CSE 331
Software Design & Implementation

Kevin Zatloukal

Spring 2020

ADT Implementation: Abstraction Functions

Implementing a Data Abstraction (ADT)

To implement an ADT:

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

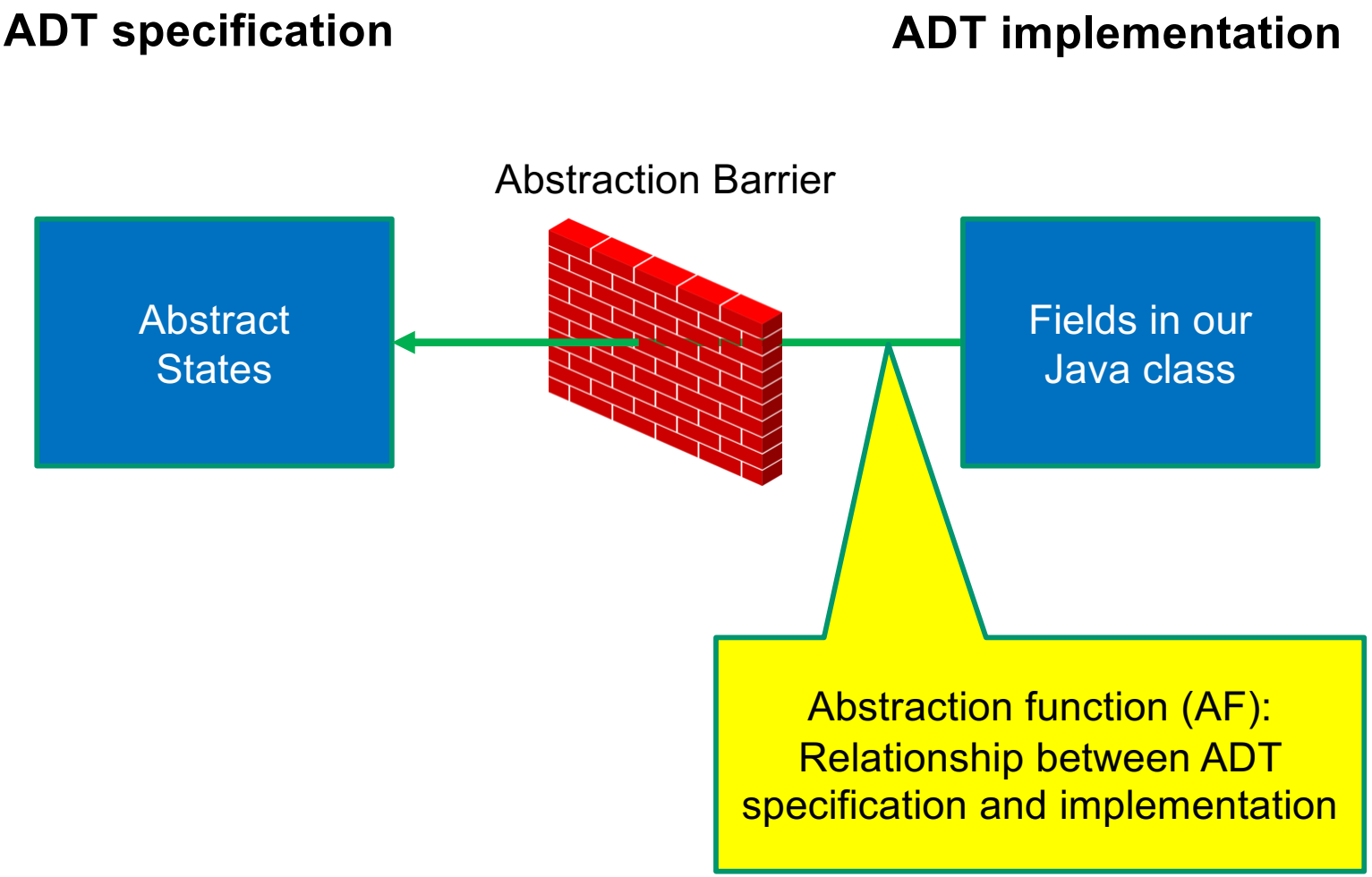
Choose a representation so that:

- 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



Connecting implementations to specs

For implementers / debuggers / maintainers of the implementation:

Abstraction Function: maps Object → 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 (x, y) and radius r
  private double x, y;
  private double r;

  //  ...

}
```

Example: Circle 3

```
/** 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;

  //  ...

