Interfaces, Stacks, and Queues

Overview

Interfaces
- Stack
- Queue
- Generic types
- Comparable
- Iterable

Stacks
- ArrayStack
- LinkedStack

Queues
- ArrayQueue
- LinkedQueue

Amortized analysis

Interfaces

```
public interface Resizable {
    public void embiggen();
    public void shrink();
    public int size();
}

public class Balloon implements Resizable {
    …
}
```

An interface:

- has only method signatures, no method bodies or instance variables.
- defines a polymorphic type.
- is a promise or contract, saying that various methods will be provided.

A class implementing the interface fulfills that promise.

Stack

```
public interface Stack {

    public boolean isEmpty();

    public double pop();

    public void push(double item);

}
```

Items are pushed onto top, popped from top

Last in, first out

Queue

```
public interface Queue {

    public double dequeue();

    public void enqueue(double item);

    public boolean isEmpty();

}
```

Items are enqueued into back, dequeued from front

First in, first out

Generic types

```
public interface Stack<T> {

    public boolean isEmpty();

    public T pop();

    public void push(T item);

}
```

`T` is a type parameter, standing for a type just as a variable stands for a value

```
Stack<String> s;
```

Actual type parameter cannot be a primitive type, but can be a wrapper class

Generics do not play well with arrays

Comparable

```
public interface Comparable<T> {
    public int compareTo(T that);
}
```

`a.compareTo(b)` returns a negative number if `a < b`, 0 if `a == b`, positive if `a > b`.

Many built-in classes (e.g., wrapper classes and String) implement Comparable.

If you define your own, `compareTo` should be consistent with `equals`.

Iterable

```
public interface Iterable<T> {
    public Iterator<T> iterator();
}

public interface Iterator<T> {
    public boolean hasNext();
    public T next();
}
```

Allows iterating through a data structure without knowing how it works.

Many built-in structures (like ArrayList) implement Iterable.

Iterables and enhanced for loops

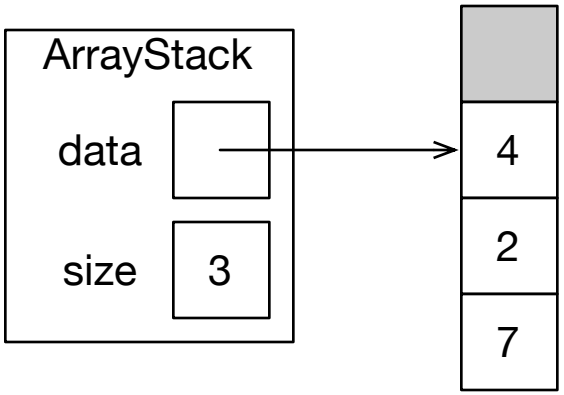
```
for (String s : list) {
   StdOut.println(s);
}
```

is exactly equivalent to:

```
Iterator<String> iter = list.iterator();
while (iter.hasNext()) {
   String s = iter.next();
   StdOut.println(s);
}
```

