Data Structures – Lab

November 2007

Road Map

- Vectors
- StringTokenizers
- Using Static Variables
- Using References with GUI Classes

Vectors

- Vectors (the **`java.util.Vector`** class) are commonly used instead of arrays, because they expand automatically when new data is added to them.
- Vectors can hold only Objects and not primitive types (eg, int).
- You must import either
  - **import** java.util.Vector; ... or,
  - **import** java.util.*;
- ArrayLists are unsynchronized and therefore faster than Vectors, but less secure in a multithreaded environment.

Vectors (cont.)

- To Create a Vector
  - Create a Vector with default initial size
    - Vector v = **new** Vector();
  - Create a Vector with an initial size
    - Vector v = **new** Vector(300);
- To Add elements to the end of a Vector
  - v.add(s); // adds s to the end of v

Vectors (cont.)

- To get the elements from a Vector
  - v.get(5); // gets the 5th element
  - v.elementAt(3); // gets the 3rd element
  - v.firstElement();
  - v.lastElement();
- To get the elements from a Vector (ListIterator)

```
ListIterator iter = v.listIterator();
while (iter.hasNext()) {
    System.out.println((String)iter.next());
}
```

StringTokenizers

- The **`java.util.StringTokenizer`** class is used to break strings into tokens (words, numbers, operators, or whatever).
- A more powerful solution is to use regular expressions, and the easiest way to do that is use the **`java.util.Scanner`** class, the **`String split(..)`** method, or the **`Pattern`** and **`Matcher`** classes.

StringTokenizers (cont.)

- A StringTokenizer constructor takes a string to break into tokens and returns a StringTokenizer object for that string.
- Each time its **`nextToken()`** method is called, it returns the next token in that string. If you don't specify the delimiters (separator characters), blanks are the default.

StringTokenizers (cont.)

- Constructors
  - StringTokenizer st = **new** StringTokenizer(s);
    - Creates a StringTokenizer for the String s that uses whitespace (blanks, tabs, newlines, returns, form feeds) as delimiters.
  - StringTokenizer st = **new** StringTokenizer(s, d);
    - Creates a StringTokenizer for the String s using delimiters from the String d.
  - StringTokenizer st = **new** StringTokenizer(s, d, f);
    - Creates a StringTokenizer for the String s using delimiters from the String d. If the boolean f is true, each delimiter character will also be returned as a token.

StringTokenizers (cont.)

- Common Methods

  Assume that st is a StringTokenizer.

  - st.hasMoreTokens()
    - Returns true if there are more tokens.
  - st.nextToken()
    - Returns the next token as a String.
  - st.countTokens()
    - Returns the int number of tokens. This can be used to allocate an array before starting, altho it can be inefficient for long strings because it has to scan the string once just to get this number. Using a Vector and converting it to an array at the end may be a better choice.

StringTokenizers (cont.)

- Example: Find the longest word in a String

```
// Assume s contains a string of words
String longestWord = "";
StringTokenizer st =  new StringTokenizer(s, " ,\t");
while (st.hasMoreTokens()) {
    String w = st.nextToken();
    if (w.length() > longestWord.length()) {
        longestWord = w;
    }
}
```

Using Static Variables

```
public class Box {
    double width;
    public static int numBoxes = 0; // static variable
      is                                          declared
      and initialized
    public Box() {
        width = 5.0;
        numBoxes++; // numBoxes is incremented to
          count
          number of objects.
    }
}
```

Using Static Variables (cont.)

```
public class TestStaticVar {
    public static void main (String args[]) {
        Box box1 = new Box();
        Box box2 = new Box();
        System.out.println("Number of objects = " +

          Box.numBoxes);
    }
}
```

