Code School
a Pluralsight company

# Practical Swift Generics

*Jon Friskics*

*August 22, 2016*

## What are generics?

- *A way to write classes and functions without declaring a specific type but still have a strong type set when the function is called*
- *Write less code while preserving type safety*

## Have we ever had anything like Generics?

| | |
|---|---|
| `NSString` | *Always an NSString* |
| `id` | *Can be any type at any point (even an NSString)* |
| `AnyObject/Any` | *Like id, but for Swift* |

*Generics live somewhere in here*

## The Book Example - Swapping Values

two Ints come in

```
func swapTwoInts(firstInt: Int, secondInt: Int) -> (Int, Int) {
    let temporary = firstInt
    let firstInt = secondInt
    let secondInt = temporary
    return (firstInt, secondInt)
}

let swappedInts = swapTwoInts(firstInt: 5, secondInt: 7)
print(swappedInts) // (7,5)
```

two Ints go out

# The Book Example - Swapping Values

two Strings come in

```
func swapTwoStrings(firstString: String, secondString: String) -> (String, String) {
    let temporary = firstString
    let firstString = secondString
    let secondString = temporary
    return (firstString, secondString)
}


let swappedStrings = swapTwoStrings(firstString: "Hello", secondString: "Friend")
print(swappedStrings)  // ("Friend", "Hello")
```

two Strings go out

## The Book Example - Swapping Values

*What if we could generalize these...*

```
func swapTwoStrings(firstString: String, secondString: String) -> (String, String) {}

func swapTwoInts(firstInt: Int, secondInt: Int) -> (Int, Int) {}

func swapTwoWhatevers...
```

*...to this?*

```
func swapTwoThings(firstThing: Thing, secondThing: Thing) -> (Thing, Thing) {}
```

# The Book Example - Swapping Values

```
func swapTwoThings(firstThing: Thing, secondThing: Thing) -> (Thing, Thing) {
    let temporary = firstThing
    let firstThing = secondThing
    let secondThing = temporary
    return (secondThing, firstThing)
}
```

```
error: use of undeclared type 'Thing'
```

*We want to declare the type, but not outside of the function*

## The Book Example - Swapping Values

*type parameter*

```
func swapTwoThings<Thing>(firstThing: Thing, secondThing: Thing) -> (Thing, Thing) {
    let temporary = firstThing
    let firstThing = secondThing
    let secondThing = temporary
    return (secondThing, firstThing)
}
```

The type **Thing** now only exists to keep the compiler happy inside of this function block

# The Book Example - Swapping Values

**Thing** gets replaced with whatever type you use to call the function

```swift
func swapTwoThings<Thing>(firstThing: Thing, secondThing: Thing) -> (Thing, Thing) {
    let temporary = firstThing
    let firstThing = secondThing
    let secondThing = temporary
    return (secondThing, firstThing)
}

let swappedThings = swapTwoThings(firstThing: 3, secondThing: 6)
print(swappedThings)  // (6, 3)
```

Int

Also an Int

# The Book Example - Swapping Values

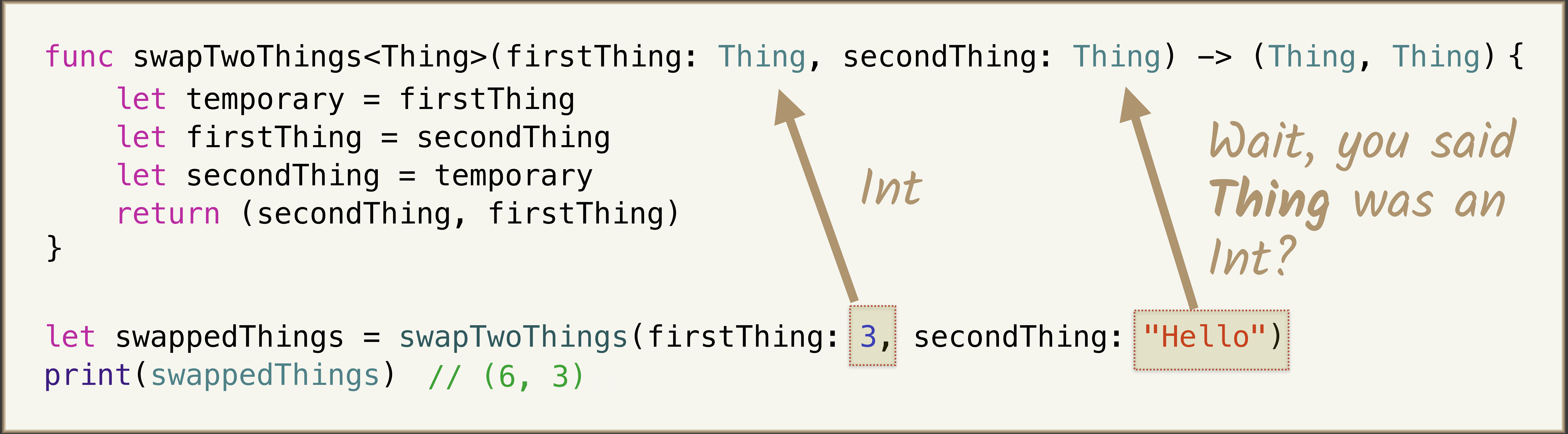
**Thing** gets replaced with whatever type you use to call the function

```swift
func swapTwoThings<Thing>(firstThing: Thing, secondThing: Thing) -> (Thing, Thing) {
    let temporary = firstThing
    let firstThing = secondThing
    let secondThing = temporary
    return (secondThing, firstThing)
}

let swappedThings = swapTwoThings(firstThing: 3, secondThing: 6)
print(swappedThings)  // (6, 3)

let swappedThings = swapTwoThings(firstThing: "Hello", secondThing: "Friend")
print(swappedThings)  // ("Friend, "Hello")
```

