Stacks & Queueueueueueueueues

"Queue" is properly spelt with as many "ue"s as you'd like.

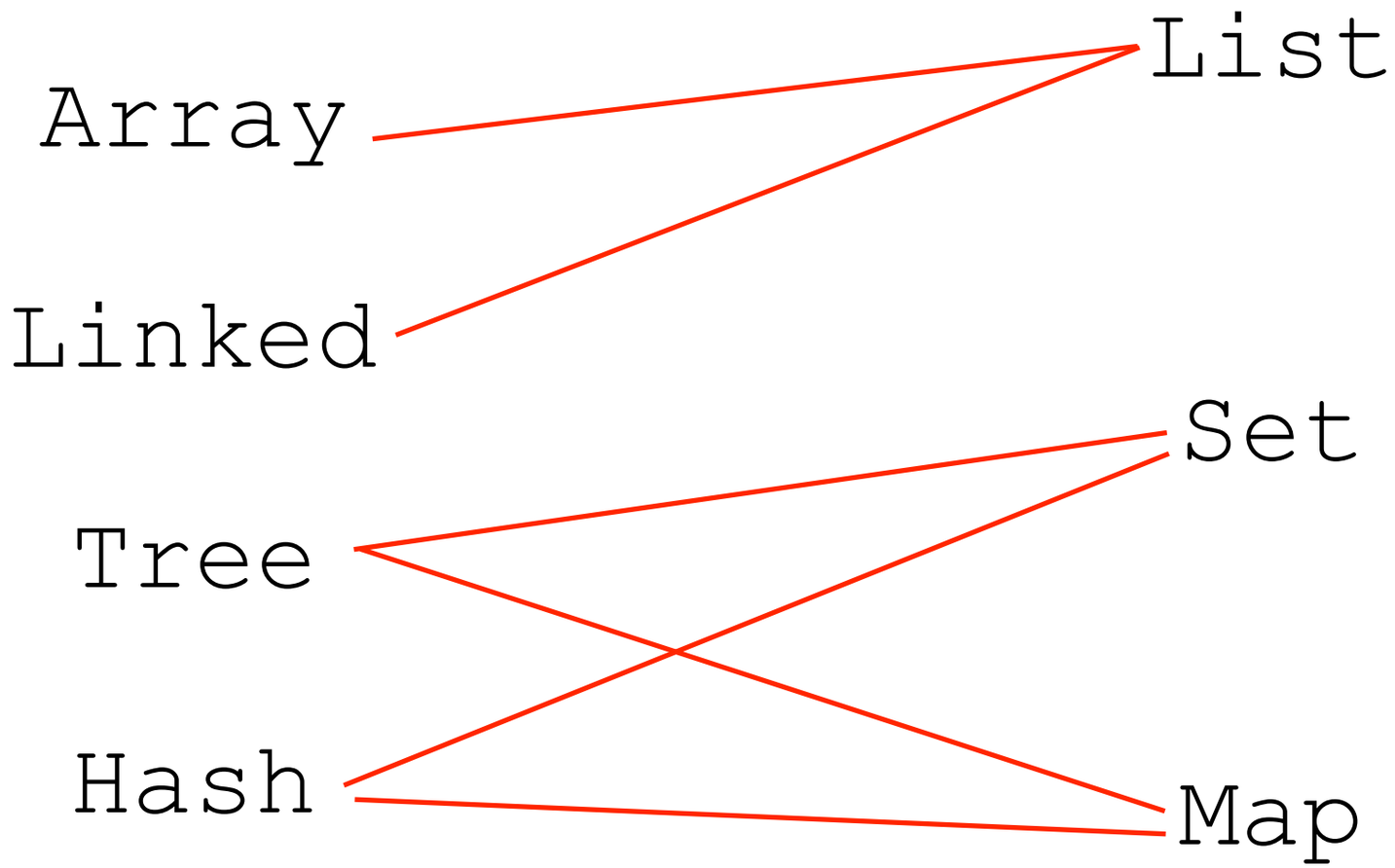
(and inner classes)

(and abstract datatypes)

(and one other thing)

