Herança e Agregação em Kotlin

Apresentação: Prof. Carlos Alberto

Herança em Kotlin

- Herança é um mecanismo onde uma classe pode herdar propriedades e métodos de outra classe. A classe que herda é chamada de **subclasse** e a classe que é herdada é chamada de **superclasse**. A herança permite reutilizar código e criar hierarquias entre as classes.

Sintaxe de Herança em Kotlin:

- Em Kotlin, a herança é implementada utilizando a palavra-chave :, e a classe base (superclasse) deve ser declarada com o modificador open para permitir que outras classes a estendam.

Herança em Kotlin

kotlin

```kotlin
// Classe base
open class Animal(val nome: String) {
    open fun fazerSom() {
        println("$nome está fazendo um som")
    }
}

// Subclasse que herda de Animal
class Cachorro(nome: String) : Animal(nome) {
    override fun fazerSom() {
        println("$nome está latindo")
    }
}

fun main() {
    val cachorro = Cachorro("Rex")
    cachorro.fazerSom()  // Saída: Rex está latindo
}
```

Explicação

- A classe Animal é a superclasse e tem uma função fazerSom().
- A classe Cachorro herda de Animal e sobrescreve a função fazerSom().
- O modificador open é necessário para que a classe Animal e a função fazerSom() possam ser estendidas e sobrescritas, respectivamente.
- A palavra-chave override é usada para indicar que a função fazerSom() está sendo sobrescrita.

Agregação em Kotlin

- A agregação é um tipo de associação entre objetos onde uma classe (a agregadora) tem uma referência a objetos de outra classe. Ao contrário da herança, a agregação não implica em uma relação "é-um" (como na herança), mas sim em uma relação "tem-um". A agregação é uma forma de modelar a composição de objetos em que um objeto "contém" outros objetos.
- **Sintaxe de Agregação em Kotlin:**
- A agregação pode ser implementada criando uma variável de um tipo diferente dentro de uma classe. Isso é feito por meio de uma simples referência a outra classe.

```kotlin
// Classe Item
class Item(val nome: String, val preco: Double)

// Classe Carrinho que agrega objetos Item
class Carrinho {
    private val itens = mutableListOf<Item>()

    fun adicionarItem(item: Item) {
        itens.add(item)
    }

    fun mostrarItens() {
        for (item in itens) {
            println("${item.nome} - ${item.preco}")
        }
    }
}

fun main() {
    val item1 = Item("Camiseta", 29.90)
    val item2 = Item("Calça", 79.90)

    val carrinho = Carrinho()
    carrinho.adicionarItem(item1)
    carrinho.adicionarItem(item2)

    carrinho.mostrarItens()  // Saída: Camiseta - 29.9, Calça - 79.9
}
```

Explicação

- A classe **Item** representa um produto que tem um nome e um preço.
- A classe **Carrinho** agrega múltiplos objetos Item. Aqui, a agregação é representada pela lista itens, que contém objetos da classe Item.
- O carrinho pode adicionar itens e exibir sua lista.
- Observe que a agregação é representada pela instância de Item dentro de Carrinho, mas as instâncias de Item existem independentemente da instância de Carrinho.

Comparação entre Herança e Agregação:

- **Herança** cria uma relação "é-um". Por exemplo, um Cachorro **é** um Animal. A classe Cachorro herda as propriedades e métodos de Animal.
- **Agregação** cria uma relação "tem-um". Por exemplo, um Carrinho **tem** itens. Um Carrinho pode ter muitos objetos Item, mas não é um Item nem é um tipo de Item.

Quando usar herança e quando usar agregação?

- **Use herança** quando uma classe pode ser uma especialização de outra. Exemplo: um Cachorro é um tipo de Animal.
- **Use agregação** quando uma classe precisar conter ou referenciar objetos de outra classe, mas sem que uma seja uma especialização da outra. Exemplo: um Carrinho contém Itens, mas Carrinho e Item são conceitos independentes.

Exemplo Combinado de Herança e Agregação:

- Podemos combinar ambos os conceitos. Por exemplo, um Carro pode ser uma subclasse de Veiculo (herança) e ainda ter um relacionamento de agregação com a classe Motor (agregação).

kotlin

Copiar

```kotlin
// Classe base Veiculo
open class Veiculo(val modelo: String) {
    open fun ligar() {
        println("$modelo está ligando")
    }
}

// Classe Motor que será agregada no Carro
class Motor(val tipo: String) {
    fun acionar() {
        println("Motor $tipo acionado")
    }
}

// Classe Carro que herda de Veiculo e agrega Motor
class Carro(modelo: String, val motor: Motor) : Veiculo(modelo) {
    override fun ligar() {
        super.ligar()
        motor.acionar()
    }
}

fun main() {
    val motor = Motor("V8")
    val carro = Carro("Fusca", motor)

    carro.ligar()
    // Saída: Fusca está ligando
    // Saída: Motor V8 acionado
}
```

O Carro herda de Veiculo e agrega um Motor, ou seja, o carro **é um** veículo e **tem um** motor.

```kotlin
*/
open class Veiculo(val modelo: String) {
    open fun ligar() {
        println("$modelo está ligando")
    }
}

// Classe Motor que será agregada no Carro
class Motor(val tipo: String) {
    fun acionar() {
        println("Motor $tipo acionado")
    }
}

// Classe Carro que herda de Veiculo e agrega Motor
class Carro(modelo: String, val motor: Motor) : Veiculo(modelo) {
    override fun ligar() {
        super.ligar()
        motor.acionar()
    }
}

fun main() {
    val motor = Motor("V8")
    val carro = Carro("Fusca", motor)

    carro.ligar()
    // Saída: Fusca está ligando
    // Saída: Motor V8 acionado
}
```