*String*

*Also a String*

# The Book Example - Swapping Values



```swift
func swapTwoThings<Thing>(firstThing: Thing, secondThing: Thing) -> (Thing, Thing) {
    let temporary = firstThing
    let firstThing = secondThing
    let secondThing = temporary
    return (secondThing, firstThing)
}

let swappedThings = swapTwoThings(firstThing: 3, secondThing: "Hello")
print(swappedThings)  // (6, 3)
```

```
error: cannot convert value of type 'String' to expected argument type 'Int'
```

Once a type parameter is set in a function block,
it can't change for that run of the function

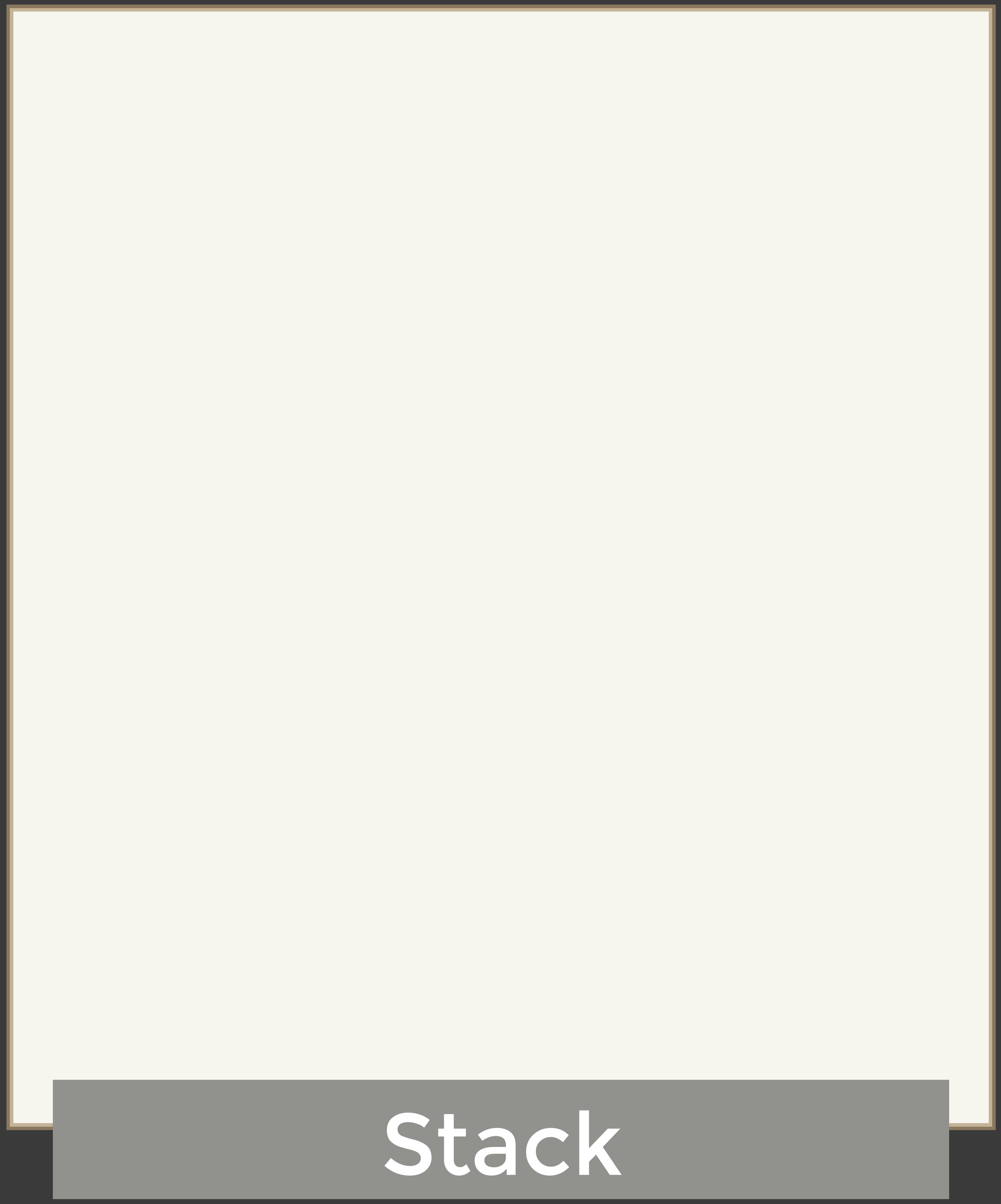
## Why not just use AnyObject?

> *id or AnyObject can change at any time, but once a generic type is defined, it's set for that block*

