



6.170 Lecture 9 Understanding ADTs



John Guttag
MIT EECS




First of a Series of Lectures

Theme is rigorous reasoning
Will be rather formal
Don't expect you to use in practice
Understanding what is involved aids informal reasoning
Something all good programmers do in practice

Topics
Reasoning about implementations of types
Reasoning about uses of types
Reasoning about subtypes
Reasoning about loops (later)
Induction a recurring theme (later)

©Srinivas Devadas/John Guttag Spring 2002 Slide 2




Representation Independence

Does restricting access to rep ensure rep. independence?
NO !

Representation independence property of using code
Program using ADT should not depend impl. of ADT

©Srinivas Devadas/John Guttag Spring 2002 Slide 3




A Data Abstraction Is Defined by a Specification

A collection of procedural abstractions
Not a collection of procedures

Together, these procedural abstractions provide
A set of values
All the ways of directly using that set of values
Creating
Manipulating
Observing

Creators and producers create new values
Mutators change value (but don't affect =)
Observers allow one to tell values apart
The key to understanding

©Srinivas Devadas/John Guttag Spring 2002 Slide 4




Implementation of an ADT Provided by a Class

To implement a data abstraction
Select representation of instances, the *rep*
Implement operations in terms of that rep

Choose rep so that
It is possible (preferably easy) to implement operations
Most frequently used operations are efficient
But which will these be?
Abstraction allows changes to rep late in game

©Srinivas Devadas/John Guttag Spring 2002 Slide 5




What Is the Rep?

A data structure?
Not exactly

It's a data structure + a set of conventions

Conventions defined by
Rep invariant
Defines set of reachable values of the data structure
Abstraction function
How the data structure is to be interpreted
I.e., how it relates to the abstract values

©Srinivas Devadas/John Guttag Spring 2002 Slide 6




CharSet Abstraction

Overview: CharSets are finite sets of chars

```

public CharSet ()
  effects: creates a fresh, empty CharSet
public void insert (char c);
  modifies: this
  effects: this' = this U {c}
public void delete (char c);
  modifies: this
  effects: this' = this - {c}
public boolean member (char c);
  returns: (c ∈ this)
public int size ();
  returns: cardinality of this
    
```

©Srinivas Devadas/John Guttag Spring 2002 Slide 7



A CharSet Implementation ?


```

class CharSet {
  private Vector elts;
  CharSet () {elts = new Vector ();}
  public void insert (char c) { //SD's code
    Character cc = new Character (c);
    elts.addElement (cc);
  }
  public void delete (char c) { //JG's code
    elts.removeElement (new Character (c));
  }
  public boolean member (char c) {
    return elts.contains (new Character (c));
  }
  public int size () {return elts.size ();}
}
    
```

```

CharSet s;
s = new CharSet ();
s.insert ('a');
s.insert ('a');
s.delete ('a');
if (s.member ('a'))
  // print wrong;
else // print right;
    
```

©Srinivas Devadas/John Guttag Spring 2002 Slide 8



So, Where Is the Error?

An important question, since
Tells you what needs to be fixed


Perhaps delete is wrong
It should remove all occurrences

Perhaps insert is wrong
It should not insert a char that is already there

We have no way of knowing
Or do we?

What representation invariants are all about

©Srinivas Devadas/John Guttag Spring 2002 Slide 9



A Rep Invariant

Captures information that must be shared across implementations of multiple operations


The way I would write it

```

class CharSet {
  private Vector elts;
  //Rep inv: noDups(elts) & allChars(elts)
  ...
}
    
```

Or, if you are the pedantic sort
 \forall indices i, j of $elts$.
 $elts.elementAt(i).equals(elts.elementAt(j)) \Rightarrow i = j$
 $\& \forall$ elements e of $elts$. e instanceof $Character$

©Srinivas Devadas/John Guttag Spring 2002 Slide 10



Now, Whose Fault Is It ?

```

public void insert (char c) { //SD's code
  Character cc = new Character (c);
  elts.addElement (cc);
}


public void delete (char c) { //JG's code
  elts.removeElement (new Character (c));
}
    
```

SD's
Obviously

Because insert breaks the rep invariant

Reason about invariants using induction

©Srinivas Devadas/John Guttag Spring 2002 Slide 11




Induction

Induction will be a recurring tool this term

Way we think about things that are potentially infinite
Or so large that they might as well be infinite


Most interesting computations fall in that class

©Srinivas Devadas/John Guttag Spring 2002 Slide 12

 **A Review of Ordinary Induction**


Done with respect to a set of values, S
Start with a set of base values, B
Choose a set of operators, O , s.t. every value in S can be generated by applying the operators in O a finite number of times to the values in B
To prove that a property, $P: S \rightarrow \text{Boolean}$, holds
Base case: $\forall s \in B. P(s)$
Induction step: $P(s) \Rightarrow P(o(s))$, for all $o \in O$
Many other kinds of induction
Complete induction (later in term)
Engineers induction
Deprecated

