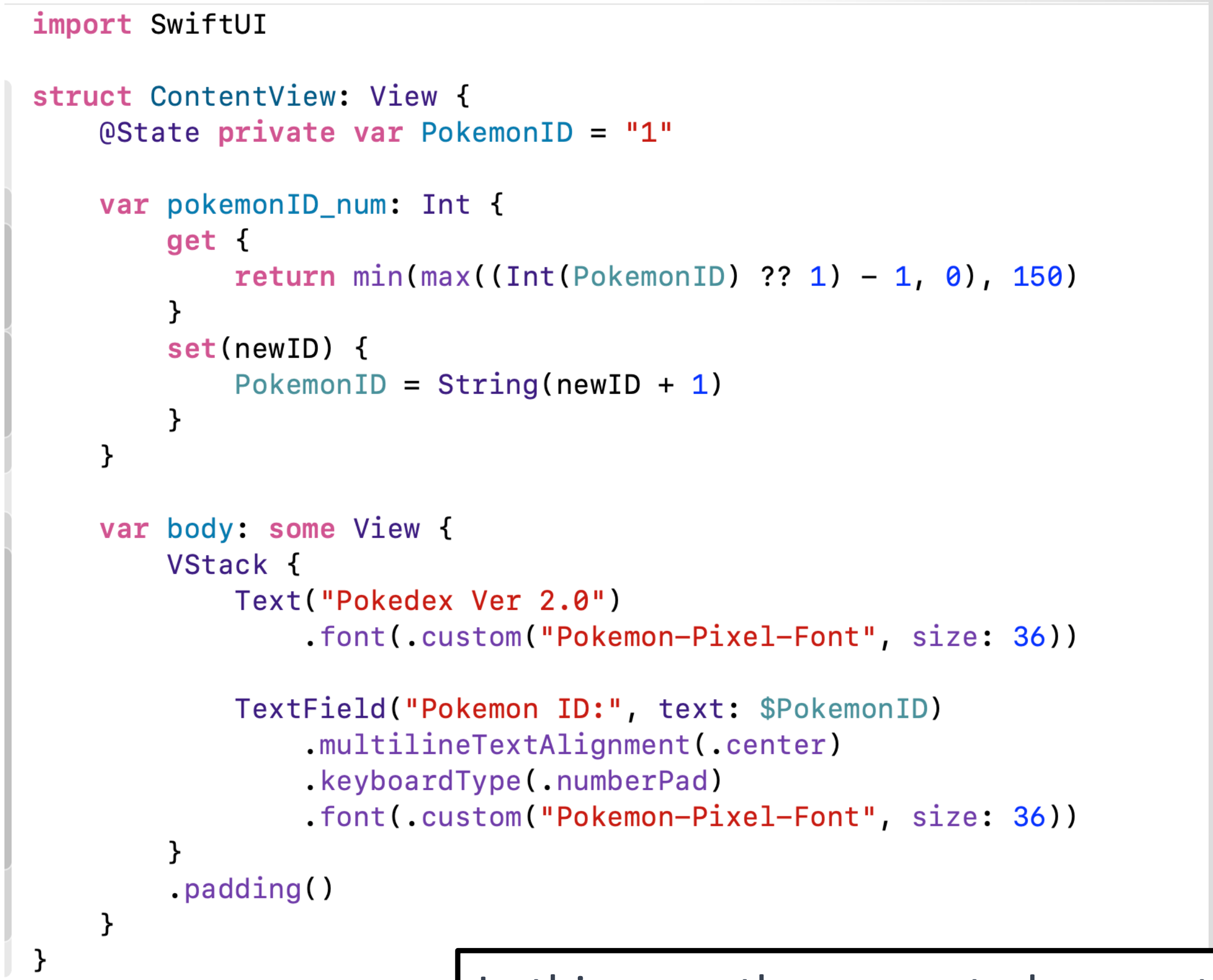
Class 9

CSC 496: iOS App Development
Swift Fundamentals: Classes (1)

