

Fundamentos de Desenvolvimento Android

MIT em Desenvolvimento Mobile - 2020

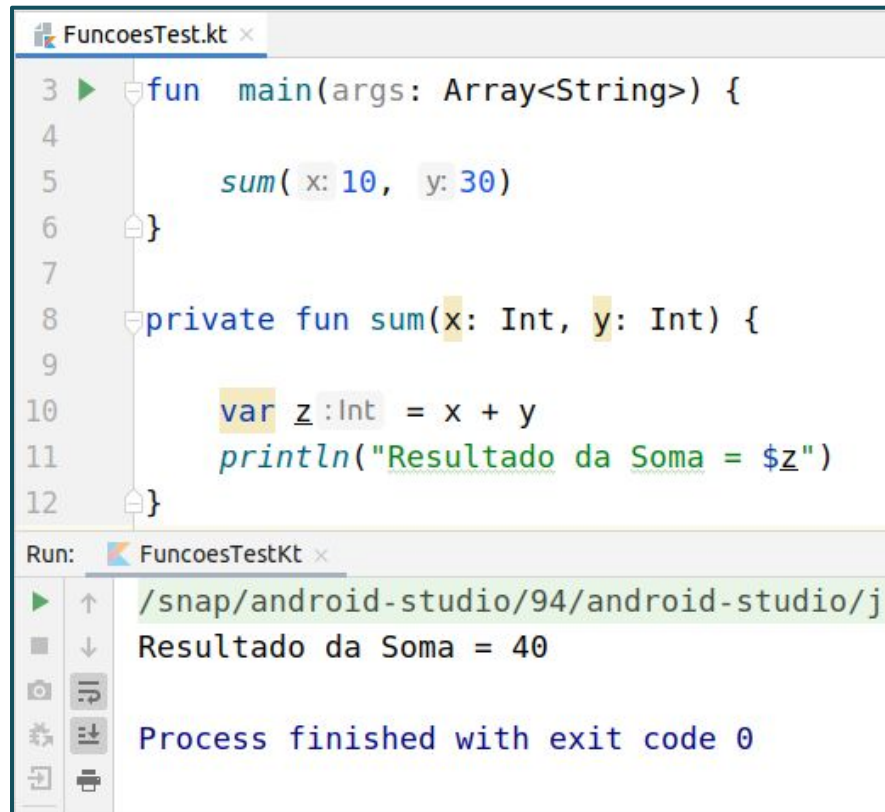
Linguagem Kotlin: Funções

Funções

Uma função é qualquer agrupamento nomeado que inclui um determinado código ou uma coleção de instruções para realizar uma operação.

{ } → representam agrupamento

Cada função tem um nome exclusivo que é usado para sua chamada dentro do programa Kotlin, sem a necessidade de duplicar instruções em vários arquivos de código-fonte.



The screenshot displays the Android Studio interface. The top editor window, titled 'FuncoesTest.kt', contains the following Kotlin code:

```
3 fun main(args: Array<String>) {  
4  
5     sum( x: 10, y: 30)  
6 }  
7  
8 private fun sum(x: Int, y: Int) {  
9  
10    var z: Int = x + y  
11    println("Resultado da Soma = $z")  
12 }
```

Below the editor, the 'Run' tab is active, showing the execution output for 'FuncoesTestKt':

```
/snap/android-studio/94/android-studio/j  
Resultado da Soma = 40  
  
Process finished with exit code 0
```

Funções

```
FuncoesTest.kt x
3 ▶ fun main(args: Array<String>) {
4
5     sum( x: 10, y: 30)
6 }
7
8 private fun sum(x: Int, y: Int) {
9
10    var z: Int = x + y
11    println("Resultado da Soma = $z")
12 }
```

Run: FuncoesTestKt x

```
▶ ↑ /snap/android-studio/94/android-studio/j
■ ↓ Resultado da Soma = 40
📷 🔄
⚙️ 📄
🔍 🖨️
```