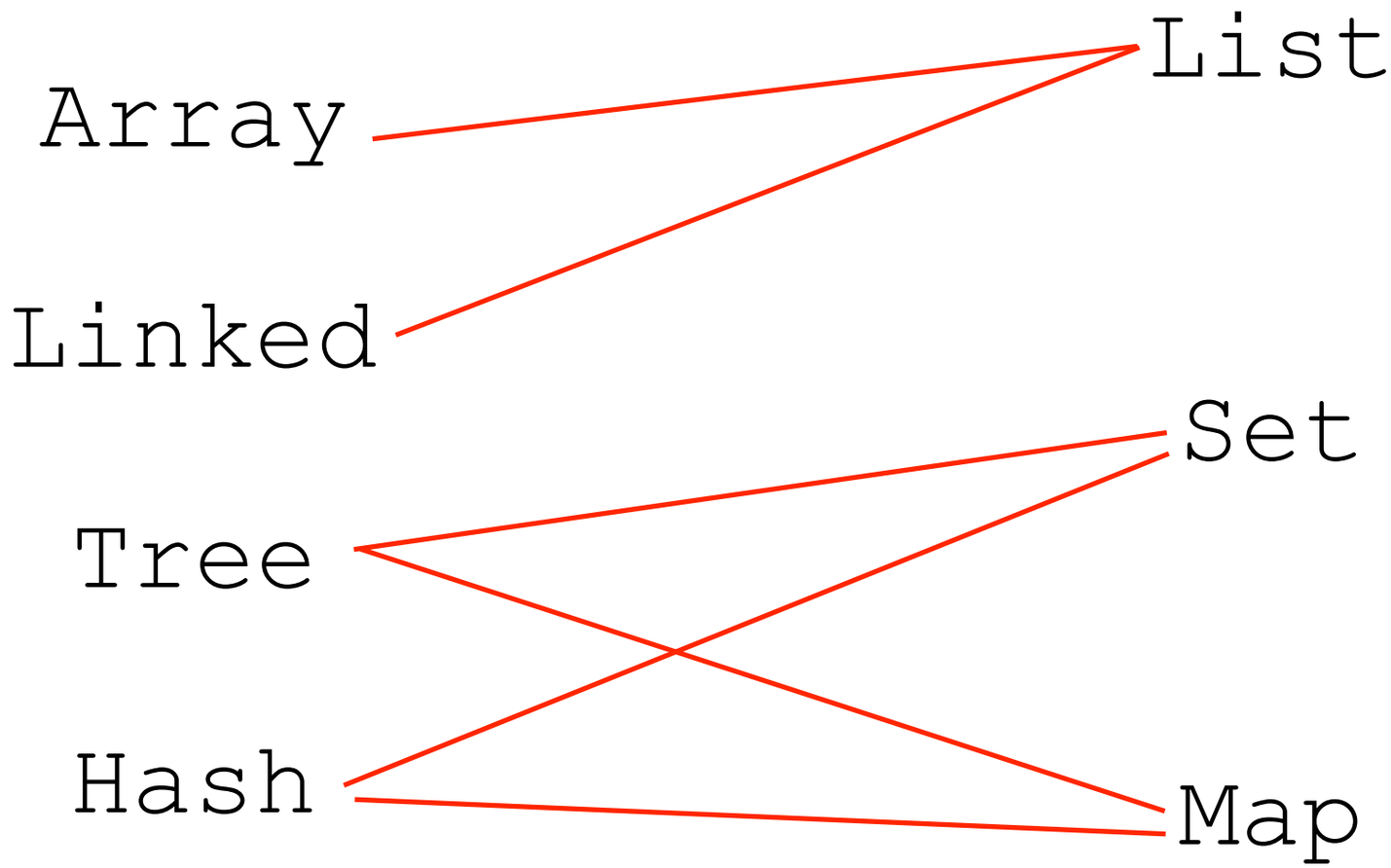
(totally *not in that order)*

Abstract Datatypes



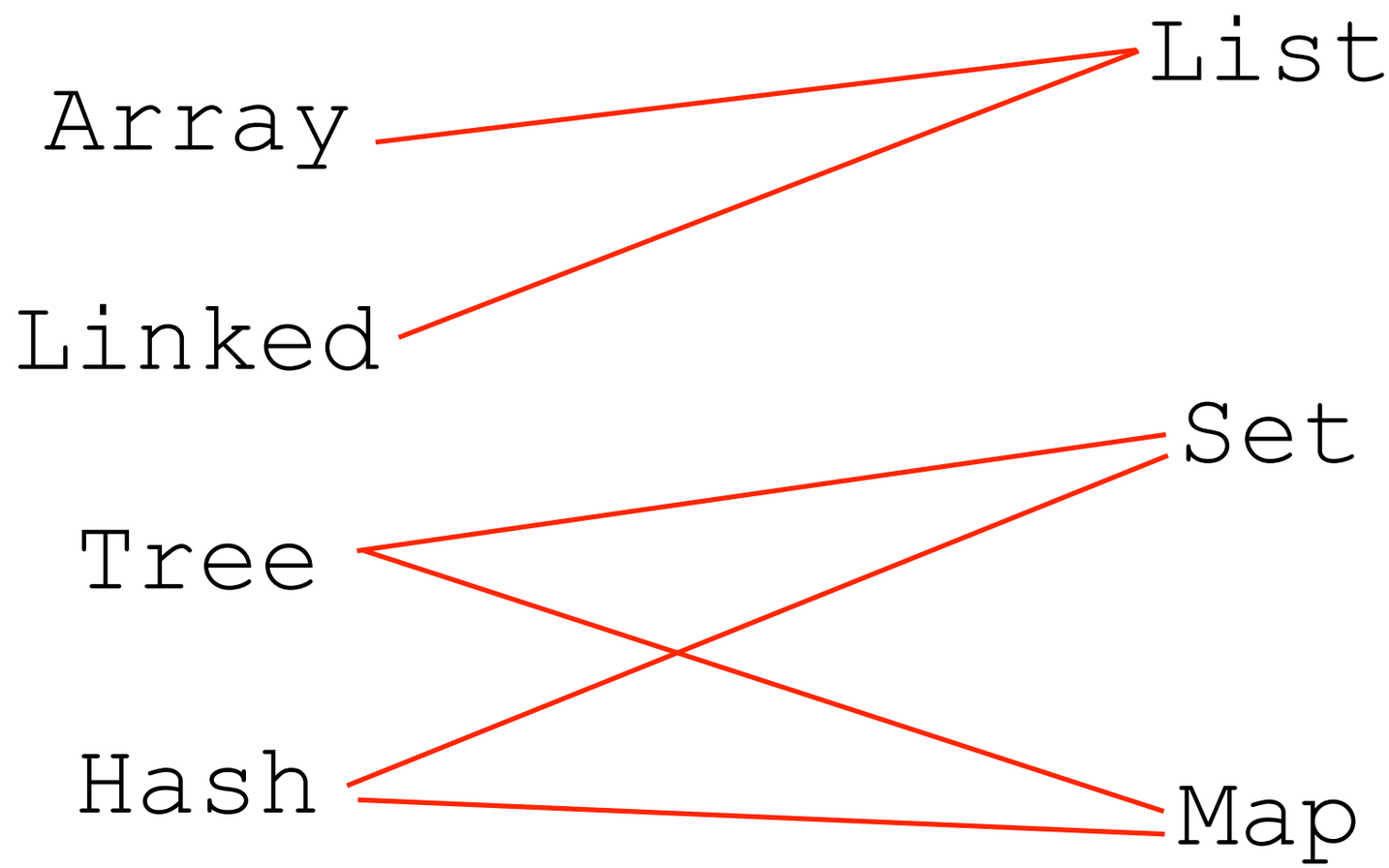
Abstract Datatypes

What it does



Abstract Datatypes

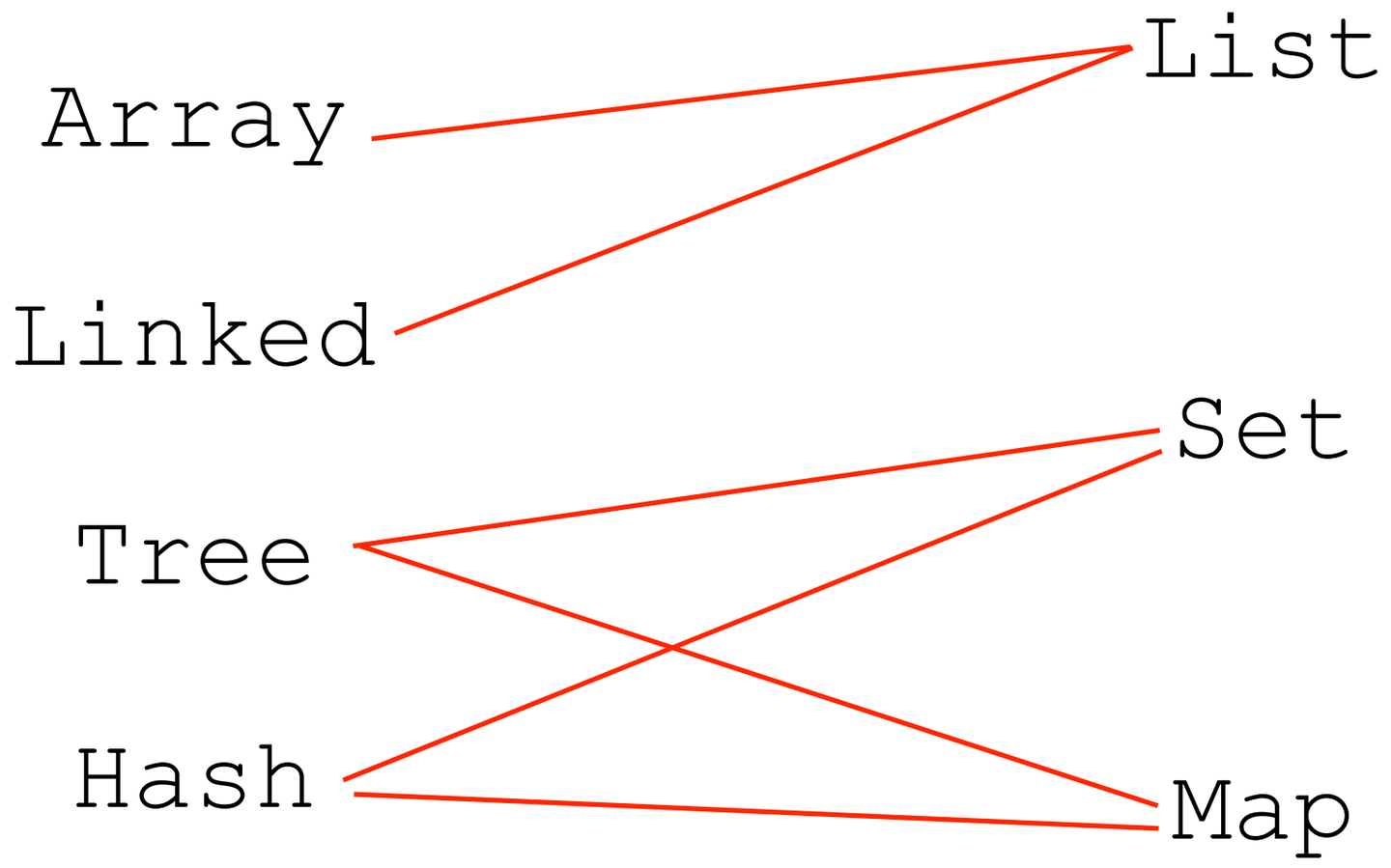
How it does it *What it does*



Abstract Datatypes

How it does it

What it does



Implementation

(or "concrete data type", if you'd like)

Abstract data type

An *Abstract Data Type* defines a *set of methods.*

```
List
  .add()
  .remove()
  .get()
  .set()
  .size()
  ...
```

```
Set
.add()
.remove()
.contains()
.size()
...
```

```
Map
.put()
.get()
.size()
.containsKey()
...
```

An *Abstract Data Type* defines a *set of methods.*

```
List
  .add()
  .remove()
  .get()
  .set()
  .size()
  ...
```

```
Set
.add()
.remove()
.contains()
.size()
...
```

```
Map
.put()
.get()
.size()
.containsKey()
...
```

How it does it

What it does

Keeping these distinct is powerful!