Si Chen (schen@wcupa.edu)

Review

Pokédex version 2 Solution (using computed property)



```swift
import SwiftUI

struct ContentView: View {
    @State private var PokemonID = "1"

    var pokemonID_num: Int {
        get {
            return min(max((Int(PokemonID) ?? 1) - 1, 0), 150)
        }
        set(newID) {
            PokemonID = String(newID + 1)
        }
    }

    var body: some View {
        VStack {
            Text("Pokedex Ver 2.0")
                .font(.custom("Pokemon-Pixel-Font", size: 36))

            TextField("Pokemon ID:", text: $PokemonID)
                .multilineTextAlignment(.center)
                .keyboardType(.numberPad)
                .font(.custom("Pokemon-Pixel-Font", size: 36))
        }
        .padding()
    }
}
```

In this case, the computed property pokemonID_num is derived from PokemonID. Any time you get or set pokemonID_num, the underlying PokemonID state variable is accessed or modified.

Computed Property

Declaration: Example

The basic syntax of a computed property involves using a code block { } after the property name to include a get block and optionally, a set block.

```swift
struct Rectangle {
    var width: Double
    var height: Double

    var area: Double {
        get {
            return width * height
        }
        set(newArea) {
            // For simplicity, assume a square shape for the new area
            width = sqrt(newArea)
            height = sqrt(newArea)
        }
    }
}
```

In this example, area is a computed property that calculates its value by multiplying width and height. You can also set area, and doing so will update width and height accordingly.

Pokédex version 2

Objective:
To create an iOS app that displays the name and profile picture of a Pokémon based on the user-inputted Pokémon ID.

Instructions:

- **Download and Setup Project:**

Download the **pokedex_ver_2.zip** file from our class website. Unzip the file and open the project in Xcode.

- **Examine the Pokemon.Swift File:**

Open the **Pokemon.Swift** file in the project. Locate the array **firstGenPokemonNames** which contains the names of the first-generation Pokémon.

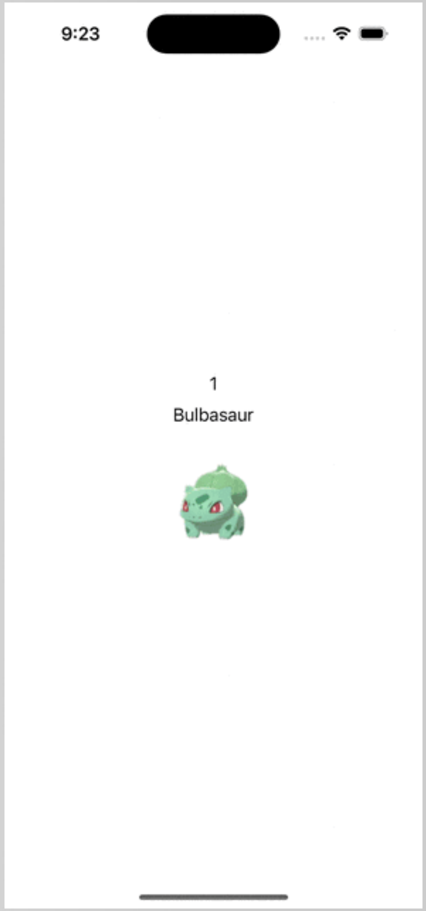
- **Implement User Input and Display in ContentView.swift:**

In **ContentView.swift**, write code to accomplish the following:

- Use the firstGenPokemonNames array to find the name of the Pokémon corresponding to the entered ID.
- Display the Pokémon's name and its corresponding profile picture based on the entered ID.

For example, if the user inputs Pokémon ID = 1, the app should display "Bulbasaur" along with its profile picture.

Tips:
You may use the .font(.custom("`Pokemon-Pixel-Font`", size: 16)) modifier to set a Pokemon font.
How would you handle invalid input (try using nil-coalescing operator ??)
How would you handle edge cases where the user input exceeds 151 or falls below 0?

