Questions on arrays so far?
Copying Arrays
public class CopyingArrays {

    public static void main(String[] args) {

        int[] myArray1 = {1,2,3,4,5};
        int[] myArray2 = new int[5];

        System.out.println("Before copy, first element of first array is "+ myArray1[0]);
        System.out.println("Before copy, first element of second array is "+ myArray2[0]);

        myArray2 = myArray1;

        System.out.println("After copy, first element of first array is "+ myArray1[0]);
        System.out.println("After copy, first element of second array is "+ myArray2[0]);

        myArray2[0] = 6;

        System.out.println("After mod, first element of first array is "+ myArray1[0]);
        System.out.println("After mod, first element of second array is "+ myArray2[0]);
    }
}

UNEXPECTED!
Copying Arrays

• Array variables are actually *references* to an array.

• So if you use an assignment statement, you are just making both variables point to the same memory location.
Copying Arrays

Before the assignment:

```
list2 = list1;
```

list1 → Contents of list1
list2 → Contents of list2

After the assignment:

```
list2 = list1;
```

list1 → Contents of list1
list2 → Contents of list2
So how do we copy?

- For Loop, set each element to be the same
- Or use a helper method in the System class, `arraycopy`
- **Note:** it’s `arraycopy`, not `arrayCopy` like most of the methods we use!
public static void main(String[] args) {

    int[] source = {1,2,3,4,5};
    int[] target = new int[source.length];

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

    System.arraycopy(source, 0, target, 0, source.length);

    // Test it out
    System.out.println(target[4]);
}

Passing arrays to methods

• Remember that primitive values are passed by value, meaning the *value* of the variable is passed to a method.

• For arrays, remember, the *value* is a reference, so that *reference* is passed.

• This is referred to as pass-by-sharing.

• If you change the array in the method, you will see the change outside the method!
Example passing an array to a method

```java
public class PassBySharing {

    public static void main(String[] args) {
        int[] nums = {1,2,3,4,5};

        System.out.println("The first element before running the method is: " + nums[0]);
        addOne(nums);
        System.out.println("The first element after running the method is: " + nums[0]);
    }

    public static void addOne(int[] numbers){
        for (int i = 0; i < numbers.length; i++){
            numbers[i] = numbers[i] + 1;
        }
    }
}
```
Variable-length argument list

- Shortcut if you’re going to pass a bunch of values of the same type, you can use the syntax:
  - typename...parameterName

- And those values will be treated as an array.

- Note: only one variable-length parameter is allowed, and it must be the last parameter
public class VariableLengthParam {

    public static void main(String[] args) {
        int[] nums = {1,2,3,4,5};

        addOne(1,2,3,4,5);
    }

    public static void addOne(int...numbers){
        for (int i = 0; i < numbers.length; i++){
            numbers[i] = numbers[i] + 1;
        }

        System.out.println(numbers[4]);
    }

}
Searching arrays

- Linear vs. binary
Searching arrays

- Linear

```java
/** Find key in array, going from start to end */
public static int linearSearch(int[] numbers, int key){

    for (int i = 0; i < numbers.length; i++){
        if (numbers[i] == key){
            return i;
        }
    }

    return -1;
}
```
Searching arrays

• For binary search, the array must first be sorted

• Cuts down search time significantly
Searching arrays

<table>
<thead>
<tr>
<th>key is 11</th>
<th>low</th>
<th>mid</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>key &lt; 50</td>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
</tr>
<tr>
<td></td>
<td>[6]</td>
<td>[7]</td>
<td>[8]</td>
</tr>
<tr>
<td></td>
<td>[9]</td>
<td>[10]</td>
<td>[11]</td>
</tr>
<tr>
<td>list</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11</td>
<td>45</td>
</tr>
<tr>
<td></td>
<td>50</td>
<td>59</td>
<td>60</td>
</tr>
<tr>
<td></td>
<td>66</td>
<td>69</td>
<td>70</td>
</tr>
<tr>
<td></td>
<td>79</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key &gt; 7</th>
<th>low</th>
<th>mid</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>[0]</td>
<td>[1]</td>
<td>[2]</td>
</tr>
<tr>
<td>list</td>
<td>2</td>
<td>4</td>
<td>7</td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11</td>
<td>45</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>key == 11</th>
<th>low</th>
<th>mid</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>list</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>10</td>
<td>11</td>
<td>45</td>
</tr>
</tbody>
</table>
Searching arrays

/** Find key in array, using binary search */
public static int binarySearch(int[] numbers, int key) {
    int low = 0;
    int high = numbers.length - 1;

    while (high >= low) {
        int mid = (low + high) / 2;
        if (key < numbers[mid]) {
            high = mid - 1;
        } else if (key == numbers[mid]) {
            return mid;
        } else {
            low = mid + 1;
        }
    }
    return -1;
}
Sorting arrays

- Selection sort

Select 1 (the smallest) and swap it with 2 (the first) in the list.

`2 9 5 4 8 1 6`

swap

The number 1 is now in the correct position and thus no longer needs to be considered.

`1 9 5 4 8 2 6`

swap

The number 2 is now in the correct position and thus no longer needs to be considered.

`1 2 5 4 8 9 6`

Select 2 (the smallest) and swap it with 9 (the first) in the remaining list.

The number 4 is now in the correct position and thus no longer needs to be considered.

`1 2 4 5 8 9 6`

The number 5 is now in the correct position and thus no longer needs to be considered.

`1 2 4 5 8 9 6`

Select 4 (the smallest) and swap it with 5 (the first) in the remaining list.

The number 6 is now in the correct position and thus no longer needs to be considered.

`1 2 4 5 6 9 8`

Select 6 (the smallest) and swap it with 8 (the first) in the remaining list.

The number 8 is now in the correct position and thus no longer needs to be considered.

`1 2 4 5 6 8 9`

Select 8 (the smallest) and swap it with 9 (the first) in the remaining list.

Since there is only one element remaining in the list, the sort is completed.
public class SelectionSort {

    public static void main(String[] args) {
        int[] numbers = {1,2,3,5,4,4,6};
        selectionSort(numbers); 

        for (int i = 0; i < numbers.length; i++) {
            System.out.println(numbers[i]);
        }
    }

    /** Selection sort */
    public static void selectionSort(int[] numbers) {

        for (int i = 0; i < numbers.length - 1; i++) {
            int currentLow = numbers[i];
            int currentLowIndex = i;

            for (int j = i + 1; j < numbers.length; j++) {
                if (numbers[j] < currentLow) {
                    currentLow = numbers[j];
                    currentLowIndex = j;
                }
            }

            if (currentLowIndex != i) {
                numbers[currentLowIndex] = numbers[i];
                numbers[i] = currentLow;
            }
        }
    }
}
java.util.arrays!

- Lots of goodies!
- Methods useful for sorting, searching, comparing, filling, and returning strings of arrays
java.util.Arrays.sort(numbers);

java.util.Arrays.binarySearch(numbers, 11));

boolean isEqual = java.util.Arrays.equals(list1, list2);

java.util.Arrays.fill(list1, 5);

System.out.println(java.util.Arrays.toString(numbers));