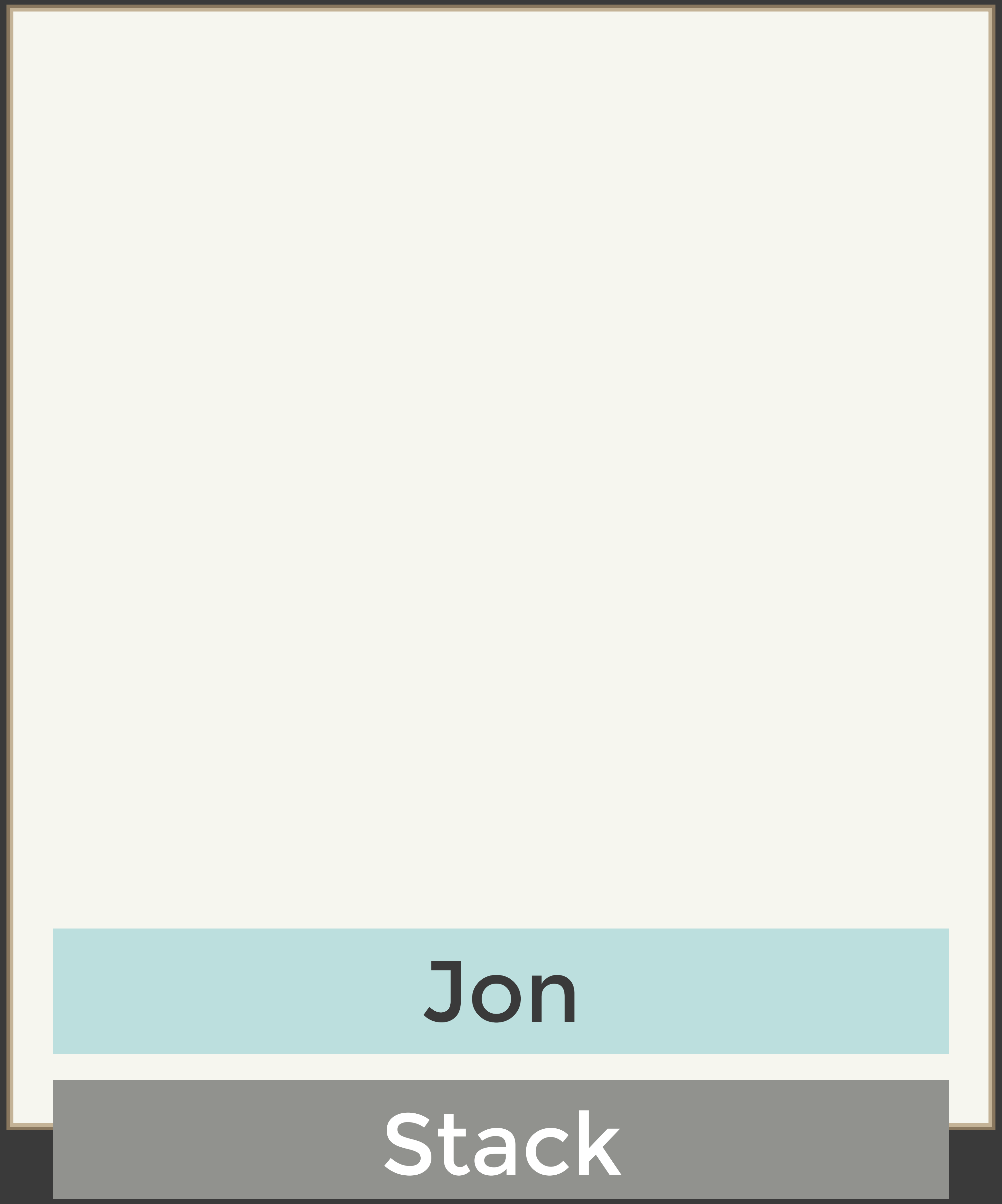
## Where should I use Generics?

> *Look for places where you're writing the same code multiple times for different types*

