CMPT 280

Topic 10: Dispensers

Mark G. Eramian

University of Saskatchewan

References

- Textbook, Chapter 10

Stacks and Queues

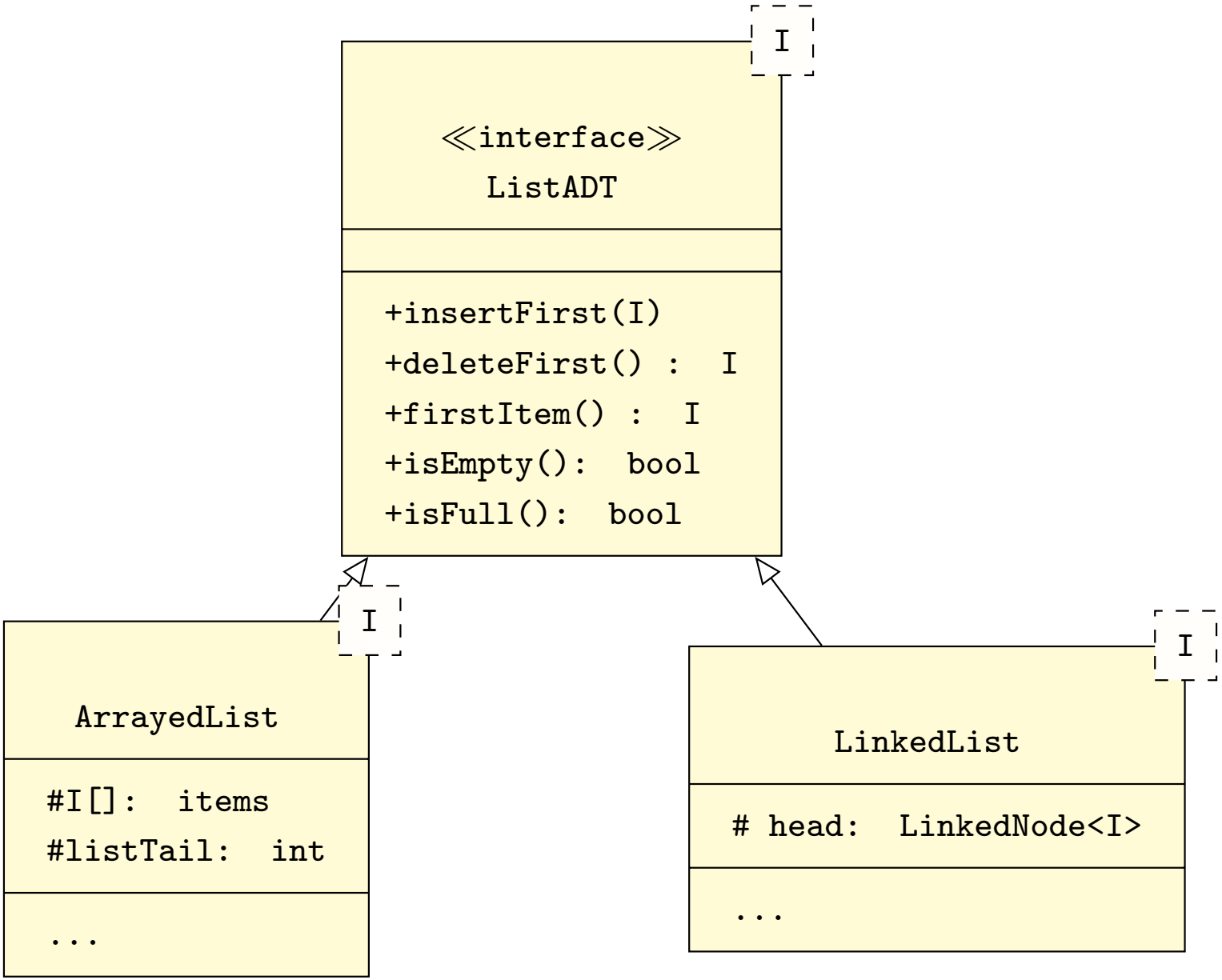
- How can we implement both an array-based stack and a linked stack with the least work?