Two new data structures!

```
Stack
  void push(String s);
  String pop();
  int size();
```

```
Queue
  void push(String s);
  String pop();
  int size();
```

Note that this explains their abstract data types, *not their* implementations!

Stack

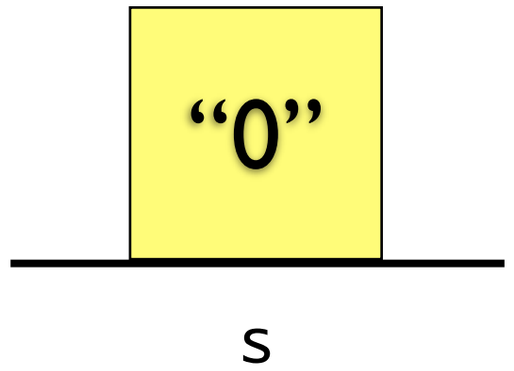
```
void push(String s);
String pop();
int size();
```

```
Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}
```

Stack

```
void push(String s);
String pop();
int size();
```

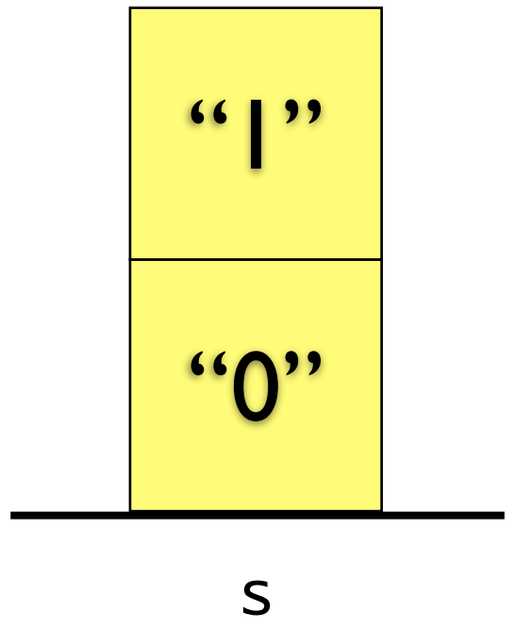
```
Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}
```



Stack

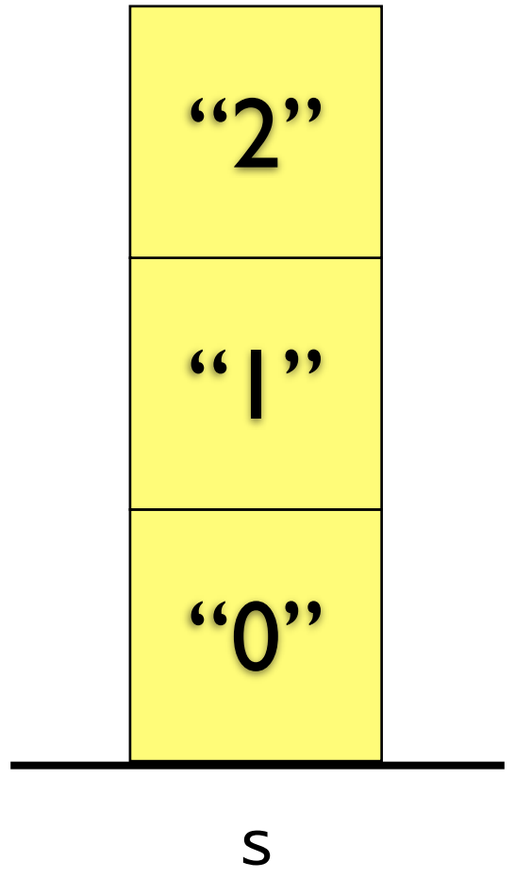
```
void push(String s);
String pop();
int size();
```

```
Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}
```



