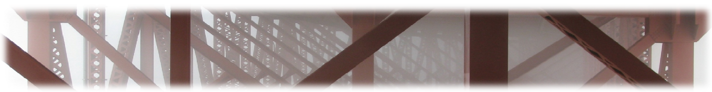
cs2220: Engineering Software

Class 7: Data Abstraction

Fall 2010
University of Virginia
David Evans



Menu

Data Abstraction

Specifying Abstract Data Types

PS2

Implementing Abstract Data Types

Managing Complexity

Procedural Abstraction

- **Divide** problem into procedures
- Use **specifications** to separate what from how

A big program can have thousands of procedures

Data Abstraction

Organize program around **abstract data types**

- **Group procedures** by the data they manipulate
- **Hide how data is represented** from how it is used

Abstract Data Types

Separate *what* you can do with data from *how* it is represented

Client interacts with data through provided operations according to their specifications

Implementation chooses how to represent data and implement its operations

What should the specification of a datatype do?

Specifying Abstract Data Types

Overview: what does the type represent

- **Mutability/Immutability**
  - e.g., A String is an immutable sequence of characters.
- **Introduce Abstract Notation**
  - e.g., A typical Set is $\{ x_1, \ldots, x_n \}$.

Operations: specifications for constructors and methods clients use

- Describe in terms of abstract notation introduced in overview.

Example: StringStack

Note: Java provides java.util.Stack,
but we'll implement our own Stack datatype.

public class StringStack
OVERVIEW: A StringStack represents a mutable last-in-first-out stack where all elements are Strings.
A typical stack is [e_n-1, e_n-2, ..., e_1, e_0] where e_n-1 is the top of the stack.

---

public class StringStack
OVERVIEW: A StringStack represents a mutable last-in-first-out stack where all elements are Strings.
A typical stack is [e_n-1, e_n-2, ..., e_1, e_0] where e_n-1 is the top of the stack.

public StringStack()
EFFECTS: Initializes this as an empty stack.

public void push(String s)
MODIFIES: this
EFFECTS: Pushes s on the top of this.
For example, if this_pre = [e_n-1, e_n-2, ..., e_1, e_0],
this_post = [s, e_n-1, e_n-2, ..., e_1, e_0]

public String pop() **throws** EmptyStackException
MODIFIES: this
EFFECTS: If this is empty, throws EmptyStackException. Otherwise, returns the element on top of this and removes that element from this.
For example, if this_pre = [e_n-1, e_n-2, ..., e_1, e_0],
this_post = [e_n-2, ..., e_1, e_0] and the result is e_n-1.

public String toString()
EFFECTS: Returns a string representation of this.

Components of Data Abstractions

Ways to create **new** objects of the type
- **Creators**: create new objects of the ADT from parameters of other types
- **Producers**: create new objects of the ADT from parameters of the ADT type (and other types)

Ways to observe properties: **observers**

Ways to change properties: **mutators**

Which of these must all (useful) types have?

---

public class StringStack
OVERVIEW: A StringStack represents a mutable last-in-first-out stack where all elements are Strings.
A typical stack is [e_n-1, e_n-2, ..., e_1, e_0] where e_n-1 is the top of the stack.

public StringStack() — **Constructor** / **Creator**
EFFECTS: Initializes this as an empty stack.

public void push(String s) — **Mutator**
MODIFIES: this
EFFECTS: Pushes s on the top of this.
For example, if this_pre = [e_n-1, e_n-2, ..., e_1, e_0],
this_post = [s, e_n-1, e_n-2, ..., e_1, e_0]

public String pop() **throws** EmptyStackException — **Observer and Mutator**
MODIFIES: this
EFFECTS: If this is empty, throws EmptyStackException. Otherwise, returns the element on top of this and removes that element from this.
For example, if this_pre = [e_n-1, e_n-2, ..., e_1, e_0],
this_post = [e_n-2, ..., e_1, e_0] and the result is e_n-1.

public String toString() — **Observer**
EFFECTS: Returns a string representation of this.