## A Generic Stack Data Structure

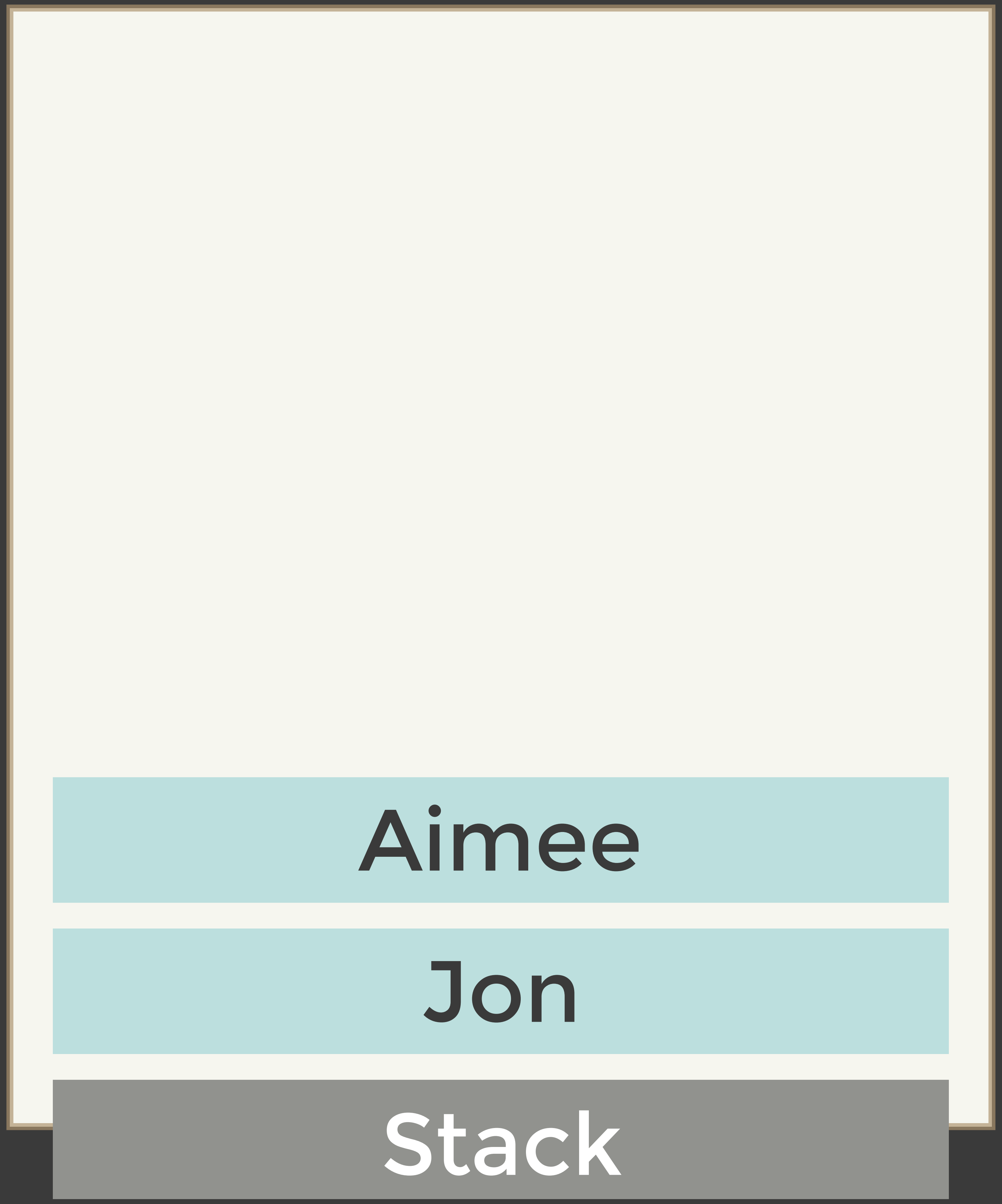


## A Generic Stack Data Structure



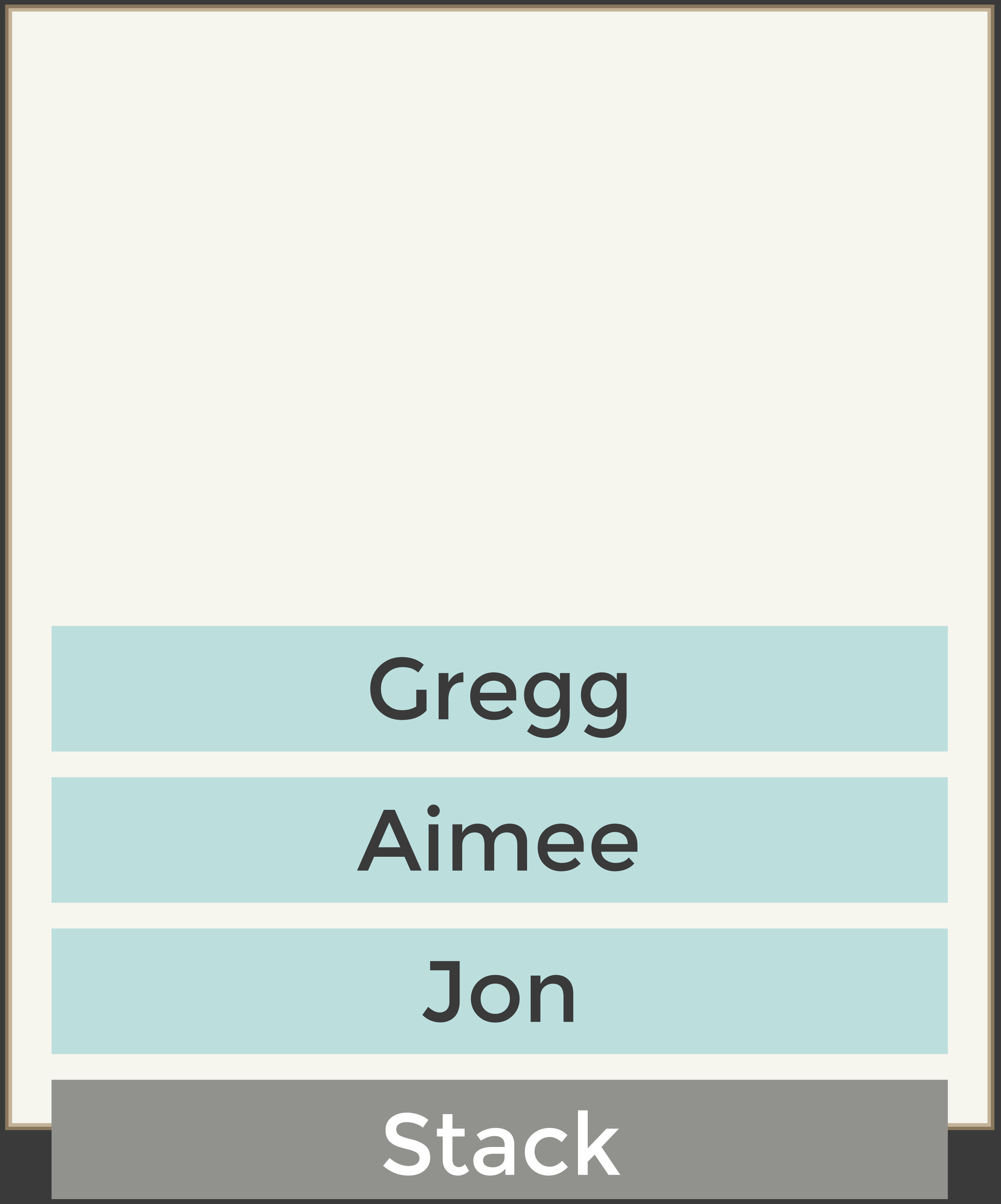
```
