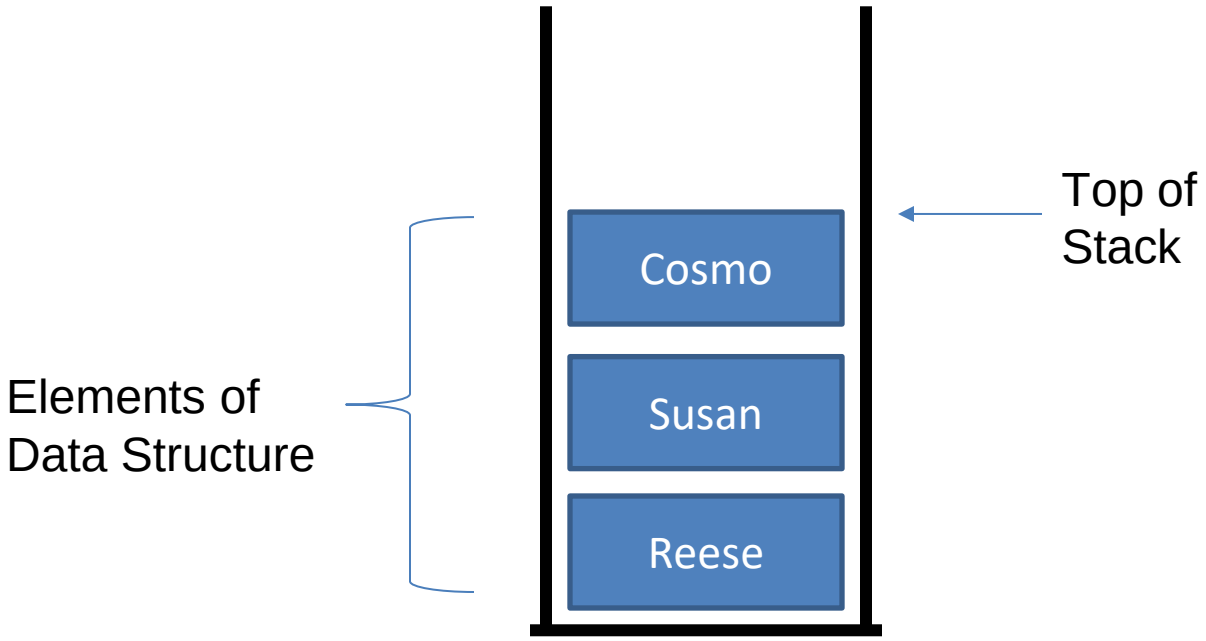
# CSCI 132:
# Basic Data Structures and Algorithms

Stacks (Array Representation)

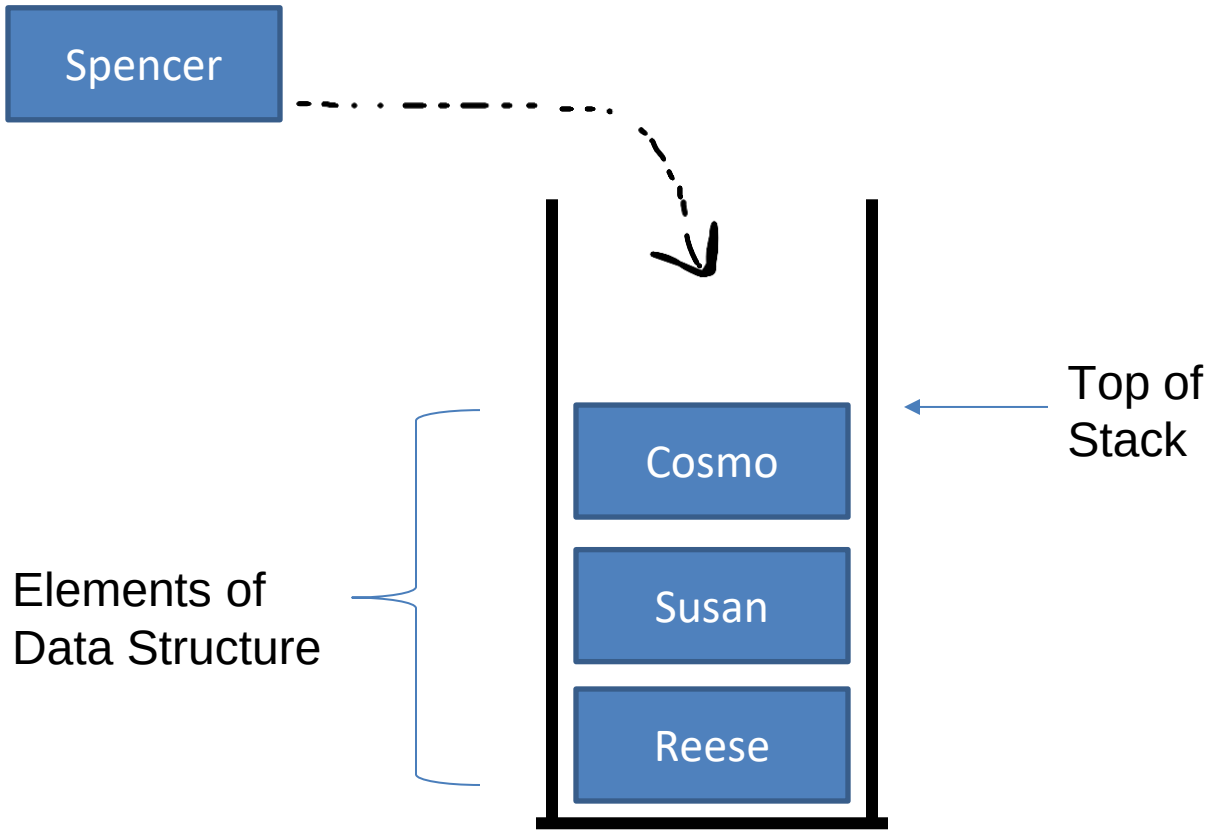
Reese Pearsall

Spring 2025

A **stack** is a data structure that can hold data, however the way we interact with a stack is a little bit different



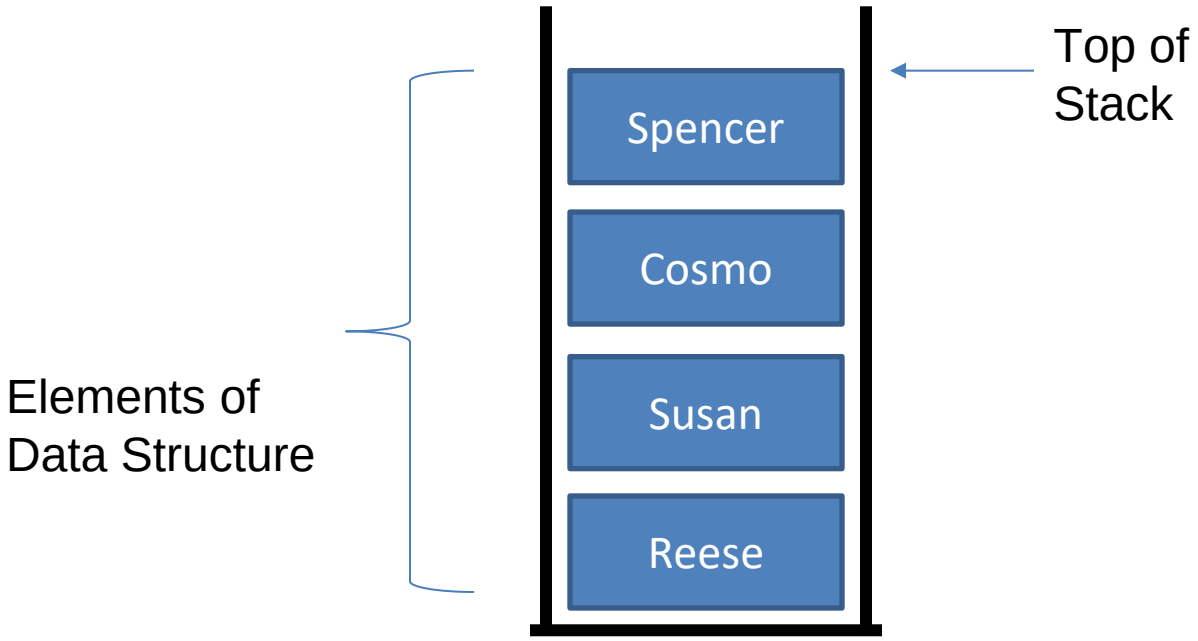
A **stack** is a data structure that can hold data, however the way we interact with a stack is a little bit different




When only interact with the top of the stack.

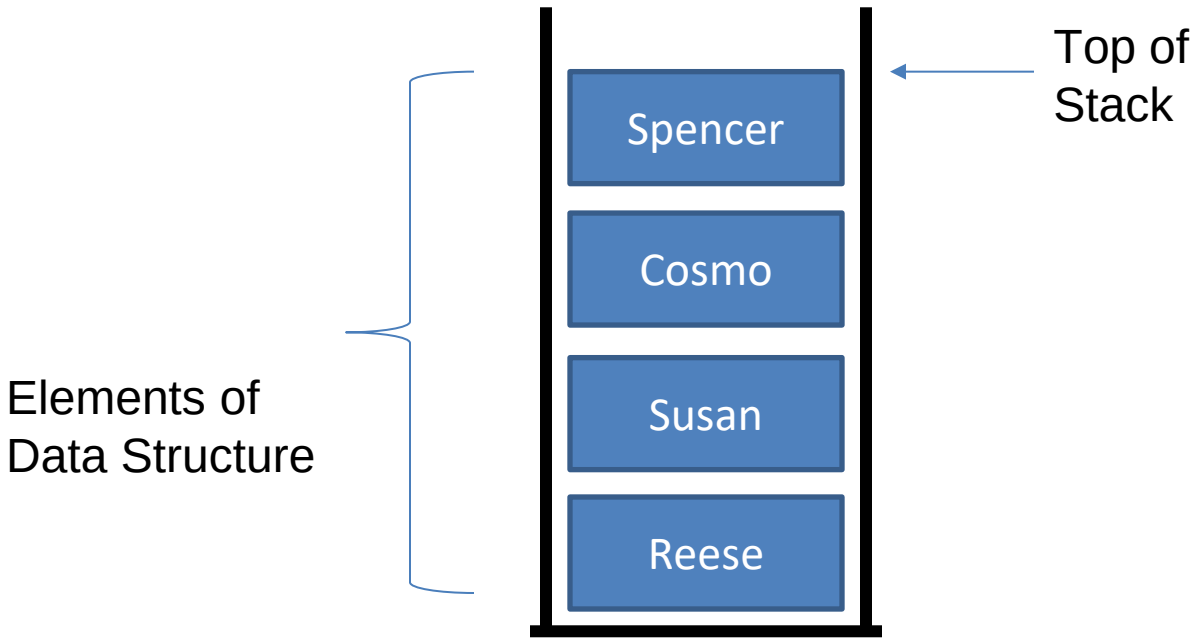
If we want to add a new element, we must put it on the top of the stack

