Introducing Protocols

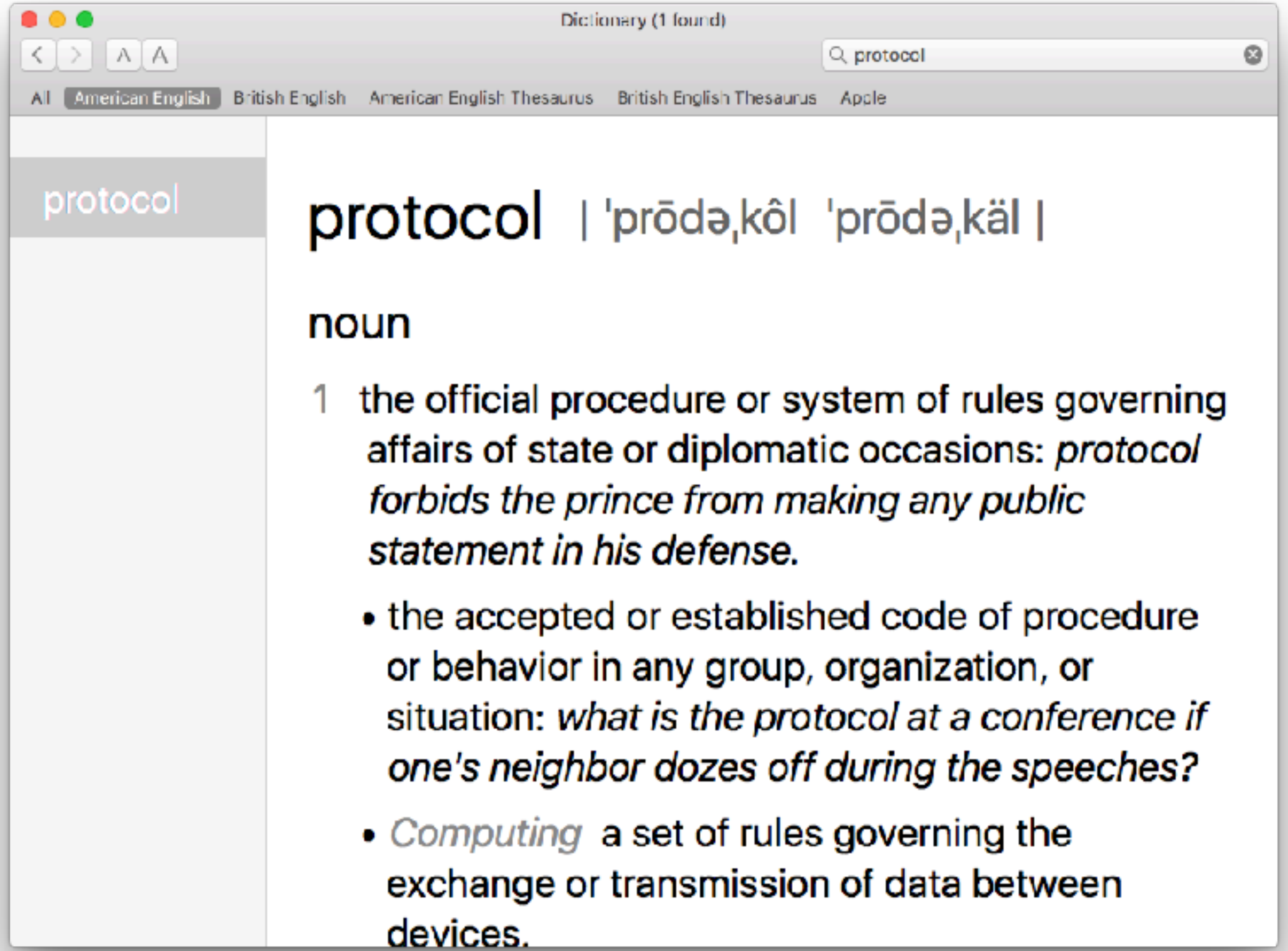
Simon Allardice
STAFF AUTHOR, PLURALSIGHT
@allardice www.pluralsight.com

Protocol

A set of rules or code of behavior

Protocol

A set of rules or code of behavior



protocol | ˈprōdəˌkôl ˈprōdəˌkäl |

noun

1 the official procedure or system of rules governing affairs of state or diplomatic occasions: *protocol forbids the prince from making any public statement in his defense.*

- the accepted or established code of procedure or behavior in any group, organization, or situation: *what is the protocol at a conference if one's neighbor dozes off during the speeches?*
- *Computing* a set of rules governing the exchange or transmission of data between devices.

```
protocol MyProtocol {

    // what methods?


    // what properties?


}
```

