

# **Odisseia Orientada a Objetos**

## **A Saga da Programação**

**Aprenda Programação Orientada a  
Objetos de forma lúdica!**

**Eduardo Costa**

01

**PILARES**

---

# 1 - PILARES

A Programação Orientada a Objetos (POO) é um paradigma de programação que organiza e estrutura o código em torno de "objetos", que combinam dados e funcionalidades relacionadas.

Ela possui 4 pilares que vamos aprender utilizando uma dinâmica de um mundo de magia.



02

**OBJETO**

---

## 2 - OBJETO

Primeiramente devemos saber o que é um objeto

Os objetos são basicamente formas de retratar algo que possui características e ações. Como um personagem de jogo que possui nome, vida e ataque.



```
Herói.swift

class Heroi {
    var nome: String
    var vida: Int

    func atacar(alvo: Monstro) {
        print("\(nome) ataca \(alvo.nome)!")
    }
}
```

03

**ENCAPSULAMENTO**

---

# 3 - ENCAPSULAMENTO

‘Ninguém mais toca além de mim!’

Toda alteração de dados deve ser feita somente por métodos da própria classe, não podendo ser modificado por fora durante o código.



Heroi.swift

```
class Heroi {  
    // Variáveis encapsuladas  
    private var nome: String  
    private var vida: Int  
  
    func receberDano(dano: Int) {  
        if dano > 0 {  
            vida -= dano  
            print("\(nome) sofreu \(dano) de dano. Vida restante: \(vida)")  
        }  
    }  
}
```

04

**HERANÇA**

3



## 4 - HERANÇA

Que tal termos heróis diferentes?

Lembra da nossa classe “Herói”?  
Podemos ter especializações que fazem tudo que os heróis fazem, e mais!



```
Herói.swift

class Guerreiro: Herói {
    var arma: String
}

class Mago: Herói {
    var poderMagico: Int
}
```

05

**POLIMORFISMO**

---

## 5 - POLIMORFISMO

Fazem coisas iguais, mas de forma diferente!

A nossa classe “Herói” possui a função “atacar” que nossa guerreira e mago herdaram, mas cada um ataca de forma diferente, né?



```
Herói.swift

class Guerreiro: Herói {
    var arma: String
    override func atacar(alvo: Monstro) {
        print("\(nome) executa um ataque com sua \(arma) em \(alvo.nome)!")
    }
}

class Mago: Herói {
    var poderMagico: Int
    override func atacar(alvo: Monstro) {
        print("\(nome) lança um feitiço com um valor mágico de \(poderMagico) em \(alvo.nome)!")
    }
}
```

06

**ABSTRAÇÃO**



## 6 - ABSTRAÇÃO

Um exemplo a ser seguido

Como nosso “Herói” é somente uma base, a classe não precisa de conteúdo em suas funções, só precisa dizer o que o guerreiro e o mago precisam saber.



```
Heroi.swift

class Heroi {
    func usarHabilidade() {
        // Método vazio a ser sobrescrito pelas subclasses
    }
}
```

## 6 - ABSTRAÇÃO

Um exemplo a ser seguido

Agora que eles sabem que precisam ter a função “usarHabilidade”, cada um usa de seu jeito.



Heroi.swift

```
class Guerreiro: Heroi {  
    override func usarHabilidade(dano: Int) {  
        print("Causou \(dano) com sua habilidade!")  
    }  
}  
  
class Mago: Heroi {  
    func usarHabilidade(qtdInimigos: Int) {  
        print("Atinge \(quantidadeInimigosAtingidos) de inimigos!")  
    }  
}
```

07

**CONCLUSÃO**



## 7 - CONCLUSÃO

Vamos criar nossos personagens!

Para a criação dos nossos objetos, as classes precisam ter um inicializador.

É uma forma de criamos nossos personagens com seus atributos iniciais, nele passamos as informações essenciais que nosso personagem precisa ter, como nome e vida.

Mago.swift

```
class Mago: Heroi {  
    init(nome: String, vida: Int) {  
        super.init(nome: nome, vida: vida)  
    }  
  
    func usarHabilidade(quantidadeInimigosAtingidos: Int) {  
        print("Atinge \((quantidadeInimigosAtingidos) de inimigos!")  
    }  
}
```



## 7 - CONCLUSÃO

Vamos criar nossos personagens!

Agora que sabemos como criar nossas classes, basta criar nossos objetos com elas!



```
let magoBom = Mago(nome: "Iluminado", vida: 120)  
let magoMau = Mago(nome: "Multi-olhos", vida: 100)
```

# AGRADECIMENTOS



# Obrigado por ler até aqui!

O objetivo deste e-book  
é totalmente didático e  
foi criado como  
atividade de curso.

As imagens utilizadas são da  
empresa Ankama de seu jogo  
Waven, os códigos foram feitos  
pelo ChatGPT e modificado por  
mim e, para apresentação,  
colocados no ShowCode.