A **stack** is a data structure that can hold data, however the way we interact with a stack is a little bit different



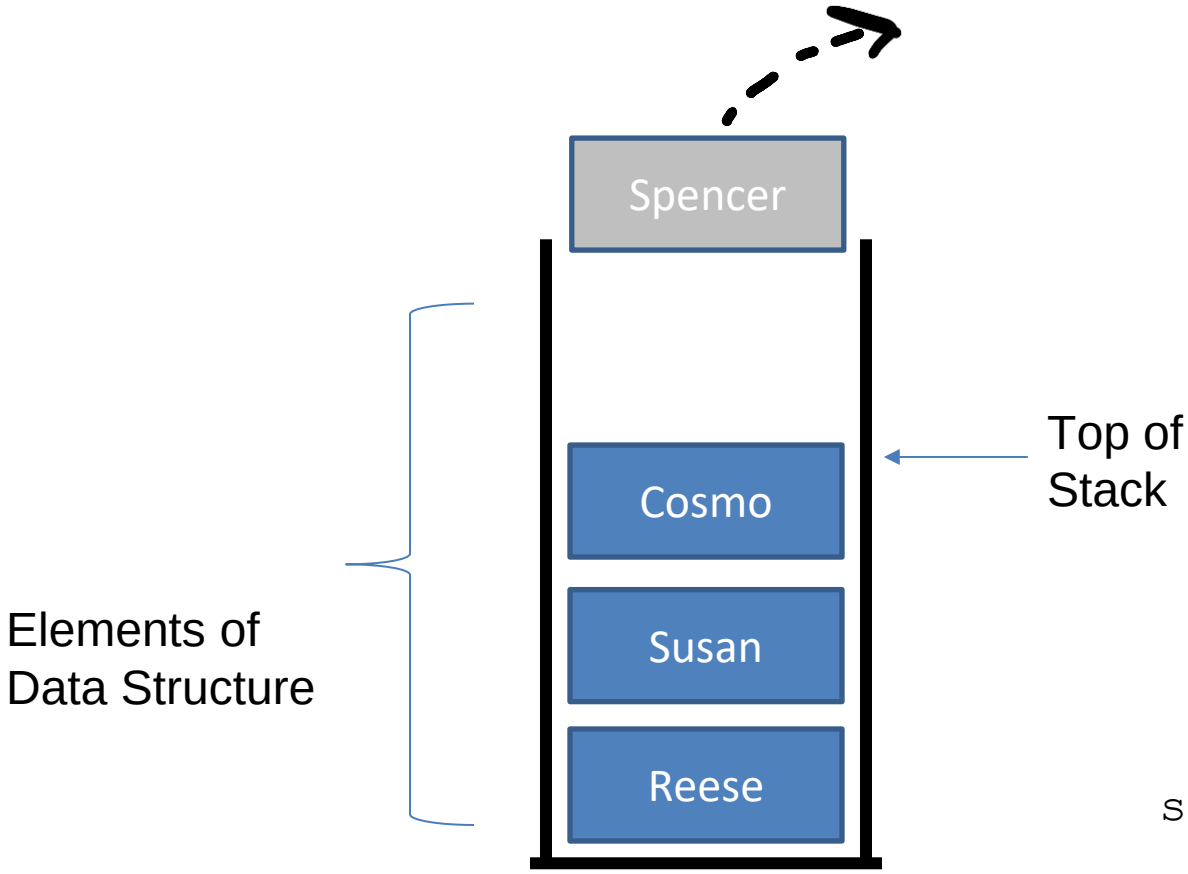

Adding something to a stack is known as the **push** operation

A **stack** is a data structure that can hold data, however the way we interact with a stack is a little bit different




If we want to remove something, we must always remove the element on the top of the stack

A **stack** is a data structure that can hold data, however the way we interact with a stack is a little bit different



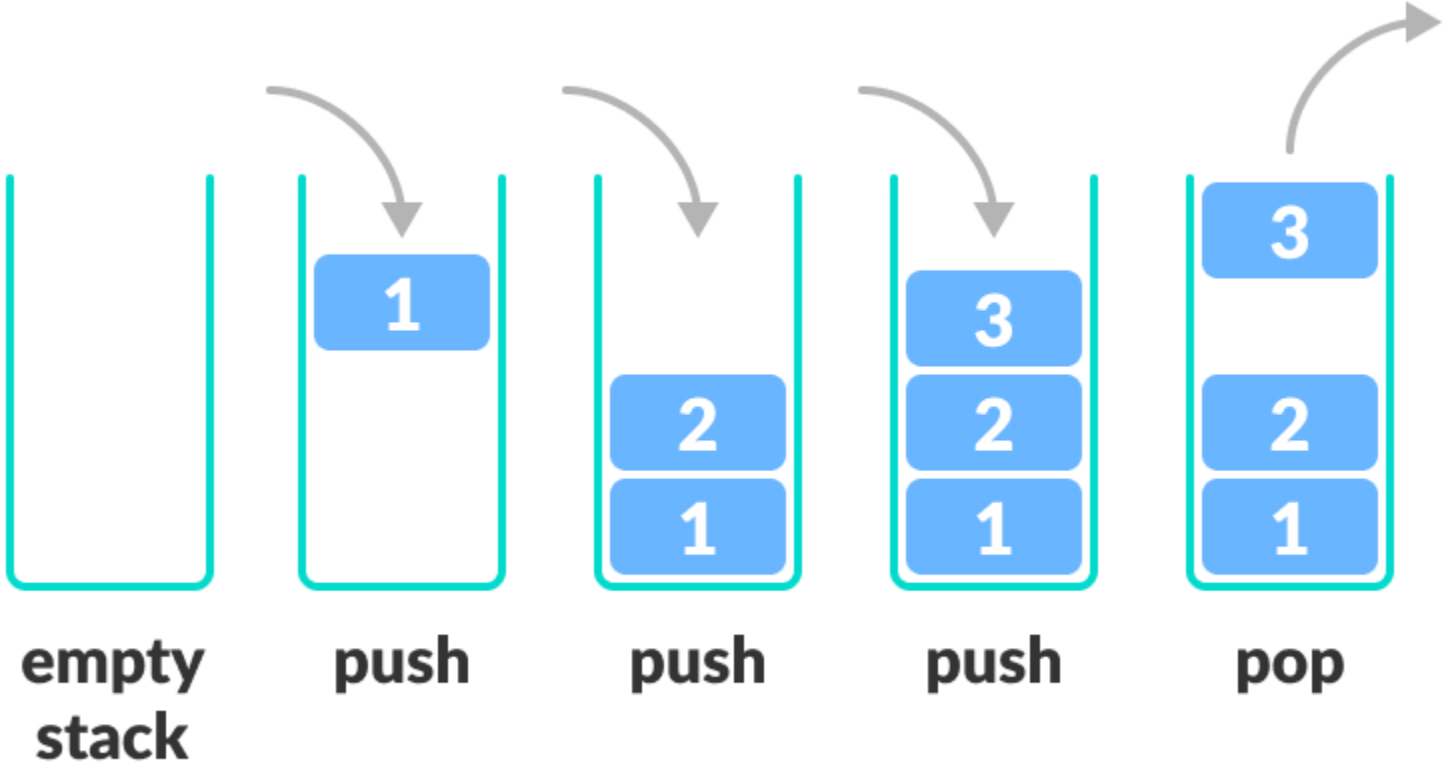

Removing an element is known as the **pop** operation

`stack.pop()` → Top node (spencer) is removed

# A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle

We can:

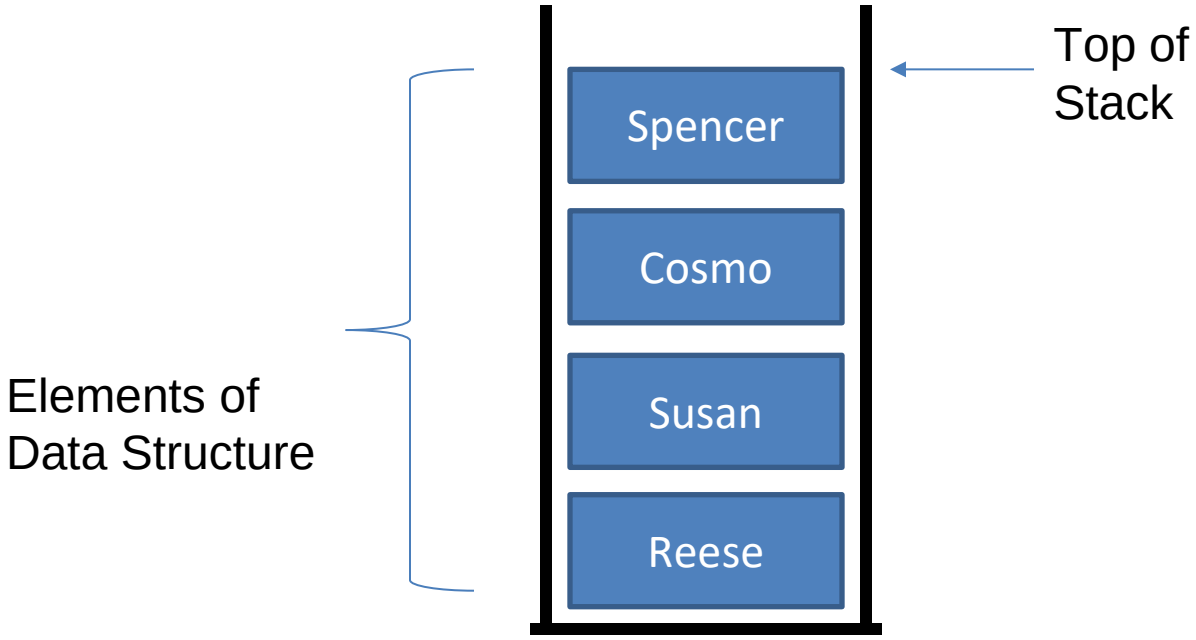
- Add an element to the top of the stack (push)
- Remove the top element (pop)