- Each protocol has a name
- A list of methods
  (names, parameters, and return types)
- A list of properties
  (name, type, get/set)

Protocol Usage

Protocol Usage

General Purpose

Creating Collections, Comparing Instances, Converting, Sorting, Debugging

Protocol Usage

General Purpose

Creating Collections, Comparing Instances, Converting, Sorting, Debugging

App-specific

Loading Data, Saving Data, Spellchecking, Resizing UIs

Inheritance and/or Protocol Adoption

Swift classes allow single class inheritance
Swift classes, structs and enums allow multiple protocols

```
class MyNewClass
```

Inheritance and/or Protocol Adoption

Swift classes allow single class inheritance
Swift classes, structs and enums allow multiple protocols

```
class MyNewClass:
```

Inheritance and/or Protocol Adoption

Swift classes allow single class inheritance
Swift classes, structs and enums allow multiple protocols

```
class MyNewClass: SomeSuperClass
```

Inheritance and/or Protocol Adoption

Swift classes allow single class inheritance
Swift classes, structs and enums allow multiple protocols

```
class MyNewClass: SomeSuperClass, SomeProtocol
```

Inheritance and/or Protocol Adoption

Swift classes allow single class inheritance
Swift classes, structs and enums allow multiple protocols

```
class MyNewClass: SomeSuperClass, SomeProtocol, OtherProtocol {
    ...
}
```

Inheritance and/or Protocol Adoption

Swift classes allow single class inheritance
Swift classes, structs and enums allow multiple protocols