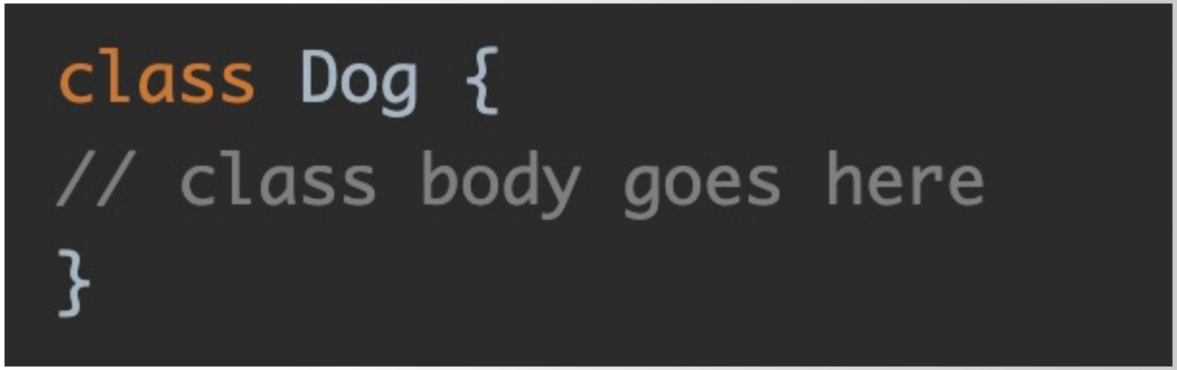


Classes

- Object-oriented programming is a common and powerful programming paradigm.
- Classes in Swift allow you to define blueprints for objects, and they are one of the building blocks of object-oriented programming (OOP).
- Classes can have properties, methods, and initializers, just like structures.
- However, they also offer **additional functionalities** not available in structures, such as inheritance, type casting, and deinitializers.

Declaration

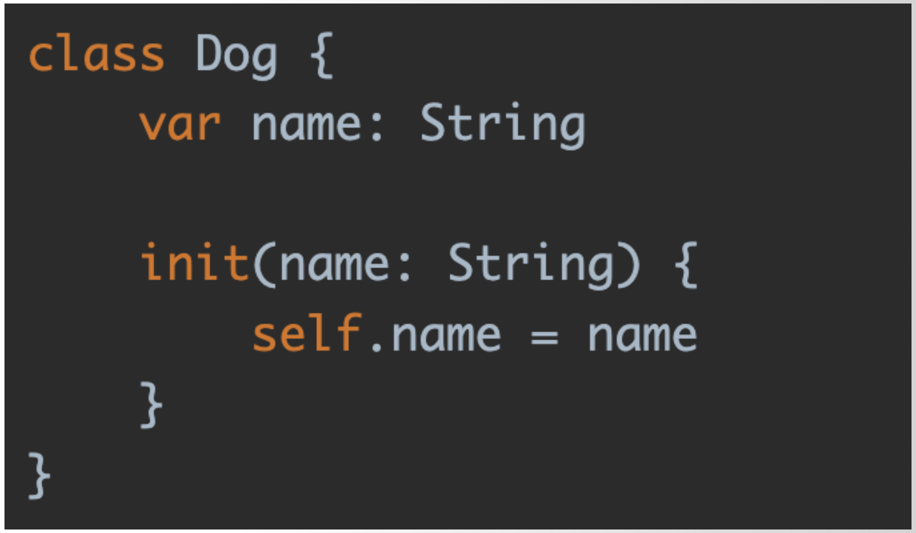
- Classes are declared with the keyword class followed by the name of the class. A basic class structure would look like this:



```
class Dog {
// class body goes here
}
```

Initialization

- A class can be initialized using an **init** method within the class. It is called when an instance of a class is created.



```
class Dog {
    var name: String

    init(name: String) {
        self.name = name
    }
}
```

To create an instance of Dog class, you just need to call Dog(name: ”Joey")