You can't have static variables in a method

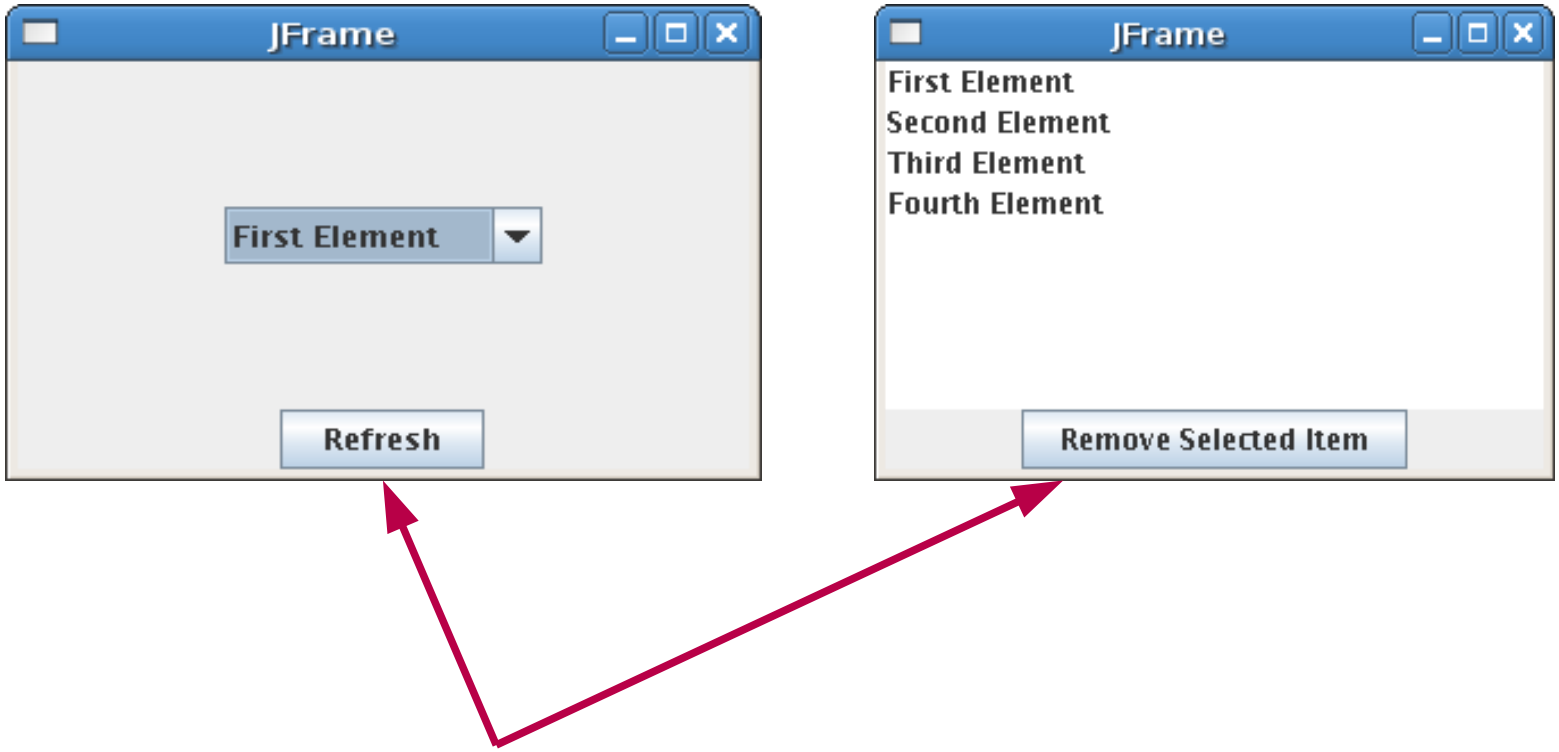
```
/**
 * Show that you can't have static variables in a method, unlike
   C/C++
 */
public class NoLocalStatics {
    public static void main(String[] argv) {
        NoLocalStatics nls = new NoLocalStatics();
        nls.process();
    }

    public void process() {
        static int a = 42;      // EXPECT COMPILE ERROR
        System.out.println("Process: " + a);
    }
}
```

Using References with GUI Classes

- Parameter Passing in Java
  - Pass by value
    - Primitive types and String
  - Pass by reference
    - All classes except String

Using References (cont.)



Vector vector = {“FirstElement”, “SecondElem
“ThirdElement”, “Fourth Element”}

Main.java

```
public static void main(String[] args) {
    Vector<String> vector = new Vector<String>();
    vector.add("First Element");
    vector.add("Second Element");
    vector.add("Third Element");
    vector.add("Fourth Element");

    ComboBoxFrame comboBoxFrame = new ComboBoxFrame();
    comboBoxFrame.setVector(vector);
    comboBoxFrame.show();

    ListFrame listFrame = new ListFrame();
    listFrame.setVector(vector);
    listFrame.show();
}
```

ListFrame.setVector(Vector)

```
public class ListFrame extends JFrame {
    ......
    public void setVector(Vector<String> vector) {
        this.vector = vector;
        DefaultListModel listModel = new
          DefaultListModel();
        Iterator<String> iterator = vector.iterator();
        while(iterator.hasNext()) {
            String nextItem = iterator.next();
            listModel.addElement(nextItem);
        }
        getJList().setModel(listModel);
    }
}
```