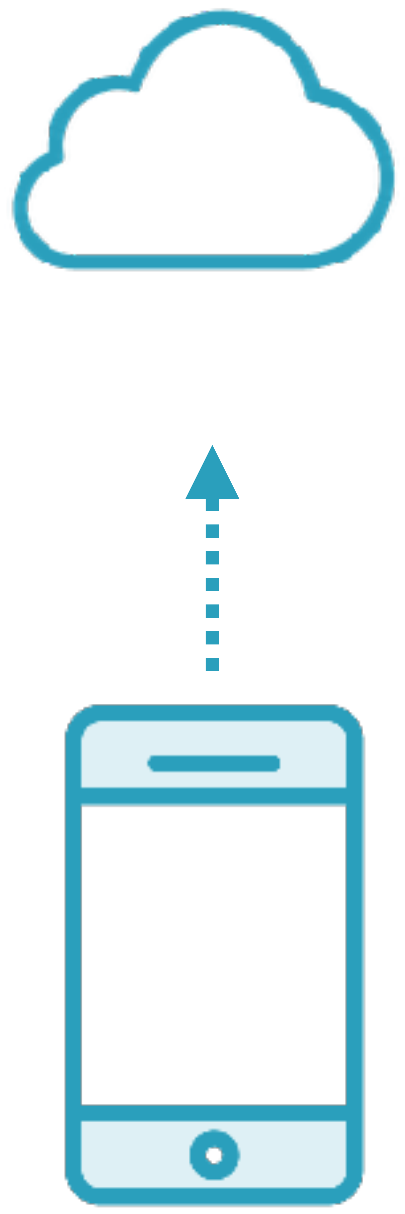
Error Handling in Swift

Error Handling in Swift

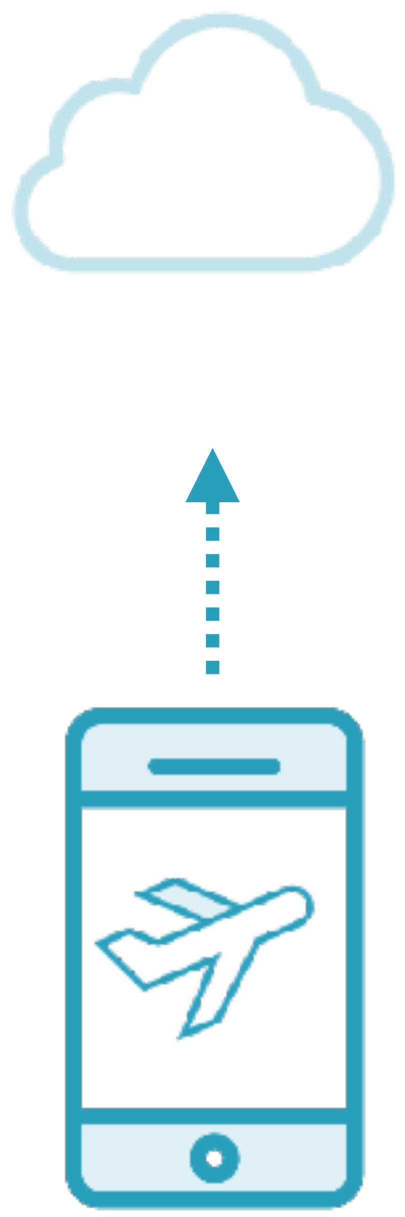
Dealing with recoverable errors

Recoverable Errors

Recoverable Errors



Recoverable Errors



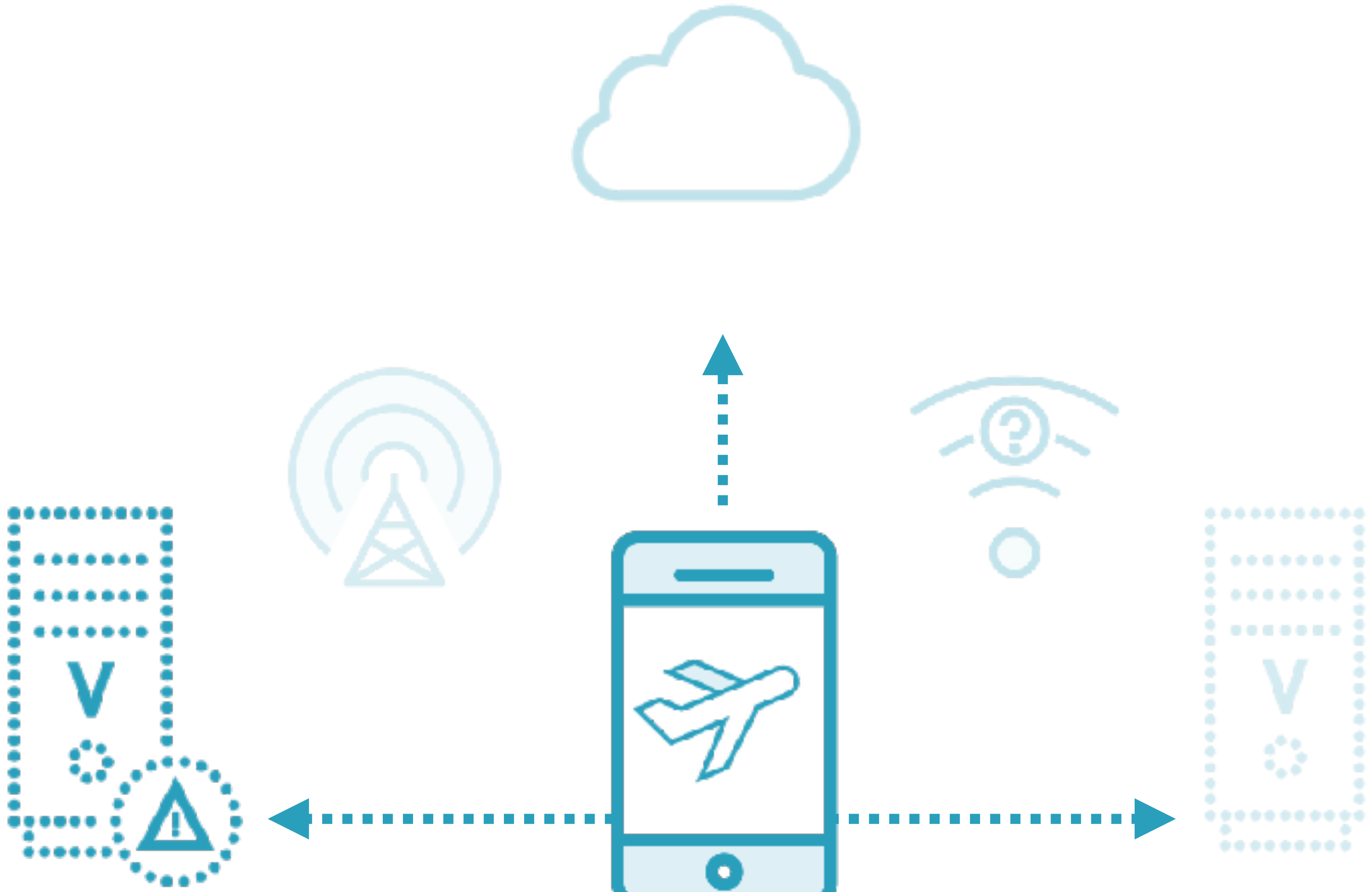
Recoverable Errors

Recoverable Errors

Recoverable Errors

Recoverable Errors

Recoverable Errors



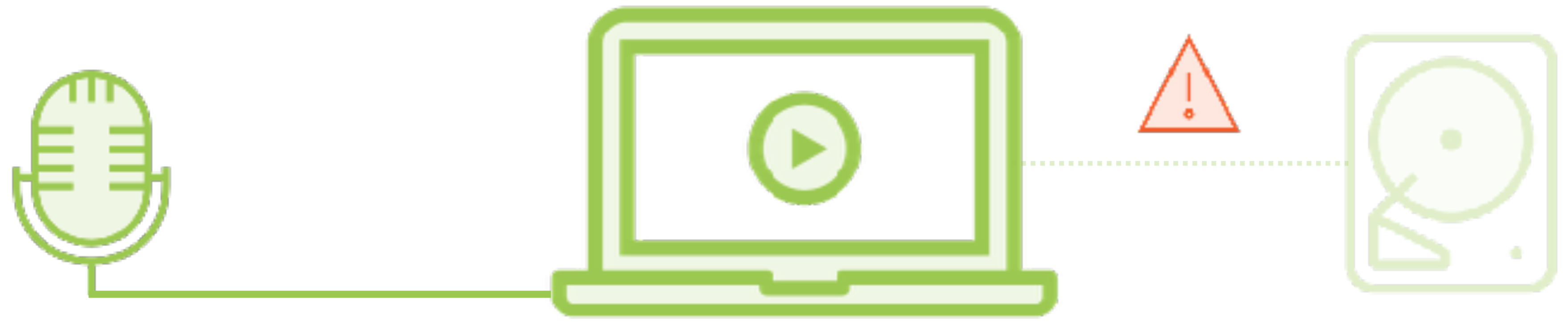
Recoverable Errors

Recoverable Errors

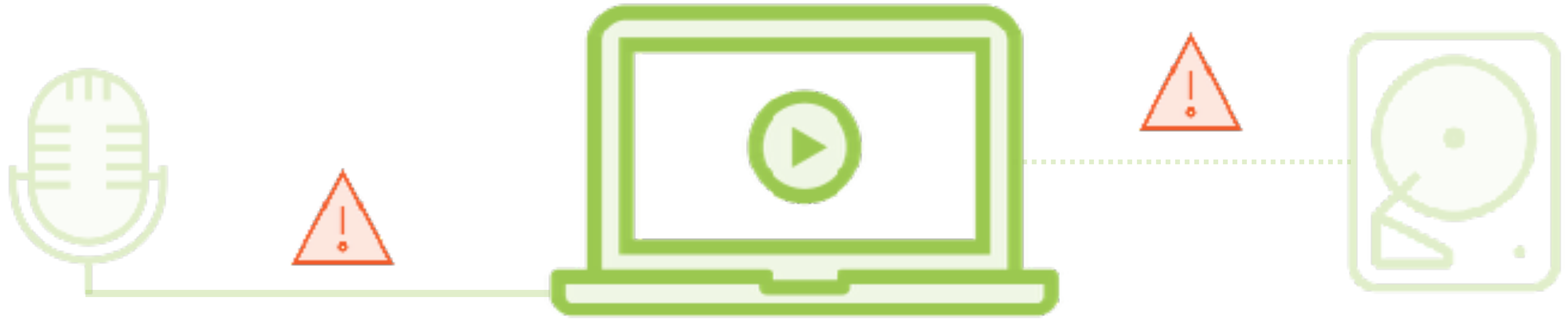
Recoverable Errors

Recoverable Errors

Recoverable Errors



Recoverable Errors



Three Parts to Error Handling

Three Parts to Error Handling

1. Define it

Three Parts to Error Handling

1. Define it

What is it?

Three Parts to Error Handling

1. Define it

What is it?

Connection error?

Three Parts to Error Handling

1. Define it

What is it?

Connection error?

Save error?

Three Parts to Error Handling

1. Define it

What is it?

Connection error?

Save error?

Calculation error?

Three Parts to Error Handling

1. Define it

What is it?

Connection error?

Save error?

Calculation error?

2. Throw it

Three Parts to Error Handling

1. Define it

What is it?

Connection error?

Save error?

Calculation error?

2. Throw it

Where and **when** can it happen?

Three Parts to Error Handling

1. Define it

What is it?

Connection error?

Save error?

Calculation error?

2. Throw it

Where and when can it happen?

3. Handle it

Three Parts to Error Handling

1. Define it

What is it?

Connection error?

Save error?

Calculation error?

