
CSE 331

Software Design & Implementation

Kevin Zatloukal

Fall 2020

ADT Implementation: Abstraction Functions

Specifying an ADT

Different types of methods:

1. **creators**
2. **observers**
3. **producers**
4. **mutators** (if mutable)

Described in terms of how they change the **abstract state**

- abstract description of what the object means
- specs have no information about concrete representation
 - leaves us free to change those in the future

IntSet, a mutable data type

```
// Overview: An IntSet is a mutable,  
// unbounded set of integers.  A typical  
// IntSet is { x1, ..., xn }.  
class IntSet {
```

(Note: Javadoc is highly simplified...)

IntSet: mutators

```
// modifies: this  
// effects:  this = this U {x}  
public void add(int x)
```

```
// modifies: this  
// effects:  this = this - {x}  
public void remove(int x)
```

Specifications written in terms of how the **abstract state** changes

Useful Building Blocks for Abstract States

Some useful “math” concepts for describing states abstractly

- numbers
- characters
- lists
- sets
- tuples (fixed length)
- objects
 - parts are named, not numbered (as in tuples)
 - e.g. {chars: “protected”, color: 3}

Implementing a Data Abstraction (ADT)

To implement an ADT:

- select the representation of instances
- implement operations using the chosen representation

Choose a representation so that:

- it is possible to implement required operations
- the most frequently used operations are efficient / simple / ...
 - abstraction allows the rep to change later
 - almost always better to start simple

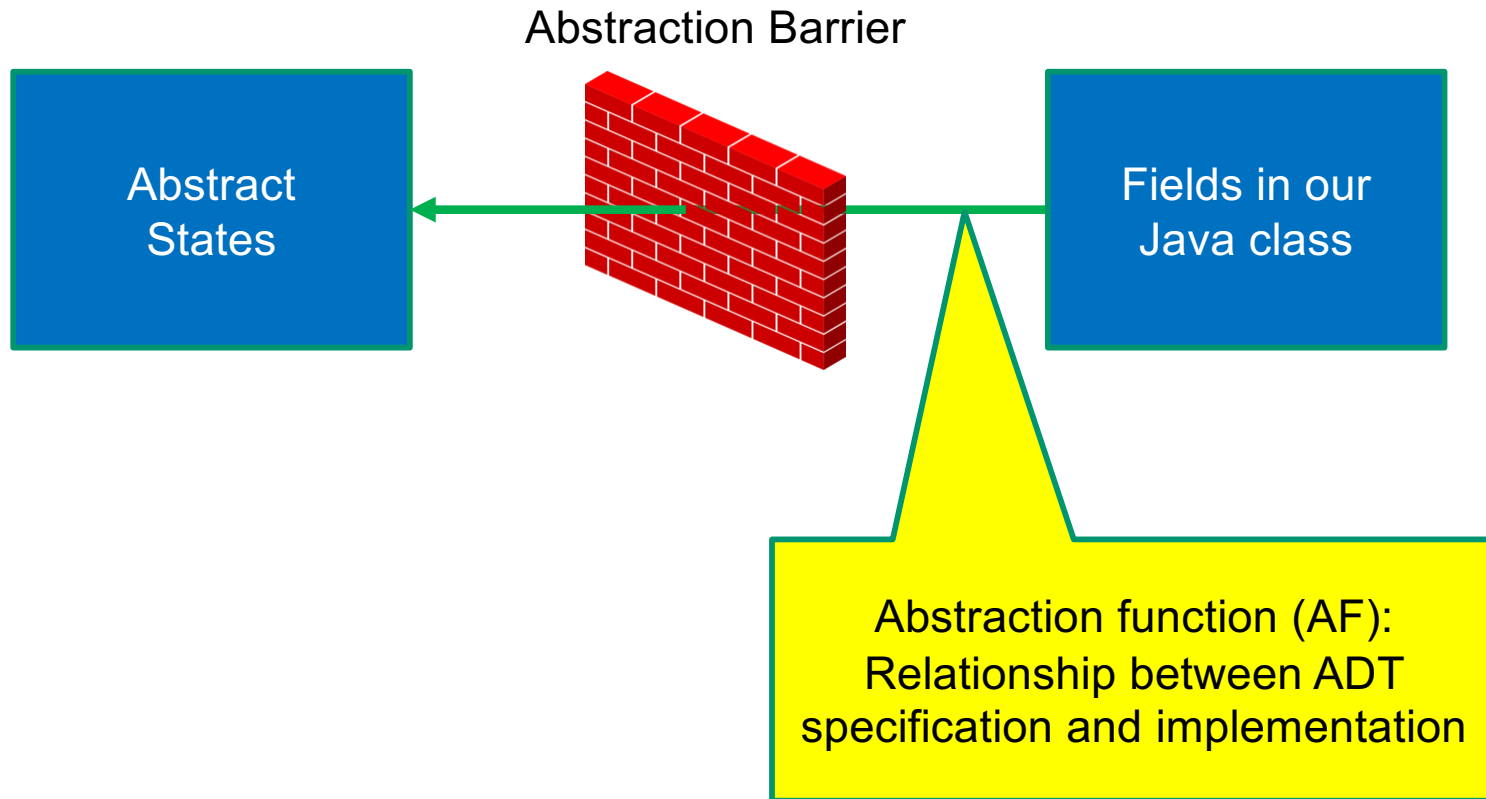
Use **reasoning** to verify the operations are correct

