cs2220: Engineering Software

Class 5: Validation

Fall 2010
University of Virginia
David Evans

Menu

PS2 Questions

ArrayList

Software Validation

List Datatype

Lists in Scheme:

Either (1) null
or (2) a Pair whose second part is a List

What are the elements of a List?

Statically-Typed Lists

In Java, every variable must have a **statically-declared type**: the elements in a list can't just be "anything", we need to declare what type they are.

```
ArrayList<String> a = new ArrayList<String>();
a.add("Hello");
String  hello = a.get(0);
```

java.util.ArrayList is a ***Parameterized Type***

```
import java.util.ArrayList;
public class TypesExample {
public static void main(String[] args) {
    ArrayList<String> as = new ArrayList<String>();
    ArrayList<Object> ao = new ArrayList<Object>();
    ArrayList<ArrayList<String>> aas
        = new ArrayList<ArrayList<String>>();
    aas.add(as);
    ~~aas.add(ao);~~
    as.add("Hello");
    ~~ao.add("Hello");~~
    String el = as.get(0);
    ~~el = ao.get(0);~~
    el = aas.get(0).get(0);
    System.out.println(el);
  }
}
```

ArrayList a = new
ArrayList();

String []

Java Collection Types

java.util.**List<E>**

java.util.**ArrayList<E>**

Closest to Scheme and Python lists

java.util.**Set<E>**

java.util.**TreeSet<E>**

java.util.**HashMap<K, V>**

Similar to Python Dictionary type

Using **HashMap**

```
package ps2;
import java.util.HashMap;
import java.util.Set;
/**
 * TallyTable provides an abstraction that maps a String to an integer value.
 * Initially, the count associated with every string is 0.
 */
public class TallyTable {
   private HashMap<String,Integer> map;
   public TallyTable() { map = new HashMap<String,Integer>(); }
   public void tally(String w) { map.put(w, getTally(w) + 1); }
   public int getTally(String w) {
      if (map.containsKey(w)) { return map.get(w); }
      else { return 0; }
  }
   …
}
```

Validation

Dictionary Definition

val·i·date

1. To declare or make legally valid.
2. To mark with an indication of official sanction.
3. To establish the soundness of; corroborate.

Can we do any of these with software?

Java’s License

READ THE TERMS OF THIS AGREEMENT AND ANY PROVIDED SUPPLEMENTAL LICENSE TERMS (COLLECTIVELY "AGREEMENT") CAREFULLY BEFORE OPENING THE SOFTWARE MEDIA PACKAGE. BY OPENING THE SOFTWARE MEDIA PACKAGE, YOU AGREE TO THE TERMS OF THIS AGREEMENT. IF YOU ARE ACCESSING THE SOFTWARE ELECTRONICALLY, INDICATE YOUR ACCEPTANCE OF THESE TERMS BY SELECTING THE "ACCEPT" BUTTON AT THE END OF THIS AGREEMENT. IF YOU DO NOT AGREE TO ALL THESE TERMS, PROMPTLY RETURN THE UNUSED SOFTWARE TO YOUR PLACE OF PURCHASE FOR A REFUND OR, IF THE SOFTWARE IS ACCESSED ELECTRONICALLY, SELECT THE "DECLINE" BUTTON AT THE END OF THIS AGREEMENT.

Java’s License

5. **LIMITATION OF LIABILITY.** TO THE EXTENT NOT PROHIBITED BY LAW, IN NO EVENT WILL SUN OR ITS LICENSORS BE LIABLE FOR ANY LOST REVENUE, PROFIT OR DATA, OR FOR SPECIAL, INDIRECT, CONSEQUENTIAL, INCIDENTAL OR PUNITIVE DAMAGES, HOWEVER CAUSED REGARDLESS OF THE THEORY OF LIABILITY, ARISING OUT OF OR RELATED TO THE USE OF OR INABILITY TO USE SOFTWARE, EVEN IF SUN HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. …

Java’s License

2. **RESTRICTIONS.** … Unless enforcement is prohibited by applicable law, you may not modify, decompile, or reverse engineer Software. You acknowledge that Software is not designed, licensed or intended for use in the design, construction, operation or maintenance of any nuclear facility. Sun disclaims any express or implied warranty of fitness for such uses.

Software Validation

Process designed to *increase our confidence* that a program *works as intended*

For complex programs, guarantees are very unusual

This is why typical software licenses don't make any claims about their program working

Increasing Confidence

Testing

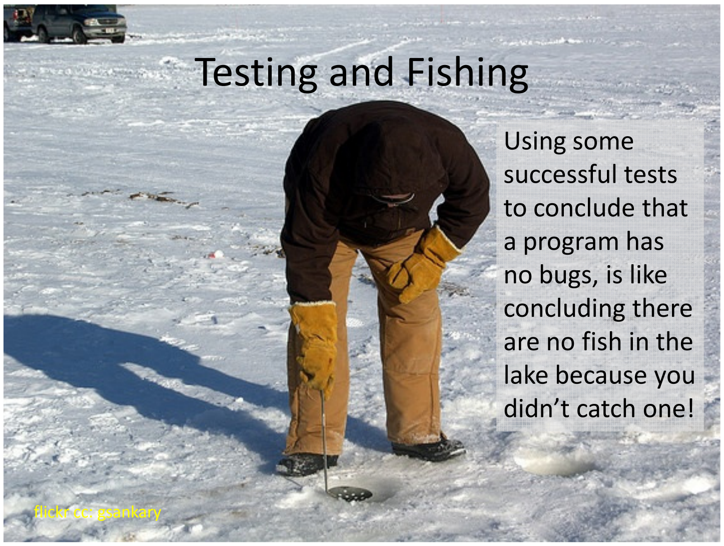
Run the program on set of inputs and check the results