> add "Jon" to Stack
```

## A Generic Stack Data Structure



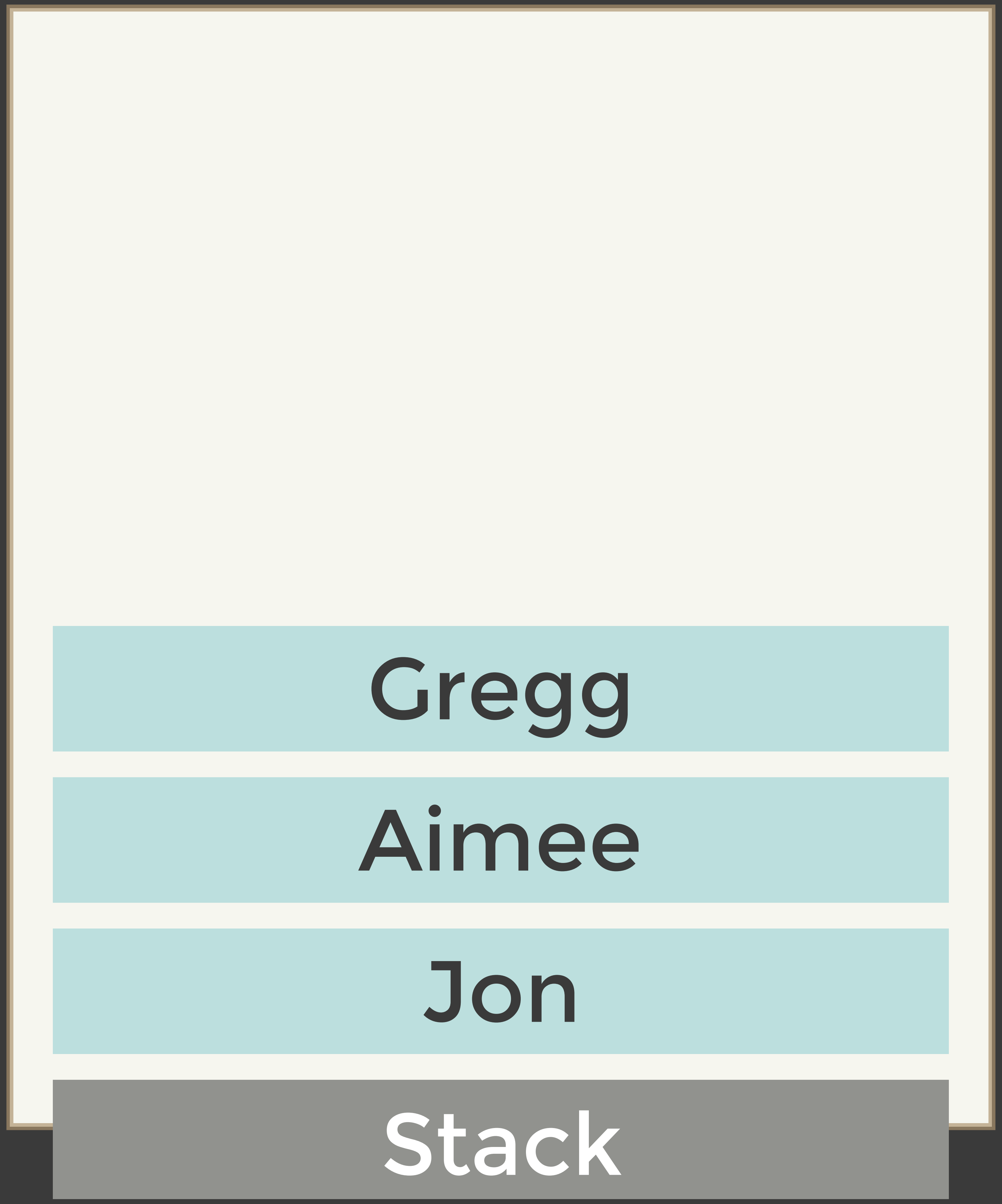
```
> add "Jon" to Stack
> add "Aimee" to Stack
```