Stacks and Queues

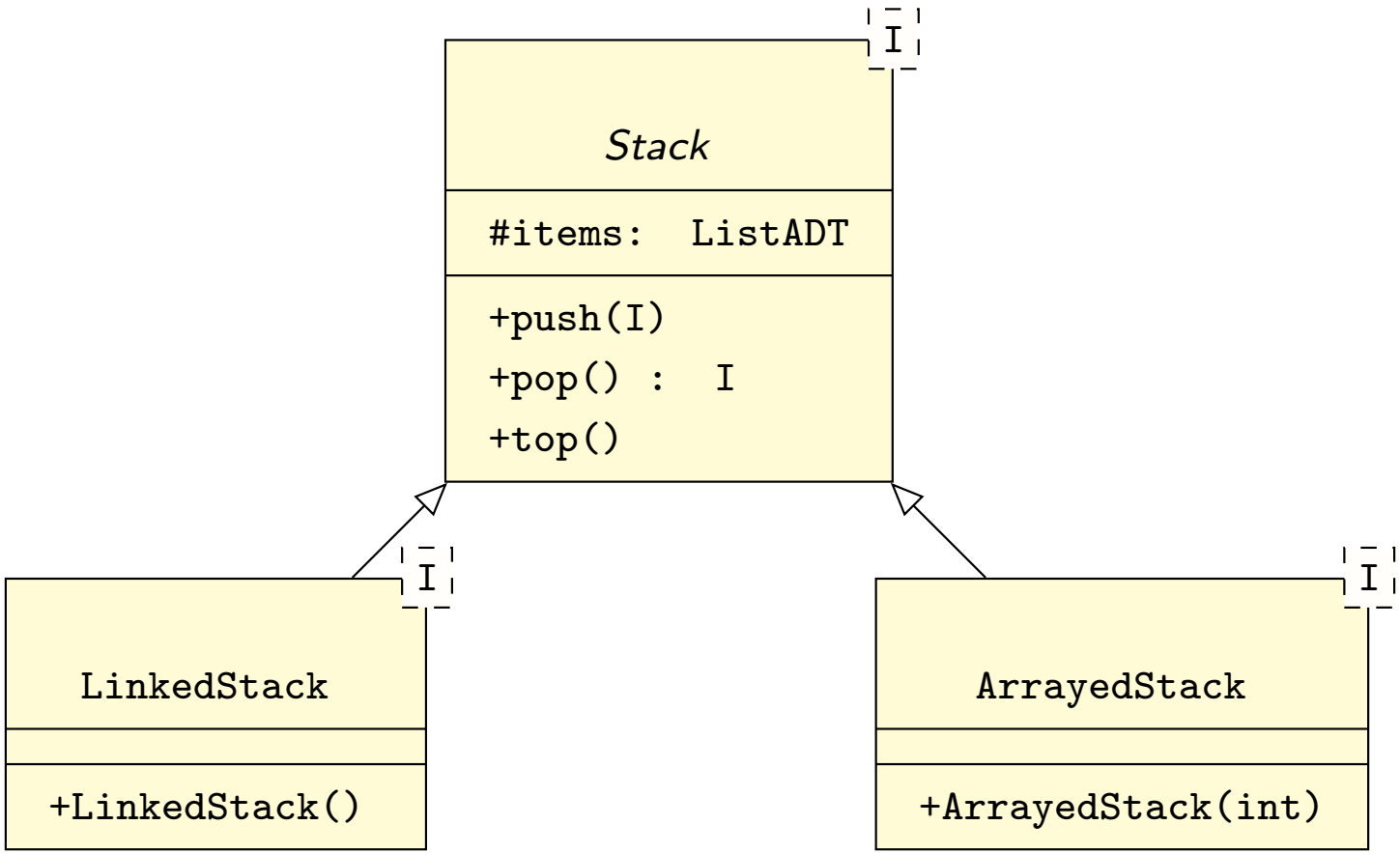
- How can we implement both an array-based stack and a linked stack with the least work?
- Take advantage of Java interfaces and class inheritance.
- Put common code in an abstract base class.
- Put code specific to each implementation in a subclass.
- Use **restriction** of lists to make our job easier.

Let's recall the common list ADT interface from back in Lecture 1...

Common List Interface



Stack Implementation Idea



Exercise 1

- Since both arrayed and linked lists share the same interface, the implementations of the stack methods are the same whether or not we restrict an arrayed list or a linked list.
- Write an abstract class `Stack` with the implementations of `top()` `push()` and `pop()` in terms of a `ListADT` interface. This class will be a *restriction* of a list.
- Hint: a variable of type `ListADT` can refer to either an `ArrayedList` or a `LinkedList`.

