CS100J 17 March 2005

Arrays. Reading: Secs 8.1, 8.2, 8.3.

The last Java feature to study in this course

The test tonight is in the Uris Auditorium at 7:30

Over the break, please listen to the following lectures on loops on your Plive CD. They are only 2-3 minutes long, and each has an insightful message for you.

1. The three lectures on Lesson page 7-6 --in fact, read the whole page.

2. The four lectures in Lesson page 7-5.

It also will help to read Secs. 8,1, 8.2, and 8.3 on arrays.

Quote for the Day: Computer science has its field of computational complexity. Mine is computational simplicity. **Gries**

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**.

Learning to keep things simple, to solve problems in a simpler way, 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 methdology for developing a loop does just that.

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

Arrays

An array is an object that can hold a fixed number of values of the same type. The array to the right contains 4 **int** values.

The **type** of the array to the right is

int[]

Here is a variable that contains the name of the array.

x a0 **int[]**

Remember that a basic declaration has the form

<type> <variable-name> ;

Therefore, a declaration of x looks as to the right. The declaration does not create the array, it only declares x. x's initial value is **null**. We'll show you later how to create the array.

int[] x ;

The elements of the array are numbered 0, 1, 2, …, x.length–1. Note that length is a variable, not a function, so don't put () after it.

	a0
0	5
1	7
2	4
3	-2

Arrays

int[] x ;

x **null** **int**[]

x= **new int**[4];

Create an array object of length 4 and store its name in x

x a0 **int**[]

	a0
0	0
1	0
2	0
3	0

x[2]= 5;

x[0]= -4;

Assign 5 to array element 2 and -4 to array element 0

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];

x[k-1]= 6;

Assign 2*x[0], i.e. -8, to x[3]
Assign 6 to x[2]

	a0
0	-4
1	0
2	6
3	-8

Difference between Vector and array --both used to contain a bunch of things

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], a[2] are in successive location in memory. Access is guaranteed to be the same, no matter which one you reference.

Elements are all the same type (a primitive type or some Object type)

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

Elements can be of any Object type (but not a primitive type), and 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;

Use an array initializer:

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

a0
5
4
7
6
5

No expression between the brackets [ ].

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

A use of an array initializer

```
public class D {
   private static 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];
   }
}
```

Variable months is made static, so that the object assigned to it will be created only once. It is private, so that it cannot be seen outside class D.

Note that months[m–1] is returned, since

months[0] = “January”, months[1] is “February”, …

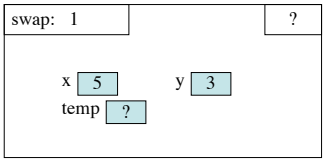
Procedure swap

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

The 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.

```
….
  swap(a, b);
```

a 5 b 3



frame for call just after frame is created.

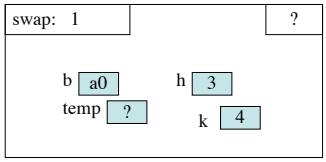
Procedure swap

```
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;
   }
}
```

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

```
….
  swap(c, 3, 4);
```

c a0



frame for call just after frame is created.