



Extensible Arrays: Vectors and StringBuffer

Wellesley College CS230
Lecture 05
Monday, February 12
Handout #12

Problem Set:

Assignment #1 due Tuesday, February 13

05 - 1



Goals of Today's Lecture

- Show that **fixed-length arrays** can be inefficient when used in straightforward ways to solve certain problems, in particular:
 1. Reading the lines of a file into an array;
 2. Filtering the elements of an array.
- Introduce the **array-doubling idiom** as a technique for efficiently solving these problems.
- Introduce Java's **Vector** and **StringBuffer** classes, which encapsulate the array-doubling idiom.
- Say a little bit about **generic classes**, of which the Vector class is an example.

05 - 2

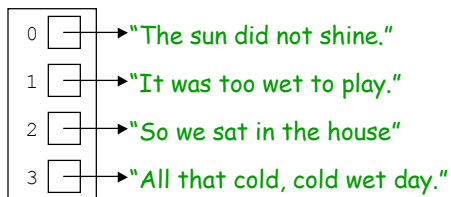


1. Reading the Lines of an File into an Array

Problem 1: read the lines of a file (w/o trailing newlines) into an array.

```
[fturbak@puma utils] cat ../text/cat-in-hat-4.txt  
The sun did not shine.  
It was too wet to play.  
So we sat in the house  
All that cold, cold, wet day.
```

`ExtensibleArray.fileToLines("../text-cat-in-hat-4.txt")` should return:



05 - 3



What's the Problem?

- When we create an array in Java, we must know its length. But when we open a file, we don't know the number of lines in advance.
- It is impossible to have a default array size that is "big enough" to store lines for any given file. (We could fail in the case where file has more lines than the default size.) Although this strategy is often used by programmers who don't know any better (especially in C), it is **unacceptable** in this course.
- We could change file format to put number of lines at beginning, but that's also **unacceptable**, since real-world files do not have this format.
- We will now study several strategies for solving this problem.

05 - 4



Strategy 1: Use fileToString() + split()

```
public static String[] fileToLinesSplitSimple (String infile) {  
    String contents = FileOps.fileToStringSimple(infile);  
    return contents.split("\n");  
}
```

↑ accumulates contents by concatenating lines to the end of a string variable.

Problem: the content-accumulation strategy is very inefficient!
(We'll see a more efficient strategy later today.)

File size	fileToStringSimple() (milliseconds)	fileToLinesSplitSimple() (milliseconds)
1000	37	52
2000	158	164
4000	1302	1403
8000	12202	12991
16000	99860	106725
32000	431255	453815

These times are not linear with file size!.

05 - 5



Digression: measuring times in Java

```
// Assume ../text/n-words.txt contains n words (one per line)  
// for n = 1000, 2000, 4000, 8000, 16000, and 32000.  
// Calculate time (in milliseconds) for processing these with fileToStringSimple()  
public static void testFileToStringSimple () {  
    for (int i = 1000; i <= 32000; i = i*2) {  
        String filename = "../text/" + i + "-words.txt";  
        System.out.print("fileToStringSimple(\"" + filename + "\"); time = ");  
        long start = System.currentTimeMillis();  
        // remember start time (long = 64-bit integer, while int = 32-bit integer)  
        FileOps.fileToStringSimple(filename); // calculate result and throw it away  
        long stop = System.currentTimeMillis(); // remember stop time  
        System.out.println((stop - start) + " milliseconds");  
    }  
}
```


05 - 6

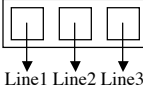


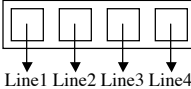
Strategy 2: Incrementing Array Size

Initially:  (An empty array.)

After first line: 

After second line: 

After third line: 

After fourth line: 

05 - 7



Strategy 2: Code

```
public static String[] fileToLinesArrayInc (String infile) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(infile));
        String[] result = new String[0];
        String line = reader.readLine(); // read the first line of the file.
        while (line != null) { // line becomes null at end of file
            // create an array one element larger than before
            String[] oneLarger = new String[result.length + 1];
            for (int i = 0; i < result.length; i++) // copy old contents to new array
                oneLarger[i] = result[i];
            oneLarger[result.length] = line; // include current line as last element
            result = oneLarger; // install new array as result
            line = reader.readLine(); // read the next line of the file
        }
        reader.close(); return result;
    } catch (IOException ex) {
        System.out.println(ex); return new String[0];
    }
}
```

05 - 8

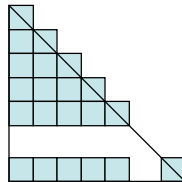


Strategy 2: Timing Results

File size	fileToLinesArrayInc() (milliseconds)
1000	8
2000	24
4000	101
8000	354
16000	1511
32000	9704

These times indicate quadratic growth, not linear growth!