}
```

The abstraction function

- Purely conceptual (not a Java function)

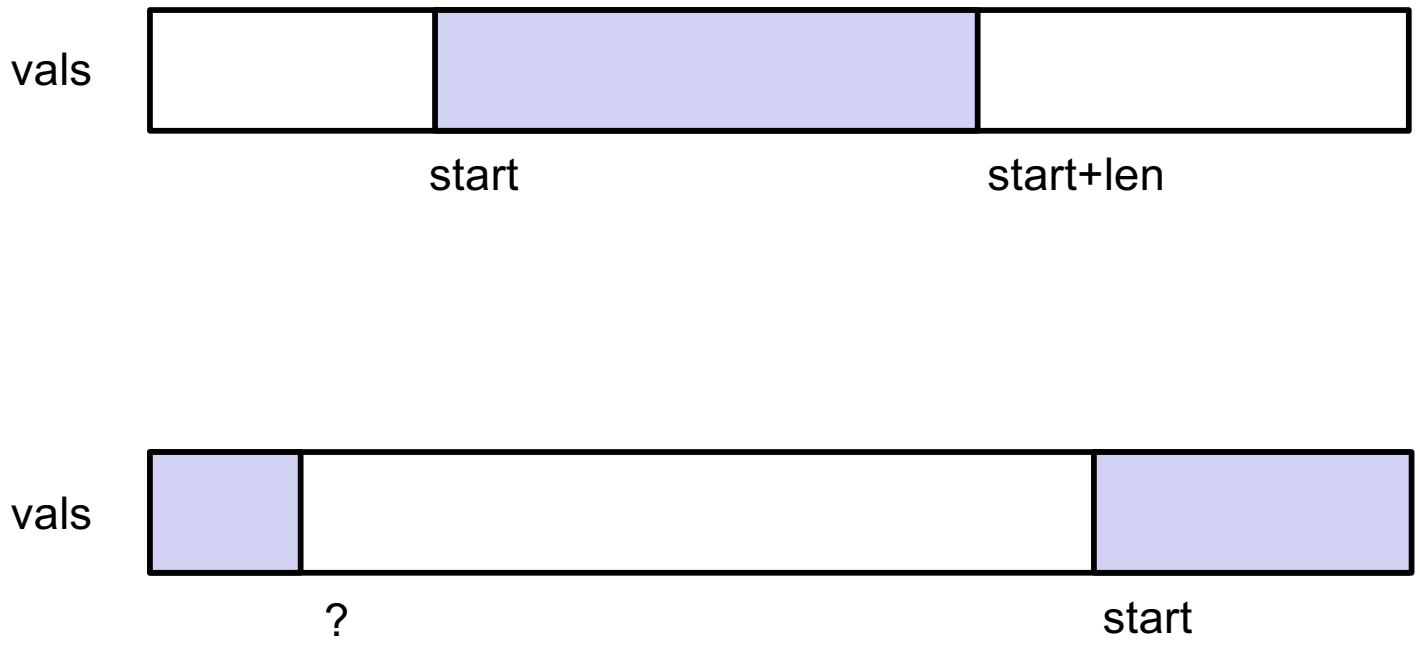
- Allows us to check correctness
  - use reasoning to show that the method leaves the abstract state such that it satisfies the postcondition

- Assume the abstract state initially satisfies precondition...
  - BUT it can be in any concrete representation corresponding to an abstract state that satisfies the precondition
  - (many possible abstract states, potentially many concrete representations for each abstract state)

More Questions?

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.


  // @requires 0 <= i < length
  // @returns this[i]
  public int get(int i) {
    if (start + i < vals.length)
      return vals[start + i];
    else
      return vals[start + i - vals.length];
  }
```

IntDeque.java

Example: Text File

Abstract state is a list of lines, each a sequence of (char, color) pairs.

How would we actually represent this?

Probably okay to store lines in an array:

- most files have < 10k lines, so copying is not too expensive
- most characters are inserted into existing lines not creating new
- (always better to start simple... can change this later)

Example: Text File

Probably not okay to make each (char, color) pair an object

- object overhead is at least 10 bytes per object
- 10 bytes * 10k lines * 40 characters = 4 Mb
- 50 files = 200 Mb of wasted space = unhappy users

Instead store characters and colors in arrays

- problem: inefficient to insert in the middle of an array
  ... **except** at the end of the array (assuming there is space)

Old trick: have two arrays

- one is the beginning of the line
- one is the end of the line in reverse order
- easy to add new characters at the split between parts

Example: TextLine

```
// Overview: Represents one line of the text file...
public class TextLine {

  // Abstraction function: AF(this) = zip sum of
  //   prefixChars[0..prefixCharsLen-1] + reverse(suffixChars[0..suffixCharsLen-1]) and
  //   prefixColors[0..prefixColorsLen-1] + reverse(suffixColors[0..suffixColorsLen-1])
  // (I.e., the list of pairs (char, color),
  //  where the i-th char is paired with the i-th color.)
  private char[] prefixChars, suffixChars;
  private int[] prefixColors, suffixColors;
  private int prefixCharsLen, suffixCharsLen;
  private int prefixColorsLen, suffixColorsLen;

  // ...
```

Example: TextLine

```
// Overview: Represents one line of the text file...
public class TextLine {

  // Abstraction function: AF(this) = zip sum of
  //   prefixChars[0..prefixCharsLen-1] + reverse(suffixChars[0..suffixCharsLen-1]) and
  //   prefixColors[0..prefixColorsLen-1] + reverse(suffixColors[0..suffixColorsLen-1])
  private char[] prefixChars, suffixChars;
  private int prefixCharsLen, suffixCharsLen;

  // @returns this (as a string)
  public String getText() {
    StringBuilder buf = new StringBuilder();
    buf.append(prefixChars, 0, prefixCharsLen);
    // Inv: buf = prefixChars[0..prefixCharsLen-1] +
    //            reverse(suffixChars[i+1..suffixCharsLen-1])
    for (int i = suffixCharsLen - 1; i >= 0; i--)
      buf.append(suffixChars[i]);
    return buf.toString();
  }
```

Example: TextLine

```
// Overview: Represents one line of the text file...
public class TextLine {

  // Abstraction function: AF(this) = ...
  //   prefixChars[0..prefixCharsLen-1] + reverse(suffixChars[0..suffixCharsLen-1]) ...

  // @requires 0 <= col <= len
  // @modifies this
  // @effects this is unchanged and prefixCharsLen = col
  private void moveSplitTo(int col) {
    if (prefixCharsLen < col) {
      prefixChars = ensureSpace(prefixChars, col);
      do {  // Inv: this is unchanged
        prefixChars[prefixCharsLen++] = suffixChars[--suffixCharsLen];
      } while (prefixCharsLen < col);
    } else if (prefixCharsLen > col) {
      // ...
    } else {
      // already in the right place
    }
  }
```

TextLine.java

(note: just chars, no colors)