Stacks

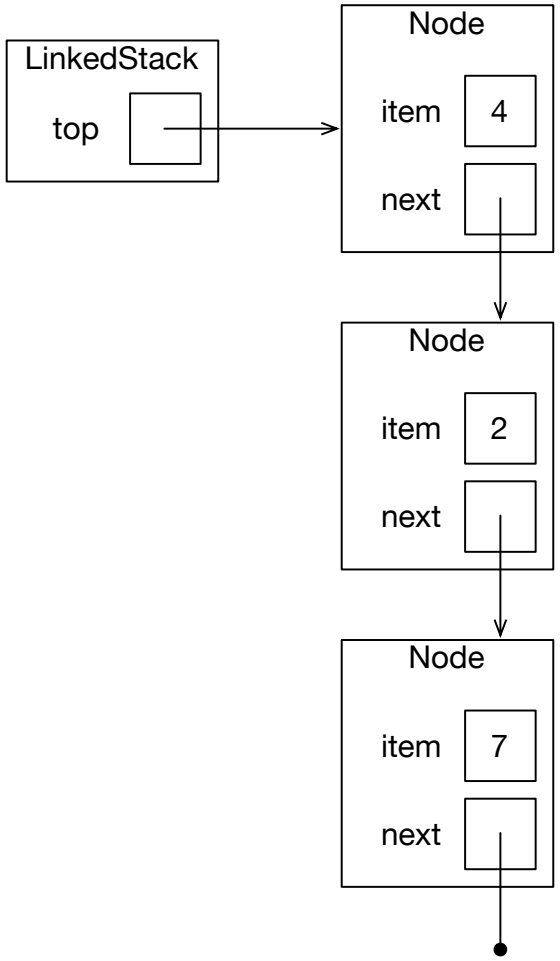
ArrayStack



`size` tells you number of items, index of next available location.

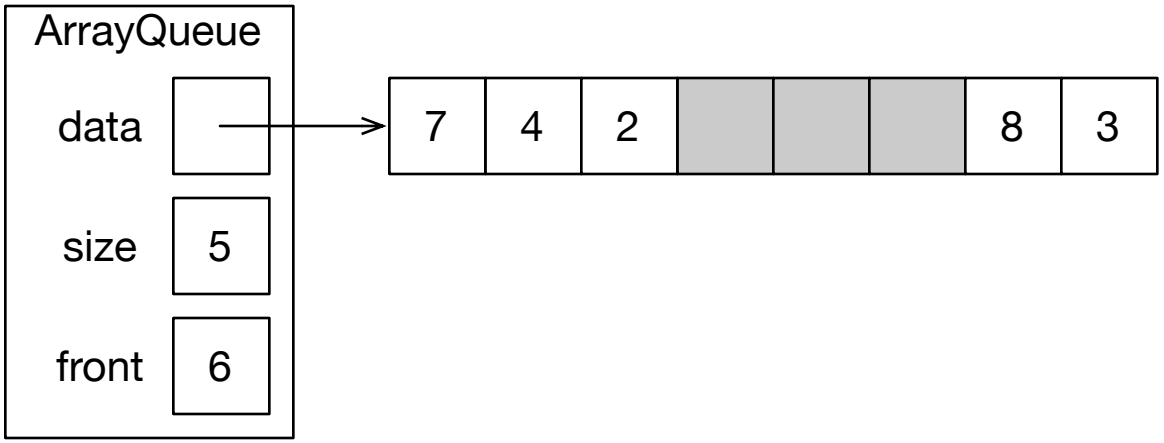
Full? Copy data into larger array.