Verification

Argue formally or informally that the program always works as intended

Analysis

Poor programmer's verification: examine the source code to increase confidence that it works as intended



Testing and Fishing

Using some successful tests to conclude that a program has no bugs, is like concluding there are no fish in the lake because you didn't catch one!

flickr cc: gsankary

Exhaustive Testing

Test **all possible inputs**

PS1: void accelerateSong(String tune, int repeats, int tempo, double rate)

How many inputs?

How many possible strings?

Integer.MAX_VALUE = 2^{31} - 1
Number of different characters (1 byte) = 2^8
Number of possible strings:

Selective Testing

We can't test everything, pick test cases with *high probability of finding flaws*

Black-Box Testing: design tests looking only at *specification*

Glass-Box Testing: design tests looking at *code*

Path-complete: at least one test to exercise each path through code

Black-Box Testing

from **WordWindow**

public void insert(String word)
REQUIRES: word does not contain a '/' character (this is necessary because **currentWindow** uses '/' to separate words in its result.
MODIFIES: this
EFFECTS: If **word** is non-null and non-empty, adds **word** as the newest element in this. If **this** already has size elements, removes the oldest element in **this**. If **word** is null or empty, does nothing.

Test all paths through the *specification*

Black-Box Testing

public void insert(String word)
REQUIRES: word does not contain a '/' character (this is necessary because **currentWindow** uses '/' to separate words in its result.
MODIFIES: this
EFFECTS: If **word** is non-null and non-empty, adds **word** as the newest element in this. If **this** already has size elements, removes the oldest element in **this**. If **word** is null or empty, does nothing.

Test all paths through the *specification*

1. Word is non-null and non-empty, this has size elements.
2. Word is non-null and non-empty, this has fewer than size elements.
3. Word is null.
4. Word is empty.

Black-Box Testing

public void insert(String word)
REQUIRES: word does not contain a '/' character (this is necessary because **currentWindow** uses '/' to separate words in its result.
MODIFIES: this
EFFECTS: If **word** is non-null and non-empty, adds **word** as the newest element in this. If **this** already has size elements, removes the oldest element in **this**. If **word** is null or empty, does nothing.

Test all paths through the *specification*

1. Word is non-null and non-empty, this has size elements.
2. Word is non-null and non-empty, this has fewer than size elements.
3. Word is null.
4. Word is empty.

Test boundary cases

1. this is empty

Glass-Box Testing

public void insert(String word)
REQUIRES: word does not contain a '/' character (this is necessary because **currentWindow** uses '/' to separate words in its result.
MODIFIES: this
EFFECTS: If **word** is non-null and non-empty, adds **word** as the newest element in this. If **this** already has size elements, removes the oldest element in **this**. If **word** is null or empty, does nothing.

Glass-Box Testing: determine test strategy and test cases based on examining the implementation code

How many paths are there through this code?

Glass-Box Testing

```
public void insert(String word) {
  if (word == null || word.length() == 0) {
     return;
  }
  assert !word.contains("/");
  words[index++] = word;
  if (index == words.length) index = 0;
  // System.out.println("Insert: " + word + " ==> " + currentWindow());
}
```

Note: there is no shame in using **println** statements to debug your code! Unless it is too much clutter, leave them in the code as comments.

How many paths are there through this code?

WordWindow Representation

```
public class WordWindow {
  // To avoid moving elements, we maintain an index into a fixed array, and
  // cycle through the array with each new element.
  private String words[]; // Array of the current words in the queue
  private int index;  // Index of the last element
  // INVARIANT: 0 <= i < words.length
```

```
public void insert(String word) {
  if (word == null || word.length() == 0) {
     return;
  }
  assert !word.contains("/");
  words[index++] = word;
  if (index == words.length) index = 0;
  // System.out.println("Insert: " + word + " ==> " + currentWindow());
}
```

Example: currentWindow

public String currentWindow()
EFFECTS: Returns a single String representation of the currentWindow which is the concatenation of all the words in order from oldest to newest, separated by '/' characters.

What would be good Black-Box test cases?

Example: currentWindow

```
public String currentWindow() {
  String res = "";
  boolean first = true;
  for (int i = index; i < index + words.length; i++) {
    if (first) {
      first = false;
    } else {
      res = res + "/";
    }
    String word = words[i % words.length];
    if (word != null) { // no word, just leave "/"s
      res = res + word;
    }
  }
  return res;
}
```

How paths tmany hrough this code?

Charge

Next class: Is it really hopeless?

PS2: Due Thursday

- My office hours: Wednesday, noon-1pm; Thursday, 11am-noon
- Robbie’s help hours: Wednesday, 2-3:30pm; 5-6:30pm

- For PS2, you should think about how to test your program (but it is not an explicit question for PS2)
- For PS3, you will need to describe a testing strategy