---

public class StringStack
OVERVIEW: A StringStack represents a mutable last-in-first-out stack where all elements are Strings.
A typical stack is [e_n-1, e_n-2, ..., e_1, e_0] where e_n-1 is the top of the stack.

public StringStack()
EFFECTS: Initializes this as an empty stack.

public void push(String s)
MODIFIES: this
EFFECTS: Pushes s on the top of this.
For example, if this_pre = [e_n-1, e_n-2, ..., e_1, e_0],
this_post = [s, e_n-1, e_n-2, ..., e_1, e_0]

public String pop() **throws** EmptyStackException
MODIFIES: this
EFFECTS: If this is empty, throws EmptyStackException. Otherwise, returns the element on top of this and removes that element from this.
For example, if this_pre = [e_n-1, e_n-2, ..., e_1, e_0],
this_post = [e_n-2, ..., e_1, e_0] and the result is e_n-1.

public String toString()
EFFECTS: Returns a string representation of this.

Using Abstract Data Types

- PS1, PS2
- Client interacts with data type using the methods as described in the specification
- Client does not know the concrete representation

Problem Set 2

Question 1, 2:
public static void sort(int[ ] a)

Specification A
From the Java SE 6 Platform API documentation:

Sorts the specified array of ints into ascending numerical order. The sorting algorithm is a tuned quicksort, adapted from Jon L. Bentley and M. Douglas McIlroy's "Engineering a Sort Function", Software-Practice and Experience, Vol. 23(11) P. 1249-1265 (November 1993). This algorithm offers n*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance. $\Theta(n^2)$

Parameters:
a – the array to be sorted.

Specification B
MODIFIES: a
EFFECTS: Rearranges the elements of a into ascending order.
e.g., if a = [3, 1, 6, 1],
a_post = [1, 1, 3, 6]

Shorter
Easy to see that a is modified
Declarative
Provides an Example
Doesn't overconstrain implementation

Might be a hint how code is guaranteed to perform: when you need to know about performance on some unknown JVM

Running Time

"This algorithm offers n*log(n) performance on many data sets that cause other quicksorts to degrade to quadratic performance."

Problems with this statement:

1. n is not defined (n = a.length)
2. "performance" is not a meaningful unit. Should be "running time in Θ(*n* log *n*) …"
3. many data sets?

Specifying Histogram

```
public static int [] histogram (int [] a)
{
  int maxval = 0;
  for (int i = 0; i < a.length; i++) {
    if (a[i] > maxval) {
      maxval = a[i];
    }
  }
  int histo [] = new int [
  for (int i = 0; i < a.leng
    histo[a[i]]++;
  }
  return histo;
}
```

Goals for a procedure specification:

1. **Declarative**
2. **Complete**
3. Clear, precise, unambiguous

Objective, attainable

Subjective, unattainable in English but we try!

REQUIRES: a is non-null
EFFECTS: Goes through the input array a, counting the number of times each element appears. Returns an array giving the histogram.

REQUIRES: a is non-null and all values in a are non-negative.
EFFECTS: Returns an array, result, where result[x] is the number of times x appears in a. The result array has maxval(a) + 1 elements. For example,
histogram ([1, 1, 2, 5]) = [0, 2, 1, 0, 0, 1]

Question 4: Remove Preconditions

REQUIRES: a is non-null and all values in a are non-negative.
EFFECTS: Returns an array, result, where result[x] is the number of times x appears in a. The result array has maxval(a) + 1 elements. For example,
histogram ([1, 1, 2, 5]) = [0, 2, 1, 0, 0, 1]

Remove the preconditions by using Exceptions:

public static int [] histogram (int [] a) **throws NegativeValue**
EFFECTS: If a contains any negative values, throws NegativeValue. If a is null, throws a NullPointerException. Otherwise, returns an array, result, … *(same as before)*

Question 5: Make it Total

REQUIRES: a is non-null and all values in a are non-negative.
EFFECTS: Returns an array, result, where result[x] is the number of times x appears in a. The result array has maxval(a) + 1 elements. For example,
histogram ([1, 1, 2, 5]) = [0, 2, 1, 0, 0, 1]