Accessing Properties

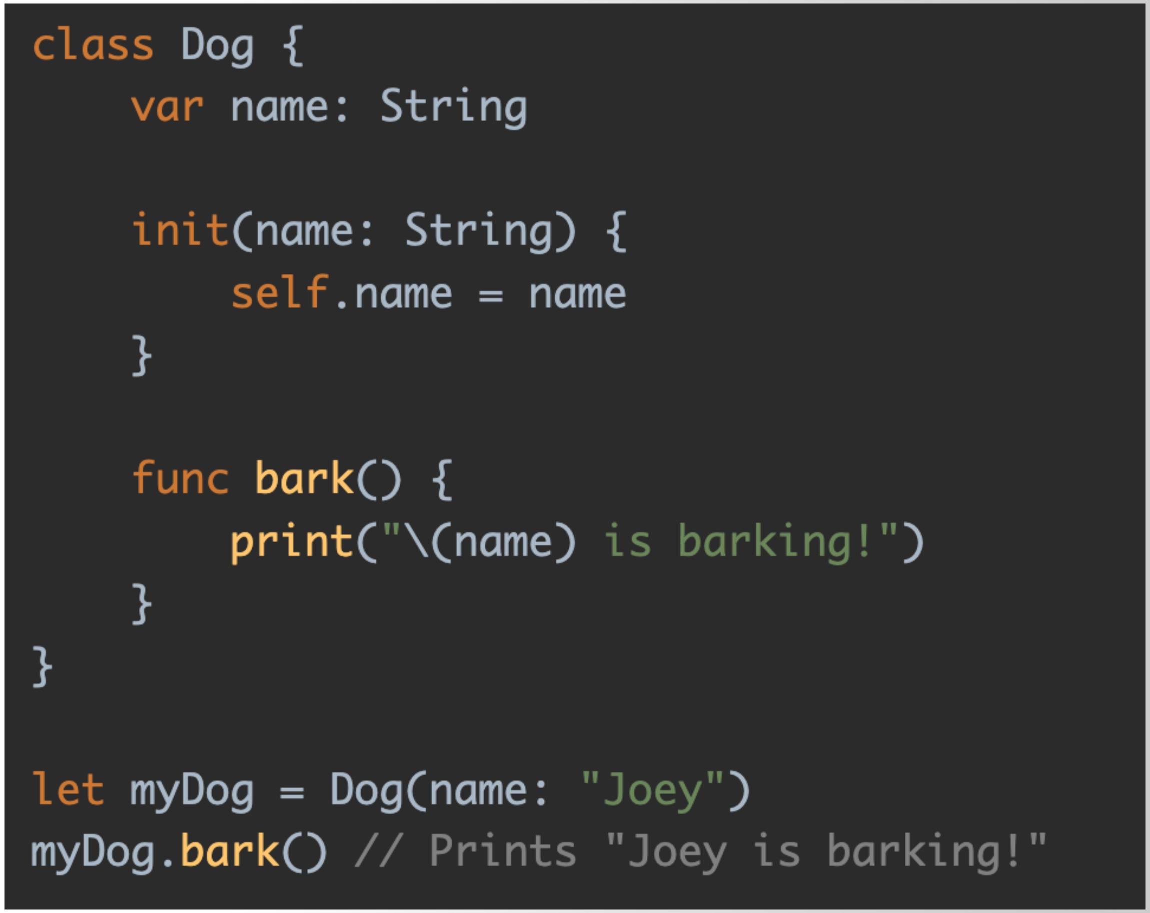
- You can access the properties of a class instance using dot syntax.

```swift
let myDog = Dog(name: "Joey")
print(myDog.name) // Prints "Joey"
```

Here, **myDog** is an instance of our **Dog** class

Methods

- Methods are functions that are associated with a particular class. They are declared within the body of the class similar to the variables declaration.

