CSE 373: Tradeoffs and Abstractions

Michael Lee

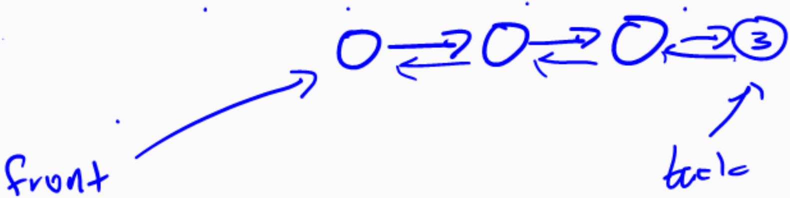
Friday Jan 5, 2017

Warmup questions:

Instructions:

- Recall: What's an ADT? What's a data structure? An implementation of a data structure?
- Skim the Queue ADT on your handout.
- Discuss: How would you implement a queue?

Possible queue implementations



enqueue?

enqueue 3

dequeue ?

Announcements

Course overload link: [redacted] (link given in lecture)

Other announcements:

- Overloading + looking for a partner? Talk to me after class.
- Project 1 out
- Important: get project setup done ASAP

Announcements

Other announcements:

- Overloading + looking for a partner? Talk to me after class.
- Project 1 out
- Important: get project setup done ASAP

Setup tips and tricks:

- Suspect the spec is out-of-date? Shift-refresh in your browser
- Use Java 8, not 9
- When running into weird Eclipse issues, try restarting it

Reviewing CSE 143 material

Places to get practice

- Section 1 handouts
- Practice-it: https://practiceit.cs.washington.edu
- CSE 143 class website (17au or older)
- Project 1

Need help? Visit office hours!

ADTs

ADTs are just a tool for *communicating* with other programmers

This course focuses on *implementing* ADTs: implementing data structures

Why?

Why?

Why can't we just use `java.util.*`?

Why?

The dream: there's One Right Way to implement each ADT

Why?

The dream: there's One Right Way to implement each ADT

The reality: nothing's perfect

Why?

The dream: there's One Right Way to implement each ADT

The reality: nothing's perfect

But we can work around many *tradeoffs* by carefully *adapting* data structures and *abstracting* algorithms!

Tradeoffs

There are (often highly *non-obvious*) ways to organize information to enable *efficient* computations over data.

However, no method is perfect: there exists unavoidable **tradeoffs**.

Tradeoffs

Examples of tradeoffs:

Tradeoffs

Examples of tradeoffs:

- Time vs space

Tradeoffs

Examples of tradeoffs:

- Time vs space
- Making one operation more efficient vs another

Tradeoffs

Examples of tradeoffs:

- Time vs space
- Making one operation more efficient vs another
- Implementing extra behavior vs performance

Tradeoffs

Examples of tradeoffs:

- Time vs space
- Making one operation more efficient vs another
- Implementing extra behavior vs performance
- Simplicity and debuggability vs performance

Tradeoffs

Examples of tradeoffs:

- Time vs space
- Making one operation more efficient vs another
- Implementing extra behavior vs performance
- Simplicity and debuggability vs performance

Core questions:

- What operations do I really need?
- What assumptions am I making about how my software will be used? (e.g. more lookups or inserts)

Case study: The List ADT

A list stores an ordered sequence of information.

Case study: The List ADT

A list stores an ordered sequence of information.

You can access each item by index.

Case study: The List ADT

A list stores an ordered sequence of information.

You can access each item by index.

A list is growable: you can add more and more elements to it.

Case study: The List ADT

A list stores an ordered sequence of information.

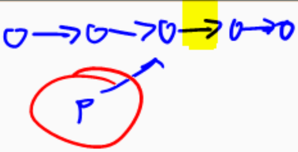
You can access each item by index.

A list is growable: you can add more and more elements to it.

It should support the following *operations*:

- **get**: returns the item at the *i*-th index
- **set**: sets the item at the *i*-th index to a given value
- **append**: add an item to the end of the list
- **insert**: insert an item at the *i*-th index
- **delete**: delete the item at the *i*-th index
- **size**: return the number of elements in the stack

Tradeoffs



Goal: implement the List ADT

Compare and contrast: array list vs linked list

- Time needed to access *i*-th element

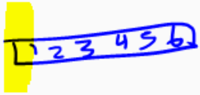
  array list : constant time

  LL : linear time

- Time needed to insert at *i*-th element

  AL : linear time

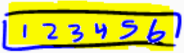
  LL : linear time

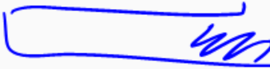


- Amount of space used overall:

  AL : sometimes wasted space

  LL : compact





- Amount of space used per element:

  AL : minimal spe

  LL : tiny extra space.

