Scarborough College
University of Toronto
Fall 1997

CSC A06F: Introduction to Computer Programming
Lecture notes 5:
Arrays and Vectors

Reading:

- Lewis and Loftus, chapter 6.

A problem

Write a program that:

- reads in a list of students and grades,
- calculates the class average, and
- reports each student's difference from the average.

Would need to declare separate variables for each expected name and grade:

```
String name1, name2, name3, name4, ... ;
int grade1, grade2, grade3, grade4, ... ;
```

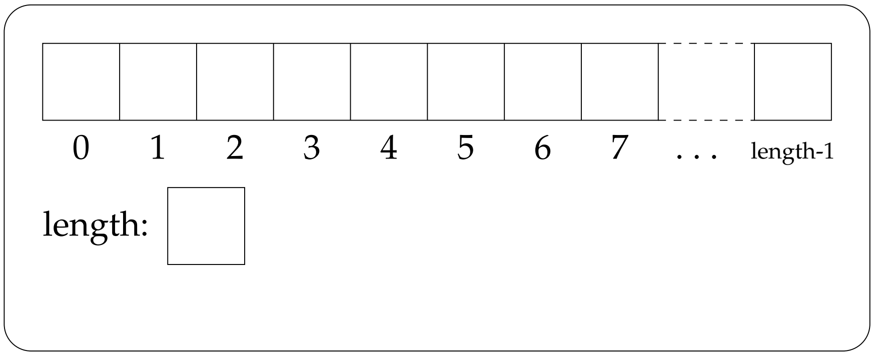
Give at least 2 problems with this.

A better solution: arrays

An **array** is an *object* that contains an ordered list of elements.

The elements can be of any type, but they must all be the same type.

The elements are *ordered* by **index** values ranging from 0 .. $(N-1)$, where N is the length of the array.



Creating an array

Declare the array:

```
float[] grades;             // an array of floats
```

Instantiate and initialize the array: must use `new` to actually create it, because arrays are objects.

```
grades = new float[500];  // makes space for 500 floats
```

Initialize and access the elements: use *`arrayname[n]`* to access element n of the array.

```
grades[0] = 76.5;
grades[1] = 82.1;
float sum = grades[0] + grades[1];
System.out.println(sum);
```

The index can be any integer expression. I.e., an expression that evaluates to an integer.

Reading input into an array

Assume that an input stream, `stdin`, has been set up in the normal way.

```
final int MAX = 500;
float grades = new float[MAX];

int count = 0;
String sGrade = stdin.readLine();

while ( ! sGrade.equals("quit") ) {
   grades[count] = Float.valueOf(sGrade).floatValue();
   count++;
   String sGrade = stdin.readLine();
}
```


Calculate the average

Complete the following loop that sums up the grades in the `grades` array, so we can calculate the average. Assume there are `count` grades in all.

```
int sum = 0;
int i = 0;        // 'i' is the loop counter

// sums up the grades in 'sum'
while (           ) {



}
int avg = sum / count;
```

Exercise: write the code to print each student's difference from the average.

Notes on arrays

- Watch out for 'of-by-one' errors. The first element is at index 0, not 1, and if the array has N elements, the last element is at index $N-1$.
- If you try to access an element before index 0, or after $N-1$, you're program will crash with an **Out of bounds** error.
- You can find out how many elements an array holds using the `length` field. E.g.,

```
int l = grades.length;
```

- You can initialize an array much like a primitive type:

```
int[] DaysInMonth = { 31, 28, 31, 30, 31, 30,
                      31, 31, 30, 31, 30, 31 };
System.out.println (DaysInMonth[4]);
```


Assignment of one array to another

Since an array is an object, you can assign one array to another:

```
float[] grades2;

grades2 = grades;
```

`grades` is an array of floats, so can assign it to `grades2`. (They are both the same type.)

Now, `grades2` refers to the *exact same* array as *grades* WHY?

Notice: there is no need to instantiate or initialize `grades2`. WHY?

Arrays of objects

An array, an object itself, can also have objects as elements.

But there are two steps to setting it up:

- Initialize the array itself (as with arrays of primitive types.
- Then, instantiate and initialize each of the objects, separately.

E.g.,

```
// array of 100 CDs
CD[] all_CDS = new CD[100];

// initialize the 0th CD
all_CDS[0] = new CD("Trace -- Son Volt", 52.5, 13);
```

Note: Instantiating the array of CDs only reserves room for the 100 references to the CDs, and ***does not*** create (i.e., instantiate) any CDs. Why doesn't it do this for you?

Passing arrays as parameters

You can also pass arrays as parameters to methods.

E.g., Consider this program:

```
import java.io.*;

// All this does is echo back the arguments put on the command line

public class EchoArgs {

   public static void main (String[] args) {

      for (int i=0; i < args.length; i++) {
         System.out.println("Argument " + i + ": " + args[i]);
      }

   }
}
```

The array `args` is an array of strings. It is passed into the `main` method by the system when you run your program.

```
> java EchoArgs arg1 arg2 arg3 ...
```

Aside: `for` loops

If you know how many iterations (i.e., times through the loop) you need to do, then you can use a `for` loop.

Syntax:

```
for (counter-initialize; condition; increment)
   statement;
```

It executes the `counter-initialize`, once, at the beginning.

Then for each iteration, it checks the `condition`. If it is true, it executes the `statement`, and then the `increment`.

If the `condition` is false, the loop ends.

Really just a shortcut for a `while` loop.

Another example

```
// method that returns the longest CD in the given array

CD LongestCD( CD[] cds, int num ) {
   double maxtime = CD[0].getTime();
   CD longest = CD[0];

   // find the maximum
   for (int i=1; i < num; i++) {
      if ( CD[i].getTime() > maxtime ) {
         maxtime = CD[i].getTime();
         longest = CD[i];
      }
   }
   return longest;
}
```

When should you use an array?

When you need to store a set of data that will be treated in a fairly uniform way, and which all need to be kept around together.

- tables, matrices, lists, etc.
- to be able to access your data randomly.
- sorting a set of names into order: have to keep scanning over the array.
- ...

When isn't an array useful

When you don't need to keep the data around. Just process it as you get it.

E.g., Printing phone bills: just read each long distance charge, print it on the bill, and throw it away.

When data aren't going to be treated uniformly.

E.g., no point to have array whose first element is a name, second is sales tax, third is a phone number, etc.

Vectors

An array has a fixed size once you create it.

Two problems:

- What if you need more space?
- It can waste memory, if you don't use all the space.

So, we have the **`Vector`** class.

A `Vector` has a **dynamic** size: it starts out with 0 elements, and grows as you add elements, and shrinks as you remove elements.

Creating a vector:

```
Vector students = new Vector();
```


Using a vector

Some of the methods of a vector:

```
void addElement (Object object)      - adds an object
void removeElement (Object object)   - removes an object
Object elementAt (int index)         - returns object at 'index'
int size ()                          - returns number of objects in it
```

Example:

```
students.addElement ("John");
students.addElement ("Mary");
students.addElement ("Dekai");

for (int i=0; i < students.size(); i++) {
   System.out.println (students.elementAt(i));
}
```