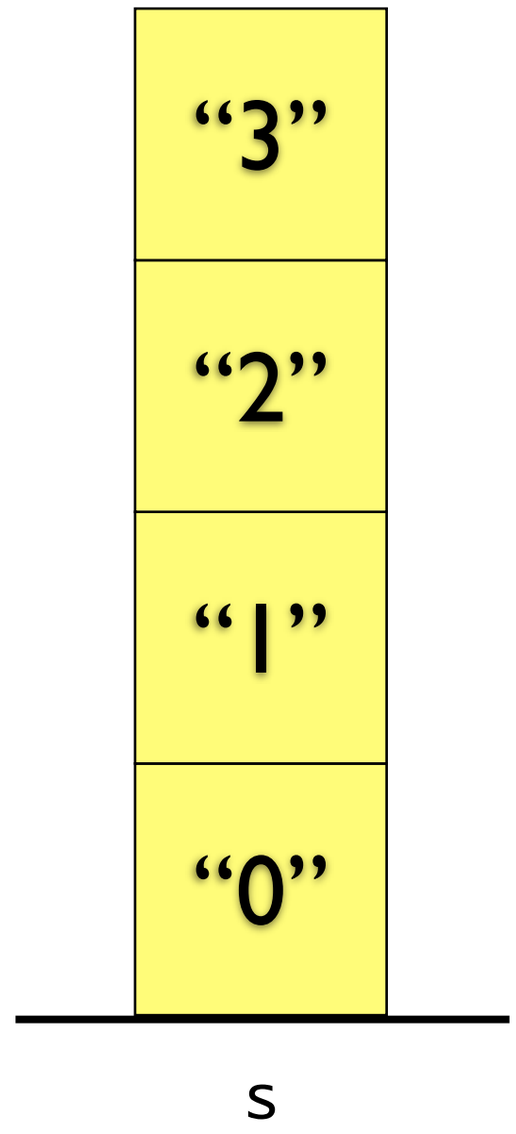
Stack

```
void push(String s);
String pop();
int size();


Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}
```



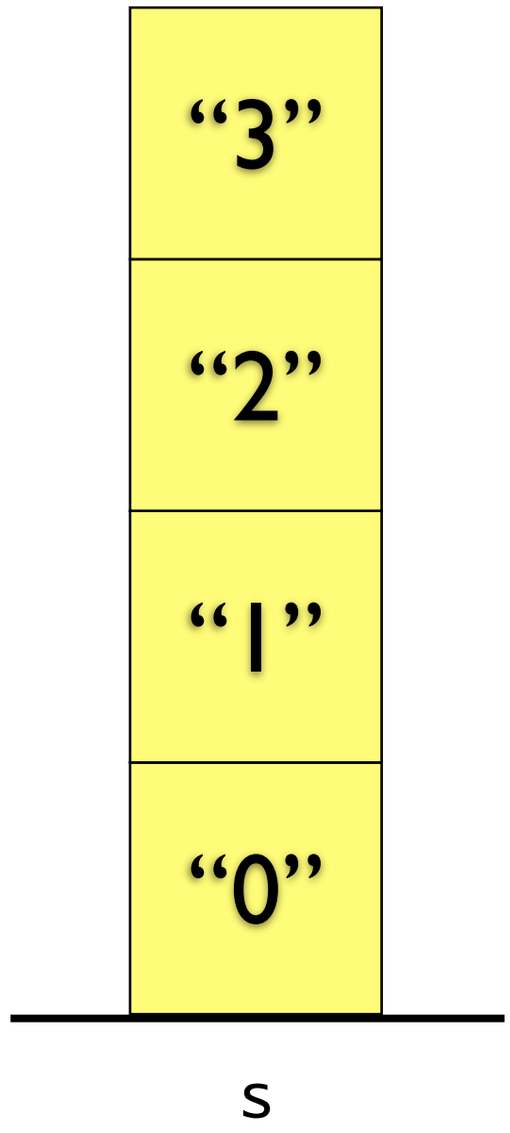
Stack

```
void push(String s);
String pop();
int size();


Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}
```



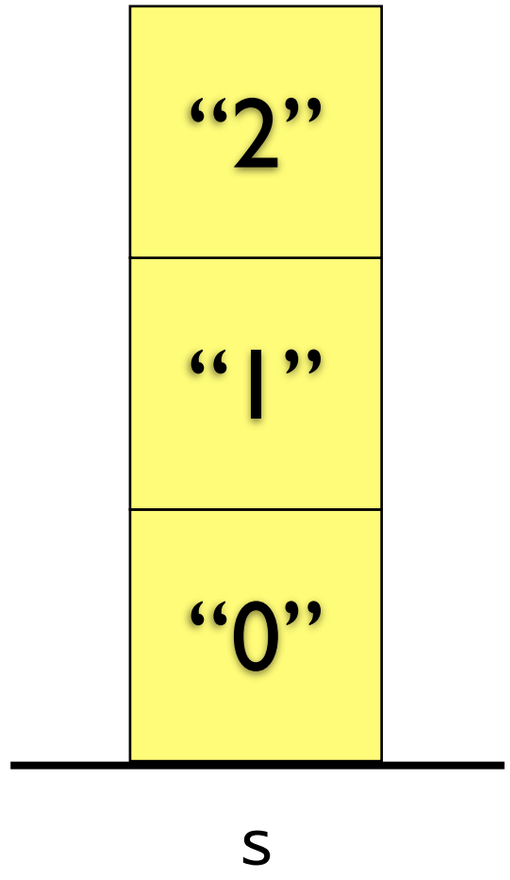
Stack

```
void push(String s);
String pop();
int size();


Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}


while (s.size() > 0) {
   String s2 = s.pop();
   System.out.print(s2 + " ");
}
```