# A Generic Stack Data Structure



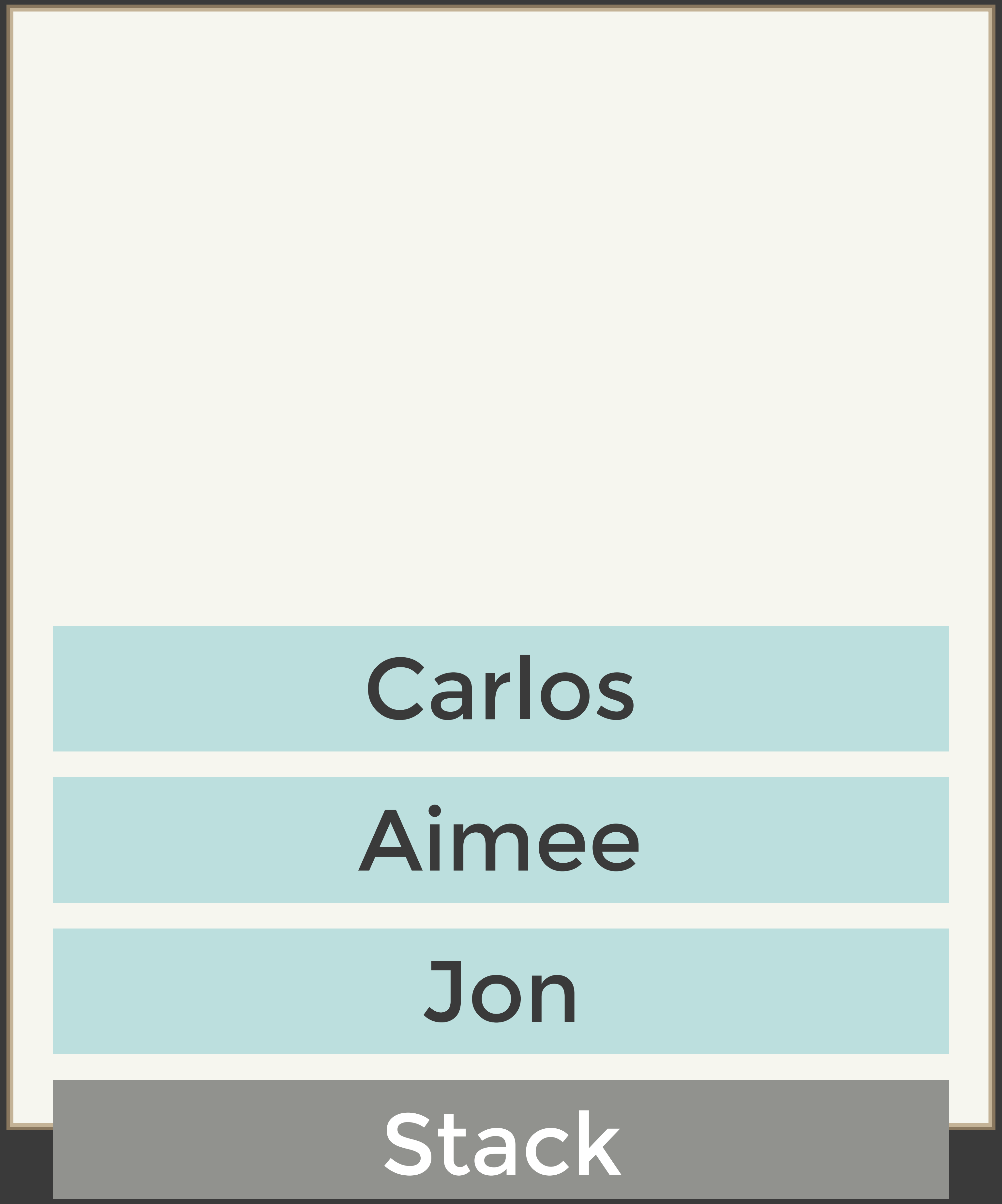
```
> add "Jon" to Stack
> add "Aimee" to Stack
> add "Gregg" to Stack
```

## A Generic Stack Data Structure



```
> add "Jon" to Stack
> add "Aimee" to Stack
> add "Gregg" to Stack
> remove thing from Stack
```

# A Generic Stack Data Structure



```
> add "Jon" to Stack
> add "Aimee" to Stack
> add "Gregg" to Stack
> remove thing from Stack
> add "Carlos" to Stack
```

# The Code for a Simple Generic Stack

```
struct Stack<T> {

    var stack: [T] = []
    var count = 0

    mutating func addToStack(item: T) {
        stack.append(item)
        count = count + 1
    }

    mutating func removeFromStack() -> T? {
        guard count > 0 else { return nil }
        count = count - 1
        return stack.removeLast()
    }
}
```

*array that holds a generic type T*

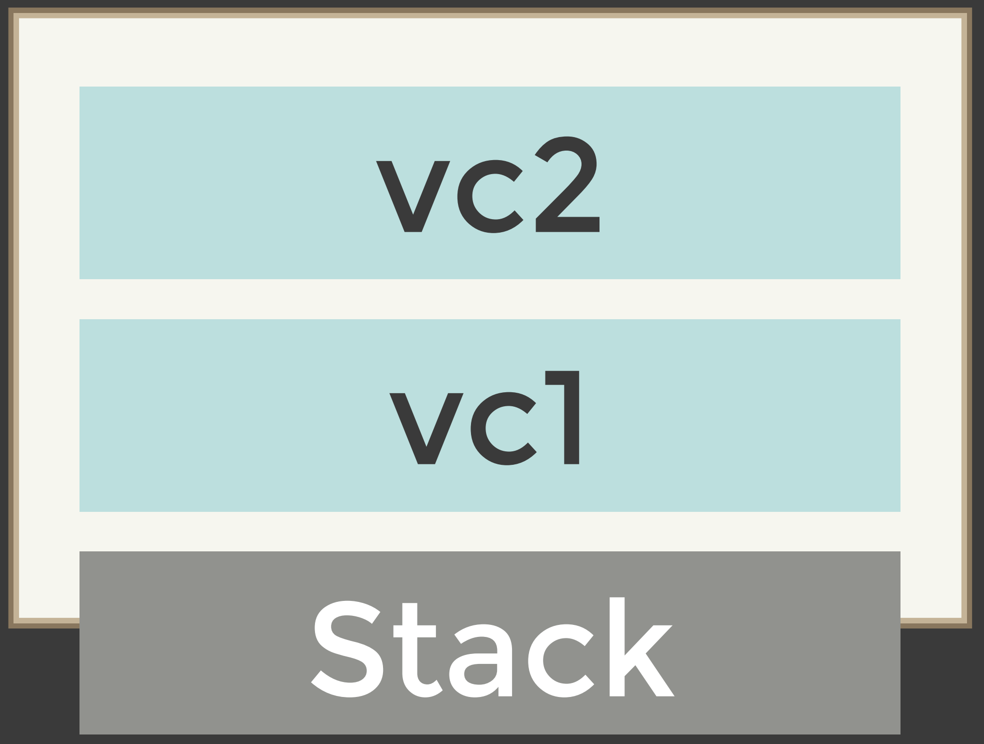
*push an item of type T onto the stack*

*remove an item of type T from the stack and return it*

## Using the Stack

```
struct Stack<T> { }

var vc1 = UIViewController()
var vc2 = UIViewController()

var s = Stack<UIViewController>()
s.addToStack(item: vc1)
s.addToStack(item: vc2)
s.count // 2
```

