iOS

Herança

Softblue
cursos online

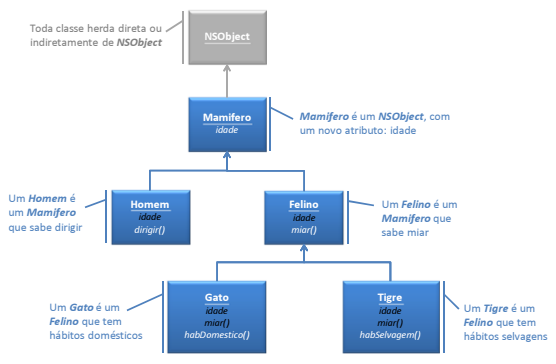
Herança

Softblue

- Permite que uma classe filha (subclasse) herde elementos não privados (***private***) de sua classe pai, também conhecida como superclasse
- Benefícios:
  - Reaproveitamento de código
  - Organização do código
  - Facilidade de manutenção
- Em algumas linguagens é possível utilizar herança múltipla
- Na omissão, herda-se da classe ***NSObject***

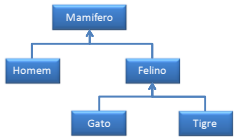
Hierarquia de classes

Softblue



Herdando uma classe

- Operador **:**




```
class Mamifero {

    var idade : Int
    var peso : Int


    func andar() -> Void {
        // Método que anda
    }

}
```

```
class Homem {

    int idade : Int
    int peso : Int
    int cpf : String

    func andar() -> Void {
        // Método que anda
    }

    func dirigir() -> Void {
        // Método que dirige
    }
}
```

```
class Homem : Mamifero {

    int idade : Int
    int peso : Int
    var cpf : String

    func andar() -> Void {
        // Método que anda
    }

func dirigir -> Void {
        // Método que dirige
    }

}
```

Sobrescrita de métodos

- Métodos herdados podem ser sobrescritos
- Subclasse se comporta de maneira diferente para a mesma invocação do método
- Métodos sobrescritos substituem para a classe em questão os métodos da superclasse
- Superclasse continua com seu comportamento habitual, não sofrendo alterações

Sobrescrevendo métodos

- Operador ***override***
- Geração de erro se não utilizar o operador corretamente

```
class Mamifero {

    func andar() -> String {
        // Método que anda de mamífero
        return 1
    }

}
```

```
class Homem : Mamifero {

    override func andar() -> String {
        // Método que anda de mamífero
        return 2
    }

}
```

```
var m : Mamifero = Mamifero()
m.andar()    // andar(mamífero): retorna 1

var h : Homem = Homem()
h.andar()    // andar(homem): retorna 2
```

Sobrecarga de métodos

- Métodos de mesmo nome mas com diferentes assinaturas (parâmetros)
- Compilador decifra qual invocar baseado na assinatura
- Permite criar diferentes comportamentos para solucionar uma mesma demanda baseada em entradas diferentes

Sobrecarregando métodos

- Não há um operador próprio
- Geração de erro se as assinaturas forem iguais

```
class Homem {

    func deslocar() -> String {
        return "Andando"
    }

    func deslocar(carro: String) -> String {
        return "Dirigindo um \(carro)"
    }

}
```

```
var h : Homem = Homem()
h.deslocar()              // retorna "Andando"
h.deslocar("Porsche")     // retorna "Dirigindo um Porsche"
```

Sobrescrita vs. sobrecarga

- Não são a mesma coisa
- Sobrescrita
  - Ocorre somente em herança
  - Diferentes comportamentos para diferentes níveis hierárquicos
- Sobrecarga
  - Pode ou não ocorrer com herança
  - Diferentes comportamentos para diferentes assinaturas (entradas de dados)

Construtores hierárquicos

- Operador ***super***

```
class Mamifero
{
    var idade : Int
    var peso : Int

    init(i: Int, p: Int)
    {
        // Validações de idade e peso
        self.idade = i
        self.peso = p
    }

}
```

```
class Homem : Mamifero
{
    var cpf : Int

    init(ih: Int, ph: Int, ch: String)
    {
        super.init(i: ih, p: ph)

        // Validações de cpf
        self.cpf = c
    }

}
```

self vs. super

```
class Mamifero
    var idade : Int

    init(idade: Int) {
        self.idade = idade
    }

    func som() -> String {
        return "Grr"
    }
```

```
class Felino : Mamifero

    func som() -> String {
        return "Miau"
    }

    func somAncestral() -> String {
        return super.som()
    }
```

```
var m : Mamifero = Mamifero()
m.som()                 // Retorna "Grr"

var f : Felino = Felino()
f.som()                 // Retorna "Miau";
f.somAncestral()        // Retorna "Grr"
```

Aulas práticas e manuais on-line

Assista agora às aulas práticas.

Clique aqui para visualizar as aulas práticas disponíveis