- specs are written in terms of *abstract states* not *actual fields*
- two intellectual tools are helpful for this...

Data abstraction outline

ADT specification

ADT implementation



Connecting implementations to specs

For implementers / debuggers / maintainers of the implementation:

Abstraction Function: maps Object \rightarrow abstract state

- says what the data structure *means* in vocabulary of the ADT
- maps the fields to the abstract state they represent
 - can check that the abstract value after each method meets the postcondition described in the specification

Representation Invariant: (next lecture)

Example: Circle

```
/** Represents a mutable circle in the plane. For example,  
 * it can be a circle with center (0,0) and radius 1. */  
public class Circle {  
  
    // Abstraction function:  
    // AF(this) = a circle with center at this.center  
    //    and radius this.rad  
    private Point center;  
    private double rad;  
  
    // ...  
  
}
```

Example: Circle 2

```
/** Represents a mutable circle in the plane. For example,  
 * it can be a circle with center (0,0) and radius 1. */  
public class Circle {  
  
    // Abstraction function:  
    // AF(this) = a circle with center at this.center  
    //   and radius this.center.distanceTo(this.edge)  
    private Point center, edge;  
  
    // ...  
  
}
```

Example: Polynomial

```
/** An immutable polynomial with integer coefficients.
 * Examples include 0, 2x, and x + 3x^2 + 5x. */
public class IntPoly {

    // Abstraction function:
    // AF(this) = sum of coeffs[i] * x^i
    //           for i = 0 .. coeffs.length-1
    private final int[] coeffs;

    // ...

}
```

