CSCI 136
Data Structures & Advanced Programming

Jeannie Albrecht

Lecture 6

Feb 21, 2014

Administrative Details

- Good job on Wed!
- Lab 2 is a little tricky (but fun)
- Graders will (hopefully) have Lab 1 graded by next Wednesday
- Let me know if TAs don't show up!

Lab 2

- Three classes:
  - Table.java
  - FrequencyList.java
  - WordGen.java
- Two Vectors of Associations
- toString() in Table and FrequencyList for debugging
- What are the key stages of execution?
  - Test code thoroughly before moving on to next stage
- Use WordFreq as example

Last Time

- Learned about Assertions and pre/post conditions
- Discussed Associations
  - Key-value pairs
  - General use class; keys and values are Objects

Review: Association Class

```
import structure5.*;
class Association {
   protected Object key;
   protected Object value;

   //pre: key != null
   public Association (Object K, Object V) {
        Assert.pre (K!=null, "Null key");
        key = K;
        value = V;
   }

   public Object getKey() {return key;}
   public Object getValue() {return value;}

   public Object setValue(Object V) {
        Object old = value;
        value = V;
        return old;
   }
}
```

Today's Outline

- Learn about Vectors
  - Dynamically resizable array
  - Easier to use (in most cases) than arrays
- How are Vectors implemented?

Recap: Dictionary.java (version 1)

```
protected Association words[] = new Association[5];
public Dictionary() {
   words[0] = new Association("perception", "Awareness of an
              object of thought");
   words[1] = new Association("person", "An individual capable of
              moral agency");
   words[2] = new Association("pessimism", "Belief that things
              generally happen for the worst");
   words[3] = new Association("philosophy", "Literally,
              love of wisdom.");
   words[4] = new Association("premise", "A statement whose
              truth is used to infer that of others");
}

// post: returns the definition of word, or "" if not found.
public String lookup(String word) {
     for (int i = 0; i < words.length; i++) {
         Association a = words[i];
         if (a.getKey().equals(word)) { //compare word to key
             // note cast to recover type from Object
             return (String)a.getValue(); //return value
          }
     }
     return "";
}
```

Recap: Dictionary.java (version 2)

```
protected Vector defs;
public Dictionary() {
   defs = new Vector();
}

public void addWord(String word, String def) {
   defs.add(new Association(word, def));
}

// post: returns the definition of word, or "" if not found.
public String lookup(String word) {
   for (int i = 0; i < defs.size(); i++) {
       Association a = (Association)defs.get(i);
       if (a.getKey().equals(word)) {
           return (String)a.getValue();
       }
   }
   return "";
}
```

Dictionary.java (version 2)

```
public static void main(String args[]) {
   Dictionary dict = new Dictionary();
   dict.addWord("perception", "Awareness of an object of
        thought");
   dict.addWord("person", "An individual capable of moral
        agency");
   dict.addWord("pessimism", "Belief that things generally
        happen for the worst");
   dict.addWord("philosophy", "Literally, love of
        wisdom.");
   dict.addWord("premise", "A statement whose truth is used to
        infer that of others");
}
```

Dictionary

```
protected Vector defs;
public Dictionary() {
   defs = new Vector();
}

public void addWord(String word, String def) {
   defs.add(new Association(word, def));
}

// post: returns the definition of word, or "" if not found.
public String lookup(String word) {
   Association a = (Association)defs.get(word);
   if (a != null)
        return (String)a.getValue();
   else
        return "";
}
```

← Does this work? No! We store Associations, not words…

Abstraction

- What "obvious" method is missing from Dictionary.java?
  - getWords() - returns Vector defs
- Why would we NOT include a getWords() method?
  - Exposes data representation to program
  - Hard to go back later and change the representation (i.e., to a List or HashMap)
- Abstraction: Don't expose any details about how Dictionary is implemented!

Importance of equals() and Vectors

- If we were implementing Vector.contains(myObject), what would we do?
  - Loop through elements and return true if one element equals myObject
- What does this require?
  - Properly defined equals() method in myObject class!

Notes About Vectors

- Primitive Types and Vectors

```
Vector v = new Vector();
v.add(5);
```

  - This (technically) shouldn't work! Can't use primitive data types with vectors…they aren't Objects!
  - (But Java is now smart about some data types, and converts them automatically for us -- called autoboxing)
- We used to have to "box" and "unbox" primitive data types:

```
Integer num = new Integer(5);
v.add(num);
…
Integer result = (Integer)v.get(0);
int res = result.intValue();
```

- Similar wrapper classes (Double, Boolean, etc) exist for all primitives

Vector Summary So Far

- Vectors: "extensible arrays" that automatically manage adding elements, removing elements, etc.
  1. Must cast Objects to correct type when removing from Vector
  2. Use wrapper classes (with capital letters) for primitive data types (use "Integers" not "ints")
  3. Define equals() method for Objects being stored if contains(), indexOf(), etc. is needed

Using Generic (or Parameterized) Data Types

- What limitations are associated with casting Objects as they are added and removed from Vectors?
  - Errors cannot be detected by compiler
  - Must rely on runtime errors
  - Compiler complains (with weird warnings)
- Instead of casting Objects, Java supports using generic or parameterized data types (Read Ch 4)
  - Instead of:

```
Vector v = new Vector();  //Vector of Objects
String word = (String)v.get(index);  //Cast to String
```

  - Say:

```
Vector<String> v = new Vector<String>(); //Vector of Strings
String word = v.get(index);  //no cast!
```

(Look at WordFreq.java with gen)

Implementing Vectors

- Vectors are really just arrays of Objects
- Key difference is that the number of elements can grow and shrink dynamically
- How are they implemented in Java?
  - What instance variables do we need?
  - What methods? (start simple)
- Constructor(s): `Vector()`, `Vector(size)`, `get(index)`, `set(index, Obj)`, `add(Obj)`, `add(index, Obj)`, `remove(index)`, `isEmpty()`, `size()` (we'll finish some of these next time!)