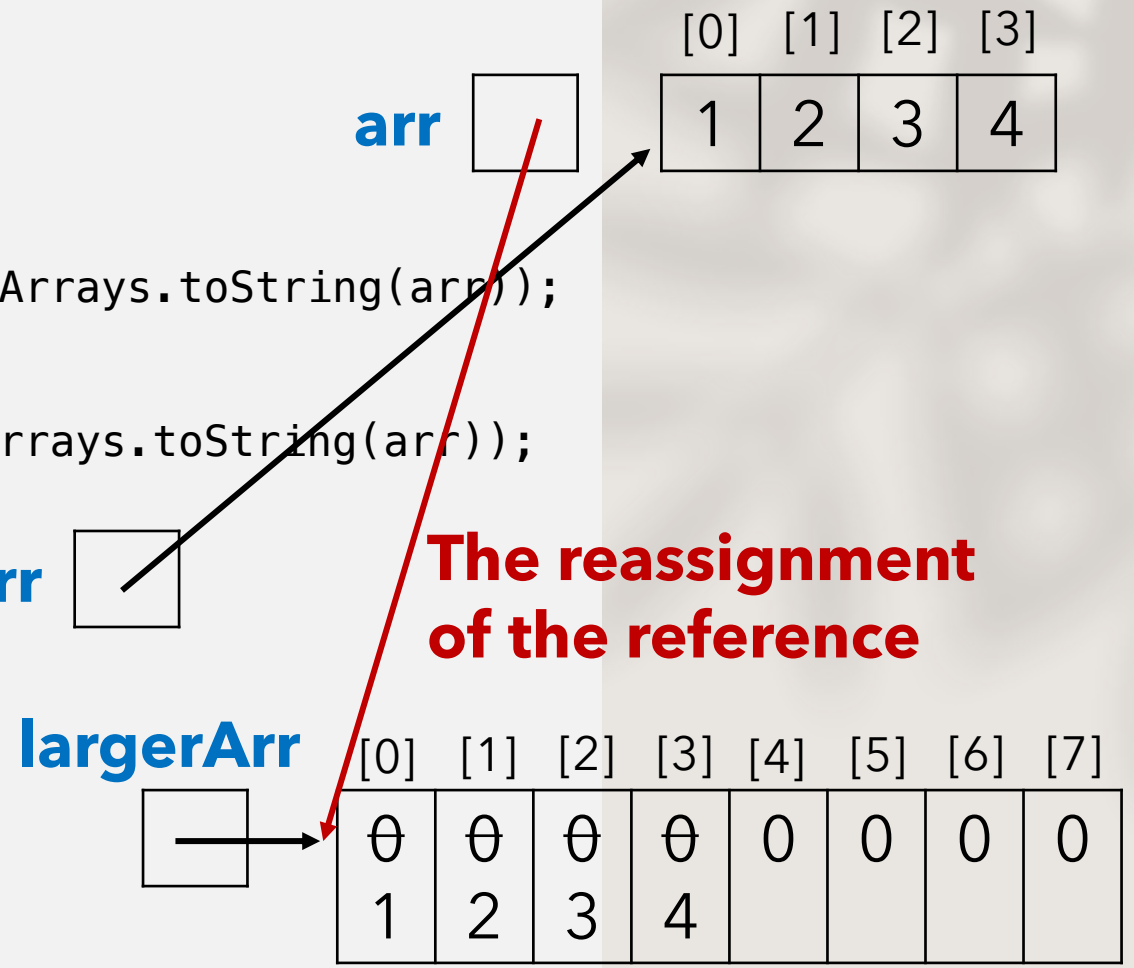
```java
import java.util.Arrays;

public class ExpandArray {
    public static void main (String[] args) {
        int[] arr = { 1, 2, 3, 4 };
        System.out.println("Before double size: " + Arrays.toString(arr));

        arr = doubleSize(arr);
        System.out.println("After double size: " + Arrays.toString(arr));
    }

    public static int[] doubleSize (int[] arr) {
        int[] largerArr = new int[arr.length * 2];









    }
}
```