©Srinivas Devadas/John Guttag Spring 2002 Slide 13

 **An Example**


$S = \mathbb{N}$, the set of natural numbers
 $B = 0$
 $O = \text{Succ}$
Axioms:
1) $\text{Succ}(0) > 0$
2) $(\text{Succ}(x) > \text{Succ}(y)) = x > y$
Prove that $\text{Succ}(x) > x$
Basis: $\text{Succ}(0) > 0$, by axiom 1
Induction: $(\text{Succ}(x) > x) \Rightarrow (\text{Succ}(\text{Succ}(x)) > \text{Succ}(x))$
 \downarrow (by axiom 2)
 $\text{Succ}(x) > x \Rightarrow \text{Succ}(x) > x$
Q.E.D.

©Srinivas Devadas/John Guttag Spring 2002 Slide 14

 **Induction, How It Works**


Need an ordering on all values of type
A finite number of least values
Prove property holds for all least values
Ordinary induction
Assume property holds for an arbitrary value
Prove it holds for all next values in ordering
For natural numbers
Ordering is distance from 0
Next is +1 (Succ)

©Srinivas Devadas/John Guttag Spring 2002 Slide 15

 **Induction, More Examples**


How about doing induction on integers?
Can still use distance from 0 as ordering
Next is +1 and -1
How about induction on real numbers?
Ordinary induction won't work
Could still choose 0 as least element
But no notion of next for enumerating rest of values
How about induction on vectors?
Can use length of vector as ordering
Will discover that this is quite useful

©Srinivas Devadas/John Guttag Spring 2002 Slide 16

 **Structural Induction (back to ADT's)**


Can be used on any abstract data type
Ordering is number of operations used to create value
Least values, created by a single call
I.e., values returned by creators
Next values, apply producers and mutators
Assume property holds for x
Prove that it holds for $x.o(\dots)$
Where no assumptions are made about arguments

©Srinivas Devadas/John Guttag Spring 2002 Slide 17

 **Finally, Reasoning About Rep Invariants**

Show that invariant holds on rep after each creator completes
Assume that invariant holds when other ops are called
Show that other ops preserve invariant
Why is this (usually) valid?
Type checking ensures that rep is modified only within implementation of type
Important consequence of modularity


©Srinivas Devadas/John Guttag Spring 2002 Slide 18

 **Proving Invariant for CharSet**

I = noDups(elts) & allChars(elts)

```
CharSet ( ) {  
  elts = new Vector ( );  
}  
public void delete (char c) {  
  elts.removeElement (new Character (c));  
}  
public void insert (char c) {  
  Character cc = new Character (c);  
  if (!this.member(c)) elts.addElement (cc);  
}  
public boolean member (char c) {  
  return elts.contains (new Character (c));  
}  
...  
}
```


©Srinivas Devadas/John Guttag Spring 2002 Slide 19

 **The Proof of I: noDups(elts) & allChars(elts)**

Base case
creator: elts = empty vector – clearly satisfies I

Induction step, assume I holds before other ops are called

©Srinivas Devadas/John Guttag Spring 2002 Slide 20


 **Induction Step, member**

I = noDups(elts) & allChars(elts)

```
public boolean member (char c) {  
  return elts.contains (new Character (c));  
}
```

member: doesn't change elts, so I preserved

©Srinivas Devadas/John Guttag Spring 2002 Slide 21


 **Induction Step, delete**

I = noDups(elts) & allChars(elts)

```
public void delete (char c) {  
  elts.removeElement (new Character (c));  
}
```

Either leaves elts unchanged or removes element
I can only be made false by adding elements
So rep invariant must be preserved

©Srinivas Devadas/John Guttag Spring 2002 Slide 22

 **Induction Step, insert**


I = noDups(elts) & allChars(elts)

```
public void insert (char c) {  
  Character cc = new Character (c);  
  if (!this.member(c)) elts.addElement (cc);  
}
```

If c in elts, elts unchanged and therefore I preserved.

If c not in elts, adding c does not introduce a duplicate, and c is a character, so I is still preserved
Q.E.D.


©Srinivas Devadas/John Guttag Spring 2002 Slide 23

 **A Question**

Q: Why did I have to look at member in proof ?
Specification says that it does not mutate set

A: Specification does not preclude modifying rep

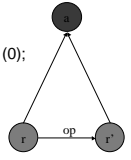
©Srinivas Devadas/John Guttag Spring 2002 Slide 24



Benevolent Side Effects

Variant of member op

```
boolean member (char c) {  
  Character c1 = new Character (c);  
  int i = elts.indexOf (c1);  
  if (i == -1) return false;  
  Character c2 = (Character) elts.elementAt (0);  
  elts.setElementAt (c1, 0);  
  elts.setElementAt (c2, i);  
  return true;  
}
```



Speeds up repeated membership tests

What's going on?
Mutates rep, but does not change *abstract* value

©Srinivas Devadas/John Guttag Spring 2002 Slide 25