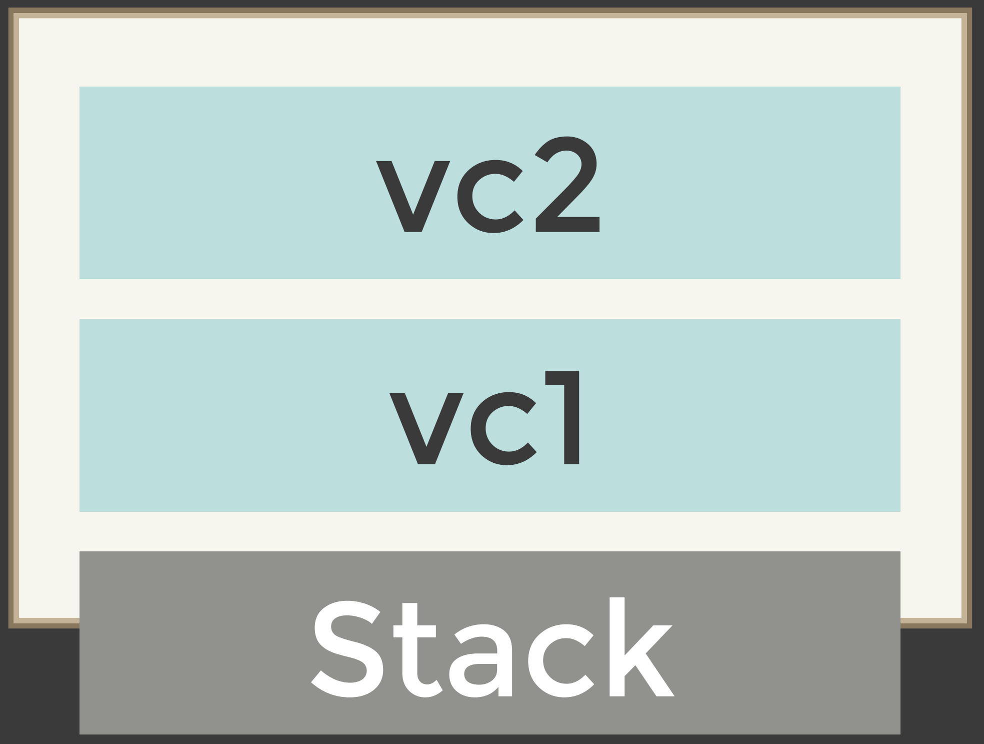
*let this Stack know it should only allow adding UIViewController objects*



## Using the Stack

```
struct Stack<T> { }

var vc1 = UIViewController()
var vc2 = UIViewController()

var s = Stack<UIViewController>()
s.addToStack(item: vc1)
s.addToStack(item: vc2)
s.count // 2

var i = s.removeFromStack()
s.count // 1
```

*i contains a UIViewController object*



## Constraining Generic Types

```
struct Stack<T> { }

var vc1 = UIViewController()
var vc2 = UIViewController()

var s = Stack<UIViewController>()
```

Right now, T could be anything, but what if we want to restrict it to just certain types?

```
struct Stack<T: Stackable> { }
```

*type constraint*

Type constraints help you restrict generic types just a little bit

# Creating and Adopting a Protocol

```
class ViewController: UIViewController, Stackable {
    internal var amIOnTheStack: Bool {
        get {
            return self.amIOnTheStack
        }
        set(newValue) {
            self.amIOnTheStack = newValue
        }
    }
}

protocol Stackable: class {
    var amIOnTheStack: Bool { get set }
}

struct Stack<T: Stackable> { }
```

*adopt the protocol*

*implement the required stuff*

*define the protocol*

## Constraints in Action

```
class ViewController: UIViewController, Stackable { }
protocol Stackable: class { }
struct Stack<T: Stackable> { }

var s1 = Stack<ViewController>()
```

*all good, ViewController is Stackable*

```
var s2 = Stack<UIButton>()
```

```
error: type 'UIButton' does not conform to protocol 'Stackable'
```

## Handling Network Responses

# A Standard Network Request

```swift
func makeRequest(urlString: String) {
    guard let url = URL(string: urlString) else { return }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            print(error!.localizedDescription)
            return
        }

        guard let data = data else { return }

        let responseString = String(data: data, encoding: String.Encoding.ascii)
        print(responseString!)
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200")
```

# A Standard Network Request

```
func makeRequest(urlString: String) {
    guard let url = URL(string: urlString) else { return }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            print(error!.localizedDescription)
            return
        }

        guard let data = data else { return }

        let responseString = String(data: data, encoding: String.Encoding.ascii)
        print(responseString!)
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200")
```

*Make sure the URL is valid*

## A Standard Network Request

```
func makeRequest(urlString: String) {
    guard let url = URL(string: urlString) else { return }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            print(error!.localizedDescription)
            return
        }

        guard let data = data else { return }

        let responseString = String(data: data, encoding: String.Encoding.ascii)
        print(responseString!)
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200")
```

*Use that URL to make a request*

*Use that request in a data task*

## A Standard Network Request

```
func makeRequest(urlString: String) {
    guard let url = URL(string: urlString) else { return }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            print(error!.localizedDescription)
            return
        }

        guard let data = data else { return }

        let responseString = String(data: data, encoding: String.Encoding.ascii)
        print(responseString!)
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200")
```

*If an error is returned, log it*

## A Standard Network Request

```
func makeRequest(urlString: String) {
    guard let url = URL(string: urlString) else { return }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            print(error!.localizedDescription)
            return
        }

        guard let data = data else { return }

        let responseString = String(data: data, encoding: String.Encoding.ascii)
        print(responseString!)
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200")
```

*Extract the response data (which is an optional)*

# A Standard Network Request

```swift
func makeRequest(urlString: String) {
    guard let url = URL(string: urlString) else { return }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            print(error!.localizedDescription)
            return
        }

        guard let data = data else { return }

        let responseString = String(data: data, encoding: String.Encoding.ascii)
        print(responseString!)
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200")
```

*Turn the response data into a String and log it*

# A Standard Network Request

```swift
func makeRequest(urlString: String) {
    guard let url = URL(string: urlString) else { return }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            print(error!.localizedDescription)
            return
        }

        guard let data = data else { return }

        let responseString = String(data: data, encoding: String.Encoding.ascii)
        print(responseString!)
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200")
```

*Call the function and pass in a URL string*

## What's wrong with that?

- *Not much control over response data*
- *Logic that processes the response is in the request code*

The solution? Completion handlers!

