Lecture 7
Abstraction Functions

Leah Perlmutter / Summer 2018
Announcements
Announcements

- HW2 due tonight 10 pm

- Wednesday, July 4 is Independence Day 🇺🇸
  - No lecture

- Section Thursday, July 5

- HW3 due Thursday, July 5 at 10 pm
  - Seek HW3 help on Tuesday; no office hours Wednesday!

- Reading 3 posted on website
  - Quiz 3 (coming soon!) due Thursday, July 5 at 10 pm
Motivation
Review

lec04
Method Specification (abstraction)

lec05
Abstract Data Type (abstraction)

IMPLEMENTS
Method Body (concrete code)

IMPLEMENTS
Data Structure (concrete code)
Example: CharSet Abstraction

// Overview: A CharSet is a finite mutable set of Characters
// @effects: creates a fresh, empty CharSet
public CharSet() {...}

// @modifies: this
// @effects: this\textsubscript{post} = this\textsubscript{pre} + \{c\}
public void insert(Character c) {...}

// @modifies: this
// @effects: this\textsubscript{post} = this\textsubscript{pre} - \{c\}
public void delete(Character c) {...}

// @return: (c ∈ this)
public boolean member(Character c) {...}

// @return: cardinality of this
public int size() {...}
Charset Representation Invariant

class CharSet {
    // Rep invariant:
    //   this.elts has no nulls and no duplicates
    private List<Character> elts = ...
    ...
}

An implementation of `insert` that preserves the rep invariant:

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

Program is wrong

- Clients observe incorrect behavior
- What client code exposes the error?
- Where is the error?
- We must consider the `meaning`
- The `abstraction function` helps us
An ADT has an abstract value

Abstract Value: An Int List is a finite sequence of integer values

size: 4
head
Integer(1)
 Integer(2)
 Integer(42)
 Integer(17)
null

size: 3
head
Integer(1)
 Integer(2)
 Integer(42)
 Integer(17)
null

size: 0
head
null

1, 2, 42, 17
?
??????
Connecting implementations to specs

**Representation Invariant**: maps Object → boolean
- Indicates if an instance is *well-formed*
- Defines the set of valid concrete values
- Only values in the valid set make sense as implementations of an abstract value
- **For implementors/debuggers/maintainers of the abstraction:** no object should ever violate the rep invariant
  - Such an object has no useful meaning

**Abstraction Function**: maps Object → abstract value
- What the data structure *means* as an abstract value
- How the data structure is to be interpreted
- Only defined on objects meeting the rep invariant
- **For implementors/debuggers/maintainers of the abstraction:** Each procedure should meet its spec (abstract values) by “doing the right thing” with the concrete representation
Functions
Set

• An unordered collection of objects
  \[ S = \{3, 1, 2, \text{mouse}\} \]
• An object can be in the set or not
  \[ 3 \in S \quad -1 \notin S \]
• Set builder notation
  \[ T = \{ x \mid x \in S \text{ and } x \text{ is an integer} \} \]
  \[ = \{ 2, 1, 3 \} \]
• Some familiar sets
  \[ \mathbb{Z} = \{ \ldots -1, 0, 1, 2, \ldots \} \quad \text{“the integers”} \]
  \[ \mathbb{Q} = \{ p/q \mid p, q \in \mathbb{Z} \} \quad \text{“the rational numbers”} \]
Function

- A relation that uniquely associates members of one set with members of another set [Wolfram]
  
  \( F : S \rightarrow Y \)  “F maps S to Y”

Range: \{animal, number\}
Example Function

\[ F : \mathbb{R} \rightarrow \mathbb{R} \]
\[ F(x) = x^2 \]

passes vertical line test
Example NOT Function

Inverse of $F(x) = x^2$

$y = \pm \sqrt{x}$

$\sqrt{25} = 5$

$\sqrt{25} = -5$

Does not pass vertical line test – Not a function!
Functions in Math and Programming

- In programming, the term “function” is often loosely used.
- Related to the concepts of “method” and “subroutine”

```java
float square(float x) {
    return x * x;
}
```
This method implements a mathematical function.

```java
void greet(String name) {
    System.out.println("Hello, " + name);
}
```
This method does not implement a mathematical function.
Abstraction
Functions
Abstraction Function

The abstraction function maps concrete representations to the abstract values they represent

\[
AF: \text{concrete rep} \rightarrow \text{abstract value}
\]

\[
AF(\text{CharSet this}) = \{ c | c \text{ is contained in this.elts} \}
\]

“set of Characters contained in this.elts”

– The abstraction function lets us reason about what [concrete] methods do in terms of the clients’ [abstract] view
  • Makes sure that all methods use the rep in the same way
– Math concept of function, not programming concept of function
  • AF not implementable in code since range is abstract values
Abstraction Function

