

Kotlin

Classes e Objetos




Softblue
cursos online

Tópicos Abordados



- Classes
- Construtores
 - Primários
 - Secundários
- Properties
- Functions
 - Tipo *Unit*
 - Valores padrão
 - Single-expression functions
- Companion Objects
- Enum Classes
- Data Classes

Classes



- Classes em Kotlin são criadas com o uso da expressão *class*

```
class Automovel {  
    var cor: String = "branco"  
    var ano: Int = 1985  
    var ligado = false;  
}
```

→ Properties

```
var auto = Automovel()  
print(auto.cor);  
print(auto.ano);  
print(auto.ligado)
```

→ Não existe o operador `new()` em Kotlin!
→ O valor das properties é acessado com o operador `"."`

Construtores



- Toda classe precisa ter um construtor primário e, opcionalmente, um ou mais construtores secundários
- Se o construtor primário não for declarado, será gerado automaticamente

```
class Automovel(var cor: String, var ano: Int, var ligado: Boolean) {  
}
```

→ Define as properties + construtor primário

```
var auto = Automovel("azul", 1980, false)  
  
print(auto.cor);  
print(auto.ano);  
print(auto.ligado)
```

→ Os parâmetros devem ser fornecidos

Construtores



```
class Automovel(corAuto: String) {  
    var cor : String = corAuto  
    var ano : Int = 1985  
    var ligado = false;  
}
```

Parâmetro do construtor
copiado para a property

```
var auto = Automovel("azul")
```

```
class Automovel(corAuto: String) {  
    var cor: String = corAuto  
    var ano: Int;  
    var ligado: Boolean  
  
    init {  
        ano = 1985;  
        ligado = false;  
    }  
}
```

O bloco init() é usado
para inicializar o objeto

Construtores secundários



- São opcionais
- Devem chamar o construtor primário (se ele foi definido)

```
class Automovel(corAuto: String) {  
    var cor: String = corAuto  
    var ano: Int;  
    var ligado: Boolean  
  
    constructor(ano: Int) : this("branco") {  
        this.ano = ano;  
    }  
}
```

→ Atribui o valor à property ano

→ Chama o construtor
primário passando a cor

Properties



- Uma property pode ser mutável (definida como *var*) ou imutável (definida como *val*)
- Métodos getters e setters são opcionais

```
class Endereco {  
    var rua: String = ""  
  
    var numero = 0  
    set(value) {  
        if (value > 0) field = value;  
    }  
  
    val numeroValido: Boolean  
    get() = numero > 0  
}
```

→ Setter

→ Getter

```
var e = Endereco()  
e.rua = "Rua das Palmeiras"  
e.numero = 200  
  
print(e.rua)  
print(e.numero)  
print(e.numeroValido)
```

Functions



- Classes podem definir *member functions*

Nome Parâmetros Tipo de retorno

```
class ContaBancaria(var saldo: Double) {  
    fun calcular(taxa: Double) : Double {  
        if (saldo == 0.0) return 0.0;  
        return saldo - saldo * taxa  
    }  
}
```

```
var b = ContaBancaria(1000.0)  
var valor = b.calcular(0.02)
```

Em Kotlin, funções também podem ser criadas fora de classes

O tipo *Unit*



- Toda função em Kotlin deve ter um tipo de retorno
- Quando uma função não precisa de um retorno explícito, o tipo retornado é *Unit*

```
fun depositar(valor: Double) : Unit {  
    saldo += valor  
    return Unit  
}
```

→ Pode ser omitido

→ Pode ser omitido

Valores padrão



- Os parâmetros definidos em funções podem possuir valores padrão
 - São assumidos se o valor não for passado explicitamente

```
fun sacar(valor: Double = 100.0) {  
    saldo -= valor;  
}
```

```
b.sacar()
```

Single-Expression Functions



- São funções definidas por uma única expressão

```
fun obterSaldoDisponivel() = Math.max(0.0, saldo - 100)
```

→ O uso das chaves é substituído pelo "="

Companion Objects



- É o recurso que o Kotlin usa para permitir a declaração de elementos estáticos (pertencentes à classe e não aos objetos)

```
class Matematica {  
    companion object {  
        fun mult(v1: Int, v2: Int) : Int = v1 * v2  
    }  
}
```

```
var v = Matematica.mult(10, 2)
```

→ Chamada feita diretamente na classe

Enum Classes



- Um enum é um tipo que permite que apenas determinados valores sejam atribuídos

```
enum class DiasUteis {  
    SEGUNDA, TERCA, QUARTA, QUINTA, SEXTA  
}
```

```
var d: DiasUteis  
d = DiasUteis.TERCA  
d = 2
```

→ Não compila!

Enum Classes



- Um enum pode receber parâmetros de inicialização

```
enum class Sexo(val sigla : String) {  
    MASCULINO("M"), FEMININO("F")  
}
```

```
var s = Sexo.MASCULINO  
print(s.sigla)
```

Data Classes



- Uma data class é uma classe responsável exclusivamente por guardar informações

```
data class Usuario(val email: String, val senha: String)
```

```
var user1 = Usuario("abc@xyz.com", "1234")  
var user2 = user1.copy(senha = "4321")
```

→ Cria uma cópia do objeto alterando apenas o que for desejado