Stack

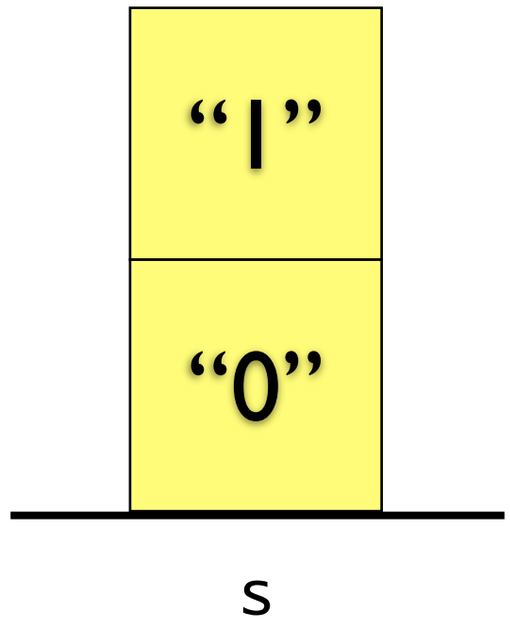
```
void push(String s);
String pop();
int size();


Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}


while (s.size() > 0) {
   String s2 = s.pop();
   System.out.print(s2 + " ");
}
```



3

Stack

```
void push(String s);
String pop();
int size();


Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}


while (s.size() > 0) {
   String s2 = s.pop();
   System.out.print(s2 + " ");
}
```



3 2

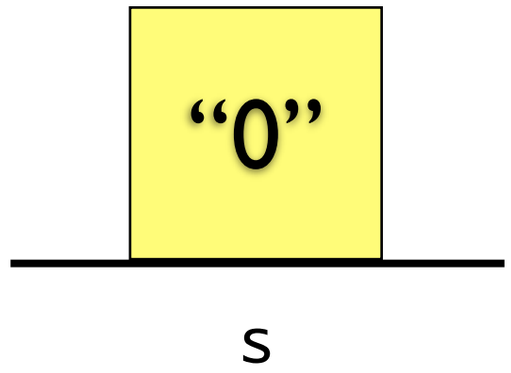
Stack

```
void push(String s);
String pop();
int size();
```

```
Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}
```

```
while (s.size() > 0) {
   String s2 = s.pop();
   System.out.print(s2 + " ");
}
```

3 2 1

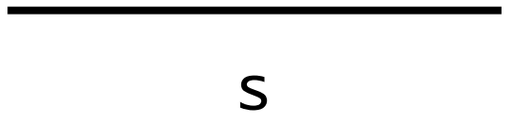


Stack

```
void push(String s);
String pop();
int size();
```

```
Stack s = new Stack();

for (int i = 0 ; i < 4 ; ++i) {
   s.push("" + i);
}
```

```
while (s.size() > 0) {
   String s2 = s.pop();
   System.out.print(s2 + " ");
}
```



3 2 1 0

This is called "LIFO" ordering

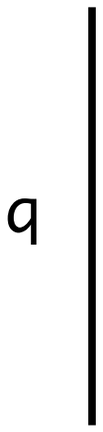
For "Last In First Out"

Queue

```
void push(String s);
String pop();
int size();
```

```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```

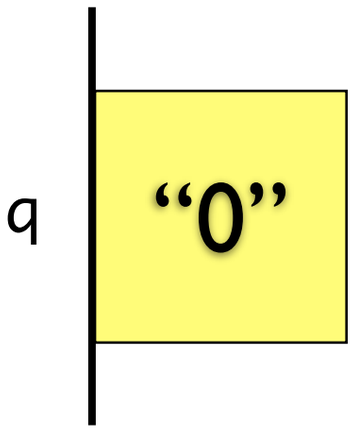
q



Queue

```
void push(String s);
String pop();
int size();
```

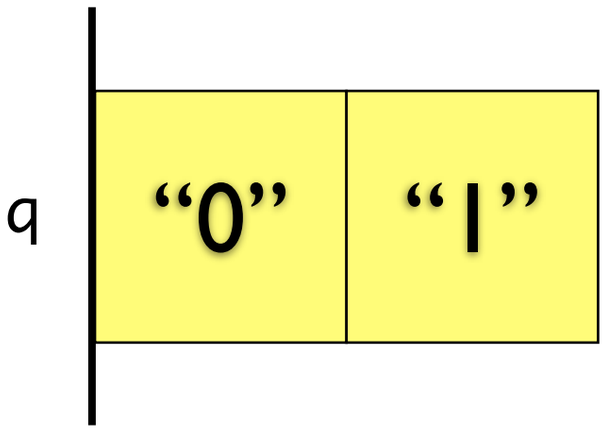
```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```



Queue

```
void push(String s);
String pop();
int size();
```

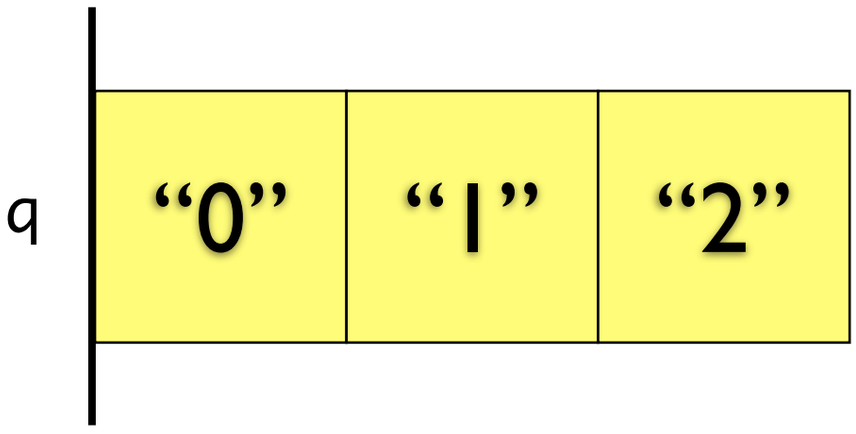
```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```



Queue