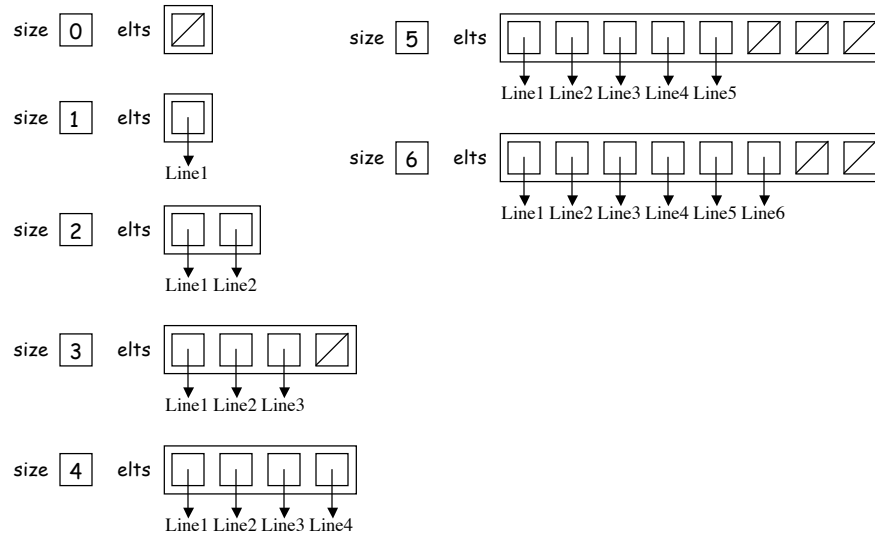
**Time of process is roughly
the number of squares in
an $n \times n$ pyramid**



05 - 9



Strategy 3: Doubling Array Size



05 - 10



Strategy 3 Code: Outer skeleton

```
/** Start with an singleton array, and double size when necessary */
public static String[] fileToLinesArrayDbl (String infile) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(infile));
        String[] elts = new String[1]; // initial array has one element
        int size = 0; // number of filled slots in elts array.
        line-reading loop goes here: see next slide!
        reader.close();
        // Copy valid slots of elts into result
        String[] result = new String[size];
        for (int i = 0; i < size; i++) {
            result[i] = elts[i];
        }
        return result;
    } catch (IOException ex) {
        System.out.println(ex);
        return new String[0];
    }
}
```

05 - 11



Strategy 3 Code: Line-reading Loop

```
String line = reader.readLine(); // read the first line of the file.
while (line != null) { // line becomes null at end of file
    if (size == elts.length) { // array isn't big enough to hold next
        element.
        // create an array double the previous size.
        String[] dbl = new String[elts.length*2];
        // copy old contents to new array
        for (int i = 0; i < elts.length; i++) {
            dbl[i] = elts[i];
        }
        elts = dbl; // install new array as elts
    }
    elts[size] = line; // include last line
    size++; // increment size to reflect new line
    line = reader.readLine(); // read the next line of the file
}
```

05 - 12



Strategy 3: Timing Results

File size	fileToLinesArrayDbI() (milliseconds)
1000	4
2000	4
4000	3
8000	5
16000	27
32000	25

Expect linear growth for this strategy. But will only see it clearly for larger file sizes!

05 - 13



Strategy 4: Java Vectors

The [Java Vector class](#) (in the java.util package) encapsulates the array-doubling strategy into an array-like class parameterized by the slot type.

Constructor method:

```
public Vector<E> (); // E.g new Vector<String> ()
```

Instance Methods:

```
public void add (E elt);  
public void add (int index, E elt);  
public E get (int index);  
public E set (int index, E elt);  
public void clear();  
public boolean contains (Object o);  
public int indexOf (Object o);  
public void remove (int index);  
public void remove (Object o);
```

There are many other constructor and instance methods: see the API.

05 - 14



Strategy 4 Code

```
public static String[] fileToLinesVector (String infile) {
    try {
        BufferedReader reader = new BufferedReader(new FileReader(infile));
        Vector<String> vec = new Vector<String>();
        String line = reader.readLine(); // read the first line of the file.
        while (line != null) { // line becomes null at end of file
            vec.add(line);
            line = reader.readLine(); // read the next line of the file
        }
        reader.close();
        String[] result = new String[vec.size()];
        return vec.toArray(result); // Copies elements from vector to given array
        // and returns array.
    } catch (IOException ex) {
        System.out.println(ex);
        return new String[0];
    }
}
```

05 - 15



Strategy 4: Timing Results

File size	fileToLinesVector() (milliseconds)
1000	2
2000	3
4000	6
8000	6
16000	27
32000	31

05 - 16



StringBuffers

The [Java StringBuffer class](#) (in the java.lang package) encapsulates the array-doubling strategy into an String-like class

Constructor method:

```
public StringBuffer (); // E.g new StringBuffer()
```

Instance Methods:

```
public void append (String s);  
public void append (char c);  
public void append (int i);  
public void append (boolean b);  
... many other append() methods ...  
public String toString ();
```

There are many other constructor and instance methods: see the API.