Arrays

Arrays: allow us to store and manipulate large quantities of data. An array is an indexed sequence of values of the same type.

Examples

- 52 playing cards in a deck.
- 17,000 undergraduates in UoE.
- 1 million characters in a book.
- 10 million audio samples in an MP3 file.
- 4 billion nucleotides in a DNA strand.
- 90 billion Google queries per year.
- 50 trillion cells in the human body.

(From 100 most popular Scottish girls' names, 2007)

Many Variables of the Same Type

How do we initialize 10 variables of the same type?

```java
double a0, a1, a2, a3, a4, a5, a6, a7, a8, a9;
a0 = 0.0;
a1 = 0.0;
a2 = 0.0;
a3 = 0.0;
a4 = 0.0;
a5 = 0.0;
a6 = 0.0;
a7 = 0.0;
a8 = 0.0;
a9 = 0.0;
...
a4 = 3.0;
a4 = 8.5;
...
double x = a4 + a5;
```

// easy alternative
```java
double[] a = new double[10];
...
a[4] = 3.0;
a[8] = 8.0;
...
double x = a[4] + a[8];
```
Many Variables of the Same Type

How do we initialize 1 million variables of the same type?

```java
// just as easy with large arrays
double[] a = new double[1000000];
...
a[123456] = 3.0;
a[987654] = 8.0;
...
double x = a[123456] + a[987654];
```

### Arrays in Java, 1

Java has special support for arrays:
- To make an array: declare, create and initialize it.

**Declare an array**

```java
int[] array0fInts;
```

**Create an array of length 10**

```java
array0fInts = new int[10];
```

### Arrays in Java, 2

Java has special support for arrays:
- To make an array: declare, create and initialize it.
- To access element \( i \) of array named \( a \), use \( a[i] \).
- Array indices start at 0.

```java
int n = 10; // size of array
double[] a; // declare the array
a = new double[n]; // create the array
for (int i = 0; i < n; i++) {
    a[i] = 0.0; // initialise each elt
}
```

**Compact alternative:**
- Declare, create and initialize in one statement.

```java
int n = 10; // size of array
double[] a = new double[n]; // declare, create, init
```
### Default Initialization of Arrays

Each array element is automatically initialized to a default value:

- **int**: 0
- **double**: 0.0
- **boolean**: false
- **char**: '\0'(null terminator)
- **String**: null

### Types of Array

All elements of a given array must be of the same type.

#### Array Types

- `int[]`
- `double[]`
- `String[]`
- `char[]`

### Alternative Initialization Syntax for Arrays

- Shorthand syntax for initializing arrays.
- Handy if you only have a few data items.

```java
String[] names = {"Rebecca", "Isla", "Brooke", "Megan", "Niamh"};
int[] mynums = {0, 7, 9, 1, 4};
double[] morenums = {2.5, -0.1, 33.0};
```

### The Length of Arrays

Given an array `a`,

- check the length of the array: `a.length`
- first element is `a[0]`
- second element is `a[1]`
- ...
- last element is `a[a.length-1]`
- If an array index is too small or too large, Java throws run-time error: `ArrayIndexOutOfBoundsException`
Arrays: Another Example

```java
public class ArrayEx {
    public static void main(String[] args) {
        String[] names = {"Rebecca", "Isla", "Brooke", "Megan", "Niamh"};
        System.out.println(names.length);
        System.out.println(names[1]);
        System.out.println(names[names.length]);
    }
}
```

Output

```
5
Isla
Exception in thread "main" java.lang.ArrayIndexOutOfBoundsException: 5
```

To get at last element, use `names[names.length-1]`.

Vector Dot Product

```
public class ArrayEx {
    public static void main(String[] args) {
        String[] names = {"Rebecca", "Isla", "Brooke", "Megan", "Niamh"};
        System.out.println(names.length);
        System.out.println(names[1]);
        System.out.println(names[names.length]);
    }
}
```

```
double[] x = {0.3, 0.6, 0.1};
double[] y = {0.5, 0.1, 0.4};
double sum = 0.0;
for (int i = 0; i < x.length; i++) {
    sum = sum + x[i] * y[i];
}
```

States

<table>
<thead>
<tr>
<th>i</th>
<th>x[i]</th>
<th>y[i]</th>
<th>x[i]*y[i]</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0.30</td>
<td>0.50</td>
<td>0.15</td>
<td>0.15</td>
</tr>
<tr>
<td>1</td>
<td>0.60</td>
<td>0.10</td>
<td>0.06</td>
<td>0.21</td>
</tr>
<tr>
<td>2</td>
<td>0.10</td>
<td>0.40</td>
<td>0.04</td>
<td>0.25</td>
</tr>
</tbody>
</table>

To get at last element, use `names[names.length-1]`.

