Lecture 08

Single-Dimensional Arrays … Part II
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
\[ \text{list2} = \text{list1}; \]

\[ \text{list1} \rightarrow \text{Contents of list1} \]

\[ \text{list2} \rightarrow \text{Contents of list2} \]

After the assignment
\[ \text{list2} = \text{list1}; \]

\[ \text{list1} \rightarrow \text{Contents of list1} \]

\[ \text{list2} \rightarrow \text{Contents of list2} \]
So how do we copy?

- For Loop, set each element to be the same
- Or use a helper method in the System class, `arraycopy`
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

key is 11
- low: [0] [1] [2] [3] [4] [5]
- mid: 50

key < 50
- low: [0] [1] [2] [3] [4] [5]
- mid: 50

key > 7
- low: [0] [1] [2] [3] [4] [5]
- mid: 10

key == 11
- mid: 10

list:
- 2 4 7 10 11 45 50 59 60 66 69 70 79
Sorting arrays

- **Selection sort**

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

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

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

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

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

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

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

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

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

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

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++) {

            // Set up our best guess for the lowest number...i
            int currentLow = numbers[i];
            int currentLowIndex = i;

            // loop through the rest of the numbers and find the lowest
            for (int j = i + 1; j < numbers.length; j++) {
                if (numbers[j] < currentLow) {
                    currentLow = numbers[j];
                    currentLowIndex = j;
                }
            }

            // if the lowest number we found is not the element we're on
            // swap it
            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));