**Stack Operations**

```
push()
pop()
peek()
isEmpty()
```

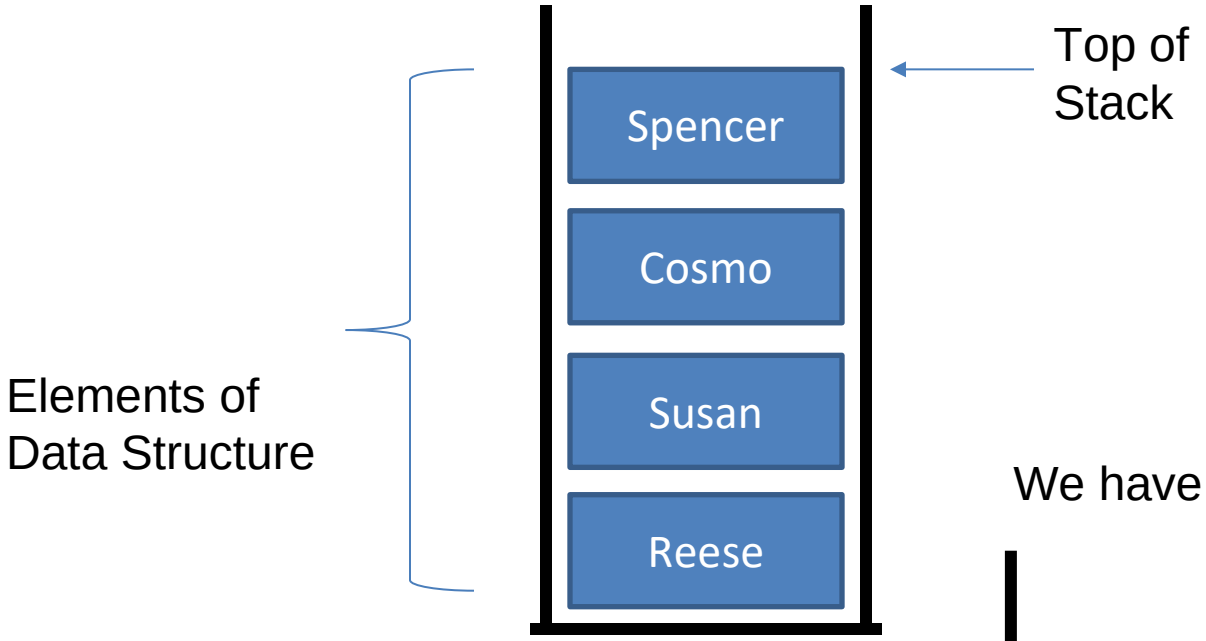
A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle




Our stack data structure needs to keep track of a few things

1. Something to hold our stack elements

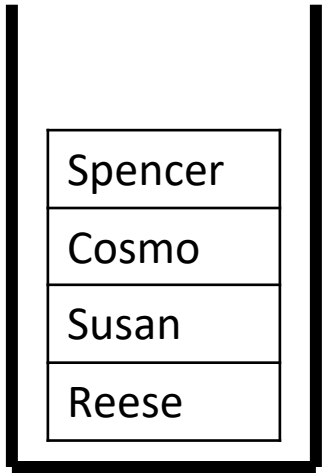
A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle




Our stack data structure needs to keep track of a few things

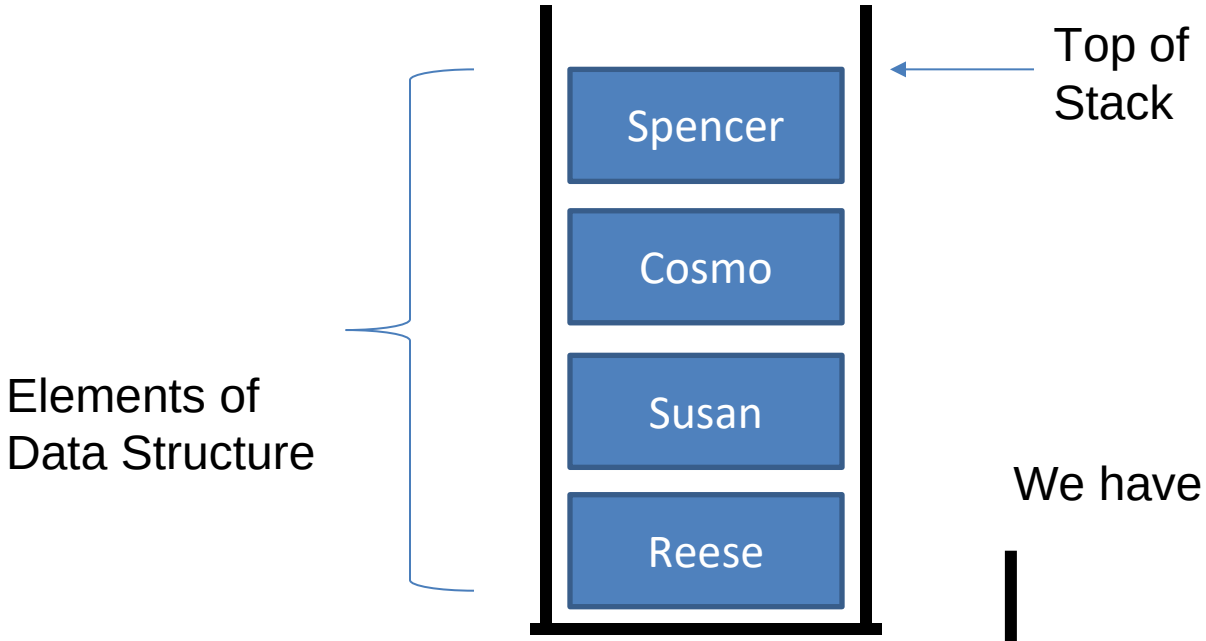
1. Something to hold our stack elements

We have a few options:




1. Array

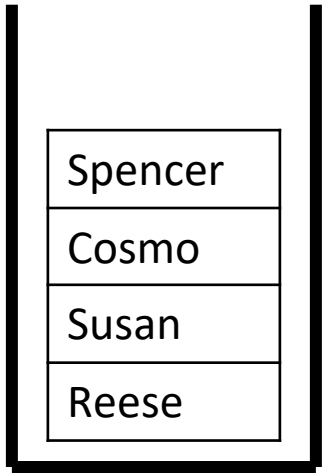
A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle




Our stack data structure needs to keep track of a few things

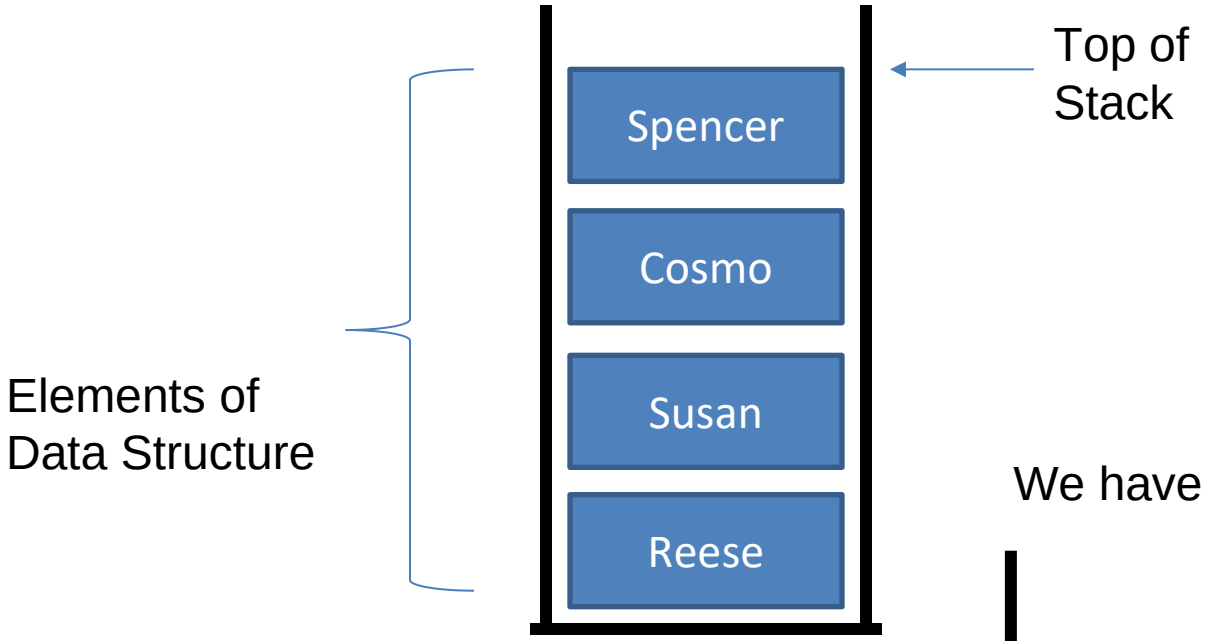
1. Something to hold our stack elements

We have a few options:




1. Array
2. ArrayList

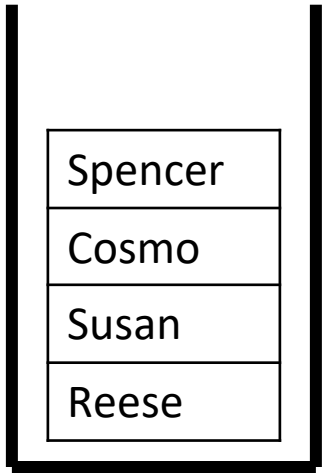
A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle



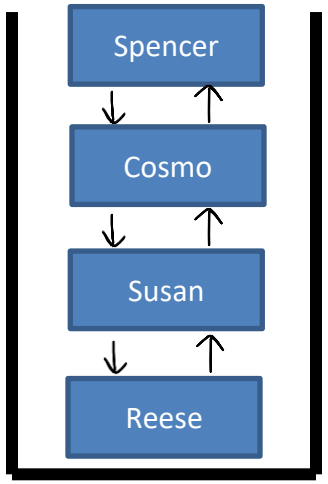

Our stack data structure needs to keep track of a few things

1. Something to hold our stack elements

We have a few options:

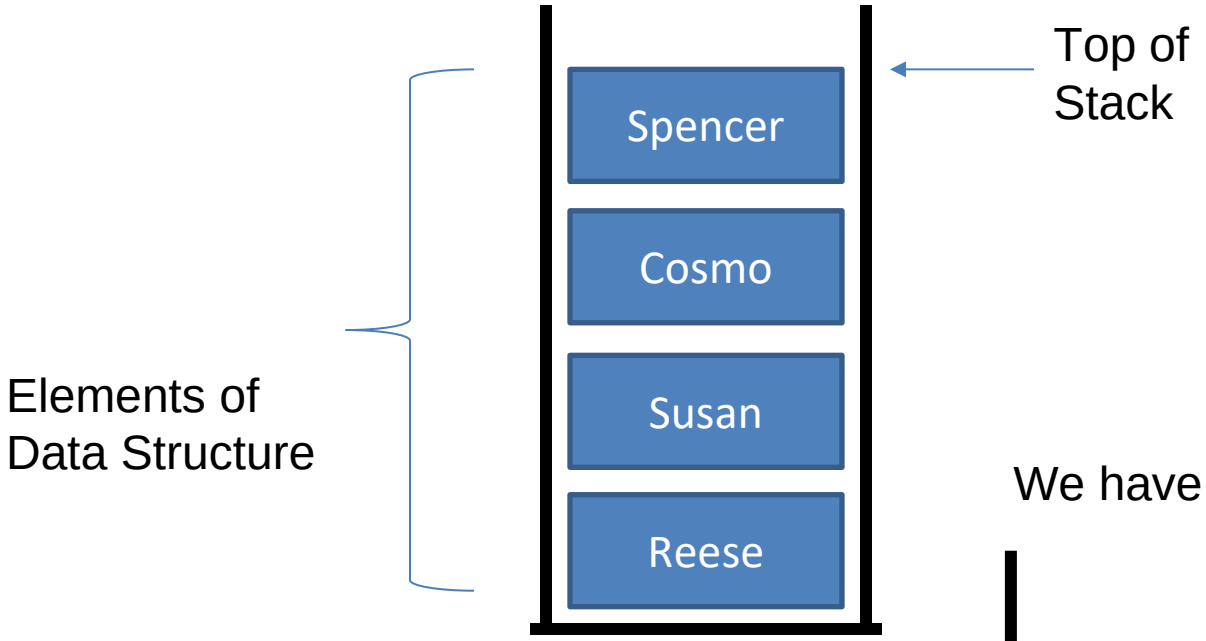



1. Array
2. ArrayList




3. Linked List

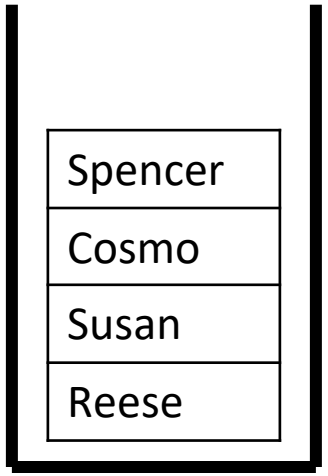
A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle



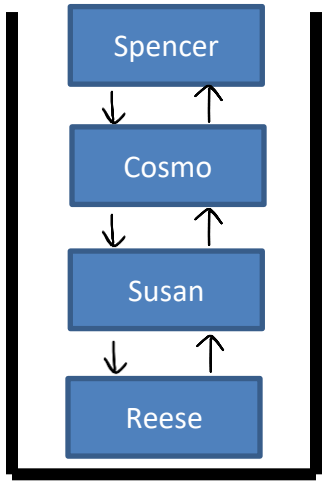

Our stack data structure needs to keep track of a few things

1. Something to hold our stack elements

We have a few options:



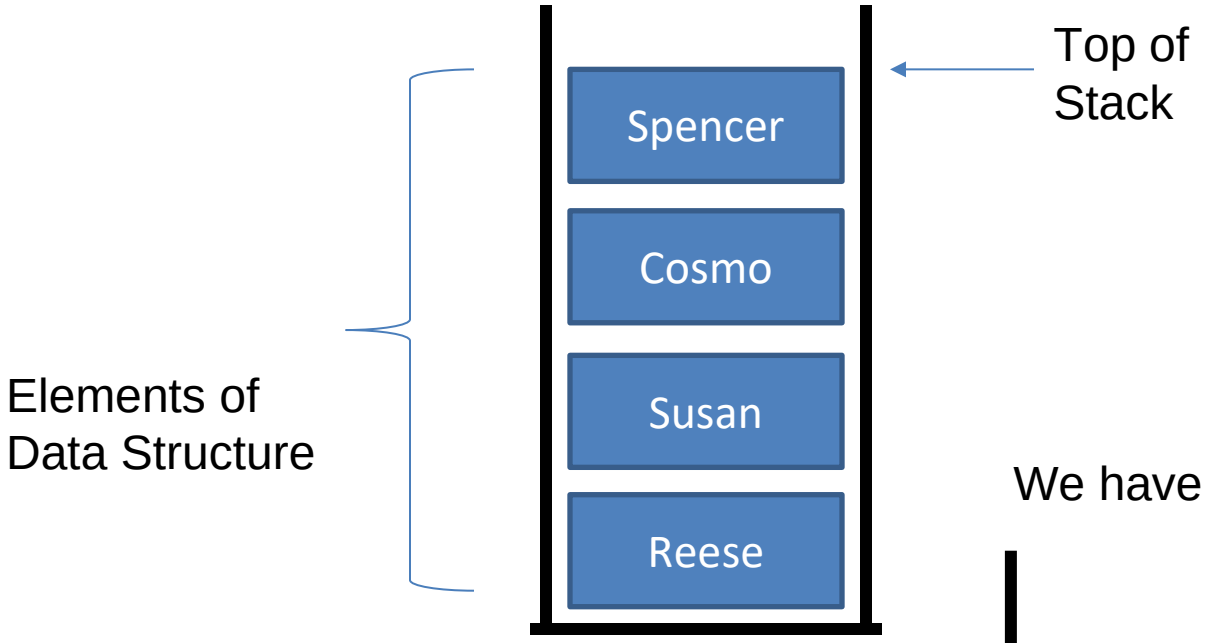

1. Array
2. ArrayList




3. Linked List

Which should you pick?

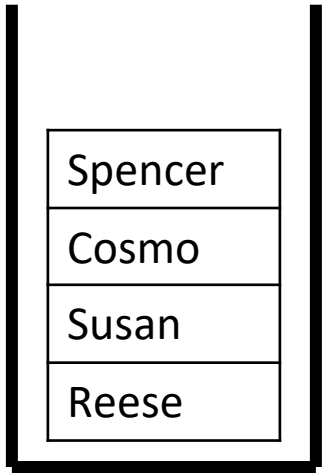
# A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle



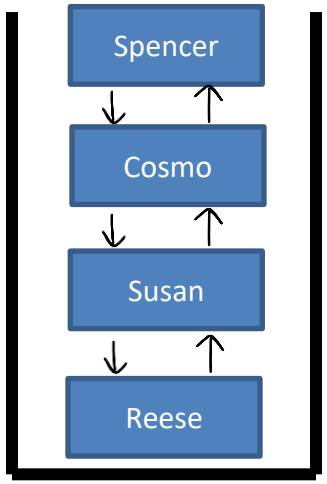

Our stack data structure needs to keep track of a few things

1. Something to hold our stack elements

We have a few options:




1. Array
2. ArrayList

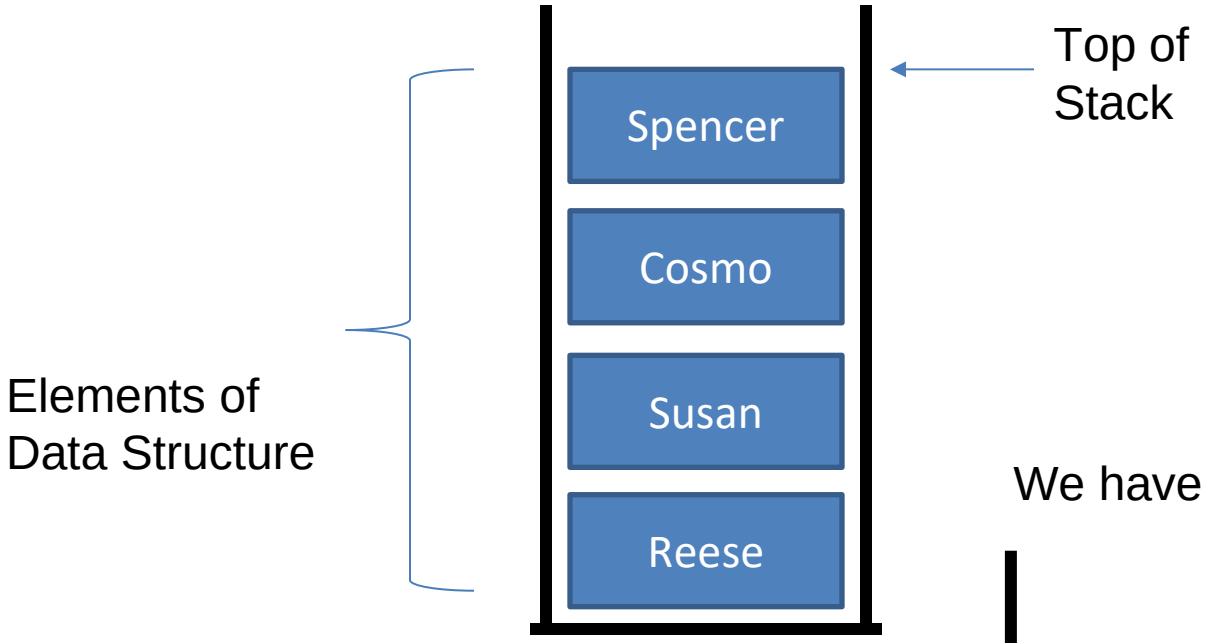



3. Linked List

Which should you pick?

- Depends on how you are using the stack

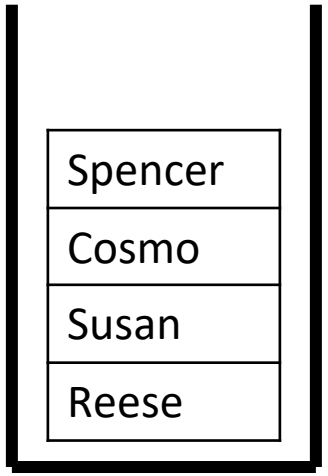
# A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle



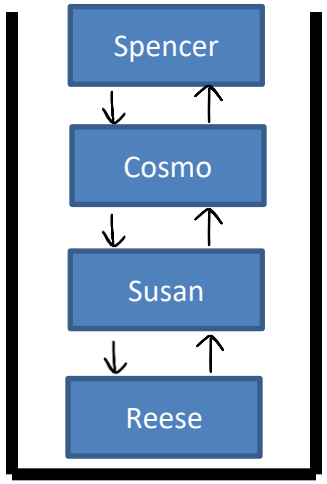

Our stack data structure needs to keep track of a few things

1. Something to hold our stack elements

We have a few options:




1. Array
2. ArrayList

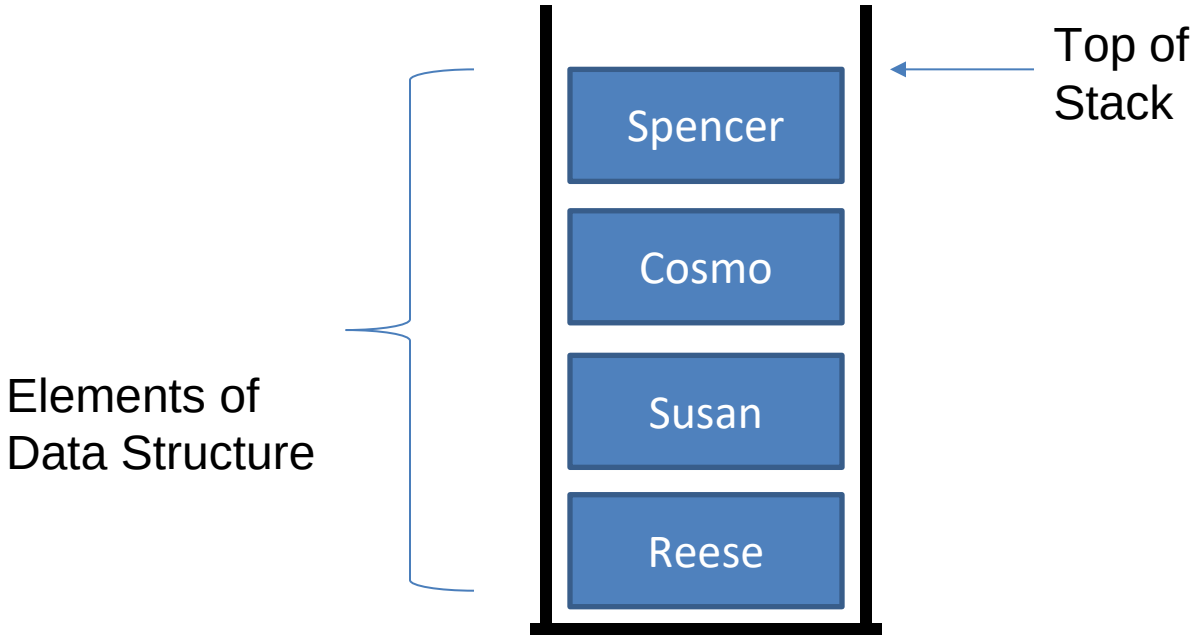



3. Linked List

## Which should you pick?

- If you know how big the stack needs to be
  → Array
- If you don't know how big the stack needs to be
  → Linked List

A **stack** is a data structure that can hold data, and follows the **last in first out (LIFO)** principle

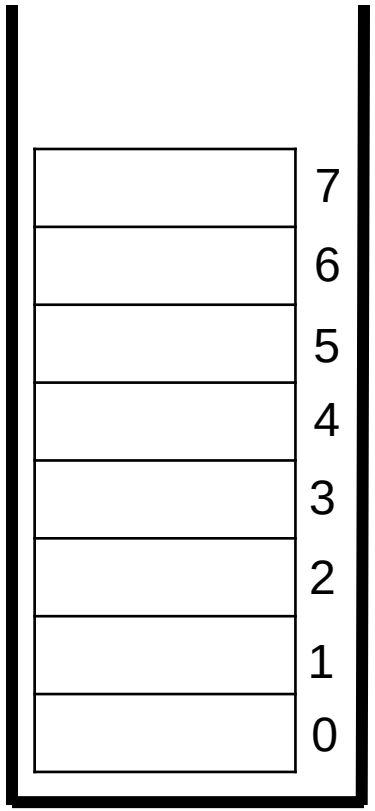



Our stack data structure needs to keep track of a few things

1. Something to hold our stack elements (Array/LinkedList)
2. Something that points the current top element of the stack
3. The size of the stack

# Stack Implementation (Array)

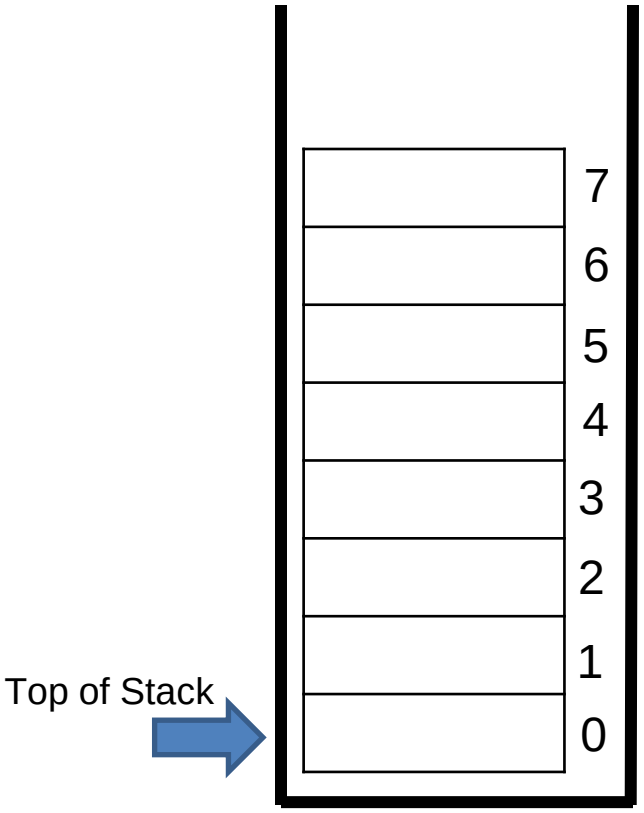
Here, we've created an array of size 8 to hold our stack data




To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data




To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

The bottom of the stack will always be at index 0, and grows towards the higher indices

```
String[] data = new String[8]
```

When the stack is empty, the index of the bottom of the stack, and the index of the top of the stack will be the same

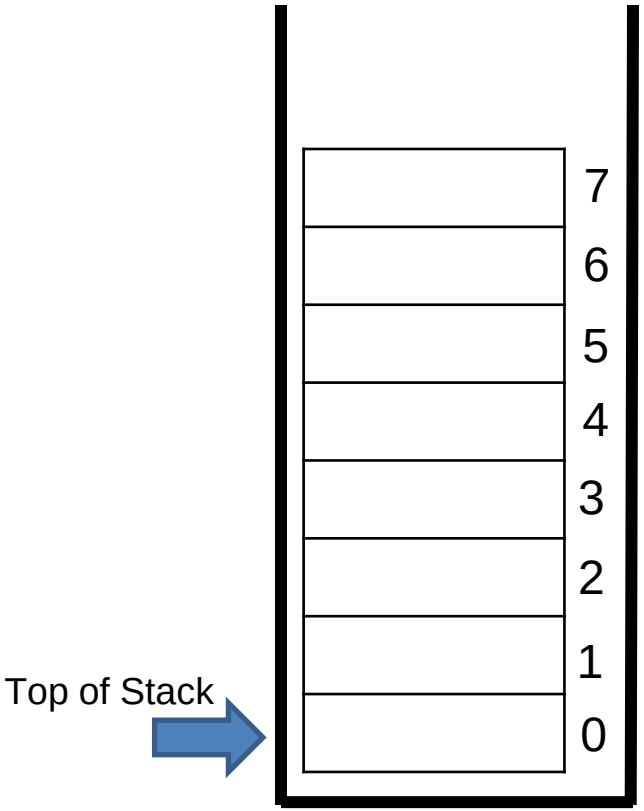
```
top_of_stack = 0
```

The size of the stack will start at 0

```
size = 0
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data




```
public void push(newElement){




}
```

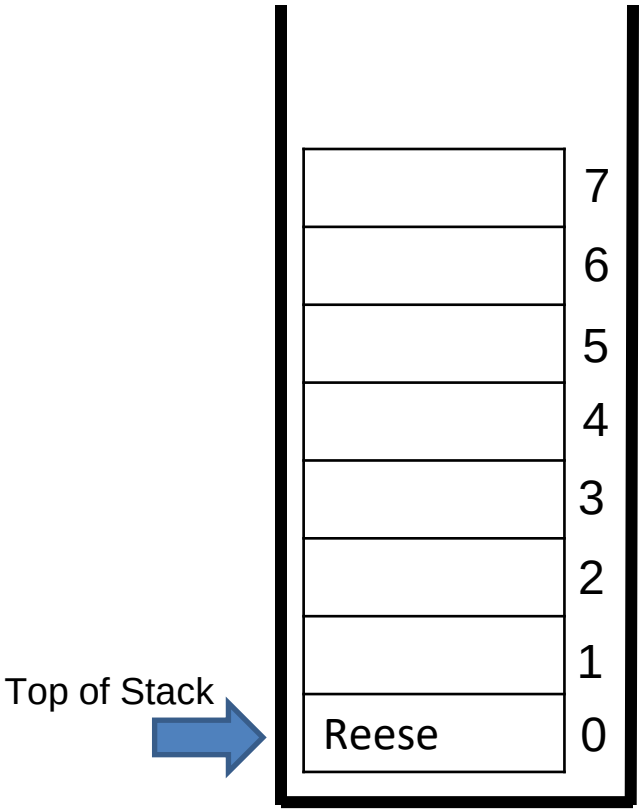
To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

Stack Instance Fields

```
String[] data = new String[8]
top_of_stack = 0
size = 0
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return




}
```

To Do List:

- Push()
- Pop()
- Peek()
- IsEmpty()

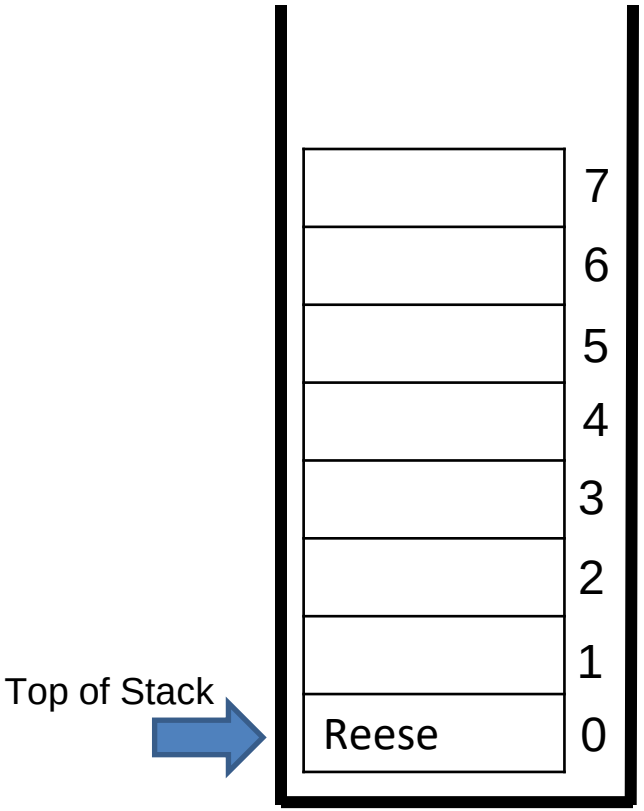
Stack Instance Fields

```
String[] data = new String[8]
                top_of_stack = 0
                        size = 1
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

