

6.170 Lecture 8

Representation invariants and abstraction functions



MIT EECS



What is an ADT?

It's more than just a data structure
data structure + a set of conventions

Specification: only in terms of the abstraction
Never mentions the representation

Representation invariant: Object \rightarrow boolean
Indicates whether a data structure is well-formed
Defines the set of valid values of the data structure

Abstraction function: Object \rightarrow abstract value
What the data structure means (as an abstract value)
How the data structure is to be interpreted
How do you compute the inverse, abstract value \rightarrow Object ?

MIT 6.170

Slide 2



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 make new values

Mutators change the value (but don't affect =)

Observers allow one to tell values apart
The key to understanding

MIT 6.170

Slide 3



Implementation of an ADT is Provided by a Class

To implement a data abstraction

Select the representation of instances, the *rep*
Implement operations in terms of that rep

Choose a representation so that

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

MIT 6.170

Slide 4



CharSet Abstraction

Overview: CharSets are finite sets of Characters

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

MIT 6.170

Slide 5




A CharSet Implementation ?

```
class CharSet {
  private ArrayList<Character> elts
  = new ArrayList<Character> ();
  public void insert (Character c) {
    elts.add (c);
  }
  public void delete (Character c) {
    elts.remove (c);
  }
  public boolean member (Character c) {
    return elts.contains (c);
  }
  public int size ( ) {
    return elts.size ();
  }
}
```

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

MIT 6.170

Slide 6



Where Is the Error?

The answer to this question 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 character that is already there

We have no way of knowing
Or do we?

Representation invariants tell us

MIT 6.170 Slide 7



A Representation Invariant

Captures information that must be shared across implementations of multiple operations


Write it this way:

```
class CharSet {
  private ArrayList<Character> elts;
  // Rep invariant: elts has no nulls and no duplicates
  ...
}
```

Or, if you are the pedantic sort:

- ∀ indices i of elts . elts.elementAt(i) != null
- ∀ indices i, j of elts .
i ≠ j ⇒ ! elts.elementAt(i).equals(elts.elementAt(j))

MIT 6.170 Slide 8




Now, where is the error?

```
public void insert (Character c) {
  elts.add (c);
}

public void delete (Character c) {
  elts.remove (c);
}
```

MIT 6.170 Slide 9




Rep invariant for Account

```
class Account {
  private List<Transaction> transactions;
  private int balance;
  ...
}
```

// real-world constraints
balance ≥ 0
balance = Σ_i transactions.get(i).amount
// implementation-related constraints
transactions != null
no nulls in transactions

MIT 6.170 Slide 10



Listing the elements of a CharSet


Consider adding the following method to CharSet
// returns: an ArrayList containing the members of this
public ArrayList<Character> getElts ();

Consider this implementation:
public ArrayList<Character> getElts () { return elts; }

Recall rep invariant: elts has no nulls and no duplicates

The implementation of getElts preserves the rep invariant
... sort of

MIT 6.170 Slide 11



Representation exposure

Consider the client code (outside implementation of the type)

```
CharSet s = new CharSet ( );
Character a = new Character('a');
s.insert (a);
s.getElts( ).add (a);
s.delete (a);
if (s.member (a)) ...
```

This is almost always evil

If you do it, document why and how
And feel guilty about it

MIT 6.170 Slide 12

6.170 Avoiding Rep Exposure

Exploit immutability

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

Character is immutable

Make a copy

```
ArrayList<Character> getElts () {
    return new ArrayList<Character>(elts);
    // or: return (ArrayList<Character>) elts.clone ();
}
```

Mutating a copy doesn't affect the original

MIT 6.170 Slide 13

6.170 Checking rep Invariants

Should code check that rep invariant holds?

Yes, if inexpensive

Yes, as debugging code (even when expensive)

It's quite hard to justify turning the checking off

Private methods need not check

MIT 6.170 Slide 14

6.170 Checking the Rep Invariant

```
public void delete (Character c) {
    checkRep ();
    elts.remove (c)
    // Is this guaranteed to get called?
    // See handouts for a less error-prone way to check at exit
    checkRep ();
}
...
/** Verify that elts contains no duplicates */
private void checkRep () {
    for (int i = 0; i < elts.size (); i++) {
        assert elts.indexOf(elts.elementAt(i)) == i;
    }
}
```

An alternative implementation: repOK() returns a boolean, and callers of repOK must check its return value

MIT 6.170 Slide 15

6.170 Practice Defensive Programming

Assume that you will make mistakes

Write and incorporate code designed to catch them

On entry:

- Check rep invariant
- Check preconditions (requires clause)

On exit:

- Check rep invariant
- Check postconditions

Checking the rep invariant helps you **discover** errors

Reasoning about the rep invariant helps you **avoid** errors

Or prove that they do not exist!

We will discuss such reasoning, later in the semester

MIT 6.170 Slide 16

6.170 Benevolent Side Effects

Different implementation of member:

```
boolean member (Character c1) {
    int i = elts.indexOf (c1);
    if (i == -1) return false;
    // move-to-front optimization
    Character c2 = elts.elementAt (0);
    elts.setElementAt (c1, 0);
    elts.setElementAt (c2, i);
    return true;
}
```

Speeds up repeated membership tests

Mutates rep, but does not change abstract value

AF maps both reps to the same abstract value

MIT 6.170 Slide 17

6.170 Rep Invariant constrains structure, not meaning

New implementation of insert that preserves invariant


```
public void insert (Character c) {
    Character cc = new Character(encrypt(c));
    if (!elts.contains(cc))
        elts.addElement(cc);
}
public boolean member (Character c) {
    return elts.contains (c);
}
```

Program still does wrong thing

Where is the error?

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

MIT 6.170 Slide 18



Abstraction functions

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


AF: Object \rightarrow abstract value

AF(CharSet this) = { c | c is contained in this.elts }
 “set of Characters contained in this.elts”
 Typically not executable

Combined with rep invariant
 Allows us to examine operators independently
 “Correctness” is now a local issue

Once again we can place the blame
 Applying the abstraction function to the result of the call to insert yields $\text{AF}(\text{elts}) \cup \{\text{encrypt}('a')\}$
 What if we used this abstraction function?
 { c | decrypt(c) is contained in this.elts }

MIT 6.170 Slide 19




The abstraction function is a function

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

It's not a function in the other direction.
 E.g., lists [a,b] and [b,a] each represent the set {a, b}

It's not as useful in the other direction.
 Can construct objects via the provided operators

MIT 6.170 Slide 20



Writing Abstraction Functions


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

Often a challenge
 Whereas writing the rep invariant is usually easy

Problem lies in denoting range of abstraction function
 Not a problem for mathematical entities like sets
 For more complex abstractions, give them fields
 AF defines the value of each “specification field”

Overview section of the specification is the key
 Ideally, provides way of writing values of abstract type
 Having a printed representation is valuable for debugging

MIT 6.170 Slide 21



Key Points

Rep invariant
 Which concrete values represent abstract values

Abstraction function
 Which abstract value each concrete value represents

Together, they modularize the implementation
 Can examine operators one at a time
 Neither one is part of the abstraction (the ADT)

In practice
 Always write a representation invariant
 Write an abstraction function when you need it
 Write an informal one for most non-trivial classes
 A formal one is harder to write and usually less useful

MIT 6.170 Slide 22