CS1110 28 Oct 2008 Arrays. Reading: Secs 8.1, 8.2, 8.3

Listen to the following lectures on loops on your Plive CD. They are only 2-3 minutes long, and each has an insightful message.

1. The 3 lectures on Lesson page 7-6 —read the whole page.
2. The 4 lectures in Lesson page 7-5.

Computational simplicity

If you are writing too much code —it gets longer and longer, with no end in sight: stop and look for a better way.

If your code is getting convoluted and you have trouble understanding it: stop and look for a better way.

Learn to keep things simple, to solve problems in simple ways. This sometimes requires a different way of thinking.

We are trying to teach not just Java but how to think about problem solving.

A key point is to break a problem up into several pieces and do each piece in isolation, without thinking about the rest of them. Our methodology for developing a loop does just that.

1

Array: object. Can hold a fixed number of values of the same type. Array contains 4 **int** values.

a0	
0	5
1	7
2	4
3	-2

The **type** of the array:

> **int**[]

Variable contains name of the array.   x | a0 | int[]

Basic form of a declaration:

> *<type> <variable-name>* ;

A declaration of x.   **int**[] x ;

Does not create array, it only declares x. x's initial value is **null**.

Elements of array are numbered: 0, 1, 2, …, x.length–1;

Make everything as simple as possible, but no simpler. Einstein

2

Notes on array length

Array length: an instance field of the array.

This is why we write x.length, not x.length( )

Length field is **final:** cannot be changed.
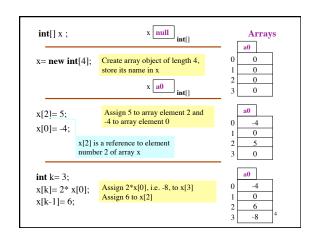
Length remains the same once the array has been created.

We omit it in the rest of the pictures.

a0	
length	4
0	5
1	7
2	4
3	-2

x | a0 | int[]

The length is not part of the array type.

The type is **int**[]

An array variable can be assigned arrays of different lengths.

3

---

int[] x ;   x | null | int[]   **Arrays**

a0	
0	0
1	0
2	0
3	0

x= **new int**[4];   Create array object of length 4, store its name in x

x | a0 | int[]

x[2]= 5;   Assign 5 to array element 2 and -4 to array element 0
x[0]= -4;

x[2] is a reference to element number 2 of array x

a0	
0	-4
1	0
2	5
3	0

int k= 3;
x[k]= 2* x[0];   Assign 2*x[0], i.e. -8, to x[3]
x[k-1]= 6;   Assign 6 to x[2]

a0	
0	-4
1	0
2	6
3	-8

4

Difference between Vector and array

Declaration:	**int**[] a;	Vector v;
	Elements of a: **int** values	Elements of v: any Objects
Creation:	a= **new int**[n];	v= **new** Vector();
	Array always has n elements	Number of elements can change
Reference:	a[e]	v.get(e)
Change element:	a[e]= e1;	v.set(e, e1);

Array locations a[0], a[1], … in successive locations in memory. Access takes same time no matter which one you reference.

Elements all the same type (a primitive type or class type)

Can't tell how Vectors are stored in memory. Referencing and changing elements done through method calls

Elements of any Object type (but not a primitive type). Casting may be necessary when an element is retrieved.

Array initializers

Instead of

> **int**[] c= **new** int[5];
>
> c[0]= 5; c[1]= 4; c[2]= 7; c[3]= 6; c[4]= 5;

a0	
	5
	4
	7
	6
	5

Use an array initializer:

> **int**[] c= **new int**[ ] {5, 4, 7, 6, 5};

No expression between brackets [ ].
(can omit this)

array initializer: gives values to be in the array initially. Values must have the same type, in this case, **int**. Length of the array is the number of values in the list

Computer science has its field called computational complexity; mine is called computational simplicity. Gries

6

Use of an array initializer

```
public class D {
    public static final String[] months= new String[]{"January", "February",
                "March", "April", "May", "June", "July", "August",
                "September", "October", "November", "December"};

    /** = the month, given its number m
        Precondition: 1 <= m <= 12 */
    public static String theMonth(int m) {
        return months[m–1];
    }
}
```

Months[m–1] is returned, since

months[0] = "January",
months[1] = "February",
…

Variable months is:

static; object assigned to it will be created only once.

public: can be seen outside class D.

final: it cannot be changed.

7

Procedure swap

```
public class D {
    /** = Swap x and y */
    public static void swap (int x, int y) {
        int temp= x;
        x= y;
        y= temp;
    }
}
….
```

A call will NOT swap a and b. Parameters x and y are initialized to the values of a and b, and thereafter, there is no way to change a and b.

a 5   b 3

swap(a, b);

swap: 1   ?

frame for call just after
frame created and args   x 5   y 3
assigned to pars: temp ?

8

Procedure swap

```
public class D {
    /** = Swap b[h] and b[k] */
    public static void swap (int[] b, int h, int k) {
        int temp= b[h];
        b[h]= b[k];
        b[k]= temp;
    }
}
….
    swap(c, 3, 4);
```

Does swap b[h] and b[k], because parameter b contains name of the array.

c a0

a0
5
4
7
6
5

swap: 1   ?

b a0   h 3

temp ?   k 4

frame for call just after frame is created.

9

Linear search

```
public class D {
    /** = index of first occurrence of c in b[h..k-1] —
        = k if c is not in b[h..k-1]*/
    public static int findFirst (int c, int[] b, int h, int k) {
        // if c is in b[h..k-1], return its index in b[h..k-1]

        // { invariant: c is not in b[h..t-1] }
        for (int t= h;  t < k;  t= t+1) {
            // Process t;
            if (b[t] == c)
                return t;
        }
        // { c is not in b[h..k-1] }
        return k;
    }
}
```

Remember

h..h-1 is the empty range

10

Two-dimensional arrays

```
   0 1 2 3   b.length      one-dimensional array
b  5 4 7 3
```

```
   0 1 2 3              rectangular array: 5 rows and 4 columns
d  0 5 4 7 3
     4 8 9 7
   1 5 1 2 3
   2
```

Type of d is **int**[][] ("**int** array array",
                "an array of **int** arrays")

To declare variable d:

   int d[][].

To create a new array and assign it to d:

   d= **new int**[3][4];

To reference element at row r column c:

   d[r][c]

number of rows

number of cols

11