Vector program patterns

Vectors contain many elements, so loops are common.

Counted processing

```
// Print the first 3 elements.
for (int i = 0; i < 3; i++)
  System.out.println(v.elementAt(i)); // no cast?

// Print all the elements.
for (int i = 0; i < v.size(); i++)
 System.out.println(v.elementAt(i));
```

Enumerators

Processing without counting

```
Enumeration enum = v.elements();
while (enum.hasMoreElements())
  System.out.println(enum.nextElement());
```

You'll take a while to get used to this approach, but it follows a good rule:

- Don't put information you don't need into your programs.

Some Vector methods

These methods all have return type "void" if not mentioned.

- addElement (Object o) adds an element at the end of the Vector
- clear ( ) removes all the elements in the Vector
- Object elementAt (int index) returns a reference to an element
- Object remove (int index) removes an element and returns a reference to it.
- int size ( ) returns the number of elements in the Vector
- setElementAt (Object o, int index) replaces the element at ***index*** with ***o***
- Enumeration elements ( ) returns an Enumeration for the Vector

Enumeration methods:

- boolean hasMoreElements ( ) returns ***true*** or ***false*** depending on whether there are still elements left to be processed.
- Object nextElement ( ) returns the next element to be processed.

Vectors of Vectors

What if we have three students, each with up to five term marks?
We need a list of students, with each item in the list owning a list of marks.

```
Vector s0 = new Vector(); // the first student's marks
Vector s1 = new Vector(); // the second student's marks
Vector s2 = new Vector(); // the third student's marks
```

Let's make a lecture section of students:

```
Vector course = new Vector();
course.addElement(s0);
course.addElement(s1);
course.addElement(s2);
```

Vectors of Vectors *continued*

Suppose the first student has three marks: 65, 89 and 47.

```
s0.addElement(new Integer(65));
s0.addElement(new Integer(89));
s0.addElement(new Integer(47));
```

Let's retrieve the second student's fourth mark:

```
Vector stu = (Vector)course.elementAt(1);
Integer mk = (Integer)stu.elementAt(3);
int mark = mk.intValue();
```

…or more briefly:

```
int mk = ((Integer) ((Vector) course.elementAt(1))
  . elementAt(3)) . intValue();
```

A more likely structure

Vectors of Vectors are less likely than a mixed structure with other fields:

```
class Student {
  private String name;
  private Vector marks = new Vector();
  public double getAverage () { ... /* what? */}
}

class Course {
  private Vector students = new Vector();
  private Employee instructor; // and so on
}
```

There's a kind of "Vector of Vectors" here, but it's "decorated".

Vectors *vs* arrays

The Vector class is useful and reasonably easy to work with, but it does have some disadvantages:

- awkward with primitive types
- inefficient with long lists

An alternative is the ***array***. Advantages:

- can have arrays of primitive types
- more efficient to process

but...

- more restrictive usage
  - size cannot change

Array notation

An array A of integers is an object that contains ***only*** ints.

We refer to the elements of A like this:

- A[0] is the array element with index 0 (the first element of A)
- A[1] is the second element
- A[N - 1] is the Nth integer

Each element of A is just like an ordinary int variable.

We can set and use the elements like this:

```
A[0] = 5;           // now A[0] == 5
int i = 3;
A[2] = i + 6;       // now A[2] == 9
A[3] = 5*A[2] + i;  // now A[3] == 48
```

No casting is necessary, because the compiler knows A's elements are ints.

An example

A sequence of many integers:

2 -3 13 428 97 -12 -96 11 26

What's the median?

- Is 2 the median?
- Is -3 the median?
- …

How long do we keep going? What do we do at each step?

Steps towards the median

Suppose we already have an array A of N integers:

```
int N = ...; // Maybe the size of N is from input?
int[] A ...; // Make A an array of N ints somehow.
```

Read the contents of A from input:

```
for (int i = 0; i < N; i++)
  A[i] = Integer.parseInt(in.readLine());
```

How many elements of A are less than 2?

```
int count = 0;
for (int i = 0; i < N; i++)
  if (A[i] < 2)
    count++;
System.out.println("number less than 2 is " +
    count);
```

Array rules

0. All elements of an array are of the same type, say "T".

1. ***An element of an array behaves exactly like an ordinary variable of the array's element type, T.***

2. An array is an object.

3. The number of elements in an array cannot change once you have instantiated the array.

Making an array

Declaring:

```
int[] A; // an array of ints
Student[] course; // an array of Students
```

- What is the type of "course"? of course's elements?

Instantiating:

```
A = new int[50]; // Now we can use A[0],...,A[49]

int classSize = Integer.parseInt(in.readLine());
course = new Student[classSize];
```

Arrays of objects

```
int classSize = Integer.parseInt(in.readLine());
Student[] course = new Student[classSize];
```

- How many Student objects exist at this point?

```
for (int s = 0; s < course.length; s++)
  course[s] = new Student();
```

Every array has a (public!) field "length". It's not a method!

Exercise

Write a method to count the members of an array that are smaller than some value:

```
int countSmaller (int refVal, int[] list)
```