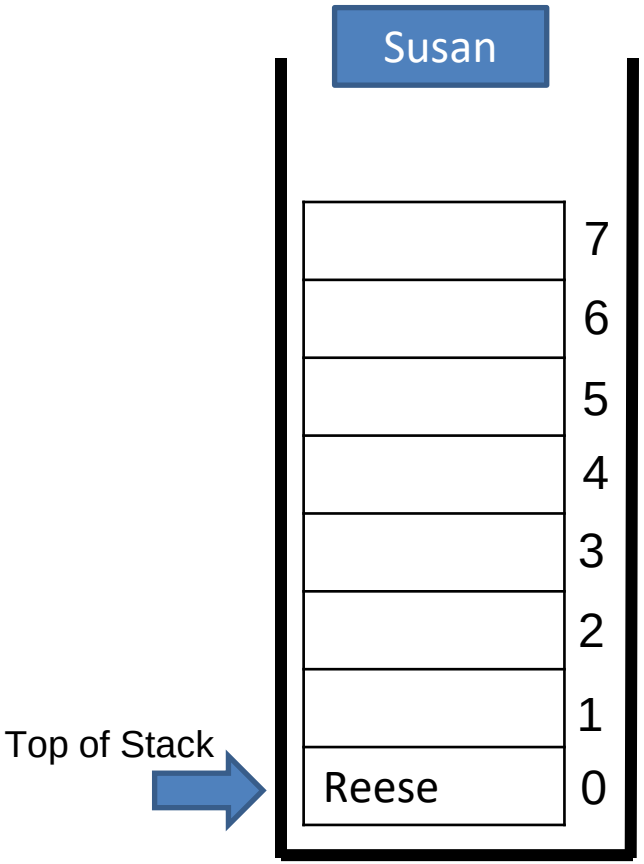
Stack Instance Fields

```
String[] data = new String[8]
            top_of_stack = 0
                    size = 1
```

# Stack Implementation (Array)

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

Here, we've created an array of size 8 to hold our stack data




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

```
stack.push("Susan")
```

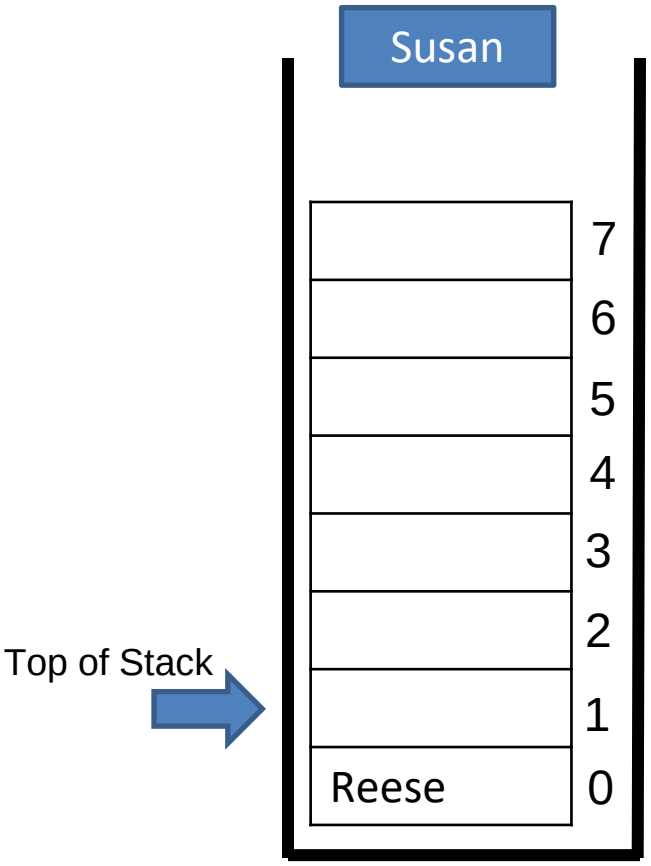
Stack Instance Fields

```
String[] data = new String[8]
top_of_stack = 0
size = 1
```

# Stack Implementation (Array)

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

Here, we've created an array of size 8 to hold our stack data




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

```
stack.push("Susan")
```

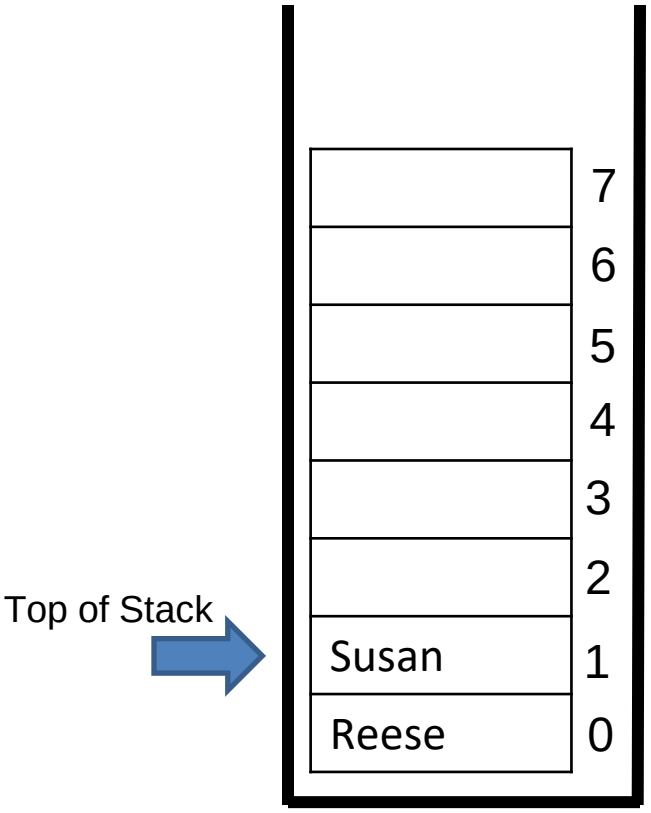
Stack Instance Fields

```
String[] data = new String[8]
top_of_stack = 1
size = 1
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

```
stack.push("Susan")
```

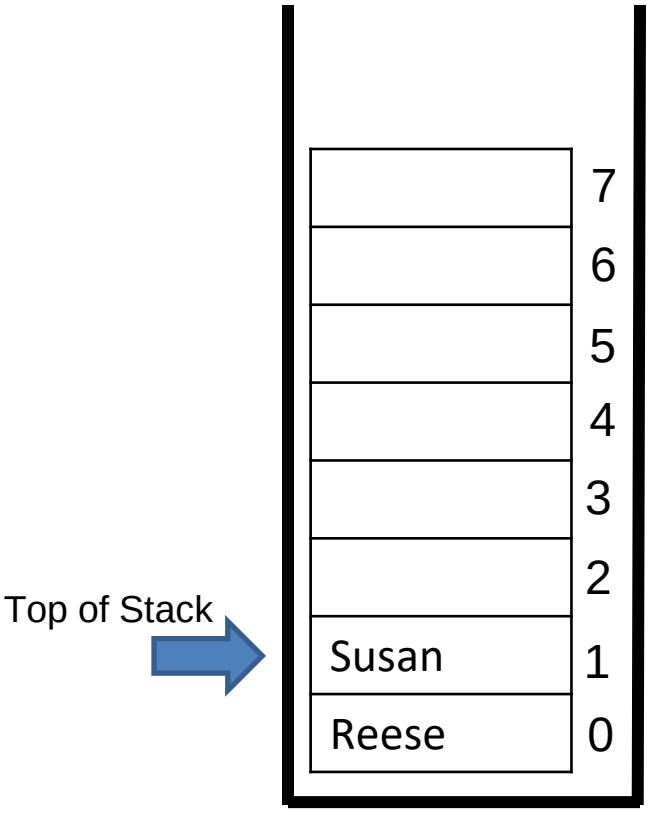
Stack Instance Fields

```
String[] data = new String[8]
top_of_stack = 1
size = 1
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

```
stack.push("Susan")
```

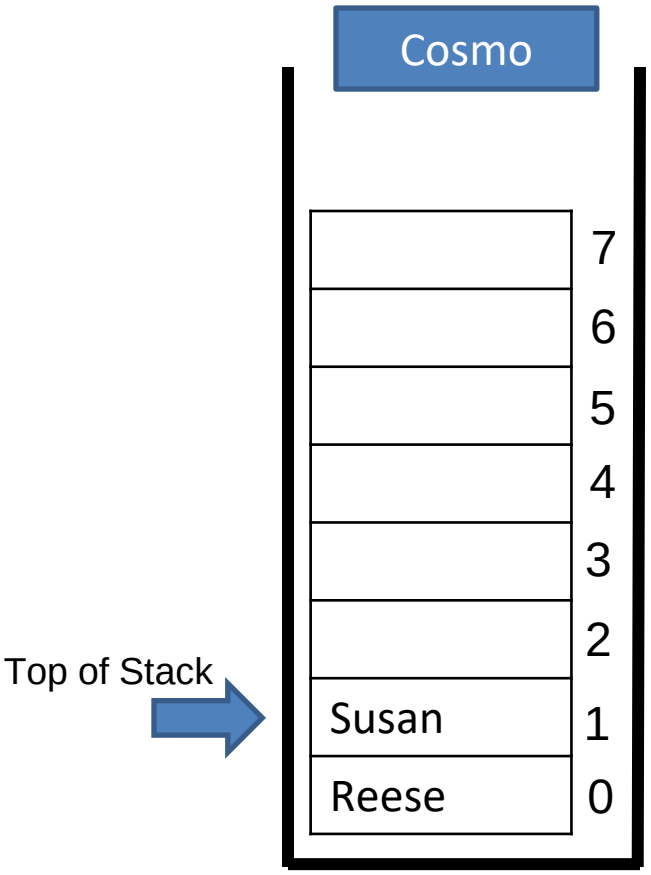
Stack Instance Fields

```
String[] data = new String[8]
top_of_stack = 1
size = 2
```