## Network Request with Completion Handler

*handler takes in a String*

```
func makeRequest(urlString: String, completionHandler:@escaping (String) -> Void) {
    guard let url = URL(string: urlString) else {
        return completionHandler("Bad URL")
    }

    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        ...
```

*call the handler w/String*

# Network Request with Completion Handler

*handler takes in a String*

```
func makeRequest(urlString: String, completionHandler:@escaping (String) -> Void) {
    ...
    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            completionHandler(error!.localizedDescription)
            return
        }
        ...
```

*call the handler w/String*

## Network Request with Completion Handler

*handler takes in a String*

```
func makeRequest(urlString: String, completionHandler:@escaping (String) -> Void) {
    ...
    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in

        ...
        completionHandler(String(data: data, encoding: String.Encoding.ascii)!)
        ...
```

*call the handler w/String*

## Network Request with Completion Handler

*handler takes in a String*

```
func makeRequest(urlString: String, completionHandler:@escaping (String) -> Void) {
    ...
    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in

        ...

makeRequest(urlString: "http://httpstat.us/200") { (response) in
    print(response)
}
```

*String entered into the handler is received here*

## Now what?

> *Satisfies the problem of having response logic in the request code, but still forced to use Strings*

*Let's try making this generic!*

# Network Request with Result Enum

```swift
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                 completionHandler:@escaping (Result<Data>) -> Void) { }


makeRequest(urlString: "http://httpstat.us/200") { (result) in
    switch result {
    case .Success(let data):
        let responseString = String(data: data, encoding: String.Encoding.ascii)!
        print("Got Data \(responseString)")
    case .Error(let error):
        print("Error \(error)")
    }
}
```

# Result Enum with Generic Parameter

```
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                completionHandler:@escaping (Result<Data>) -> Void) { }


makeRequest(urlString: "http://httpstat.us/200") { (result) in
    switch result {
    case .Success(let data):
        let responseString = String(data: data, encoding: String.Encoding.ascii)!
        print("Got Data \(responseString)")
    case .Error(let error):
        print("Error \(error)")
    }
}
```

Enum with a generic type parameter

Force the same type for Success cases

OK to still return error logs as a String

# Result Enum with Generic Parameter

```
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                 completionHandler:@escaping (Result<Data>) -> Void) { }

makeRequest(urlString: "http://httpstat.us/200") { (result) in
    switch result {
    case .Success(let data):
        let responseString = String(data: data, encoding: String.Encoding.ascii)!
        print("Got Data \(responseString)")
    case .Error(let error):
        print("Error \(error)")
    }
}
```

*this forces the **T** type to be **Data** in this function*

# Processing the Result Response

```
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                 completionHandler:@escaping (Result<Data>) -> Void) { }


makeRequest(urlString: "http://httpstat.us/200") { (result) in
    switch result {
    case .Success(let data):
        let responseString = String(data: data, encoding: String.Encoding.ascii)!
        print("Got Data \(responseString)")
    case .Error(let error):
        print("Error \(error)")
    }
}
```

*switch on the result enum and do something for each case*

# Handling the Error cases with Result

```
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                 completionHandler:@escaping (Result<Data>) -> Void) { }
    guard let url = URL(string: urlString) else {
        return completionHandler(.Error("Invalid URL"))
    }
    ...
}
```

*Call the handler with the Error case and a String value for that error message*

```
makeRequest(urlString: "http://httpstat.us/200") { (result) in ... }
```

# Handling the Error cases with Result

```
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                 completionHandler:@escaping (Result<Data>) -> Void) { }
    ...
    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        if error != nil {
            return completionHandler(.Error(error!.localizedDescription))
        }
        ...
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200") { (result) in ... }
```

# Handling the Error cases with Result

```
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                 completionHandler:@escaping (Result<Data>) -> Void) { }
    ...
    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        ...
        guard let data = data else {
            return completionHandler(.Error("No data in response"))
        }
        ...
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200") { (result) in ... }
```

# Handling the Success case with Result

```swift
enum Result<T> {
    case Success(T)
    case Error(String)
}
func makeRequest(urlString: String,
                 completionHandler:@escaping (Result<Data>) -> Void) { }
    ...
    let request = URLRequest(url: url)
    URLSession.shared.dataTask(with: request) { (data, response, error) in
        ...
        completionHandler(.Success(data))
    }.resume()
}
makeRequest(urlString: "http://httpstat.us/200") { (result) in ... }
```

## Generic Data Source Example

> *Xcode time! (if we have time)*

## Next steps to learn more

- *The Swift Programming Language official book*
- *Chris Eidhof + co's books and talk.objc.io*
- *Austin Zheng's* ***Generics in Swift*** *blog posts*
- *Find something in one of your apps and start small*

Code School

## Thanks!

*Jon Friskics*

*@jonfriskics*

*jon@codeschool.com*