Enumerations and Vectors

CS111 Lecture 18

Thursday, April 6, 2000

Enumeration Contract

A Java Enumeration is an abstract collection of Objects that can be enumerated one at a time until there are no more:

```
public  interface  java.util.Enumeration  {
```

```
public abstract boolean hasMoreElements();
```

Returns true if there are more elements in this enumeration, and false otherwise.

```
public abstract Object nextElement();
```

Returns the next Object in this enumeration. (Note: you must cast the result if it should have a more specific type.)

```
}
```

Interfaces

In Java, an **interface** is a “pure” contract that has no implementation. A class can only be a subclass of one other class, but it can implement arbitrarily many interfaces. E.g

```
public class OrderedSet
       extends SetImpl
       implements Sequence, Set
```

Listing the Words in a File

```
// An applet that prints out the words of a file in the order of their appearance followed by
// A word count.  Punctuation marks are considered to be words.
public class Words extends TextApplet {

  public void run () {
    String filename = WordEnumeration.chooseFilename();
    // State variables for iteration
    WordEnumeration words = WordEnumeration.fileToWords(filename);
    int count = 0;
    println("Processing file " + filename + "\n");
    println("Here are the words in the file in order of appearance:");
    println("----------------------------------------");
    while (words.hasMoreElements()) {
      println(words.nextElement());
      count = count + 1;  // Could also say count++
    }
    println("----------------------------------------");
    println("The file has " + count + " words");
  }
}
```

Vectors: What and Why?

In Java a Vector is an **extensible** indexed collection of objects.

- As with arrays access time to Vector slots is constant time.
- Unlike with arrays, the size of a Vector can change dynamically as objects are inserted or removed.
- As with ObjectList, every Vector element must be an Object. This implies lots of casting!
- We draw Vector instances just like arrays, except with the title “Vector” at the top.

Vector Contract

Below is a contract for a subset of Java's Vector class.

See the JDK 1.0.2 API for details.

```
public  class  java.util.Vector
{
  // Constructors
  public Vector();

  // Instance Methods
  public final void addElement(Object  obj);
  public final Object elementAt(int  index);
  public final Enumeration elements();
  public final void insertElementAt(Object  obj, int  index);
  public final void removeElementAt(int  index);
  public final void setElementAt(Object  obj, int  index);
  public final int size();
}
```

Listing the Distinct Words in a File

```
// An applet that prints the number of distinct words in the file,
// followed by a list of distinct words in dictionary order.
public class WordsDistinctSorted extends TextApplet {

  public void run () {
    String filename = WordEnumeration.chooseFilename();
    println("Processing file " + filename + "\n");
    // State variables for iteration
    WordEnumeration words = WordEnumeration.fileToWords(filename);
    Vector set = new Vector(); // set contains sorted sequence of strings seen so far.
    . . . Main loop goes here. See next slide . . .
    // Print results
    println("There are " + set.size() + " distinct words in the file:");
    println("----------------------------------------");
    Enumeration distinct = set.elements();
    while (distinct.hasMoreElements()) {
      println(distinct.nextElement());
    }
    println("----------------------------------------");
  }
}
```

Insertion Loop for Distinct Words Program

```
// This code belongs in the context of the previous slide.
// Insert all words into the set
while (words.hasMoreElements()) {
  String word = (String) words.nextElement();
  int index = StringVectorOps.binarySearchSorted(word, set);
  // Only insert word if its not already in set.
  if ((index == set.size())
     || (! (word.equals(set.elementAt(index))))) {
    set.insertElementAt(word, index);
  }
}
```

Linear Search of an Unsorted Vector of Strings

```
// If x is in vec, returns the least index at which vec appears.
// (There may be more than one.)
// If x is not in vec, returns the index at which x should be inserted.
// Use linear left-to-right search to find the index.
public static int linearSearchUnsorted(Object x, Vector vec) {
  for (int i = 0; i < vec.size(); i++) {
    if (x.equals(vec.elementAt(i))) {// Cast unnecessary for .equals()
      return i;
    }
  }
  // Only reach this point if x is not equal to any element in vec,
  // in which case insertion point is at end of vec.
  return vec.size();
}
```

Linear Search of a Sorted Vector of Strings

```
// Assume that string in vec are sorted from low to high
// according to the string compareTo() method.
// If x is in vec, returns the least index at which x appears.
// (There may be more than one.)
// If x is not in vec, returns the index at which x should be
// inserted in vec in sorted order.
// Use linear left-to-right search to find the index.
public static int linearSearchSorted(String x, Vector vec) {
  for (int i = 0; i < vec.size(); i++) {
    if (x.compareTo((String) vec.elementAt(i)) <= 0) {
      return i;
    }
  }
  // Only reach this point if x is greater than all other elements
  // in which case insertion point is at end of vec.
  return vec.size();
}
```

Binary Search of a Sorted Vector of Strings

```
// Assume that objects in vec are sorted from low to high
// according to the string compareTo() method.
// If x is in vec, returns an index at which vec appears.
//  (There may be more than one.)
// If x is not in vec, returns the index at which x should be
// inserted in vec in sorted order.
// Use binary search to find the index.
public static int binarySearch(String x, Vector vec) {
   int lo = 0;
   int hi = vec.size() - 1;
   // Loop invariants:
   // * All elements at indices < lo are less than x.
   // * All elements at indices > hi are greater than x.
   . . . Main loop goes here (see next slide) . . .
   // lo must be hi + 1 at this point.
   // By invariants, insertion point must be lo.
   return lo;
}
```

Binary Search: Main Loop

```
// This code belongs in the context of the previous slide.
while (lo <= hi) {
        int mid = (lo + hi) / 2;
        String midElt = (String) vec.elementAt(mid);
        int comp = x.compareTo(midElt);
        if (comp == 0) {
                return mid;
        } else if (comp < 0) {
                hi = mid - 1;
        } else { // (comp > 0)
                lo = mid + 1;
        }
}
```