Object Oriented Programming

The Swift Programming Language Part 5, Section 4

iCodeWave Community

Inheritance, Abstraction & Polymorphism

Today’s Agenda 😍

iCodeWave Community

OOP

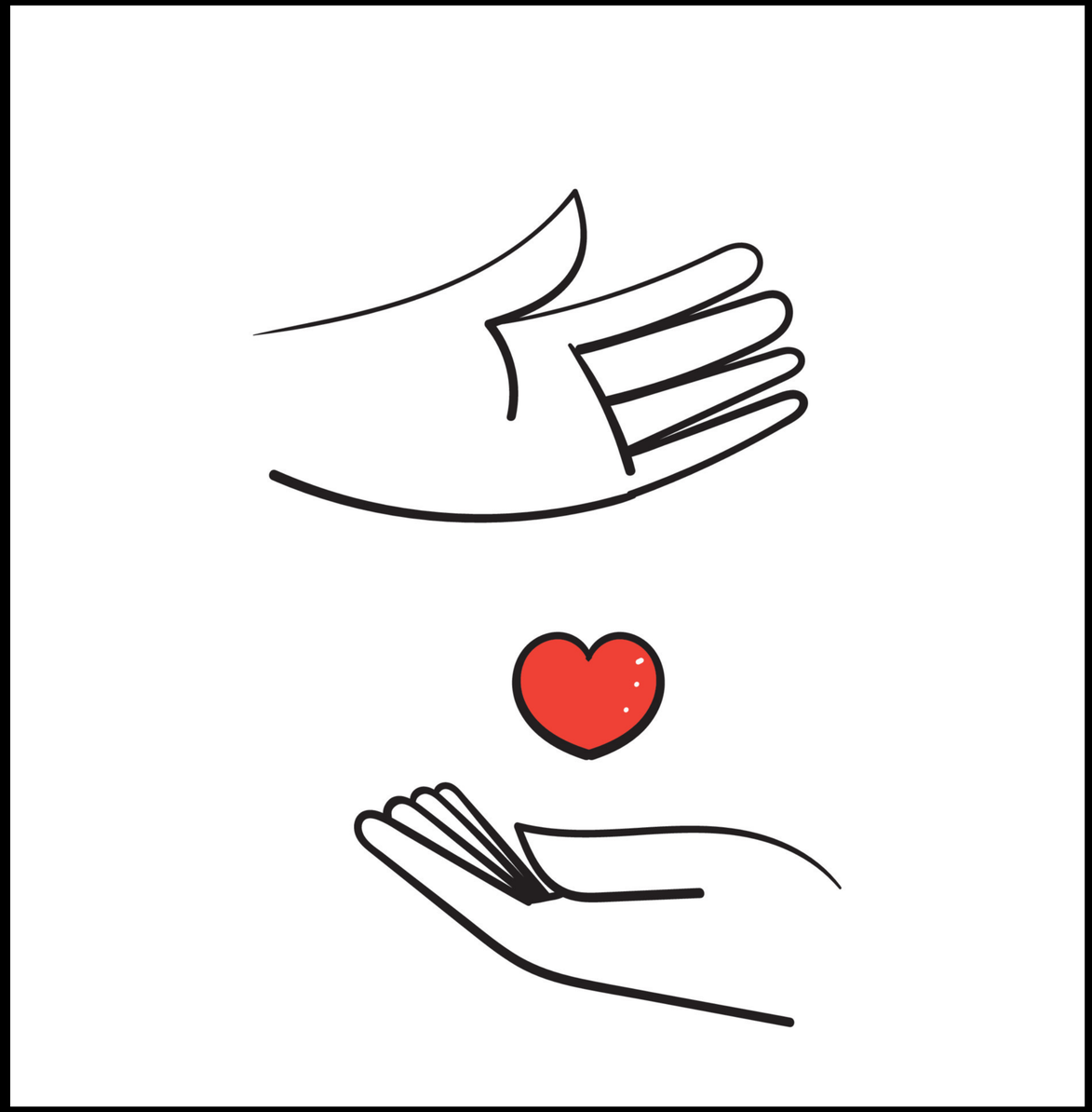
What’s inheritance?

- Inheritance allows us to create a new class from an existing class
- The new class that is created is known as subclass (child or derived class) and the existing class from which the child class is derived is known as superclass (parent or base class)

OOP

Why inheritance?

- Inheritance promotes code reuse by allowing subclasses to inherit behaviors and properties from a superclass
- Subclasses can extend or modify the behavior of the superclass, adding new functionality as needed
- Allows for the creation of hierarchical relationships between classes, enabling more organized and structured code



OOP

Inheritance – Example Code

```
class Animal { // Superclass to be inherit (parent)
    var name: String

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

    func makeSound() {
        print("Animal \(name) makes a sound")
    }
}

// Inherited class (child)
class Dog: Animal {
    override func makeSound() {
        print("Dog \(name) barks")
    }
}

let myDog = Dog(name: "Buddy")
myDog.makeSound() // Output: Dog Buddy barks
```

OOP

What’s abstraction?