Exercise 2

- Write the `ArrayedStack` and `LinkedStack` classes. What methods will they need?

What was the point of all that?

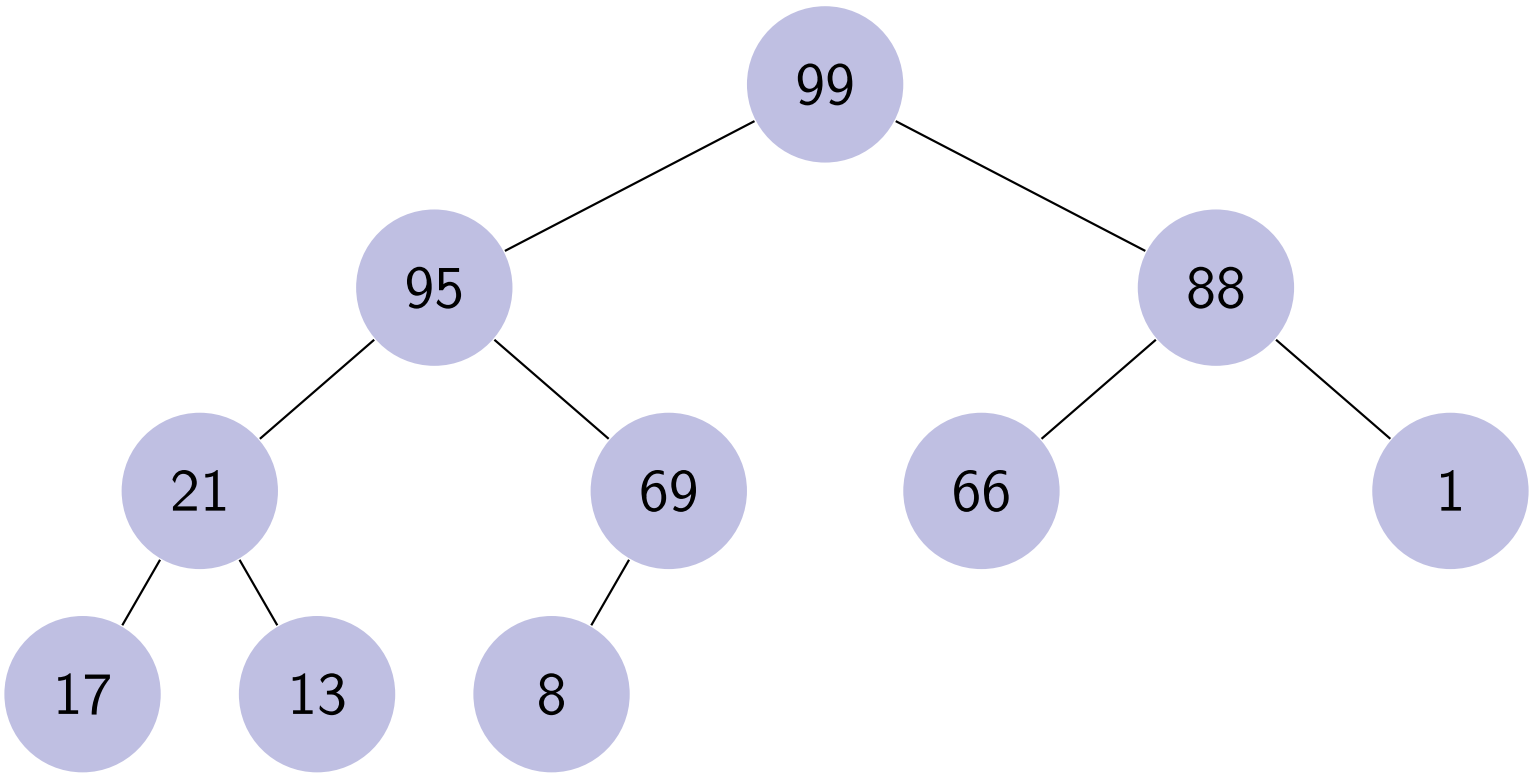
- Common functionality is in the base class.
- Since the `ArrayedList` and `LinkedList` share the same interface, the logic for the stack methods are identical for the two versions of the stack.
- Thus we can have **one** copy of that logic in our code, and the only difference between `LinkedStack` and `ArrayedStack` is the class that implements the list operations on which the stack operations are based.
- The stack logic only occurs in one place (the `Stack` class), which makes software maintenance easier.
- We got all the functionality of two different flavours of stack without having to code any new data structures — we just used the two flavours of list we already had and the concept of *restriction*. Again easier maintenance and debugging.