Array-processing Examples, 1

Create an array with random values
```
double[] a = new double[n];
for (int i = 0; i < n; i++) {
    a[i] = Math.random();
}
```

Print the array values, one per line
```
for (int i = 0; i < n; i++) {
    System.out.println(a[i]);
}
```

Find the maximum of the array values
```
double max = a[0];
for (int i = 1; i < n; i++) {
    if (a[i] > max) max = a[i];
}
```

Array-processing Examples, 2

Compute the average of the values in an array (length `n`) of doubles.
```
double sum = 0.0;
for (int i = 0; i < n; i++) {
    sum += a[i];
}
double average = sum / n;
```

▶ Exercise: Figure out (i) why this algorithm fails for a long list of numbers, (ii) how to fix it?
▶ For the keen: How to fix it if we didn't know the number of values in advance?
Copy one array (called \(a\), of doubles, length \(n\)) to another.

double[] \(b\) = new double[\(n\)];
for (int \(i\) = 0; \(i\) < \(n\); \(i++\)) {
    \(b[i]\) = \(a[i]\);
}

▶ Note, it's not “just” \(b=a\) as for primitive types.
▶ See also: clone, System.arraycopy, Arrays.copyOf.

Reverse the elements within an array (called \(a\), of doubles, length \(n\)).

for (int \(i\) = 0; \(i\) < \(n/2\); \(i++\)) {
    double temp = \(a[i]\);
    \(a[i]\) = \(a[n-1-i]\);
    \(a[n-1-i]\) = temp;
}

Setting Array Values at Run Time
Print a random card.

String[] rank = { "2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace" };
String[] suit = { "Clubs", "Diamonds", "Hearts", "Spades" };

int \(i\) = (int) (Math.random() * 13);  // between 0 and 12
int \(j\) = (int) (Math.random() * 4);  // between 0 and 3
System.out.println(rank[\(i\)] + " of " + suit[\(j\)]);

Q: In what order does the program print the deck?

Output 1

7 of Spades
...
Jack of Diamonds
...

Output 2

2 of Clubs
2 of Diamonds
2 of Hearts
2 of Spades
3 of Clubs
4 of Clubs
5 of Clubs
6 of Clubs
Shuffling

Given an array, rearrange its elements in random order.

**Shuffling algorithm:**

1. In iteration i, pick random card from **deck[i]** through **deck[N-1]**, with each card equally likely.
2. Exchange it with **deck[i]**.

---

Remark on capitalisation

In Java, roughly, everything except class names (and a few related things) starts with a lower case letter as a matter of convention. But **SUITs** etc. were written in capitals. Why?

Because if we were teaching the whole language these would have been *constant fields* — not only constant, but enforced as constant by the compiler. But we aren’t teaching the features that let you do that in this course, so always start your variable and method names with lower case.

---

Shuffling a Deck of Cards

```java
public class Deck {
    public static void main(String[] args) {
        String[] suit = {"Clubs", "Diamonds", "Hearts", "Spades"};
        String[] rank = {"2", "3", "4", "5", "6", "7", "8", "9", "10", "Jack", "Queen", "King", "Ace"};
        int SUITS = suit.length;
        int RANKS = rank.length;
        int N = SUITS * RANKS;
        String[] deck = new String[N];
        for (int i = 0; i < RANKS; i++)
            for (int j = 0; j < SUITS; j++)
                deck[SUITs * i + j] = rank[i] + " of " + suit[j];
        for (int i = 0; i < N; i++) {
            int randCard = i + (int) (Math.random() * (N - i));
            String temp = deck[randCard];
            deck[randCard] = deck[i];
            deck[i] = temp;
        }
        for (int i = 0; i < N; i++)
            System.out.println(deck[i]);
    }
}
```

---

Output

```
% java Deck
Jack of Clubs
4 of Spades
5 of Clubs
10 of Diamonds
2 of Hearts
Queen of Clubs
8 of Hearts
5 of Hearts
3 of Clubs
7 of Hearts
10 of Hearts
6 of Hearts
Jack of Spades...
3 of Hearts
```
**Two-Dimensional Arrays**

Examples of two-dimensional arrays:
- Table of data for each experiment and outcome.
- Table of grades for each student and assignment.
- Table of grayscale values for each pixel in a 2D image.