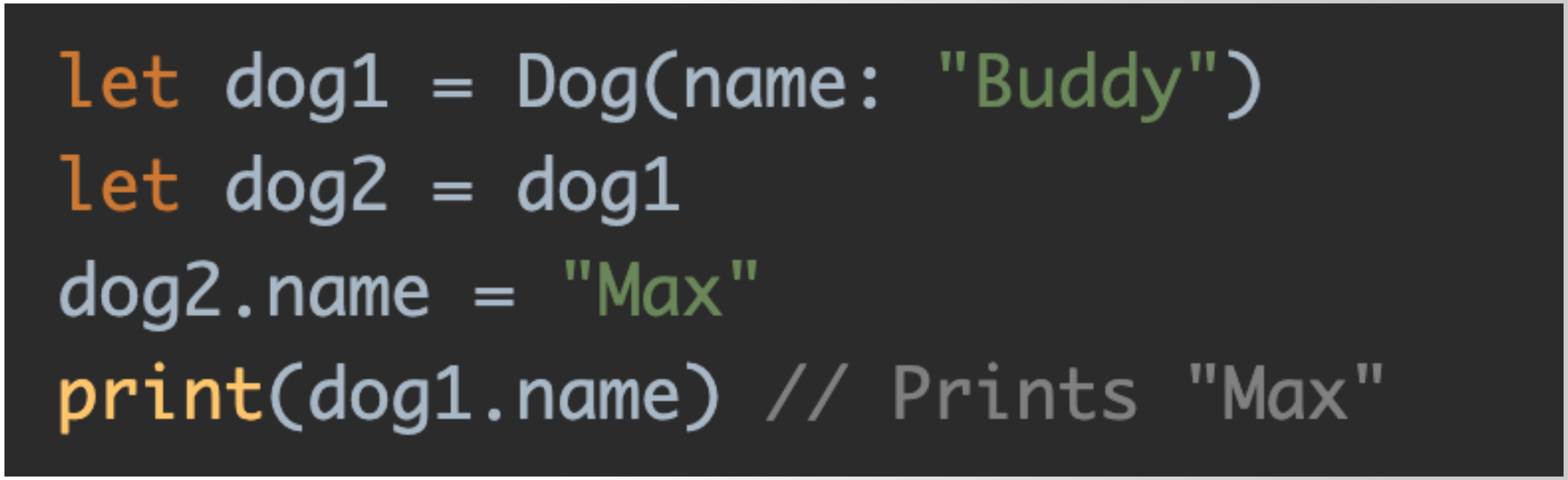


```
class Dog {
    var name: String

    init(name: String) {
        self.name = name
    }

    func bark() {
        print("\(name) is barking!")
    }
}

let myDog = Dog(name: "Joey")
myDog.bark() // Prints "Joey is barking!"
```

The method **bark()** is called on the instance **myDog**, causing it to print a message to the console.

Mutability

- Swift classes are reference types and hence their instances (or objects) are mutable.



```swift
let dog1 = Dog(name: "Buddy")
let dog2 = dog1
dog2.name = "Max"
print(dog1.name) // Prints "Max"
```

Even though **dog1** is assigned as a **let** constant, because it's a reference to an object, we can change the properties on **dog1** through **dog2**.