Example: Polynomial 2

```
/** An immutable polynomial with integer coefficients.
 * Examples include 0, 2x, and x + 3x^2 + 5x. */
public class IntPoly {

    // Abstraction function:
    // AF(this) = sum of monomials in this.terms
    private final List<IntTerm> terms;

    // ...

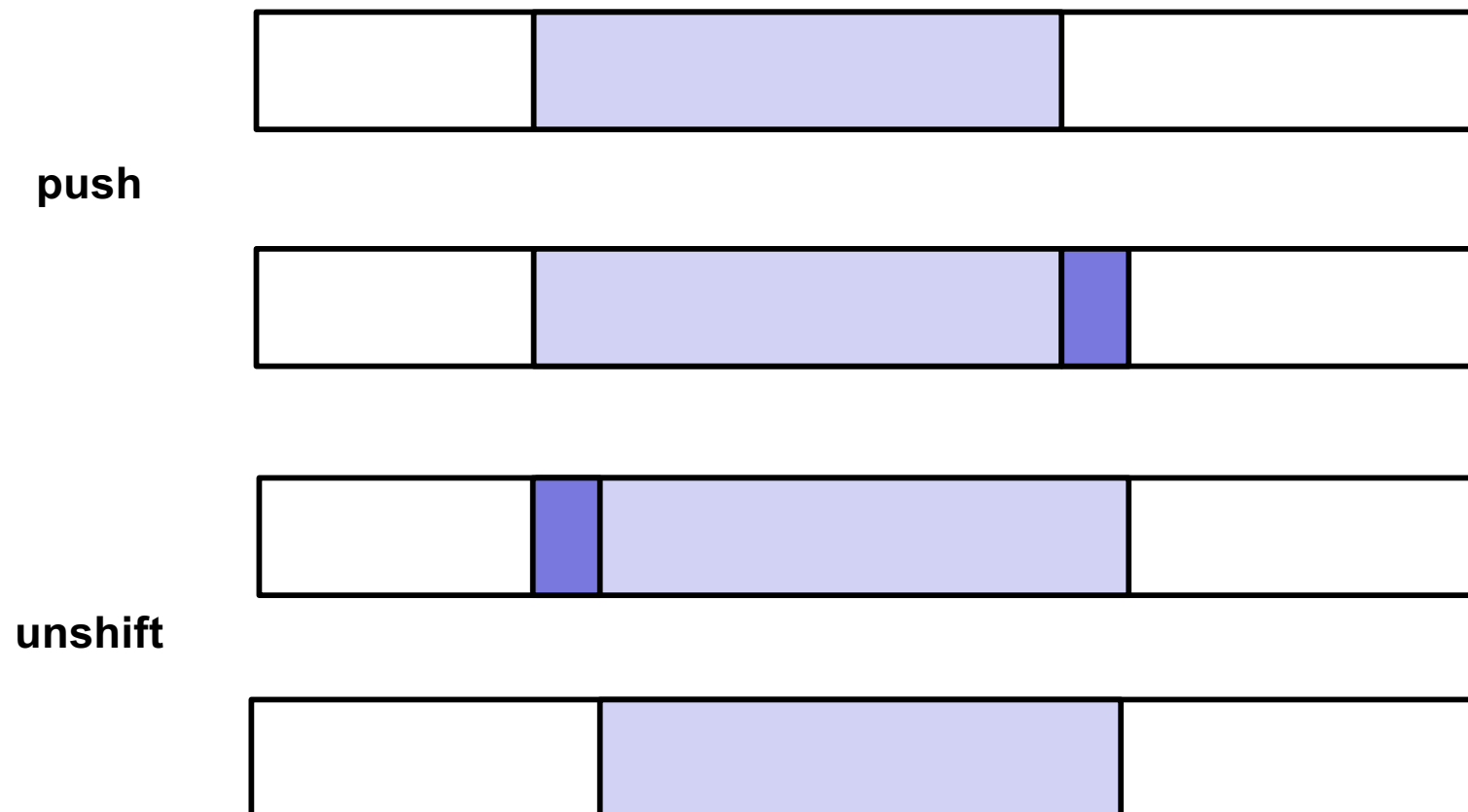
}
```

The abstraction function

- Purely conceptual (not a Java function)
- Allows us to check correctness
 - use reasoning to show that the method leaves the abstract state such that it satisfies the postcondition

Example: IntDeque

// List that only allows insert/remove at ends.



Example: IntDeque

// List that only allows insert/remove at ends.



push + unshift



push + unshift

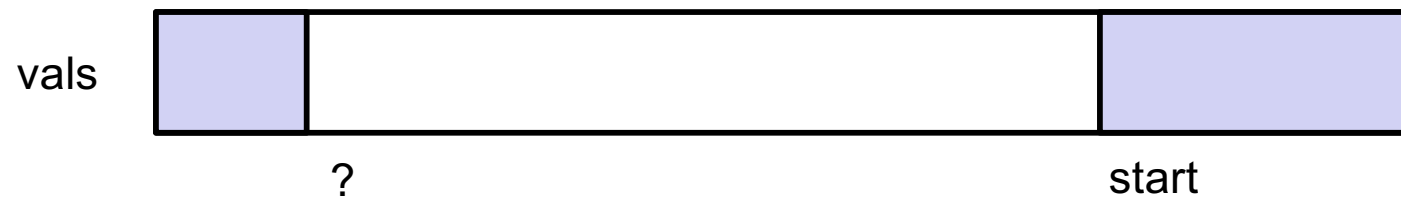


push + unshift



Example: IntDeque

// List that only allows insert/remove at ends.