- Focus on essential characteristics while hiding unnecessary details
- Model real-world objects more effectively by emphasizing relevant information and hiding implementation details
- Use of protocol as application
- Every class that implement of protocol must conform all of declared properties & method available in protocol

OOP

Why abstraction?

- Promotes encapsulation by hiding internal details and exposing only necessary functionalities
- Enables the creation of modular, reusable code by defining clear interfaces and separating implementation details
- Becomes easier to understand, maintain, and extend

OOP

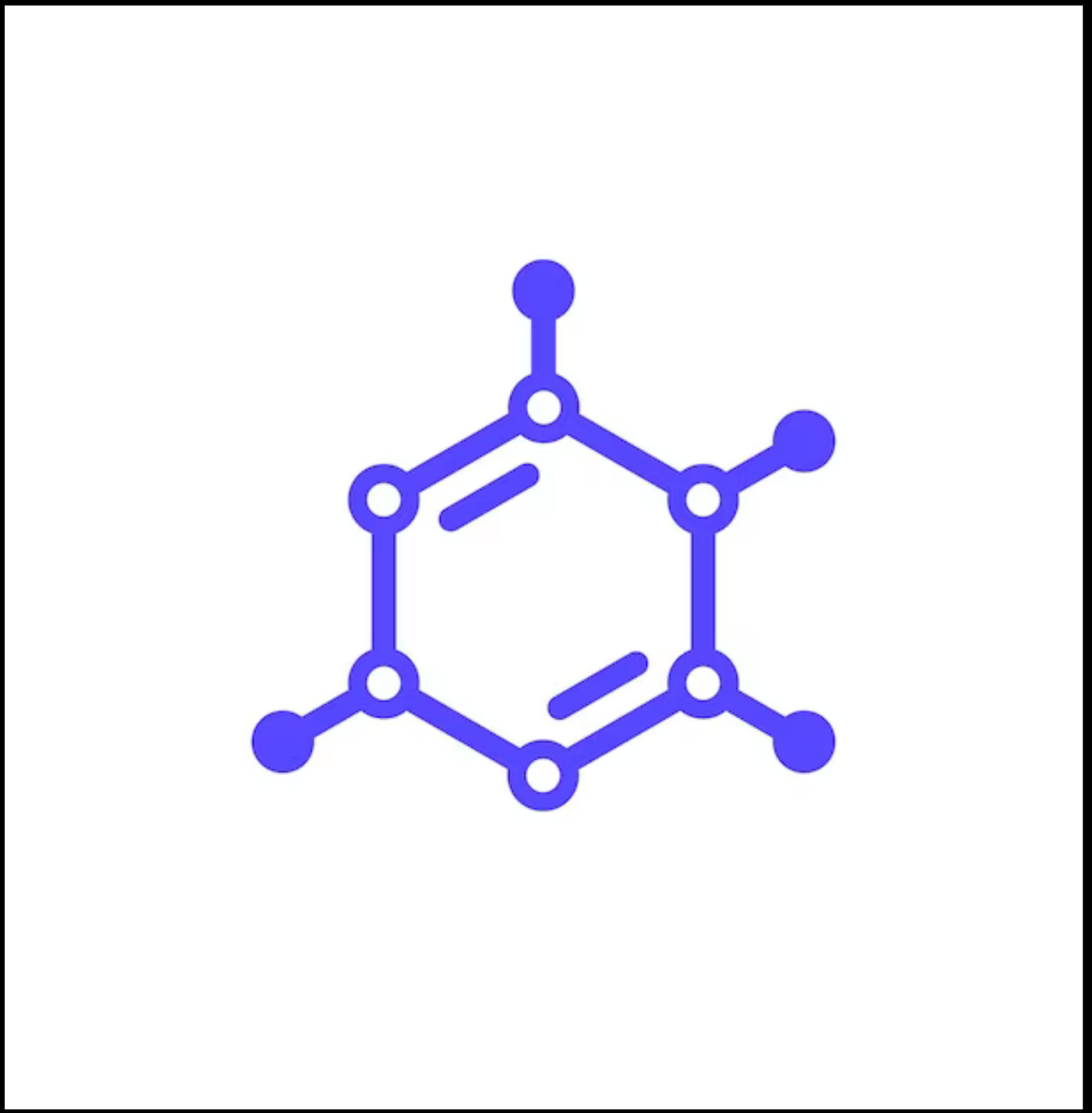
Abstraction – Example Code

```swift
protocol Shape { // this is abstraction that only declare properties/methods only
    var area: Double { get }
    func draw()
}

// class Circle must conform all properties & methods in protocol Shape
class Circle: Shape {
    var radius: Double

    init(radius: Double) {
        self.radius = radius
    }

    var area: Double { // this conform protocol Shape
        return Double.pi * radius * radius
    }

    func draw() { // this also conform protocol Shape
        print("Drawing a circle with radius \(radius)")
    }
}
```

OOP

What’s polymorphism?