Values allowed by the Data Structure – all concrete values

Well Formed Values – concrete values that have a corresponding abstract value Rep Invariant Holds
Abstraction Function

Domain
- All concrete values
- Well Formed concrete values

Codomain
- All Abstract Values

Range
- Concretely Representable Abstract Values
Abstraction Function

**Domain**
- Well Formed concrete values
  - Integer(1)
  - Integer(2)
  - Integer(42)
  - Integer(17)
  - null

**Codomain**
- All Abstract Values
- Concretely Representable Abstract Values

**Range**
- All concrete values

size: 4
head

1, 2, 42, 17
Abstraction Function

- **Domain**
  - Well Formed concrete values
  - Integer(2)
  - Integer(42)
  - Integer(17)

- **Codomain**
  - All Abstract Values
  - Concretely Representable Abstract Values

- **Range**
  - null
  - size: 0
  - head
Abstraction Function

All concrete values

Well Formed concrete values

Domain

Codomain

All Abstract Values

Concretely Representable Abstract Values

Range

0, 1, 2^{10,000}
Summary so far:

The abstraction function maps concrete representations to the abstract values they represent

AF: concrete rep $\rightarrow$ abstract value

- Well Formed concrete values
- Concretely Representable Abstract Values
The abstraction function is a function

Why do we map concrete to abstract and not vice versa?

• It’s not a function in the other direction
  – Example: lists [a, b] and [b, a] might each represent the set \{a, b\}

• It’s not as useful in the other direction
  – Purpose is to reason about whether our methods are manipulating concrete representations correctly in terms of the abstract specifications
Writing an abstraction function

Domain: all representations that satisfy the rep invariant
Range: concretely representable abstract values

Overview section of the specification should provide a notation of writing abstract values
  – Could implement a method for printing in this notation
    • Useful for debugging
    • Often a good choice for toString
Abstraction Function and Stack

/** A last-in, first-out stack. A typical stack is e0, e1, ... en where en is the top element of the stack and is most recently pushed and first available to be popped. */

public class Stack {
   // Rep invariant:
   // 0 <= this.top <= this.a.length
   // this.a != null
   // Abstraction Function:
   // AF(this) = A last-in, first-out stack defined by an ordered sequence of integers
   // this.a[0] ... this.a[this.top-1]
   // where the rightmost integer in the sequence is at the top of the stack
   private int[] a;
   private int top;
   ...
}
Stack AF example

recall: \( \text{top} \) points to the array element just after the top of the stack

<table>
<thead>
<tr>
<th>new()</th>
<th>0</th>
<th>0</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>push(17)</td>
<td>17</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>push(-9)</td>
<td>17</td>
<td>-9</td>
<td>0</td>
</tr>
<tr>
<td>pop()</td>
<td>17</td>
<td>-9</td>
<td>0</td>
</tr>
</tbody>
</table>

Abstract states are the same
\[ 17 = 17 \]

Concrete states are different
\[ \langle [17,0,0], \text{top}=1 \rangle \neq \langle [17,-9,0], \text{top}=1 \rangle \]

AF is a function
Inverse of AF is not a function
Benevolent side effects

Different implementation of `member`:

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

- Move-to-front speeds up repeated membership tests
- Mutates rep, but does not change abstract value
  - *AF maps both reps to the same abstract value*
  - Precise reasoning/explanation for “clients can’t tell”
Abstract and Concrete operations

Abstract Object → Abstract Operation → Abstract Object’

Concrete Object ← Concrete Operation ← Concrete Object’

AF
Abstraction Function and Charset

The AF tells us what the rep means...

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

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

The two methods assume different abstraction functions! BAD!!!
Charset Abstraction Function

class CharSet {
    // Rep invariant:
    //   this.elts has no nulls and no duplicates
    // Abstraction Function:
    //   AF(this) = { c | c is contained in this.elts }
    private List<Character> elts = ...
    ...
}

- Defined in terms of the representation (this.elts)
- Internal comment (not javadoc)
  - located just inside of the class definition at the very beginning
  - internal comment: located just inside of the class definition at the very beginning

- Now we can re-implement insert to respect the AF
Data Abstraction: Summary

**Representation Invariant** describes what makes the concrete representation valid (green area)

**Abstraction Function** maps valid concrete values to abstract values

- Neither one is part of the ADT’s specification
- Both are needed to reason an implementation satisfies the specification
Closing
Closing Announcements

• HW2 due tonight 10 pm
• HW3 due Thursday, July 5 at 10 pm
• Quiz 3 (coming soon!) due Thursday, July 5 at 10 pm

• Happy Independence Day! 🇺🇸