Total: a function that is defined for all inputs
In Java: produce an output, not an exception, for all inputs

public static int [] histogram (int [] a)
EFFECTS: If a is null, returns []. Otherwise, returns an array, result, where result[minValue(a) + x] is the number of times x appears in a and minValue(a) is the lowest value in a. The result array has maxValue(a) - minValue(a) + 1 elements. For example,
histogram ([1, 1, 2, 5]) = [2, 1, 0, 0, 1]
histogram ([-2, 0, 1, -2]) = [2, 0, 1, 1]

Is there a better solution?

Question 5: Make it Total

public static java.util.HashMap<Integer,Integer> histogram (int [] a)
EFFECTS: Returns a HashMap where the value associated with x is the result is the number of times x appears in a. That is, if result.containsKey (x) the number of appearances of x in a is result.get (x). Otherwise, the number of appearances of x in a is 0.

Question 6

Problem 6. Write a program that takes as input a list of file names and outputs a list of pairs of files **sorted** by the number of 3-length sequences they have in common.

```
// imports removed
public class CompareDocuments {
  public static void main(String[] args) {
    ArrayList<Document> docs = new ArrayList<Document> ();
    LabeledGraph g = new LabeledGraph();

    for (String file : args) {
      Document d;
      try {
        d = new Document(file, 3);
        docs.add(d);
        g.addNode(file);
      } catch (FileNotFoundException fnfe) { System.err.println("Error: cannot open file: " + file + " [" + fnfe + "]");
      } catch (DuplicateNodeException e) { System.err.println("Error: duplicate file: " + file); }
    }

    for (int i = 0; i < docs.size(); i++) {
      Set<String> keys = docs.get(i).keys();
      for (int j = i + 1; j < docs.size(); j++) {
        int similarity = 0;
        for (String key : keys) { if (docs.get(j).contains(key)) { similarity++; } }
        if (similarity > 0) {
          try {
            g.addEdge(docs.get(i).getName(), docs.get(j).getName(), similarity);
          } catch (NoNodeException e) { assert false;
          } catch (DuplicateEdgeException e) { assert false; }
        }
      } // for j
    } // for i

    ArrayList<EdgeRecord> edges = g.getSortedEdges();
    System.out.println ("Common Sequences: " + edges);
  }
}
```

This code is formatted densely to fit on one slide! Your code should be more spacious.

```
for (String key : keys) {
  if (docs.get(j).contains(key)) {
    similarity++;
  }
}
```

```
import ps2.*;
import java.io.FileNotFoundException;
import java.util.ArrayList;
import java.util.Set;

public class CompareDocuments {
  public static void main(String[] args) {
    int window = 3;
    ArrayList<Document> docs = new ArrayList<Document> ();
    LabeledGraph g = new LabeledGraph();

    for (String file : args) {
      Document d;
      try {
        d = new Document(file, window);
        docs.add(d);
        g.addNode(file);
      } catch (FileNotFoundException fnfe) {
        System.err.println("Error: cannot open file: " + file + " [" + fnfe + "]");
      } catch (DuplicateNodeException e) {
        System.err.println("Error: duplicate file: " + file);
      }
    }

    for (int i = 0; i < docs.size(); i++) {
      Set<String> keys = docs.get(i).keys();
      for (int j = i + 1; j < docs.size(); j++) {
        int similarity = 0;
        for (String key : keys) {
          if (docs.get(j).contains(key)) {
            // System.out.println(docs.get(i).getName() + " <-> " + docs.get(j).getName() + ": " + key);
            similarity++;
          }
        }
        if (similarity > 0) {
          try {
            g.addEdge(docs.get(i).getName(), docs.get(j).getName(), similarity);
          } catch (NoNodeException e) {
            assert false;
          } catch (DuplicateEdgeException e) {
            assert false;
          }
        }
      } // for j
    } // for i

    ArrayList<EdgeRecord> edges = g.getSortedEdges();
    System.out.println ("Common Sequences: " + edges);
  }
}
```