Arrays – continued

Kees Huizing
September 2009
Multi-dimensional arrays

- to model grids, matrices, screens, tables, etc.
- example: student scores

- student nr. $i$ has scored an $s$ for course nr. $c$

<table>
<thead>
<tr>
<th>course → student ↓</th>
<th>0</th>
<th>1</th>
<th>2</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>good</td>
<td>avg</td>
<td>good</td>
<td>NV</td>
</tr>
<tr>
<td>1</td>
<td>avg</td>
<td>avg</td>
<td>bad</td>
<td>exc</td>
</tr>
<tr>
<td>2</td>
<td>avg</td>
<td>avg</td>
<td>avg</td>
<td>avg</td>
</tr>
</tbody>
</table>
String[][] table;

table = new String[4][3];

//print table
for (int r=0; r < table.length; r++) {
    for (int c=0; c < table[r].length; c++) {
        System.out.print( table[r][c] );
    }
    System.out.println();
}

//etc.

2-dim array is an array of arrays
Two-dimensional arrays

• conventions
  
  • pixels, geometry, etc.: first index is x-coordinate, second is y-coordinate
  
  • matrices: first index is row, second is column
  
  • different :

• it doesn’t matter what you choose, as long as it is consistent
Ragged arrays

- what does this produce?

```
//declaration
int[][] triangle;
//creation
triangle = new int[5][];
for (int r = 0; r < 5; r++) {
    triangle[r] = new int[r+1];
}
//filling
for (int r = 0; r < triangle.length; r++) {
    for (int c = 0; x < triangle[r].length; c++) {
        triangle[r][c] = 10*r + c;
    }
}
//printing
for (int r = 0; r < triangle.length; r++) {
    for (int c = 0; x < triangle[c].length; c++) {
        System.out.print(triangle[r][c] + " ");
    }
    System.out.println();
}
```
Aliasing

- `int[ ] a;` declares a variable that holds a reference to an array
- `int[ ] b;` ditto
- `a = new int[5];` creates 5 variables and a “points to them”
- `b = a;` makes `b` point to the same 5 variables `a` points to

```
• System.out.println( b[1] );
  • will print 0
```

```
• a[1] = 7;
  System.out.println( a[1] );
  • will print 7
```

```
• System.out.println( b[1] );
  • will print 7
```
Aliasing

- `int[] a;` declares a variable that holds a reference to an array
- `int[] b;` ditto
- `a = new int[5];` creates 5 variables and `a` “points to them”
- `b = a;` makes `b` point to the same 5 variables `a` points to

- `System.out.println( b[1] );`
  - will print 0
- `a[1] = 7; System.out.println( a[1] );`
  - will print 7
- `System.out.println( b[1] );`
  - will print 7
Aliasing ctd

- assignment `b = a` only copies *reference*
- to copy *content* use loop
- or `b = Arrays.copyOf(a, 5);`
Frequency count

• Suppose we want to count how often each grade (1-10) occurs in a list of 100 grades

```java
int[] frequencies = new int[11];
//assert all elts of frequencies are 0

for (int i = 0; i<100; i++) {
    int grade = scanner.nextInt();
    assert 1<=grade && grade<=10;
}
```
Frequency count

• Suppose we want to count how often each grade (1-10) occurs in a list of 100 grades

```java
int[] frequencies = new int[11];
//assert all elts of frequencies are 0

for (int i = 0; i<100; i++) {
    int grade = scanner.nextInt();
    assert 1<=grade && grade<=10;

    frequencies[ grade ] += 1;
}
```
Array as return type

Calculate frequency of digits in number \( n \)

```java
int[] digitFreq(int n) {
    assert n>=0;
    int[] freqs = new int[10];
    //assert all elts of freqs are 0;

    int m=n;
    // special case
    if (m==0) { freqs[0] += 1; }

    while (m>0) {
        int d = m % 10;
        assert 0<=d && d<10;
        freqs[d] += 1;
        m /= 10;
    }

    return freqs;
}
```

d is last digit of \( m \)
last digit is removed from \( m \)
More on arrays

• quick initialization:

    String[] beers = {“Duvel”, “Leffe Dubbel”, “Hoare Tripel”};

• elsewhere:

Specification
Specification

- method builder has to assume that method is called correctly:
  - **precondition** (requires clause)
- method caller has to assume that method produces correct result
  - **postcondition** (ensures clause)
Specification

- See method as a *product*
  - developer offers a product
  - programmer uses product
  - delivery contract with rights and duties of parties
Specification example

What are the pre- and postcondition?

//pre: n>=0
//post: \esult == r && r*r <= n && n < (r+1)*(r+1)
f(int n) {
    int r = 0;
    while (r+1)*(r+1) <= n {
        r +=1;
    }
}

pre: n ≥ 0

post: r^2 ≤ n < (r+1)^2
or r = \lfloor \sqrt{n} \rfloor (floor)

Usage

assert n >= 0;
w = f( 10 );
assert w*w <= n && n < (w+1)*(w+1);

apply pre- and post to actual parameters