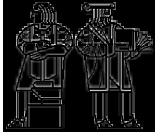


## 6.170 Lecture 13 Understanding ADTs (continued)



John Guttag  
MIT EECS

### Structural Induction Not Always Valid

#### Consider adding following to CharSet

```
Vector get_elts () {  
    //effects: returns a vector containing the members of this  
    return elts;}  
}
```

#### Now, consider the code

```
CharSet s = new CharSet ();  
s.insert ('a');  
s.get_elts().addElement ('a');  
s.delete ('a');  
if (s.member ('a')) ...
```

#### What we have here, is a case of

**INDEPENDENT  
EXPOSURE**

In the jargon, we have *exposed the rep*

© John Chapin/John Guttag

Spring 2000

Slide 2

### Exposing the Rep

This is almost always evil

If you do it, document why and how

It's not against the law

But it ought to be !



© John Chapin/John Guttag

Spring 2000

Slide 3

### Avoiding Rep Exposure

#### Exploit immutability

Is this safe?

```
Character choose () {  
    return (Character) elts.elementAt (0);  
}
```

Yes, because Character is immutable

#### Make a copy

Is this safe?

```
Vector get_elts () {return (Vector) elts.clone ();}
```

Yes, because mutating copy doesn't affect original

© John Chapin/John Guttag

Spring 2000

Slide 4

### Back to Rep Invariants

**Q:** Should code check that rep invariant holds?

**A1:** Yes, if inexpensive

**A2:** Yes, as debugging code (even when expensive)

© John Chapin/John Guttag

Spring 2000

Slide 5

### Checking the Rep Invariant

```
...  
public void delete (char c) {  
    if (!this.checkRepInv ())  
        System.out.println ("Rep Inv. Failure, start of CharSet.delete");  
    elts.removeElement (new Character (c));  
}
```

```
...  
private boolean checkRepInv () {  
    for (int i = 0; i < elts.size (); i++) {  
        if (!(elts.elementAt (i) instanceof Character)) return false;  
        if (!(elts.lastIndexOf (elts.elementAt (i)) == i))  
            return false;  
    }  
    return true;  
}
```

© John Chapin/John Guttag

Spring 2000

Slide 6

**6.170 Rep Invariant Not the Whole Story**

**Demonstrated to Professor Chapin the error of his ways**  
His implementation of insert failed to preserve invariant  
So, he went away and spent last night re-programming

```

public void insert (char c) //JC's code
  Character cc;
  cc = new Character (encrypt(c));
  if (!elts.contains(cc))
    elts.addElement(cc);
}

public boolean member (char c) //JG's code
  return elts.contains (new Character (c));
}

```

```

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

```

**Program does wrong thing**  
But, alas, once again cannot affix blame

© John Chapin/John Guttag Spring 2000 Slide 7

**6.170 The Rest of the Story**

**Abstraction function relates the concrete representation to the abstract value it represents**

**toCharSet(elts) = {c | c is contained in elts}**  
Typically not executable

**Combined with rep invariant**  
Allows us to examine operators independently  
"Correctness" now a local issue

**Once again we can safely place the blame**  
Applying the abstraction function to the result of the call to insert yields {encrypt(a)}  
Once again, JC is at fault

© John Chapin/John Guttag Spring 2000 Slide 8

**6.170 A Question**

**Q: Why do we map concrete to abstract rather than vice versa?**

**A: It's not a mapping in other direction.**  
E.g., [a,b] and [b,a] each represent the set {a, b}

© John Chapin/John Guttag Spring 2000 Slide 9

**6.170 Writing Abstraction Functions**

**Abstraction function needs to be defined properly on all representations that satisfy the rep invariant**

**Often a challenge**  
Whereas writing rep invariant usually easy

**Problem lies in denoting range of abstraction function**  
Not a problem for things like sets  
But what about *RoomReservation*?

**Overview section of specification the key**  
Ideally, provides way of writing values of abstract type  
Having print forms of abstractions valuable for debugging

© John Chapin/John Guttag Spring 2000 Slide 10

**6.170 Writing Abstraction Functions, An Example**

**Overview**  
A room reservation is a mapping from a room and a date to a reservation holder or "free," e.g., <34-101, 3/6/00> → 6.170 and <34-101, 2/21/00> → free

**Rep: rVector, dVector, hVector**

**Inv:**  
Vectors are of same length  
rVector contains only rooms, dVector only dates, and hVector only subjects or null  
if rVector[j] = rVector[k] & dVector[j] = dVector[k], then hVector[j] = hVector[k], i.e., it is a mapping

**Abstraction function:**  
<r, d> → h is in the mapping iff ∃ j s.t. rVector[j]=r, dVector[j]=d, and hVector[j]=h or h=free and hVector[j]=null

© John Chapin/John Guttag Spring 2000 Slide 11

**6.170 Writing Rep Invariants, reprised**

**Abstraction function sometimes convenient here**  
E.g., elts.lastIndex = |toCharSet(elts)|  
Where toCharSet is the abstraction function

**I'm not a big fan of this**  
Not obvious how to write checking code

© John Chapin/John Guttag Spring 2000 Slide 12

**6.170** **Key Points**

---

**Rep invariant**  
Which concrete values represent abstract values  
Use induction to show that is indeed an invariant

**Abstraction function**  
Which abstract value each concrete value represents

**Together, they modularize implementation**  
Can examine operators one at a time

**In practice**  
Rep invariant almost always worth writing  
Abstraction function harder to write and usually less useful

---

© John Chapin/John Guttag Spring 2000 Slide 13

**6.170** **Reasoning About Uses of ADT's**

---

**Structural induction works here too**  
**But look at specs, not code when reasoning**  
What I did when reasoning about natural numbers

**Basis -- values created by constructors**

**Ops -- mutators**

**Axioms -- specs of operations, and axioms of types used in overview parts of specifications**

---


© John Chapin/John Guttag Spring 2000 Slide 14

**6.170** **An Example**

---

Overview: HorseSets are finite sets of colored horses

```
public HorseSet ( )
  effects: creates a fresh, empty HorseSet
public void insert (Horse h);
  modifies: this
  effects: this' =
    if  $\exists h1 \in S$  s.t.  $h1.color = h.color$ 
      then this
    else this U {h}
public boolean member (horse h);
  effects: returns (h in this)
public int size ( );
  effects: returns |this|
```



---

© John Chapin/John Guttag Spring 2000 Slide 15

**6.170** **A Proof**

---

**Prove:**  $|S| > 1 \Rightarrow (\exists h1, h2 \in S [h1.color \neq h2.color])$

**Basis:**  $S = \{ \}$ , vacuously true

**Induction (on insert):**

**Assume:**  $|S| > 1 \Rightarrow (\exists h1, h2 \in S [h1.color \neq h2.color])$

**Show:**  $|S1| > 1 \Rightarrow (\exists h3, h4 \in S1 [h3.color \neq h4.color])$ ,  
where  $S1 =$  if  $\exists h5 \in S$  s.t.  $h5.color = h.color$  then  $S$  else  $S \cup \{h\}$

If  $S1 = S$ , then it holds by induction hypothesis

If  $S1 = S \cup \{h\}$ , there are three cases to consider

$|S| > 1$ : If  $S$  has horses of different colors,  
by the meaning of union, so does  $S \cup \{h\}$

$|S| = 0$ : Vacuous case, since hypothesis of thm. false

$|S| = 1$ : We know that  $S$  did not contain a horse of  $h.color$ ,  
so again we know thm. holds by meaning of union

---

© John Chapin/John Guttag Spring 2000 Slide 16