2. Throw it

Where and **when** can it happen?

3. Handle it

What are you going to do about it?

Creating Errors

Some languages have predefined error types.

```
Error myError =  new Error();
```

Creating Errors

Some languages have predefined error types.

```
Error myError =  new Error();
myError.description = "Connection failure";
```

Creating Errors

Some languages have predefined error types.

```
Error myError =  new Error();
myError.description = "Connection failure";
myError.priority = 1;
```

Creating Errors

Some languages have predefined error types.

```
Error myError =  new Error();
myError.description = "Connection failure";
myError.priority = 1;
```

Creating Errors

Some languages have predefined error types. Swift does not.

Swift Errors

Can be created from any type

```swift
struct SomeKindOfError {
    // whatever you need...
 }
```

Swift Errors

Can be created from any type

```
class SomeKindOfError {
    // whatever you need...
}
```

Swift Errors

Can be created from any type

```swift
enum  SomeKindOfError {
    // whatever you need...
}
```

Swift Errors

Can be created from any type

Using Guard and Defer

Guard Statement

```
guard itemsRequested < itemsInStock else  {
    print("Cannot fulfil request.")
    return
}
```

Guard Statement

Guard Statement

```
guard some-condition-i-need-to-be-true else  {
   what-we-do-if-it-isn't
}
```

Guard Statement

```
guard customerBalance > requiredAmount else  {
    what-we-do-if-it-isn't
}
```

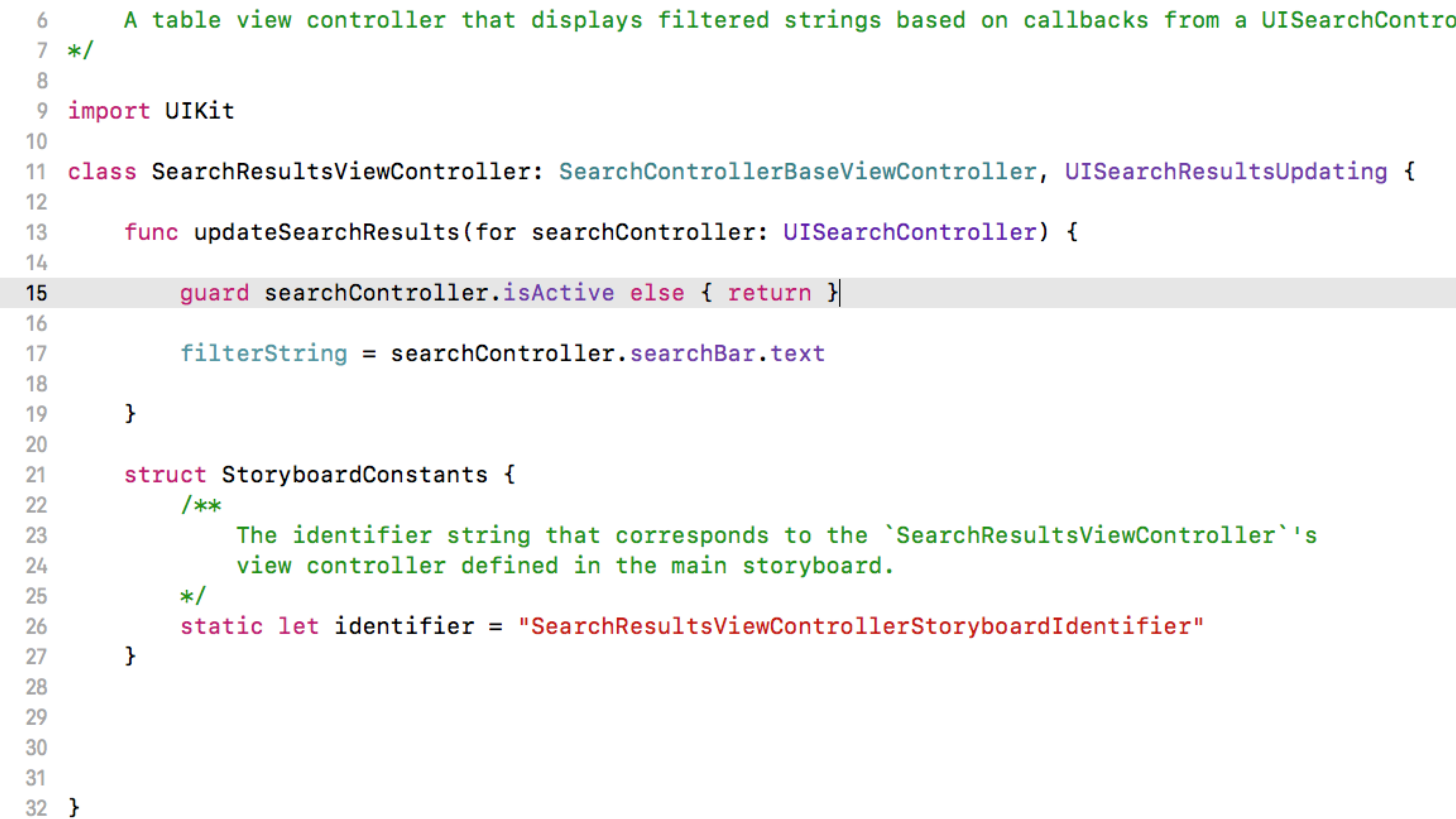
Guard Statement

```
guard  let unwrappedVal = optionalVal  else  {
    what-we-do-if-it-isn't
}
```