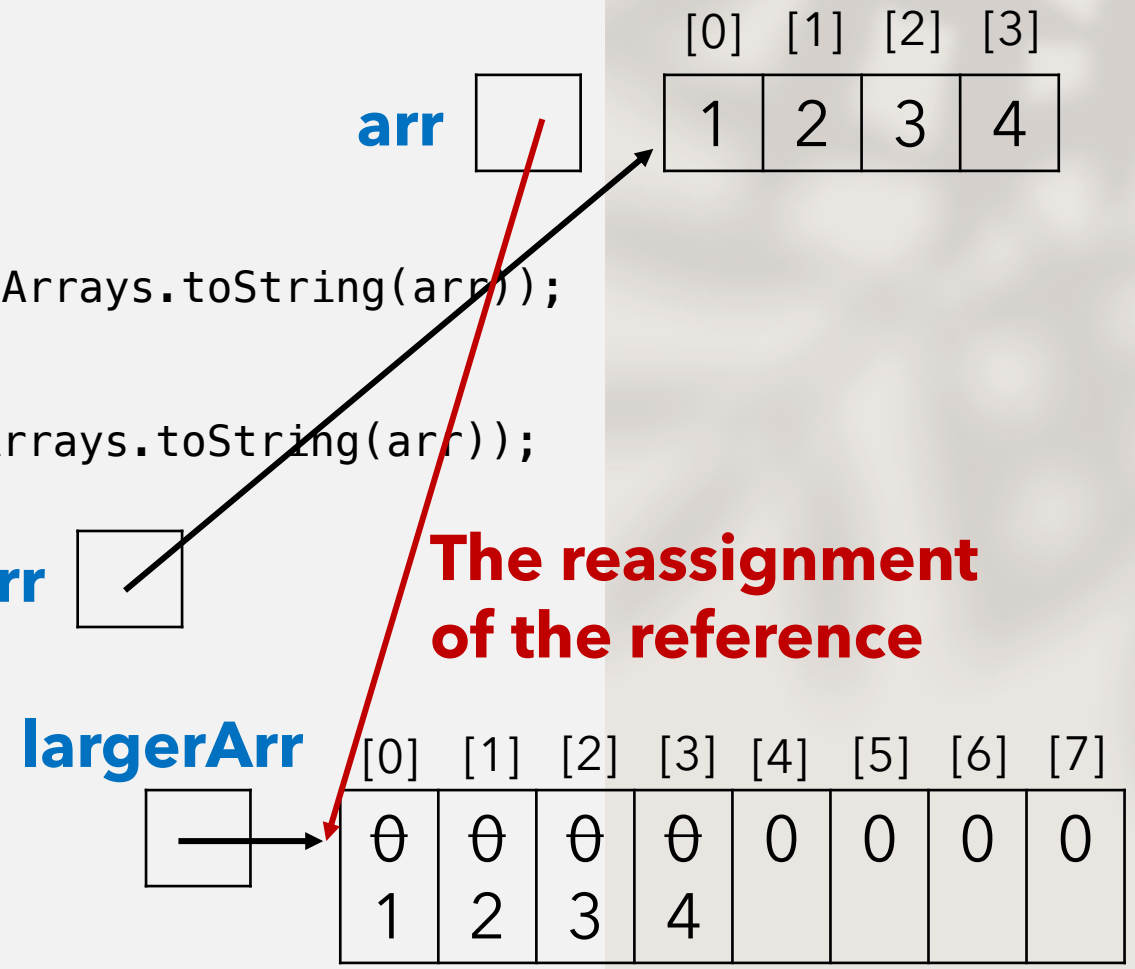
```java
import java.util.Arrays;

public class ExpandArray {
    public static void main (String[] args) {
        int[] arr = { 1, 2, 3, 4 };
        System.out.println("Before double size: " + Arrays.toString(arr));

        arr = doubleSize(arr);
        System.out.println("After double size: " + Arrays.toString(arr));
    }

    public static int[] doubleSize (int[] arr) {
        int[] largerArr = new int[arr.length * 2];

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


    }
}
```



```java
import java.util.Arrays;

public class ExpandArray {
    public static void main (String[] args) {
        int[] arr = { 1, 2, 3, 4 };
        System.out.println("Before double size: " + Arrays.toString(arr));

        arr = doubleSize(arr);
        System.out.println("After double size: " + Arrays.toString(arr));
    }

    public static int[] doubleSize (int[] arr) {
        int[] largerArr = new int[arr.length * 2];

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

        return largerArr;
    }
}
```



```java
import java.util.Arrays;

public class ExpandArray {
    public static void main (String[] args) {
        int[] arr = { 1, 2, 3, 4 };
        System.out.println("Before double size: " + Arrays.toString(arr));

        arr = doubleSize(arr);
        System.out.println("After double size: " + Arrays.toString(arr));
    }

    public static int[] doubleSize (int[] arr) {
        int[] largerArr = new int[arr.length * 2];

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

        return largerArr;
    }
}
```




```
$ javac ExpandArray.java
$ java ExpandArray
Before double size: [1, 2, 3, 4]
After double size: [1, 2, 3, 4, 0, 0, 0, 0]
```

TopHat Activity

Q: What does the following code segment print?

```
int[] a1 = {4, 5, 2, 12, 14, 14, 9};
int[] a2 = a1; // refer to same array as a1
a2[0] = 7;
System.out.println(a1[0]);
```

Answer: 7