GENERAL ASSEMBLY

ERROR HANDLING

Zeke Abuhoff

Lead iOS Instructor, General Assembly

LEARNING OBJECTIVES

+ Define an error
+ Use ‘do’, ‘try’ and ‘catch’ keywords
+ Create an enum that conforms to the Error protocol
+ Throw an exception

LAST TIME ON iOS NETWORKING...

COMPLETION HANDLERS

Step Two - Parse data

Swift can't guarantee the validity of data from a response, so its APIs for handling such data are couched in optionals and possible thrown errors.

```
do {
     let jsonObject = try JSONSerialization.jsonObject(with: responseData, options:
JSONSerialization.ReadingOptions.allowFragments)
     let jsonString = String(jsonObject)
     print("RESPONSE JSON: \(jsonString)")
} catch { return }
```

ERRORS

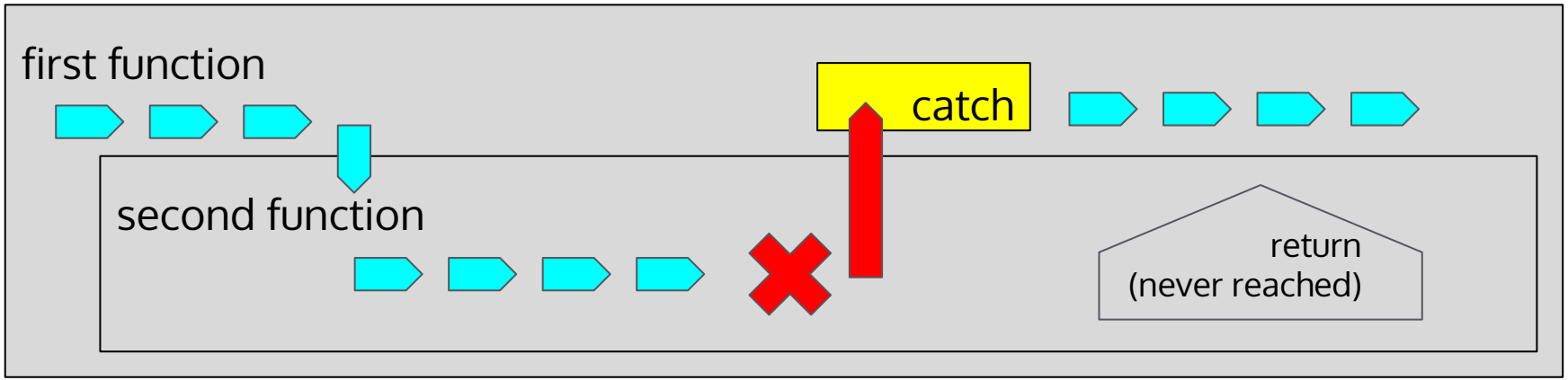
Before we can understand how to use `do`, `try` and `catch`, we have to understand what an error is.

In Swift, an **error** is an object representing undesired behavior.

Error handling is the practice of acknowledging errors and working around them. Without error handling, our apps would be less stable.

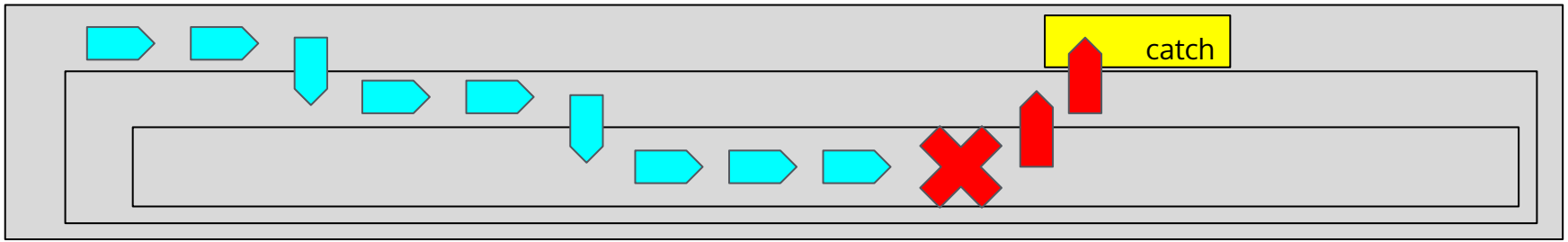
CATCHING ERRORS

An error's defining behavior is its ability to be **thrown**. When a function throws an error, that function ends. A thrown error will propagate to the function that called the throwing function. Eventually, the error must be caught.