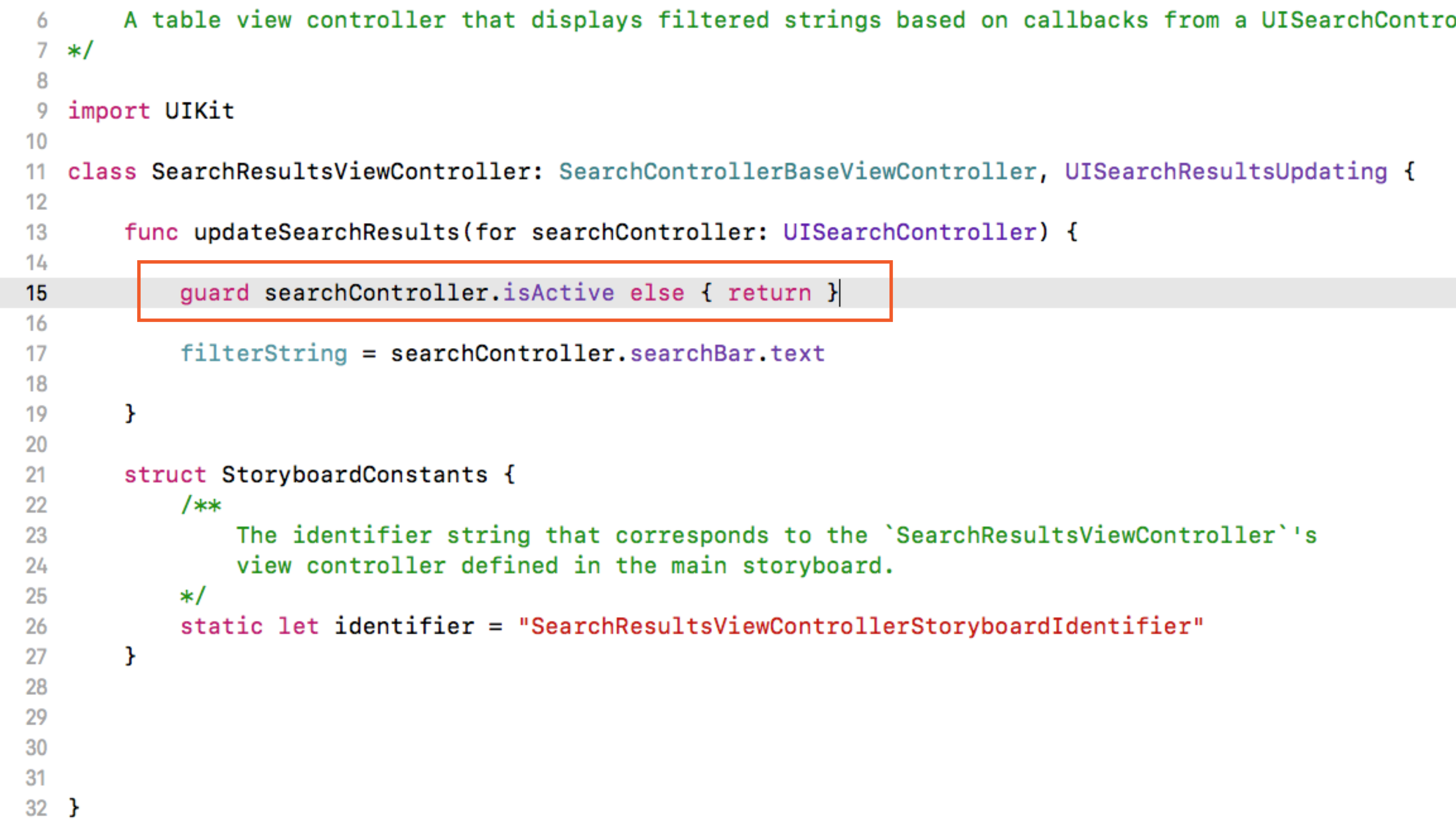
Guard Statement

```
guard  let unwrappedVal = optionalVal  else  {
     return / throw / break / continue
}
```

