Arrays

An array is a list of values. In Java, the components of an array can be of any type, basic or object. An array is a special kind of object with special language support.

```java
int[] nums = new int[3] ;
```

Arrays have special constructor: “new int[3]” creates a 3-element array of int. You can access the elements of the array using the “[ ]” operator, starting with element 0.

```java
nums[0] = 12 ;
nums[1] = 701 ;
nums[2] = 5 ;
```

Arrays in Memory

The declaration of an n-element array sets aside n adjacent slots in memory.

Accessing Arrays

The position of an element in an array is called its array index.

- The index of the first element in an array is 0.
- The index of the i-th element in an array is i - 1.
- The index of the last element in an n-element array is n - 1.

The number of elements in an array is available in a public instance field named length. Bounds-checking is done on array indices at run time.

```java
public void printInts(int[] nums) {
    for( int i=0 ; i < nums.length ; i++ )
        System.out.println(nums[i]) ;
}
```

```java
public void printLastX(int[] nums) {
    System.out.println( nums[nums.length] ) ; /* Error: nums.length exceeds array bounds */
}
```

```java
public void printLast(int[] nums) {
    System.out.println( nums[nums.length-1] ) ;
}
```

Array Declarations

int[] is a type “array of int” distinct from int.

```java
int[] a, b, c ;
/* Declares 3 arrays: a, b and c */
```

Alternatively, you can place the brackets next to the identifier.

```java
int a[], b, c ;
/* Declares 1 array: a and two ints: b and c */
```

Array Declarations: 2

Arrays are like any other object: the value of an array variable is null until you create an array object using new.

```java
int[] grades ;
int n = grades.length ; /* Error: can’t access null array */
```

```java
grade = new int[20] ;
n = grades.length ; /* OK: n=20 */
```

Array Elements

The elements of an array are just like any regular variable of their type. Once you’ve indexed the array, you can do all the normal things.

```java
int[] nums = new int[100] ;
boolean[] flags = new boolean[10] ;
Foo foos = new Foo[12] ;
int i, j, k ;

... if( boolean[i] )
    nums[i] = nums[j]++ * foos[k].getX() ;
```

Arrays of Objects

When you create an array of objects, all of the elements are set to null. You need to create an instance for each element individually.

```java
Foo foos = new Foo[12] ;
for( int i=0 ; i < foos.length ; i++ )
    foos[i] = new Foo() ;
```
Array Initializers

If you know the values of the elements at compile time, you can create and initialize an array without using `new`.

```java
int[] grades = { 87, 66, 93 };
```

is the same as

```java
int[] grades = new int[3]
grades[0] = 87;
grades[1] = 66;
grades[2] = 93;
```

Array Parameters

Arrays are like any other object parameter: the reference is immutable, but the elements can change.

```java
/* This method will successfully reverse the elements in an array */
public void reverse(int[] nums) {
    for (int i = 0; i < nums.length / 2; i++) {
        int tmp = nums[i];
        nums[i] = nums[nums.length-(i+1)];
        nums[nums.length-(i+1)] = tmp;
    }
}
```

Command-line Arguments

Now we know everything we need to understand the signature of the `main` method.

```java
public class CommandLine {
    public static void main(String[] args) {
        ... 
    }
}
```

The parameter `args` is an array of `String`. The elements of `args` are the command-line arguments used to run the class.

Two-dimensional Arrays

You can create an entire 2-D array with one call to new, or you can create each “inner” array separately. The inner arrays don’t necessarily have to be the same size.

```java
/* Full 10x12 array of int */
int[][] matrix = new int[10][12];

/* Create the “outer” array first */
int[][] ragged = new int[5][];

/* Inner arrays are of lengths 1, 2, 3, 4 and 5 */
for (int i = 0; i < ragged.length; i++)
    ragged[i] = new int[i+1];
```

Multi-dimensional Arrays

There’s no reason you can’t create arrays of 3, 4 and 5 dimensions (“an array of arrays of arrays of arrays...”), but beyond 2 dimensions, they get harder and harder to manage.

Before you create a many-dimensional array, think hard about other ways to attack the problem. Often, an object carrying all the same information is a better idea.

```java
private int[][] board = {
    {EMPTY, EMPTY, EMPTY},
    {EMPTY, EMPTY, EMPTY},
    {EMPTY, EMPTY, EMPTY}
};
```

Sorting Integers

Now that we can make lists of things using arrays, we might like to be able to organize our lists.

A sorting algorithm takes an unordered array of objects and returns an array of objects in ascending or descending order. What constitutes “ascending order” on a general object `Foo` is not always clear, so for now we’ll concentrate on integers.
**Bubble Sort**

Input: List of integers $X_0, X_1, \ldots, X_n$.
Output: $X_0, X_1, \ldots, X_n$ in ascending order.

1. $i := 0$, $swap := 0$
2. If $X_i > X_{i+1}$, swap $X_i$ and $X_{i+1}$, $swap := 1$.
3. $i := i + 1$
4. If $i < n$, go to Step 2.
5. If $swap > 0$, go to Step 1.
6. Return sorted list $X$.

**Bubble Sort: Example**

| 5 9 2 1 6 | 2 1 5 6 9 |
| 5 9 2 1 6 | 2 1 5 6 9 |
| 5 2 9 1 6 | 2 1 5 6 9 |
| 5 2 9 1 6 | 2 1 5 6 9 |
| 2 5 1 6 9 | 1 2 5 6 9 |
| 2 5 1 6 9 | 1 2 5 6 9 |

**Selection Sort**

Input: List of integers $X_0, X_1, \ldots, X_n$.
Output: $X_0, X_1, \ldots, X_n$ in ascending order.

1. $i := 0$.
2. Find the index $j$ of the minimum item, such that $j \geq i$.
3. Swap $X_i$ and $X_j$.
4. $i := i + 1$
5. If $i < n$, go to Step 1.
6. Return sorted list $X$.

**Selection Sort: Example**

| 5 9 2 1 6 | 2 1 5 6 9 |
| 1 9 2 5 6 | 2 1 5 6 9 |
| 1 2 9 5 6 | 1 2 5 6 9 |
| 1 2 5 6 9 |

**Insertion Sort**

Input: List of integers $X_0, X_1, \ldots, X_n$.
Output: $X_0, X_1, \ldots, X_n$ in ascending order.

1. $i := 1$.
2. $n := X_i$, $j := i - 1$.
3. If $j < n$ or $j < 0$, go to Step 5.
4. $X_{j+1} := X_j$, $j := j - 1$, go to Step 3.
5. $X_{j+1} := n$, $i := i + 1$.
6. If $i < n$, go to Step 2.
7. Return sorted list $X$.

**Insertion Sort: Example**

| 5 9 2 1 6 |
| 2 5 9 1 6 |
| 1 2 5 6 9 |