Mathematical abstraction: matrix
Java abstraction: 2D Array

---

**Setting 2D Array Values at Compile Time**

Initialize 2D array of doubles by listing values. Each element of the array is itself an array of type `double[]`.

```java
double[][] p = {
    { 0.02, 0.92, 0.02, 0.02, 0.02 },
    { 0.02, 0.92, 0.32, 0.32, 0.32 },
    { 0.02, 0.02, 0.02, 0.02, 0.02 },
    { 0.02, 0.02, 0.02, 0.02, 0.02 },
    { 0.47, 0.02, 0.47, 0.02, 0.02 },
};
```

---

**Matrix Addition**

Matrix Addition: given two n-by-n matrices `a` and `b`, define `c` to be the n-by-n matrix where `c[i][j]` is the sum `a[i][j] + b[i][j].`

```java
int m = 10;
int n = 3;
double[][] a = new double[m][n];
for (int i = 0; i < m; i++) {
    for (int j = 0; j < n; j++) {
        a[i][j] = 0.0;
    }
}
for (int i = 0; i < m; i++) {
    double[][] b = new double[m][n];
    for (int j = 0; j < n; j++) {
        b[i][j] = 0.0;
    }
}
for (int i = 0; i < m; i++) {
    double[][] c = new double[m][n];
    for (int j = 0; j < n; j++) {
        c[i][j] = a[i][j] + b[i][j];
    }
}
```
Matrix Multiplication

Matrix Multiplication: given two n-by-n matrices \( a \) and \( b \), define \( c \) to be the n-by-n matrix where \( c[i][j] \) is the dot product of the \( i^{th} \) row of \( a[][] \) and the \( j^{th} \) column of \( b[][] \).

```java
double[][] c = new double[n][n];
for (int i = 0; i < n; i++) {
    for (int j = 0; j < n; j++) {
        for (int k = 0; k < n; k++) {
            c[i][j] += a[i][k] * b[k][j];
        }
    }
}
```

Enhanced for loop, 1

Ordinary for loops are easy to get wrong! Often there's a better way:

```java
int[] numbers = {2, 5, 6, 1, 0, 5};
```

**Ordinary for loop**

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

**Enhanced for loop**

```java
for (int num : numbers) {
    System.out.println(num);
}
```

Enhanced for loop, 2

▶ Also called *for-each* loop, with : pronounced “in”.
▶ On each iteration, an element of the iterable gets assigned to the loop variable.
▶ Loop gets executed once for each element in the iterable.
▶ Easier and more concise: no need to initialise loop counter, increment, set termination condition...
▶ ... but less flexible; no access to the loop counter.
▶ Use them whenever you don’t need access to the loop counter.
▶ Typical use: when you need access to all the elements of an array, but you don’t care about their indexes.

General form:

```java
for ( variable declaration : iterable ){
    ...
}
```

NB the variable must have same type as elements in *iterable*.

Enhanced for loop, 3

Another Example: Right

```java
String[] words = {"hello", "world", "yes", "we", "can"};
for (String w : words) {
    System.out.println(w);
}
```

Another Example: Wrong

```java
String[] words = {"hello", "world", "yes", "we", "can"};
for (int w : words) {
    System.out.println(w);
}
```
Summary

Arrays:
- Method of storing large amounts of data.
- Almost as easy to use as primitive types.
- We can directly access an element given its index.

Enhanced for loop:
- Good alternative to ordinary for loop where you just want to iterate over an array, and don’t care about the indexes.

This Week’s Reading

Java Tutorial
pp51-57
i.e. now it’s time to read carefully the section on Arrays within Chapter 3, *Language Basics*, that I suggested skimming over before.