Guard Statement



```
    A table view controller that displays filtered strings based on callbacks from a UISearchContro
*/

import UIKit

class SearchResultsViewController: SearchControllerBaseViewController, UISearchResultsUpdating {

    func updateSearchResults(for searchController: UISearchController) {

        guard searchController.isActive else { return }

        filterString = searchController.searchBar.text

    }

    struct StoryboardConstants {
        /**
            The identifier string that corresponds to the `SearchResultsViewController`'s
            view controller defined in the main storyboard.
        */
        static let identifier = "SearchResultsViewControllerStoryboardIdentifier"
    }



}
```



```swift
    A table view controller that displays filtered strings based on callbacks from a UISearchContro
*/

import UIKit

class SearchResultsViewController: SearchControllerBaseViewController, UISearchResultsUpdating {

    func updateSearchResults(for searchController: UISearchController) {

        guard searchController.isActive else { return }

        filterString = searchController.searchBar.text

    }

    struct StoryboardConstants {
        /**
            The identifier string that corresponds to the `SearchResultsViewController`'s
            view controller defined in the main storyboard.
        */
        static let identifier = "SearchResultsViewControllerStoryboardIdentifier"
    }




}
```

Using Optional Binding with Guard

```swift
if let unwrappedName = optionalName {
   print("We have the value \(unwrappedName)")
} else {
   print("It was nil.")
}
```

Using Optional Binding with Guard

```swift
if let unwrappedName = optionalName {
   print("We have the value \(unwrappedName)")
} else {
   print("It was nil.")
}

print(unwrappedName) // error - no longer available
```

Using Optional Binding with Guard

Using Optional Binding with Guard

```
guard let unwrappedName = optionalName else {
    return
}
```

Using Optional Binding with Guard

```swift
guard let unwrappedName = optionalName else {
      return
}

print(unwrappedName) // yes - still available
```

Using Optional Binding with Guard

Using Optional Binding with Guard

```
guard let unwrappedTrack = optionalTrack else { return }
guard let unwrappedArtist = optionalArtist else { return }
guard let unwrappedAlbum = optionalAlbum else { return }
```

Using Optional Binding with Guard

```
guard let unwrappedTrack = optionalTrack else { return }
guard let unwrappedArtist = optionalArtist else { return }
guard let unwrappedAlbum = optionalAlbum else { return }

// if we get to this line, they're all unwrapped
print("\(unwrappedAlbum) \(unwrappedArtist) \(unwrappedAlbum)")
```

Using Optional Binding with Guard

```swift
guard let unwrappedTrack = optionalTrack else { return }
guard let unwrappedArtist = optionalArtist else { return }
guard let unwrappedAlbum = optionalAlbum else { return }
```