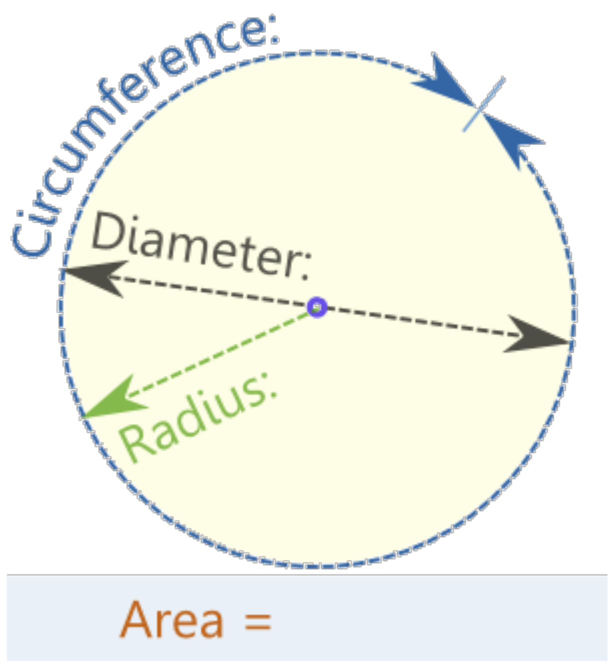
Structs vs Classes

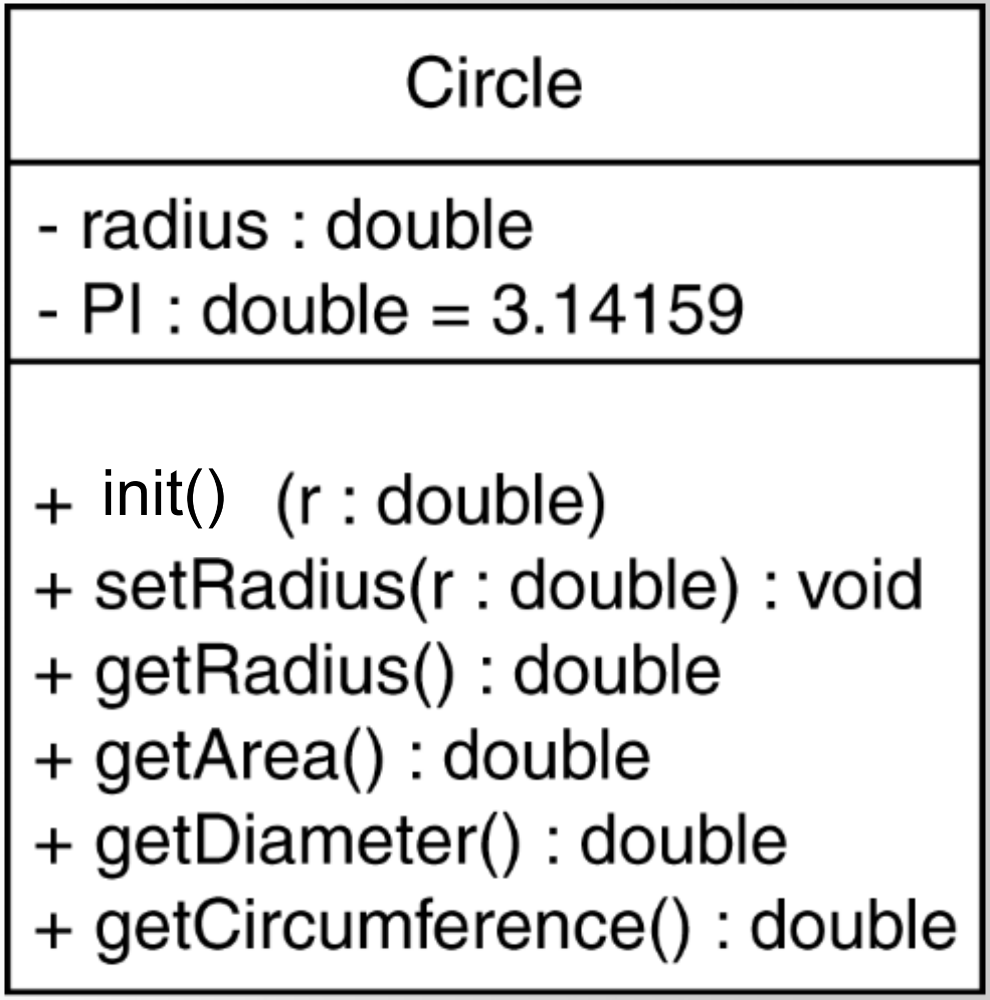
- In Swift, structs and classes have a lot in common:
  - both can have properties and methods.
- But there are some key differences you should be aware of:
  - Unlike classes, structs are **always copied** when they are **passed around your code**.
  - Structs can't be subclassed.

The key difference is that **classes are reference types** and **structs are value types**!