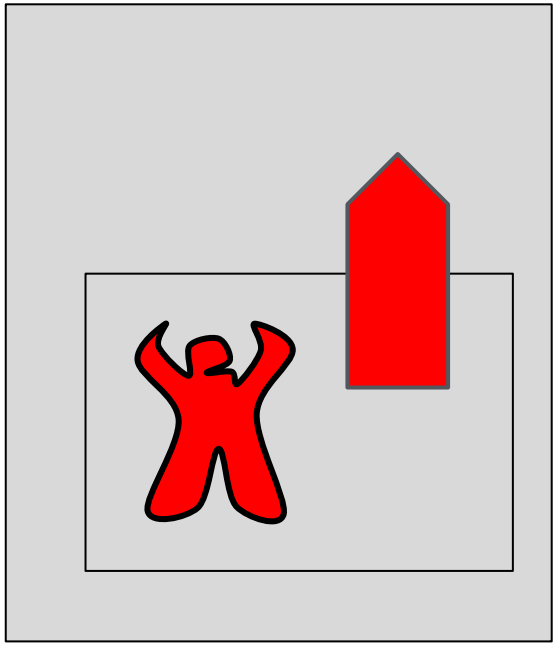
CATCHING ERRORS

I write that the error must “eventually” be caught because a function doesn’t have to catch a thrown error if that function itself is capable of throwing errors.



This way, errors can propagate up a chain of error-throwing functions until the lowest non-throwing function catches the error. And that function *must* catch the error. This mandate is enforced at compile-time.

CATCHING ERRORS



Think of it this way: a zookeeper enters the gorilla enclosure, hoping to take a charming photo of the gorillas as they sleep.

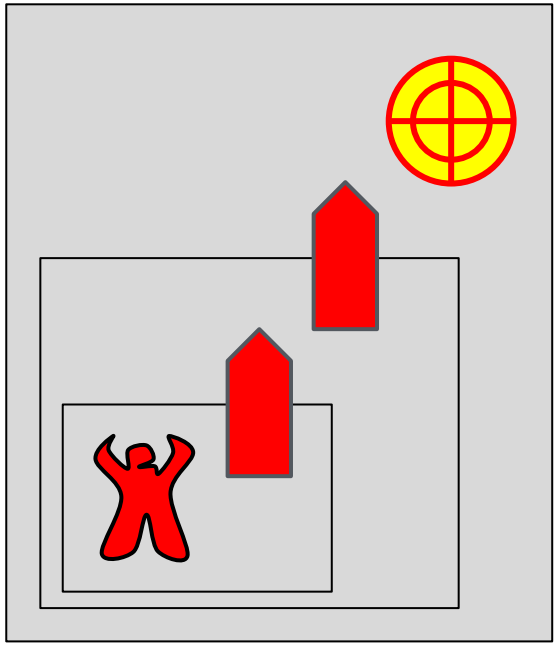
Moments later, the zookeeper runs out the enclosure, screaming. Not only has he not gotten the photo, but also he is being chased by an angry gorilla.

CATCHING ERRORS

Perhaps there is another zookeeper there, ready to stop the rampaging gorilla.

If there isn't, the gorilla will escape to rampage through the whole primates building. If he is not stopped there, he will rampage through the whole zoo.

At some point, the gorilla must be stopped. Society cannot abide a freely raging gorilla, so the error will be caught, using lethal force, if necessary.



DO, TRY, CATCH

In Swift, when we want to use a function that may throw an error, we tag it with the `try` keyword and wrap it in a `do` block. This is the quarantine area where we will contain the gorilla.

```
do {
    let usefulData = try myRiskyFunction()
    return usefulData
} catch {
    print("Oh, no! An error!")
    return nil
}
```

The `do` block must be followed with a `catch` block, whose code will run if an error is thrown. This is where we write what will happen when the gorilla is caught.

DO, TRY, CATCH

Practice:

Complete part 1 of the test functions in this lesson's accompanying project. These tests don't start out failing, but follow the instructions anyway.

ERROR ENUMS

Errors come in many flavors. Swift uses objects called **enums** to list these flavors.

You can write your own enums and make them error enums by adding ": Error" to their name where they're defined.

```
enum Mistakes: Error {
    case Small
    case NotThatBadReally
    case Huge
}
```

ERROR ENUMS

Do, try, catch can account for these error cases by having multiple catch blocks.

This way, different code is executed, depending on what sort of error was thrown.

```
do {
    try toMakeThingsRight()
} catch .Small {
    return “Sorry!”
} catch .NotThatBadReally {
    return “Eh. You’ll get over it.”
} catch .Huge {
    return “I’ve made a huge mistake.”
}
```

THROWING ERRORS

You, too, can write functions that throw errors.

All you have to do is write `throws` before your function's return type, then write `throw` and the error in the body of the function where appropriate.

```
func thisMightWork() throws -> Bool {
    if things.workOut() {
        return true
    } else if backupPlan.worksOut() {
        return false
    } else {
        throw Mistake.Huge
    }
}
```

THROWING ERRORS

Practice:

Complete part 2 of the test functions in this lesson's accompanying project.