Example: IntDeque

```
/** List that only allows insert/remove at ends. */
public class IntDeque {

    // AF(this) =
    //   vals[start..start+len-1]      if start+len < vals.length
    //   vals[start..] + vals[0..len-(vals.length-start)-1]  o.w.
    private int[] vals;
    private int start, len;

    // Creates an empty list.
    public IntDeque() {
        vals = new int[3];
        start = len = 0;
    }
}
```

← AF(this) = vals[0..-1] = []

Example: IntDeque

```
/** List that only allows insert/remove at ends. */
public class IntDeque {

    // AF(this) =
    //   vals[start..start+len-1]      if start+len < vals.length
    //   vals[start..] + vals[0..len-(vals.length-start)-1]  o.w.
    private int[] vals;
    private int start, len;

    // ...

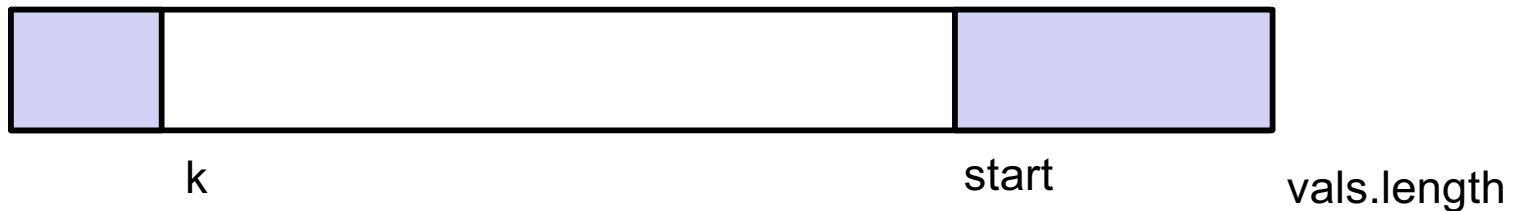
    // @returns length of the list
    public int getLength() {
        return len;
    }
}
```

Example: IntDeque

// List that only allows insert/remove at ends.



#items = len



#items = vals.length - start + k

#items = len iff $k = len - (vals.length - start)$

Example: IntDeque

```
/** List that only allows insert/remove at ends. */
public class IntDeque {

    // AF(this) =
    //   vals[start..start+len-1]      if start+len < vals.length
    //   vals[start..] + vals[0..len-(vals.length-start)-1]  o.w.
    private int[] vals;
    private int start, len;

    // ...

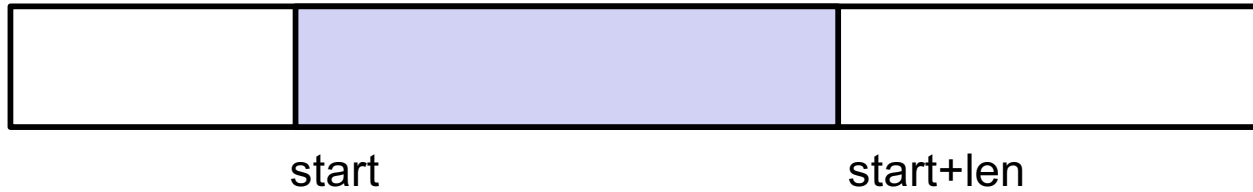
    // @returns length of the list
    public int getLength() {
        return len;
    }
}
```

Example: IntDeque

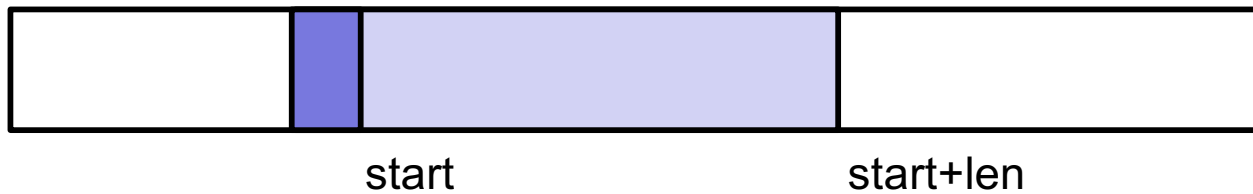
```
/** List that only allows insert/remove at ends. */  
public class IntDeque {  
  
    // ...  
  
    // @requires 0 <= i < length  
    // @returns this[i]  
    public int get(int i) { ... }
```

Example: IntDeque

// List that only allows insert/remove at ends.



unshift



Example: IntDeque

```
// AF(this) =
//   vals[start..start+len-1]           if start+len < vals.length
//   vals[start..] + vals[0..len-(vals.length-start)-1]   o.w.

// @requires 0 < list length
// @returns value at the front of the list
// @modifies this
// @effects first element of list removed
public int unshift() {
    int val = get(0);
    if (start + 1 < vals.length)
        start += 1;
    else
        start = 0;
    len -= 1;
    return val;
}
```

IntDeque.java