Exercise: Circle Class

Circle Class





```
Circle
- radius : double
- PI : double = 3.14159

+ init()  (r : double)
+ setRadius(r : double) : void
+ getRadius() : double
+ getArea() : double
+ getDiameter() : double
+ getCircumference() : double
```

Bulbasaur Class

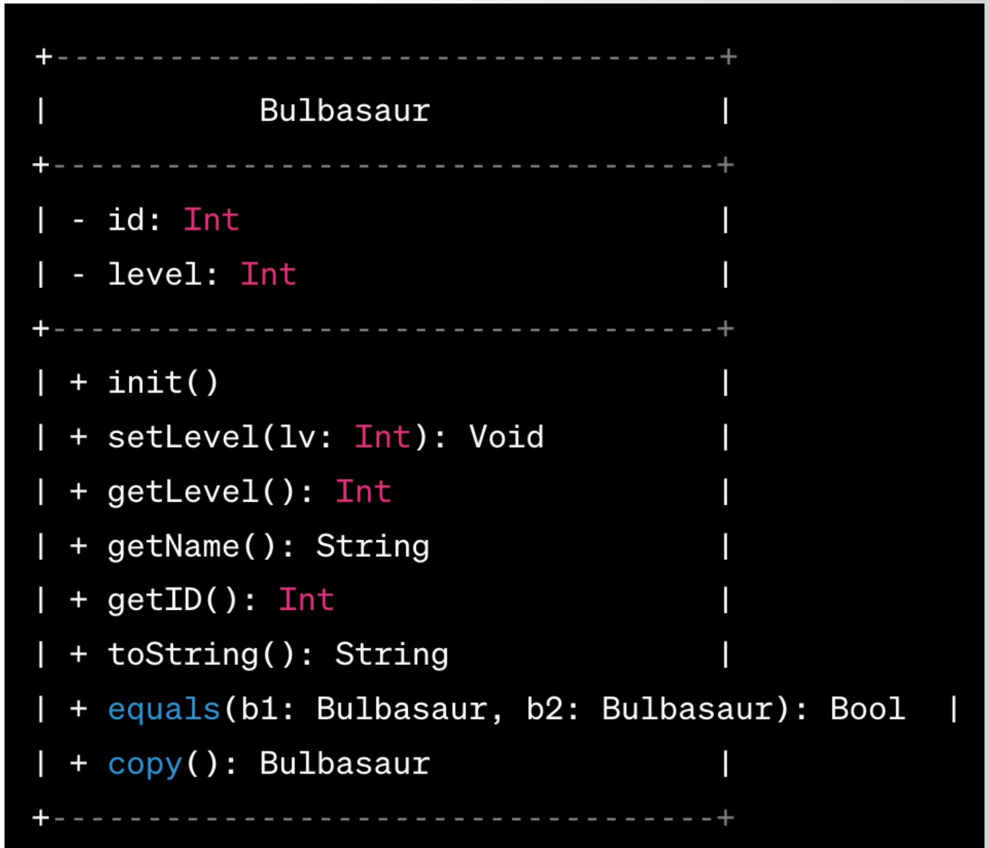
Bulbasaur, known as Fushigidane in Japan, is a Pokemon species in Nintendo and Game Freak's Pokemon franchise. Designed by Ken Sugimori, their name is a combination of the words ``bulb" and ``dinosaur". First appearing in Pokemon Red and Blue.

Exercise: Bulbasaur Class

We know that **Bulbasaur** evolves into **Ivysaur** when it reaches **level 16** and subsequently transforms into **Venusaur** upon attaining **level 32**.

UML Diagram for Bulbasaur Class

Please refer to the attached UML diagram to design a class for Bulbasaur (Bulbasaur.swift):



```
+-----------------------------------+
|           Bulbasaur               |
+-----------------------------------+
| - id: Int                         |
| - level: Int                      |
+-----------------------------------+
| + init()                          |
| + setLevel(lv: Int): Void         |
| + getLevel(): Int                 |
| + getName(): String               |
| + getID(): Int                    |
| + toString(): String              |
| + equals(b1: Bulbasaur, b2: Bulbasaur): Bool  |
| + copy(): Bulbasaur               |
+-----------------------------------+
```

Bulbasaur Class

Attributes for Each Bulbasaur Object

•**id**: Represents the evolutionary stage of the Pokemon. It is set to 1 for Bulbasaur, 2 for Ivysaur, and 3 for Venusaur. The initial value is 1.