# Stack Implementation (Array)

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

Here, we've created an array of size 8 to hold our stack data




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

Stack Instance Fields

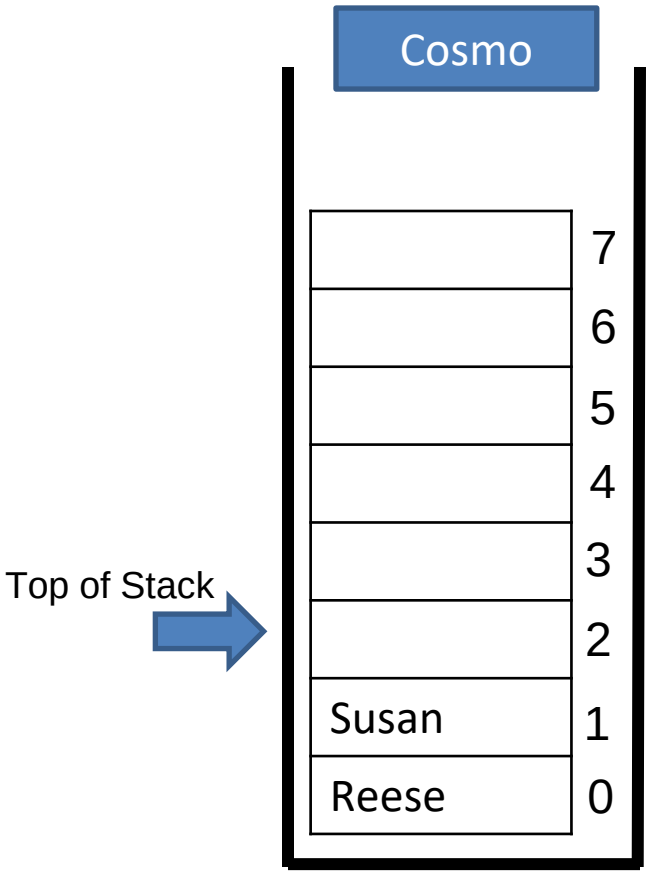
```
String[] data = new String[8]
top_of_stack = 1
size = 2
```

```
stack.push("Cosmo")
```

# Stack Implementation (Array)

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

Here, we've created an array of size 8 to hold our stack data




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

Stack Instance Fields

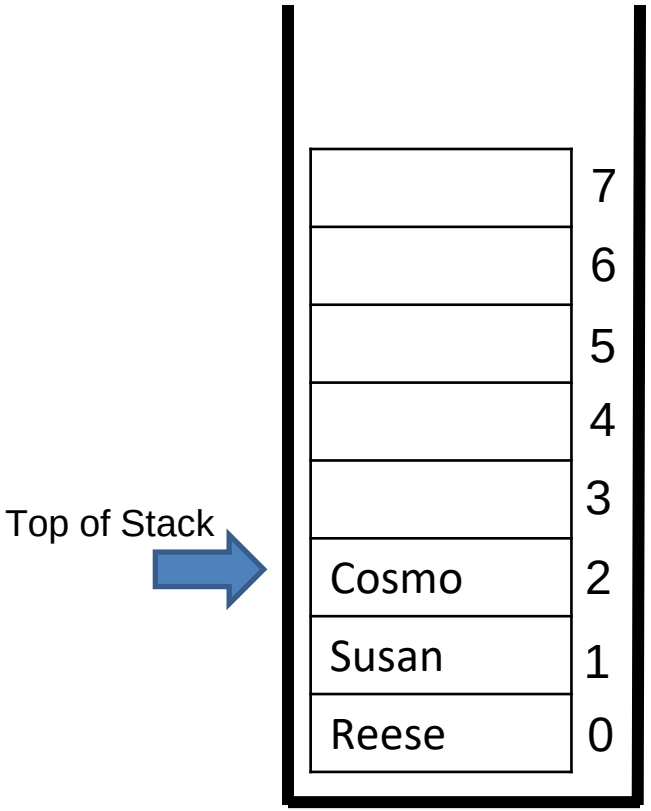
```
String[] data = new String[8]
top_of_stack = 2
size = 2
```

```
stack.push("Cosmo")
```

# Stack Implementation (Array)

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

Here, we've created an array of size 8 to hold our stack data




```
public void push(newElement){

  if stack is empty:
      place newElement at current top_of_stack
      size++


  if stack if full:
      return


  else:
       top_of_stack++;
       place newElement at index top_of_stack
       size++
}
```

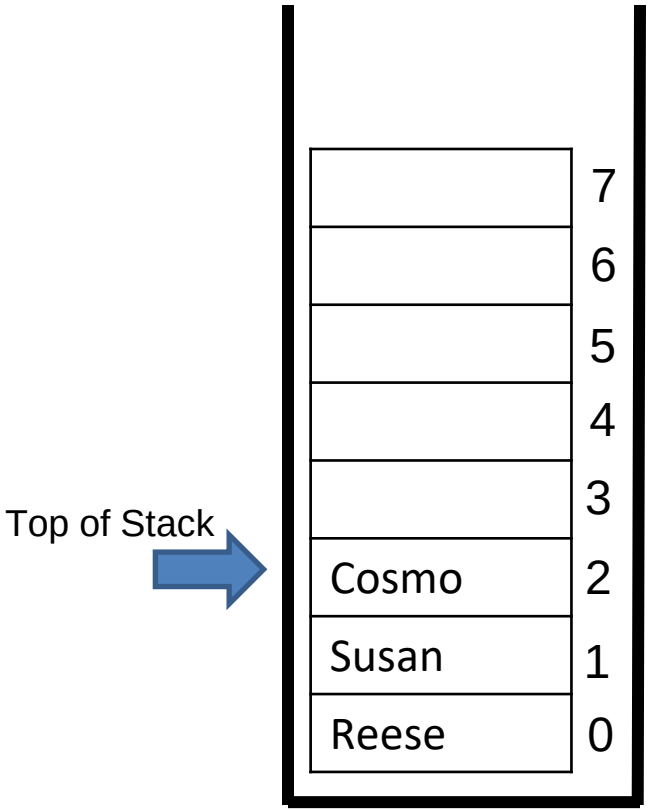
Stack Instance Fields

```
String[] data = new String[8]
                top_of_stack = 2
                        size = 3
```

```
stack.push("Cosmo")
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data




```
public void pop(){

}
```

The pop method will always remove the element on the top of the stack

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()

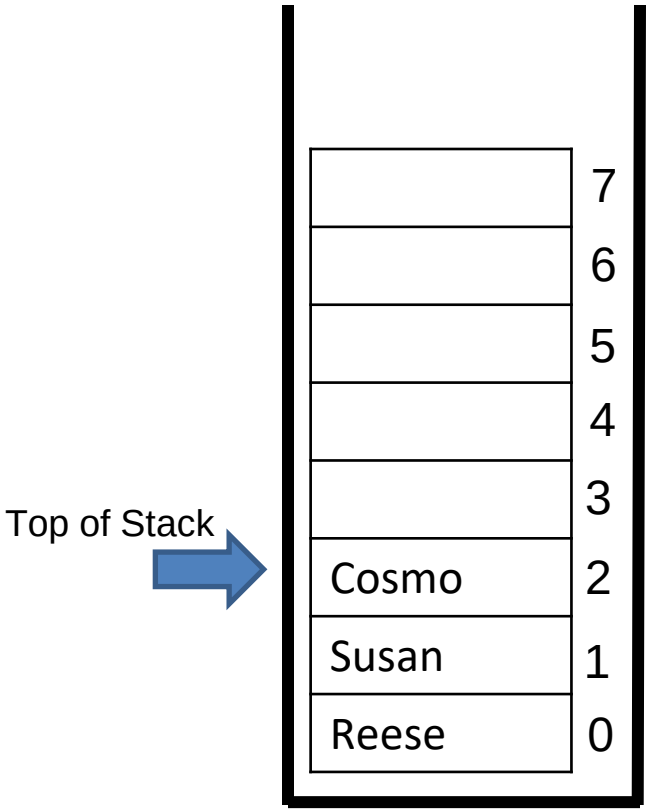
Stack Instance Fields

```
String[] data = new String[8]
top_of_stack = 2
size = 3
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void pop(){

   if stack is empty:
       return

   Set index top_of_stack to be null
   top_of_stack--
   size--

}
```

Stack Instance Fields

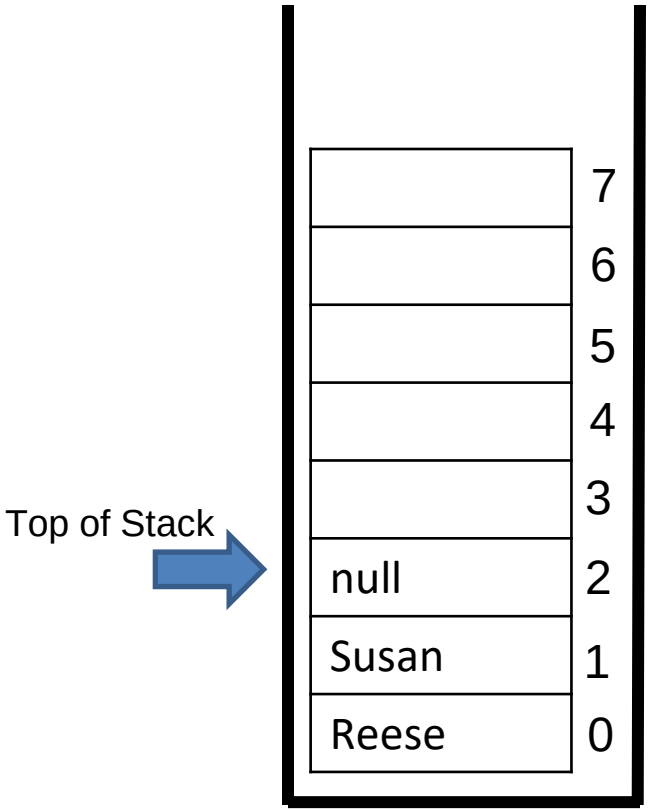
```
String[] data = new String[8]
                top_of_stack = 2
                        size = 3
```

```
stack.pop()
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void pop(){

   if stack is empty:
       return

   Set index top_of_stack to be null
   top_of_stack--
   size--

}
```

Stack Instance Fields

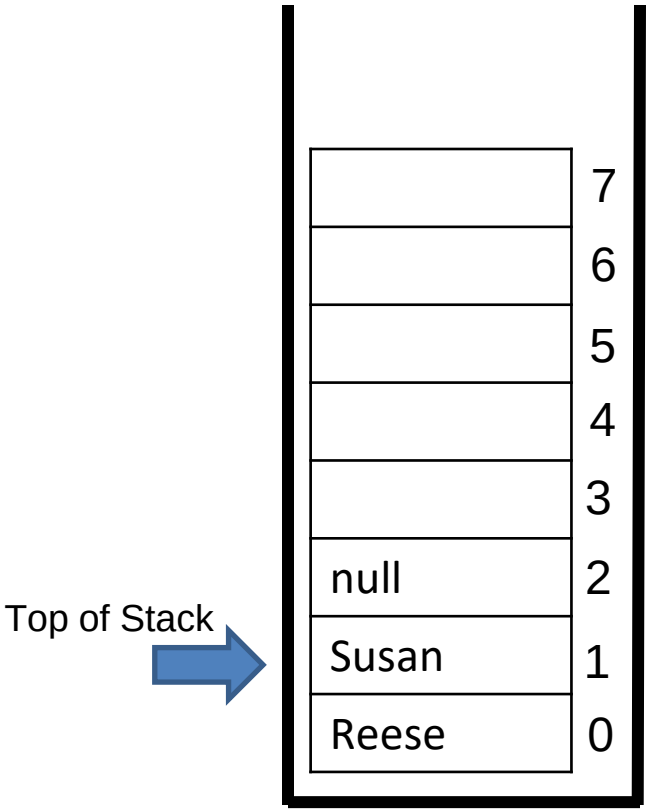
```
String[] data = new String[8]
                top_of_stack = 2
                        size = 3
```