Tradeoffs

Goal: implement the List ADT

Compare and contrast: array list vs linked list

- Time needed to access *i*-th element
  - Array list: immediate (constant time)
  - Linked list: must iterate to find *i*-th node
- Time needed to insert at *i*-th element
- Amount of space used overall:
- Amount of space used per element:

Tradeoffs

Goal: implement the List ADT

Compare and contrast: array list vs linked list

- Time needed to access *i*-th element
  - Array list: immediate (constant time)
  - Linked list: must iterate to find *i*-th node
- Time needed to insert at *i*-th element
  - Array list: must shift elements
  - Linked list: must iterate to *i*-th node
- Amount of space used overall:
- Amount of space used per element:

Tradeoffs

Goal: implement the List ADT

Compare and contrast: array list vs linked list

- Time needed to access *i*-th element
  - Array list: immediate (constant time)
  - Linked list: must iterate to find *i*-th node
- Time needed to insert at *i*-th element
  - Array list: must shift elements
  - Linked list: must iterate to *i*-th node
- Amount of space used overall:
  - Array list: Potentially wastes space (after doubling)
  - Linked list: No wasted space
- Amount of space used per element:

Tradeoffs

Goal: implement the List ADT

Compare and contrast: array list vs linked list

- Time needed to access *i*-th element
  - Array list: immediate (constant time)
  - Linked list: must iterate to find *i*-th node
- Time needed to insert at *i*-th element
  - Array list: must shift elements
  - Linked list: must iterate to *i*-th node
- Amount of space used overall:
  - Array list: Potentially wastes space (after doubling)
  - Linked list: No wasted space
- Amount of space used per element:
  - Array list: No wasted space
  - Linked list: Slightly more space per element

A question:

How do we print out all the elements inside of a list?

A question:

How do we print out all the elements inside of a list?

One idea:

```
for (int i = 0; i < myList.size(); i++) {
    System.out.println(myList.get(i));
}
```

How efficient is this if `myList` is an array list? A linked list?

AL: linear time

LL: quad

A problem:

We want to make linked list iteration fast. How?

A problem:

We want to make linked list iteration fast. How?

Idea!

- **Adapt** the list ADT
- **Abstract** the idea of iteration

A solution?

```java
Iterator<String> iter = myList.iterator();
while (iter.hasNext()) {
    String item = iter.next();
    System.out.println(item);
}
```

Case study: The List ADT

A list stores an ordered sequence of information.

You can access each item by index.

A list is growable: you can add more and more elements to it.

It should support the following *operations*:

- **get**: returns the item at the *i*-th index
- **set**: sets the item at the *i*-th index to a given value
- **append**: add an item to the end of the list
- **insert**: insert an item at the *i*-th index
- **delete**: delete the item at the *i*-th index
- **size**: return the number of elements in the stack

Case study: The List ADT

A list stores an ordered sequence of information.

You can access each item by index.

A list is growable: you can add more and more elements to it.

It should support the following *operations*:

- **get**: returns the item at the *i*-th index
- **set**: sets the item at the *i*-th index to a given value
- **append**: add an item to the end of the list
- **insert**: insert an item at the *i*-th index
- **delete**: delete the item at the *i*-th index
- **size**: return the number of elements in the stack
- **iterator**: returns an iterator over the list

The Iterator ADT

An iterator “wraps” some sequence.

The Iterator ADT

An iterator “wraps” some sequence.

It yields each subsequent element one by one on request.

The Iterator ADT

An iterator “wraps” some sequence.

It yields each subsequent element one by one on request.

An iterator “remembers” what it needs to yield next.

The Iterator ADT

An iterator “wraps” some sequence.

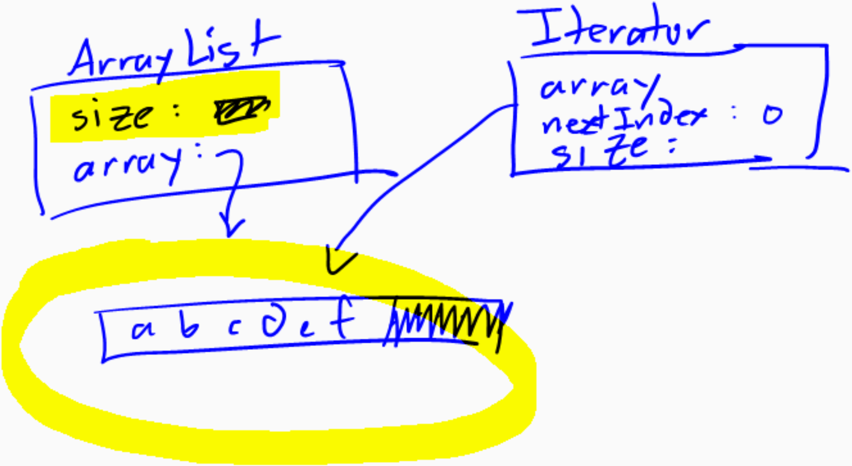
It yields each subsequent element one by one on request.

An iterator “remembers” what it needs to yield next.

Supported operations:

- **hasNext**: returns ‘true’ if there’s another element left to yield and false otherwise
- **next**: returns the next element (if there is one)

Implementing an iterator: A plan of attack



Next time...

What is this ‘efficiency’ thing anyways?

Parting thoughts

Reminder: Overloading/partner concerns, talk to me after class

Supplemental resources: see resources page on class website for...

- Strategies on effectively testing code
- Info on JUnit
- Math review (logs, exponents, summations)

Have suggestions for more resources docs we should write?
Use feedback form.