Using Optional Binding with Guard

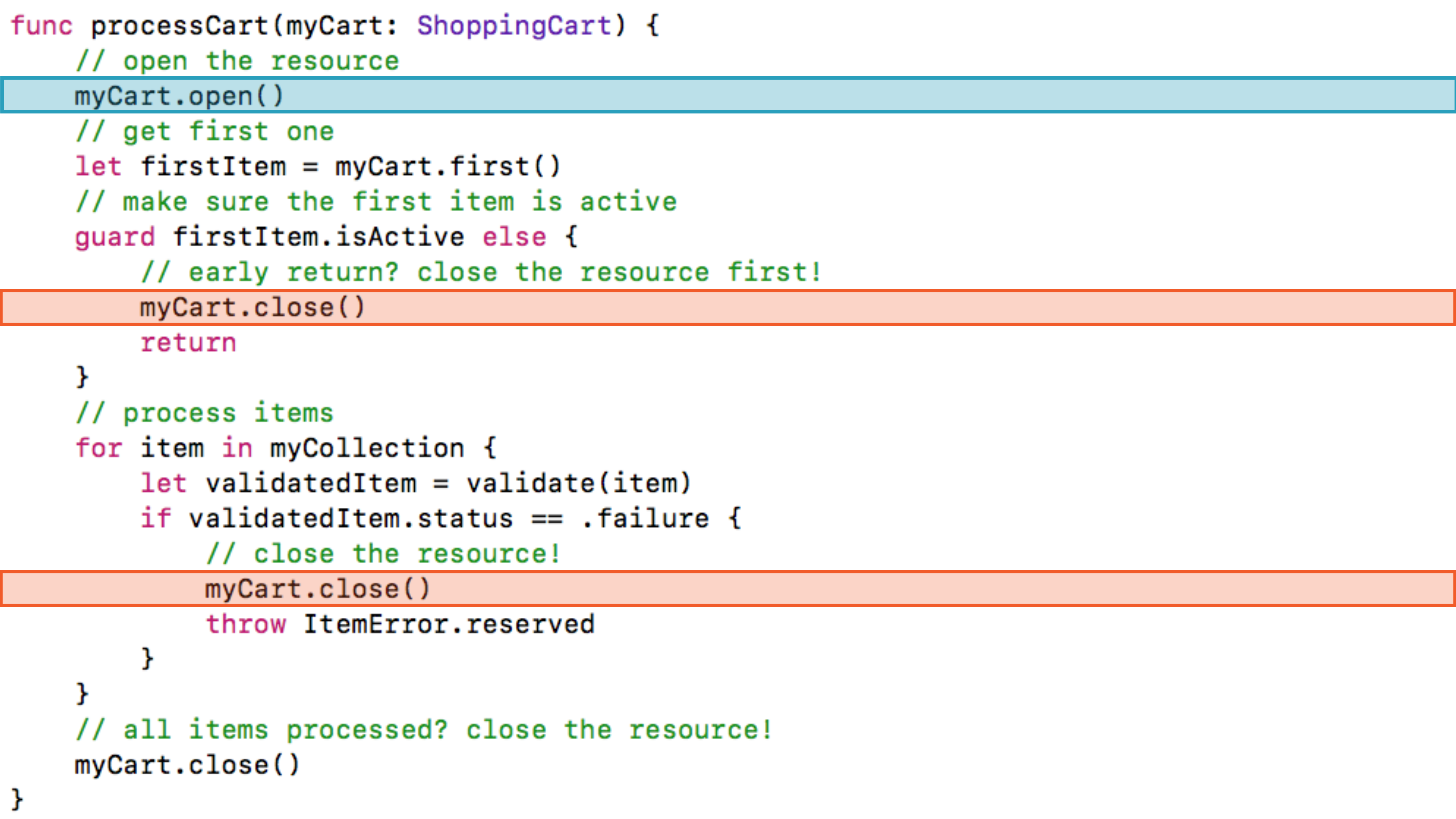
```
guard let unwrappedTrack = optionalTrack ,
      let unwrappedArtist = optionalArtist ,
      let unwrappedAlbum = optionalAlbum else { return }
```

Using Optional Binding with Guard

```swift
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        myCart.close()
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            // close the resource!
            myCart.close()
            throw ItemError.reserved
        }
    }
    // all items processed? close the resource!
    myCart.close()
}
```

```
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        myCart.close()
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            // close the resource!
            myCart.close()
            throw ItemError.reserved
        }
    }
    // all items processed? close the resource!
    myCart.close()
}
```

```swift
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        myCart.close()
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            // close the resource!
            myCart.close()
            throw ItemError.reserved
        }
    }
    // all items processed? close the resource!
    myCart.close()
}
```



```swift
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        myCart.close()
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            // close the resource!
            myCart.close()
            throw ItemError.reserved
        }
    }
    // all items processed? close the resource!
    myCart.close()
}
```

```swift
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        myCart.close()
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            // close the resource!
            myCart.close()
            throw ItemError.reserved
        }
    }
    // all items processed? close the resource!
    myCart.close()
}
```

Defer Statement

```
func someFunction() {





    // code...
}
```

Defer Statement

```
func someFunction() {

     defer {
        // your cleanup code
     }

     // code...
}
```

Defer Statement

```swift
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    defer {
        myCart.close()
    }
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            throw ItemError.reserved
        }
    }
}
```

```
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    defer {
        myCart.close()
    }
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            throw ItemError.reserved
        }
    }
}
```

```swift
func processCart(myCart: ShoppingCart) {
    // open the resource
    myCart.open()
    defer {
        myCart.close()
    }
    // get first one
    let firstItem = myCart.first()
    // make sure the first item is active
    guard firstItem.isActive else {
        // early return? close the resource first!
        return
    }
    // process items
    for item in myCollection {
        let validatedItem = validate(item)
        if validatedItem.status == .failure {
            throw ItemError.reserved
        }
    }
}
```