```
stack.pop()
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void pop(){

   if stack is empty:
       return

   Set index top_of_stack to be null
   top_of_stack--
   size--

}
```

Stack Instance Fields

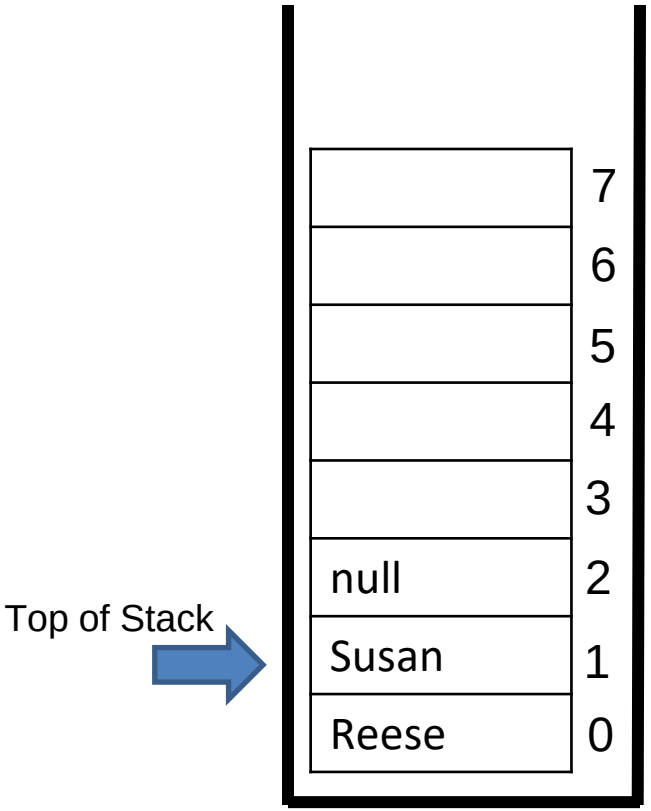
```
String[] data = new String[8]
                top_of_stack = 1
                        size = 3
```

```
stack.pop()
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void pop(){

   if stack is empty:
       return

   Set index top_of_stack to be null
   top_of_stack--
   size--

}
```

Stack Instance Fields

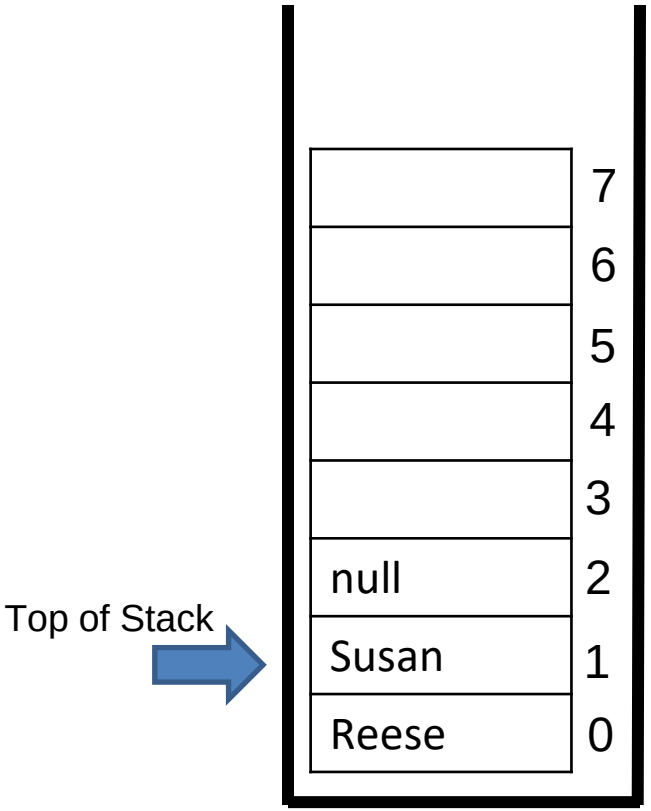
```
String[] data = new String[8]
top_of_stack = 1
size = 2
```

```
stack.pop()
```

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public void pop(){

   if stack is empty:
       return

   Set index top_of_stack to be null
   top_of_stack--
   size--

}
```

Stack Instance Fields

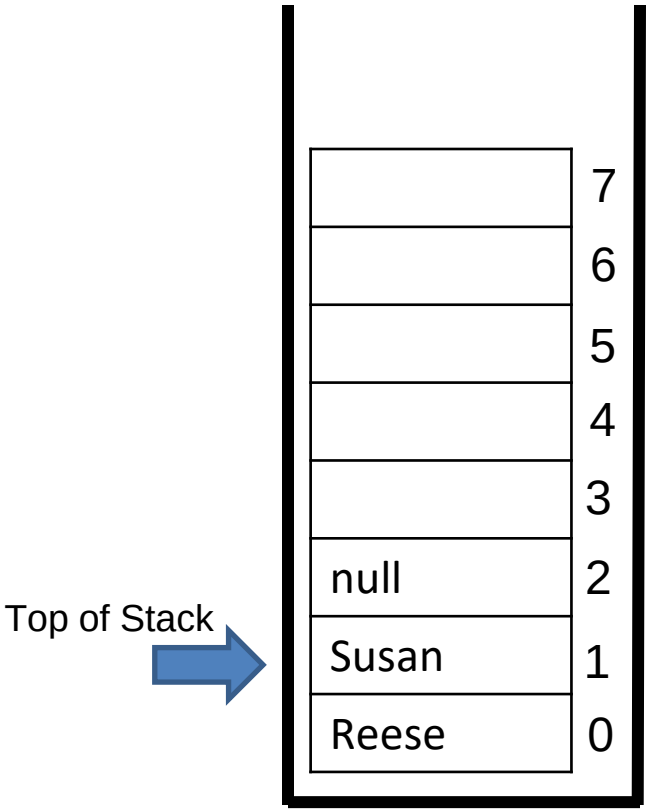
```
String[] data = new String[8]
                top_of_stack = 1
                        size = 2
```

Note: This method does not return the element that was removed, however there may be times where the pop() method returns the element that got removed

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public String peek(){



}
```

Stack Instance Fields

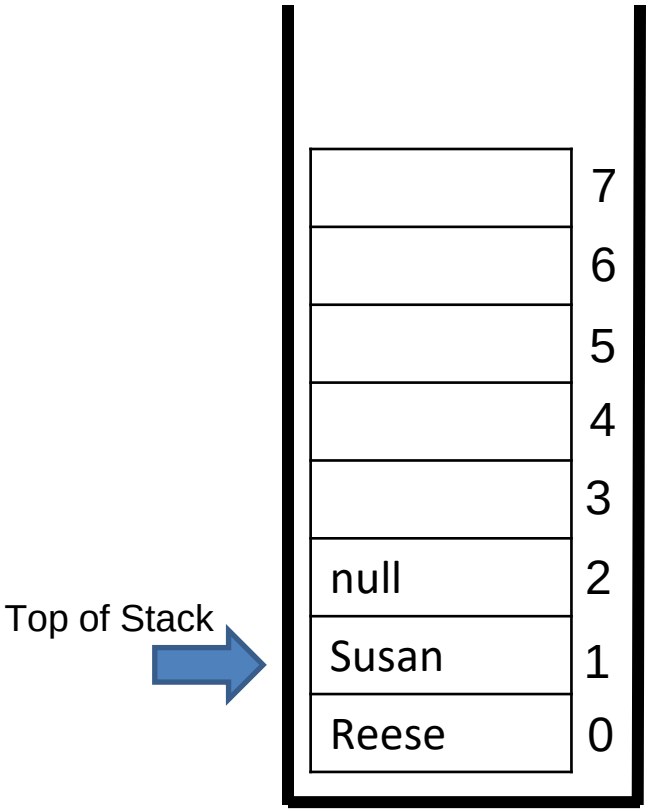
```
String[] data = new String[8]
top_of_stack = 1
size = 2
```

The `peek()` method returns the element that is currently on the top of the stack

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public String peek(){


  If stack is not empty:
     return data[top_of_stack]


}
```

Stack Instance Fields

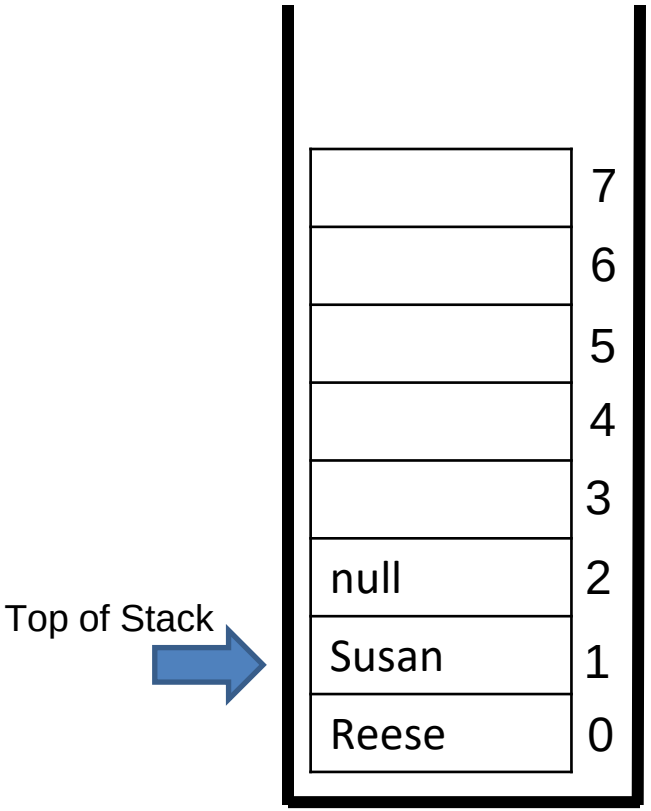
```
String[] data = new String[8]
top_of_stack = 1
size = 2
```

The `peek()` method returns the element that is currently on the top of the stack

# Stack Implementation (Array)

Here, we've created an array of size 8 to hold our stack data

To Do List:
- Push()
- Pop()
- Peek()
- IsEmpty()




```
public boolean isEmpty(){


    if size == 0:
        return true

    else:
        return false
}
```

Stack Instance Fields

```
String[] data = new String[8]
top_of_stack = 1
size = 2
```

The `isEmpty()` method returns a boolean: true if the stack is empty, false if the stack is not empty