# CS 240

## Data Structures and Algorithms I

Alex Vondrak

ajvondrak@csupomona.edu

October 24, 2011

# A Generic ArrayStack

```
class ArrayStack<E> implements Stack<E> {
  public void push(E value) { ... }

  public E pop()
     throws StackUnderflowException { ... }

  public E top()
     throws StackUnderflowException { ... }

  public boolean isEmpty() { ... }

  public int size() { ... }
}
```

Let's play with the file `ArrayStack.java` from the course website

# Java API

- For the sake of discussion, we've been implementing stacks by hand
- Java has many data structures included with its libraries
- Because it's so extensive, it's important to be familiar with the Java libraries' APIs
- `http://download.oracle.com/javase/7/docs/api/java/util/Stack.html`

# One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

| **Operation** | **Worst-Case Running Time** |
|---|---|
| `size` | |
| `isEmpty` | |
| `top` | |
| `pop` | |
| `push` | |

# One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

| **Operation** | **Worst-Case Running Time** |
|---|---|
| `size` | $O(1)$ |
| `isEmpty` | |
| `top` | |
| `pop` | |
| `push` | |

# One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

| **Operation** | **Worst-Case Running Time** |
|---|---|
| `size` | $O(1)$ |
| `isEmpty` | $O(1)$ |
| `top` | |
| `pop` | |
| `push` | |

# One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

| **Operation** | **Worst-Case Running Time** |
|---|---|
| `size` | $O(1)$ |
| `isEmpty` | $O(1)$ |
| `top` | $O(1)$ |
| `pop` | |
| `push` | |

# One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

| **Operation** | **Worst-Case Running Time** |
|---|---|
| `size` | $O(1)$ |
| `isEmpty` | $O(1)$ |
| `top` | $O(1)$ |
| `pop` | $O(1)$ |
| `push` | |

# One Final Talking Point...

Q: How efficient are the stack operations?

A: Depends on the implementation!

Let's take a look at our `ArrayStack`:

| **Operation** | **Worst-Case Running Time** |
|---|---|
| `size` | $O(1)$ |
| `isEmpty` | $O(1)$ |
| `top` | $O(1)$ |
| `pop` | $O(1)$ |
| `push` | $O(n)$ |

# Amortized Analysis

- At worst, `push` takes $O(n)$ time to resize the array...
- ...But how often does that happen?
- Idea: average out the cost of $n$ operations performed in sequence

```
public void push(E value) {
  if (size() == data.length)
    grow();
  data[++top] = value;
}
```

Q: What if `grow()` increases the size of the array by 1?

Q: What if `grow()` doubles the size of the array?