```
void push(String s);
String pop();
int size();
```

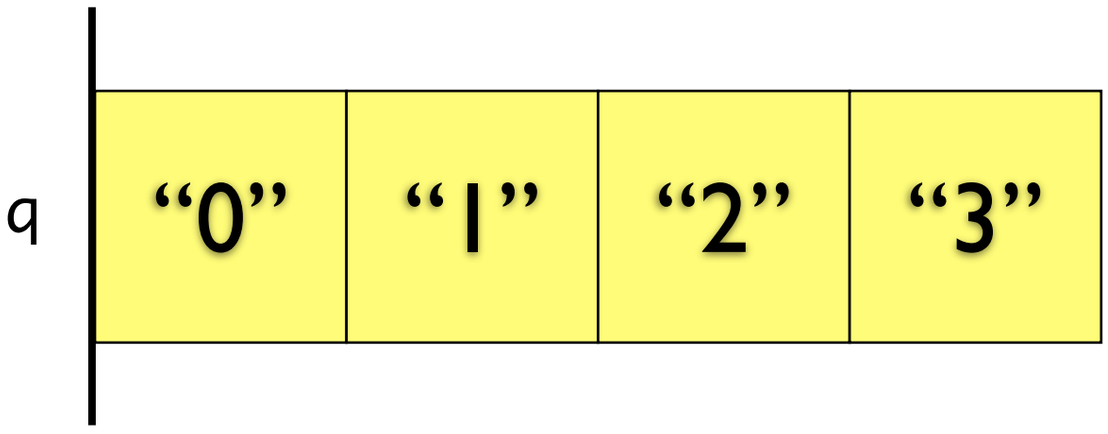
```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```



Queue

```
void push(String s);
String pop();
int size();
```

```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```

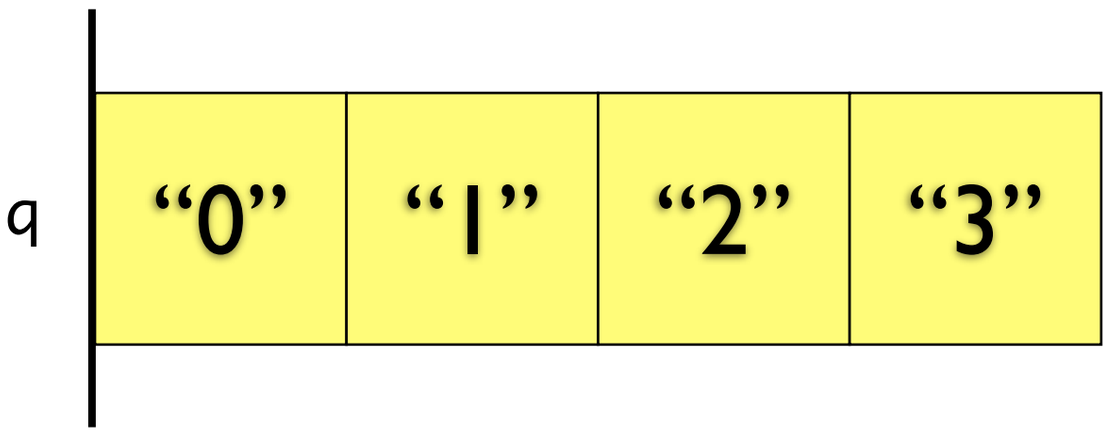


Queue

```
void push(String s);
String pop();
int size();
```

```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```

```
while (q.size() > 0) {
   String s = s.pop();
   System.out.print(s + " ");
}
```



Queue

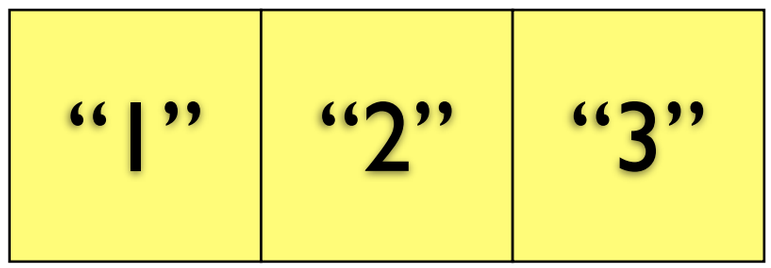
```
void push(String s);
String pop();
int size();
```

```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```

```
while (q.size() > 0) {
   String s = s.pop();
   System.out.print(s + " ");
}
```

0

q



Queue

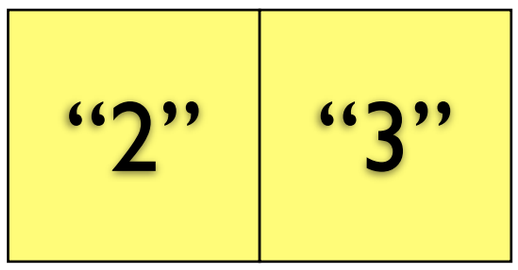
```
void push(String s);
String pop();
int size();
```