LinkedStack



Queues

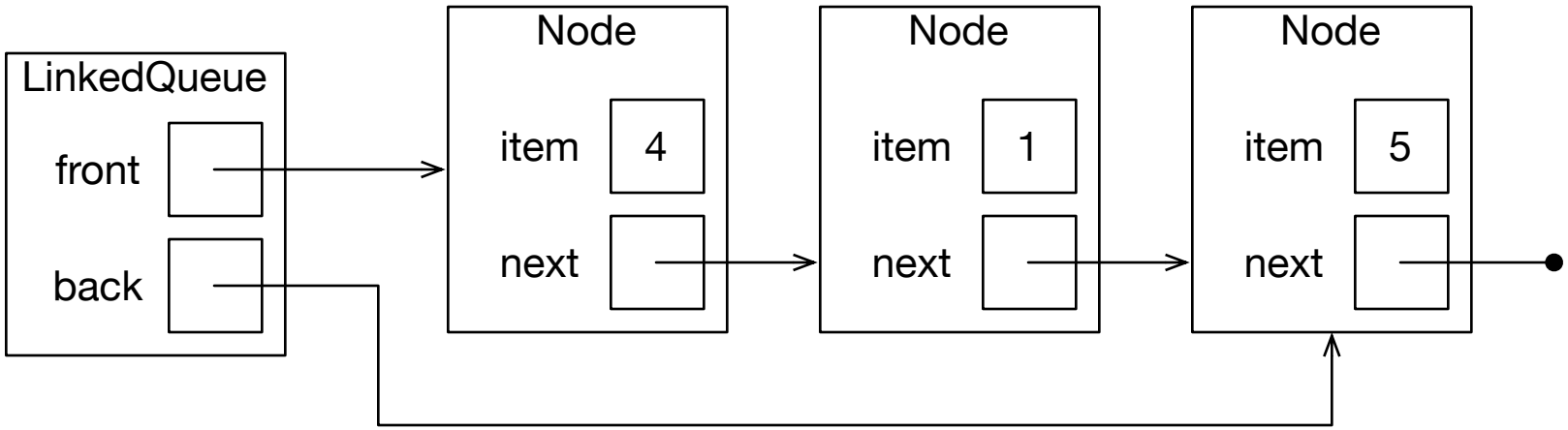
ArrayQueue



Wraps around when it hits the end of the array.

Full? Copy data into larger array.

LinkedQueue



Analysis

Best: what's the cost of the best individual event?

Average: what's the cost per event over an average sequence?

Amortized: what's the cost per event of the worst possible sequence?

Worst: what's the cost of the worst individual event?

Each is at least as good as the one below it.

Amortized is usually the same as worst, but …

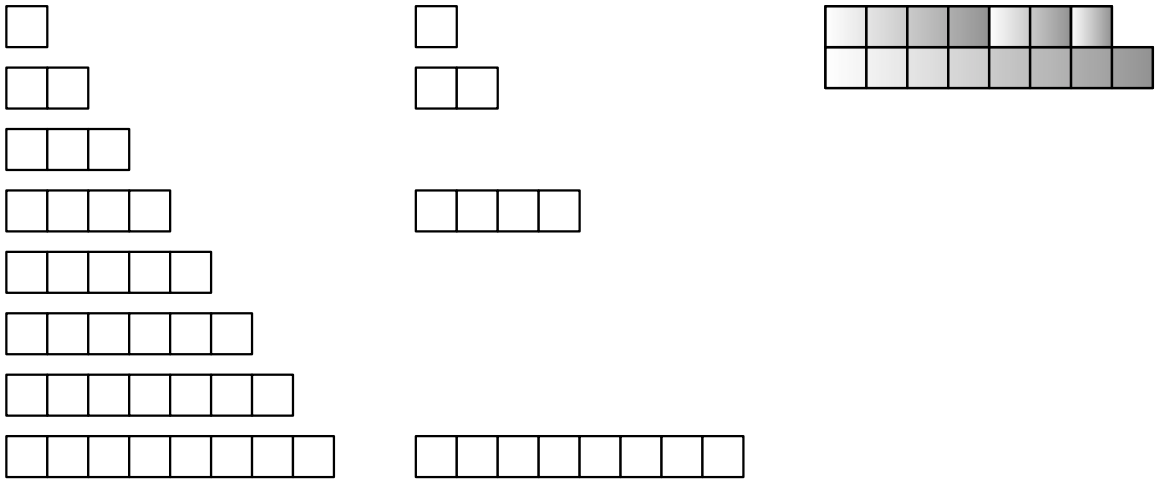
Amortized analysis of array stack operations

Best: $\Theta(1)$

Average: $\Theta(1)$

Amortized: $\Theta(1)$

Worst: $\Theta(n)$



Review

Interfaces define polymorphic types without committing to an implementation.

Generic types allow type parameters to be specified later.

Useful interfaces include comparable and iterable.

Stacks are LIFO, queues are FIFO.

Either can be implemented using an array or linked nodes.

All stack and queue operations take $\Theta(1)$ amortized time.