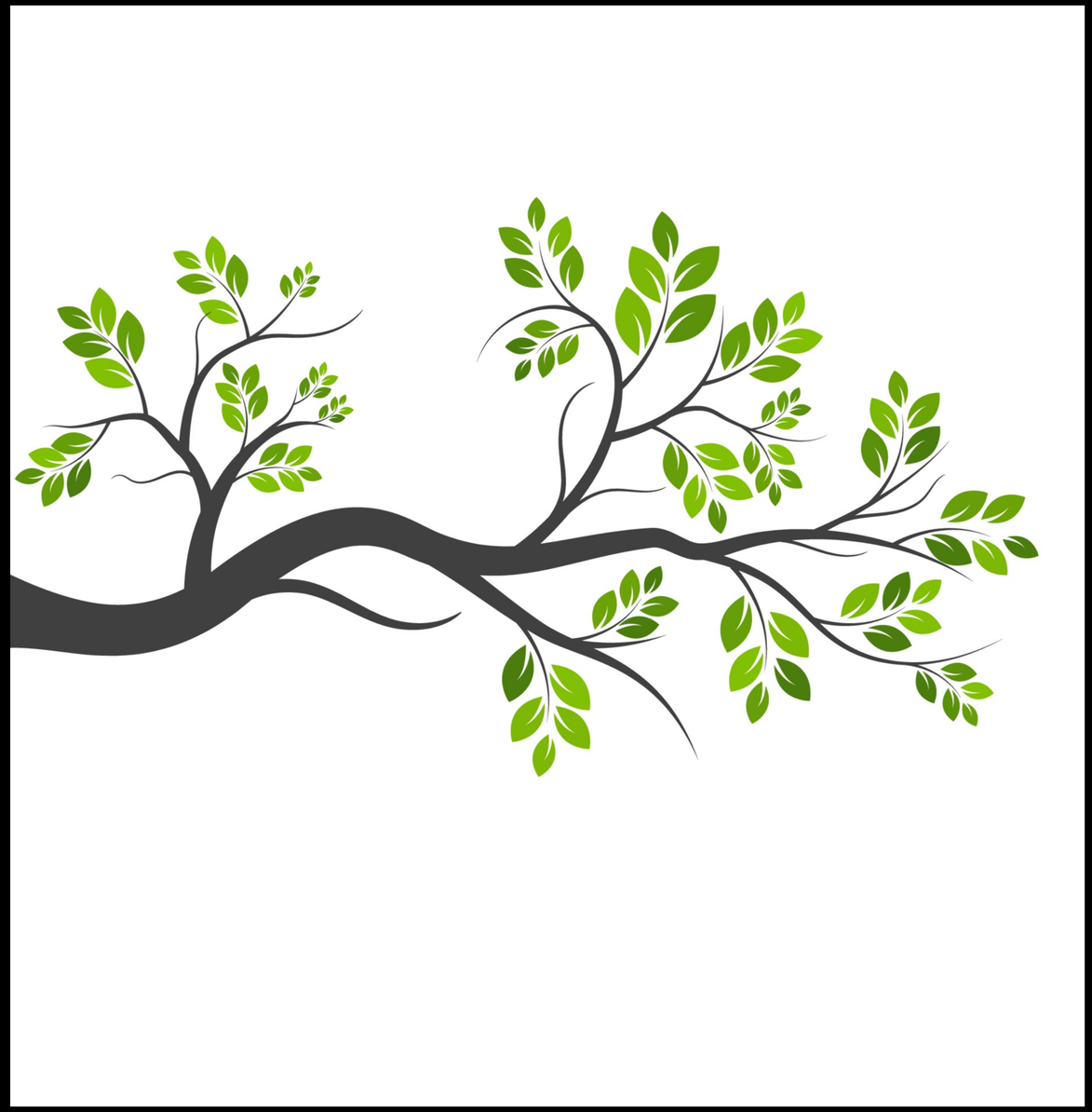
- Allows objects to be treated as instances of their superclass, enabling dynamic behavior based on the specific object type
- Achieved through method overriding
- Promotes code reusability, flexibility, and maintainability



OOP

Why polymorphism?

- Allows for the reuse of code across different classes and objects
- Enables the creation of more adaptable and scalable software
- Simplifies maintenance and updates by promoting modular and extensible code



OOP

Polymorphism – Example Code

```
class Animal { // Superclass
    func makeSound() {
        // Default implementation
        print("Animal makes a sound")
    }
}

class Dog: Animal {
    override func makeSound() { // this is polymorphism
        print("Dog barks")
    }
}

class Cat: Animal {
    override func makeSound() { // this is also polymorphism
        print("Cat meows")
    }
}

let animals: [Animal] = [Dog(), Cat()]
for animal in animals {
    animal.makeSound() /* Output:
                          Dog barks
                          Cat meows
                          */
}
```

iCodeWave Community

Thanks For Your Attendance Today!

iCodeWave Community