```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```

```
while (q.size() > 0) {
   String s = s.pop();
   System.out.print(s + " ");
}
```

0 1

q



Queue

```
void push(String s);
String pop();
int size();
```

```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```

```
while (q.size() > 0) {
   String s = s.pop();
   System.out.print(s + " ");
}
```

0 1 2

q

Queue

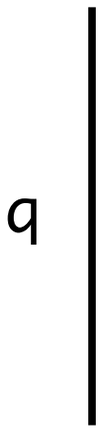
```
void push(String s);
String pop();
int size();
```

```
Queue q = new Queue();

for (int i = 0 ; i < 4 ; ++i) {
   q.push("" + i);
}
```

```
while (q.size() > 0) {
   String s = s.pop();
   System.out.print(s + " ");
}
```

0 1 2 3

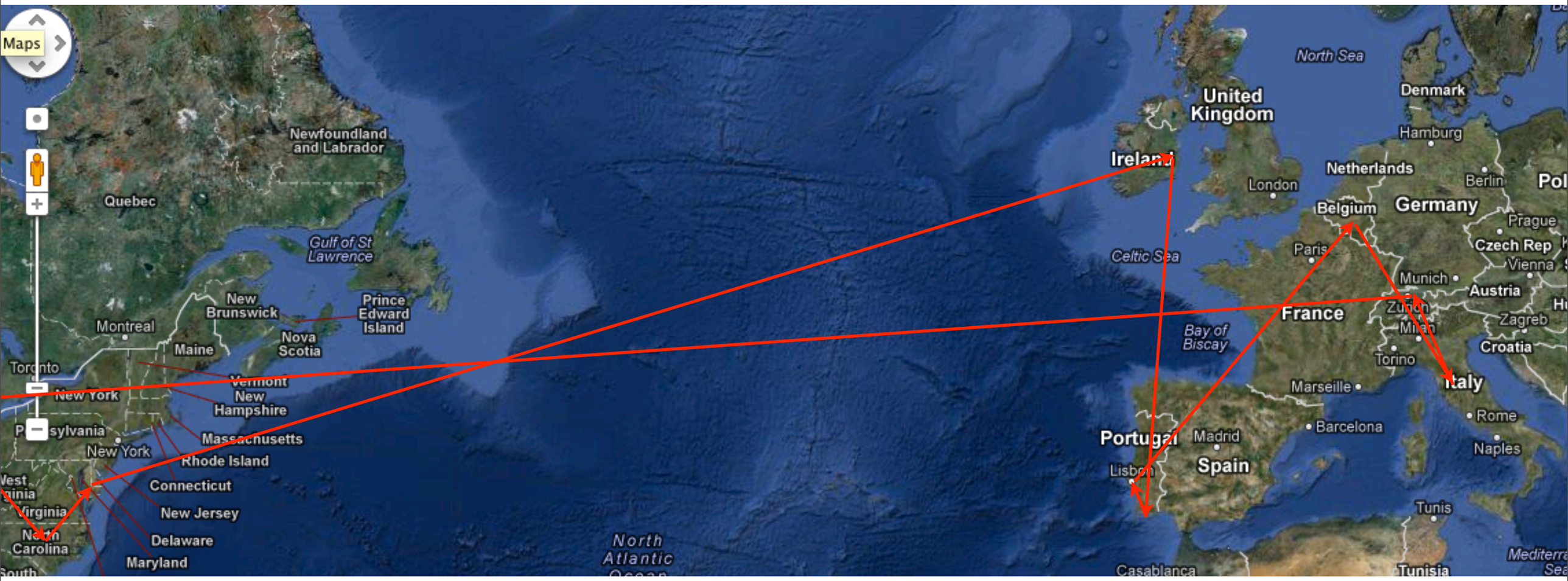
q



This is called "FIFO" ordering

For "First In First Out"

Take six minutes. Find an example of a stack and a queue from daily life. Be ready to tell the class.



Wednesday, October 17, 12

A stack!

A queue!

Java's name for ADTs: *Interfaces*

```
public interface IStack {
   // How many elements are in my stack?
   int size();

   // Push a String onto my stack.
   void push(String s);

   // Remove, and return, the top element of my stack.
   String pop();
}
```

By convention, not-built-in interfaces start with I, to show that they aren't classes.

If you think this looks a lot like a class definition, but without any bodies for the methods, you're spot on.

implements

```
public class MacStack implements IStack {
  // ...code...
}
```

The compiler will point out any methods you've promised but not delivered.

implements

```
public class MacStack implements IStack {
  // ...code...
}
```

Eclipse will point out any methods you've promised but not delivered.

Interfaces are types!

```
public String doSomethingWithAStack(IStack stack) {
  // ...code...
}
```

Limits you to only those methods defined in the interface; also frees you from caring about how it's implemented!

```
IStack s = new MacStack();
```

Inner classes (recap!)

Fun fact: you can define classes inside other classes.

StringLinkedList, for example.

When should you define an *inner* (or *nested*) class?

Implementation *Abstract* data type

Inner classes (recap!)

Fun fact: you can define classes inside other classes.

StringLinkedList, for example.

When should you define an *inner* (or *nested*) class?

Implementation

When's it's part of the implementation, but not part of the interface.

Abstract data type

Demo time!