

2





• What if the input string is not a well-formed integer? An integer with a possibly invalid or missing value

Expecting string

representing an integer

• How to represent a invalid or missing value? NULL null nil 0 throw an exception

Representing a Missing Value

• Consider the following code

Int

let inputString = input from user let num = Int(inputString) • What should the type of num be?

DEPAUL UNIVERSITY

3

9

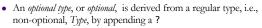
Special Value nil

- Swift adopts a different approach for Java, C++, even Objective-C
- Introducing a special value nil
 - Represents a missing value, i.e., the absence of a valid value or
 - · Not equal to any valid value or object. It is not 0, -1, etc.
 - Can only be used in an optional type
 - Different from regular types.
- · Does not belong to any non-optional type
- Values of non-optional types, i.e., regular types, can never

DEPAUL UNIVERSITY

A new concept.

Optional Types



· Any non-optional type, basic type or class type, can become an

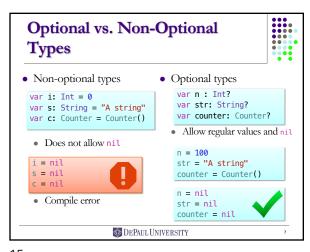
Counter?

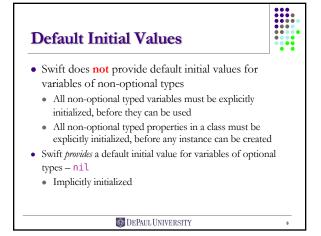
- · An optional type is a wrapper that can wrap any type
- An optional type Type? means, it contains either
 - A valid value or object of Type, or
 - No value at all, which is represented as nil
- Synonyms: nullable types, non-nullable types

DEPAUL UNIVERSITY

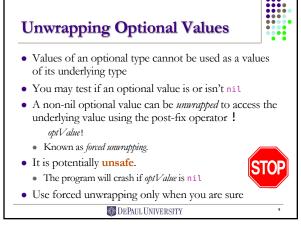
11

10





15



Unwrapping Optional Values

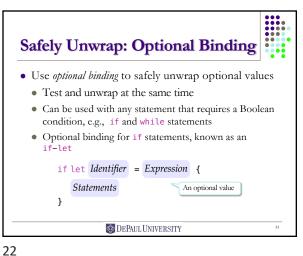
• Test optional values before forced unwrapping

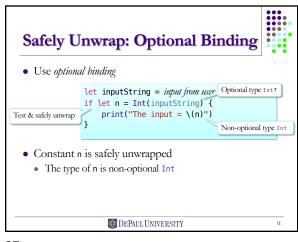
let inputString = input from user
let num = Int(inputString) Optional type Int?
if num != nil {
 print("The input = \(num!)")
}

Forced unwrapping

DEPAUL UNIVERSITY

17





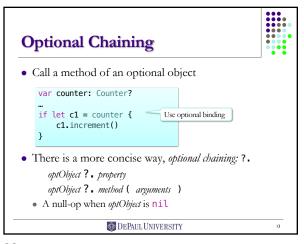
27

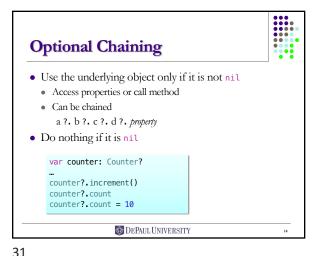
16

21

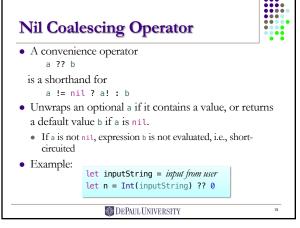
© Xiaoping Jia, 2015-2021

2





30



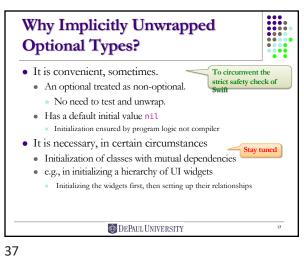
Implicitly Unwrapped
Optional Types

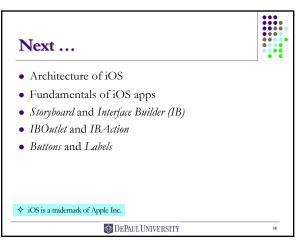
Consider a non-optional type: Type
Guaranteed by the compiler that its value can never be nil anywhere

An implicitly unwrapped optional type: Type!
One can safely assert that its value is not nil within a given context

Ensured by the design or the logic of the program
Weaker guarantee (not by the compiler), but not your responsibility (guaranteed by other parts of the program).
Usually in a limited context, not everywhere.
You can safely use it within the context.

32





38

33

37