Reading Roundup - Heaps

- What is a heap?
- What is the heap property?
- At any given time, what item(s) can be accessed or removed from a heap?

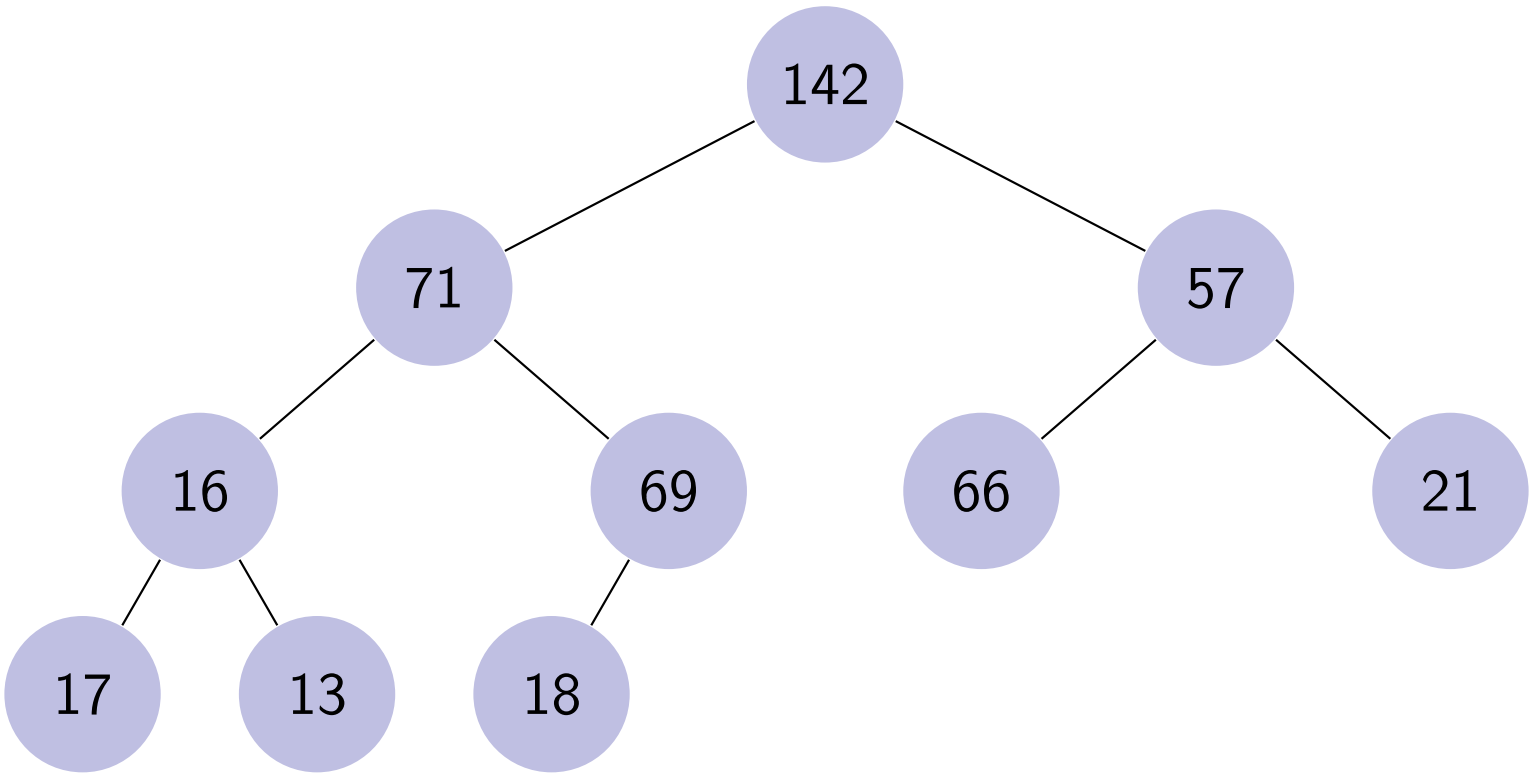
Heaps

Is this a heap?



Heaps

Why is this not a heap?



Next Class

- Next class reading: Chapter 11