•**level**: Denotes the current level of the Bulbasaur object, initialized to 1.

Methods for Each Bulbasaur Object

•**setLevel(lv: int): void**: Takes an integer value as an argument to update the object's level. If the new level falls within the range [16-31], the Pokemon evolves into Ivysaur. If the level is 32 or higher, it evolves into Venusaur.

•**getLevel(): int**: Returns an integer representing the current level of the Pokemon.

•**getName(): String**: Returns the name of the Pokemon as a string ("Bulbasaur" for id = 1, "Ivysaur" for id = 2, and "Venusaur" for id = 3).

•**getID()**: Returns the current id value.

•**toString()**: Outputs the current level and id of the Bulbasaur object.

•**equals()**: Compares the level and id of two different Bulbasaur objects to determine if they are equal.

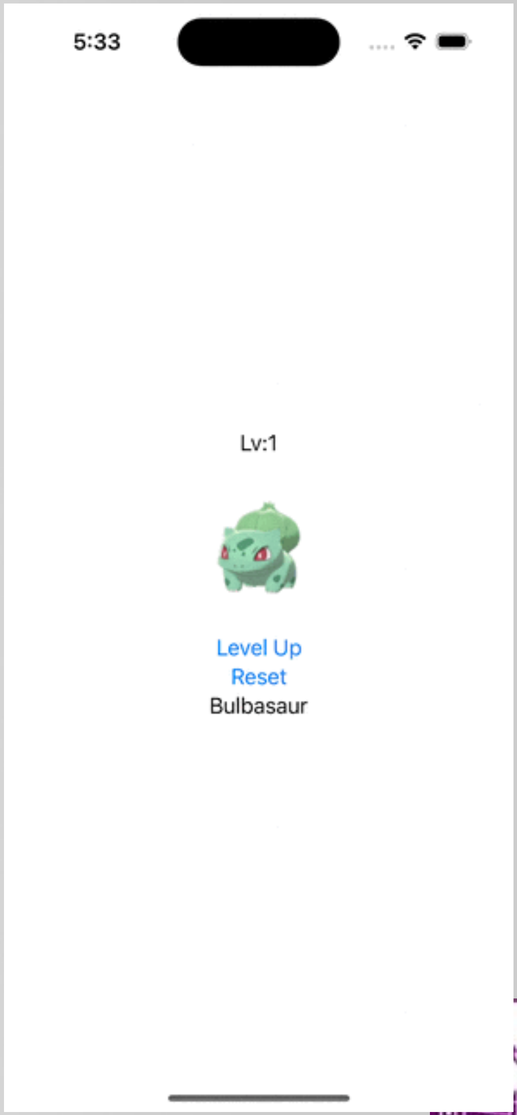
•**copy()**: Creates a clone of a Bulbasaur object with the same level and id values.

Bulbasaur Level-Up App

Objective:
Use the **Bulbasaur class** and design a SwiftUI iOS application that manages the state of a Bulbasaur as it levels up and evolves.

Requirements:
1.UI Elements:
The application should have the following UI elements:

1. A text label displaying the current level of the Pokémon (e.g., "Lv: 1").
2. An image displaying the current form of the Pokémon (Bulbasaur, Ivysaur, or Venusaur). The image assets for these forms are already provided in the course website, please download and add them to your project. The filename for each image corresponds to the Pokémon's name (e.g., "Bulbasaur.png", "Ivysaur.png", "Venusaur.png").
3. A button labeled "Level Up" that, when pressed, increments the Pokémon's level by 1 and updates its form if it evolves.
4. A ”reset” button to reset level back to 1



Q & A