Process finished with exit code 0

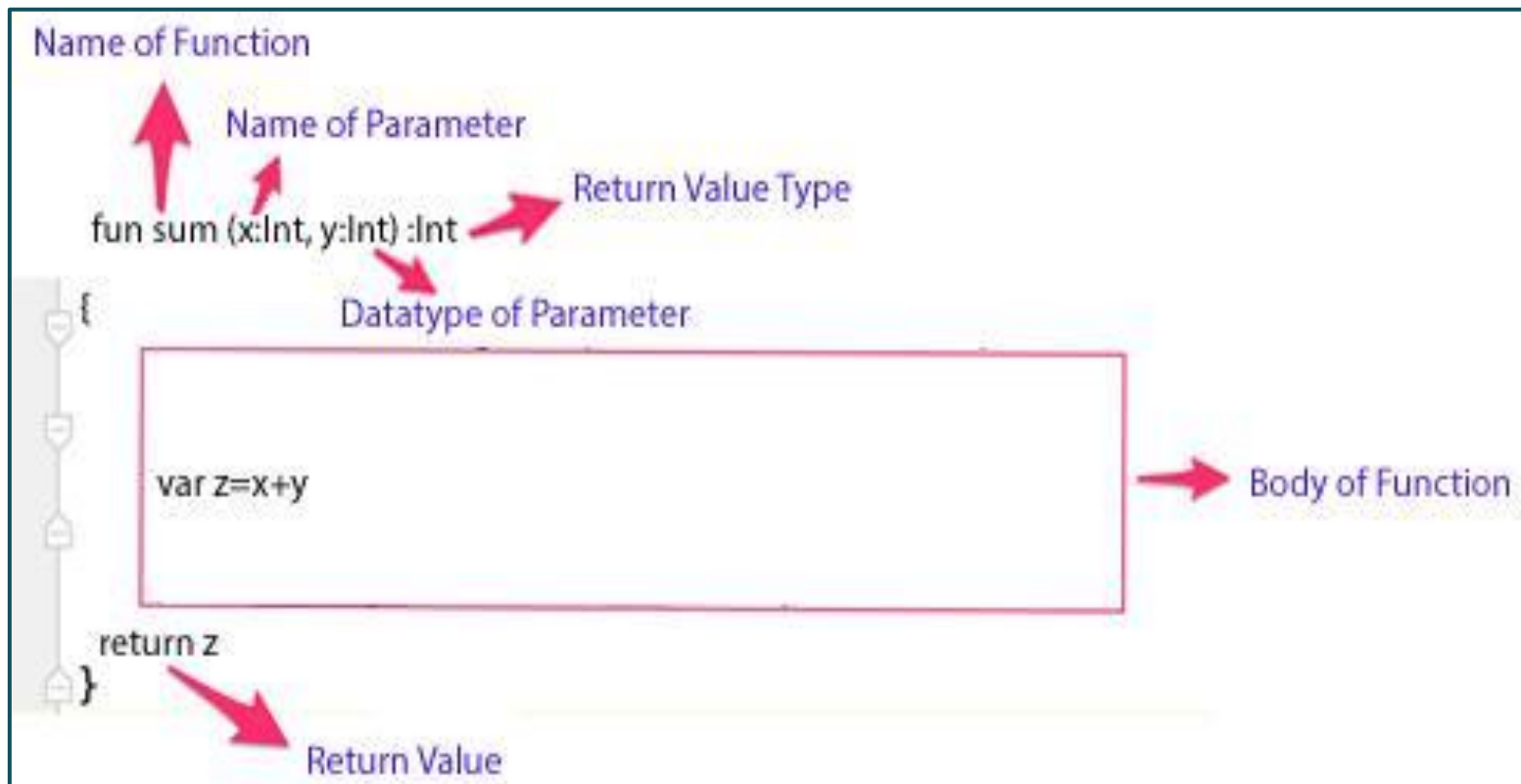
```
FuncoesTest.kt x
3 ▶ fun main(args: Array<String>) {
4
5     println("Resultado da Soma = ${sum( x: 10, y: 30)}")
6 }
7
8 private fun sum(x: Int, y: Int) : Int { 2
9
10    var z: Int = x + y
11    return z 1
12 }
```

Run: FuncoesTestKt x

```
▶ ↑ /snap/android-studio/94/android-studio/jre/bin/java ...
■ ↓ Resultado da Soma = 40
📷 🔄
⚙️ 📄
🔍 🖨️
```

Process finished with exit code 0

Funções



Escopo das Variáveis

A localização das variáveis - seja dentro ou fora da função principal - tem um efeito importante na saída da função.

O exemplo a seguir mostra como o fluxo do programa será afetado se alterarmos a localização das variáveis dentro ou fora das funções.

```
EscopoTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 var minhaFrase = "MIT Mobile" 1
4
5 fun main(args: Array<String>) {
6
7     nomear( minhaFrase: "Infnet")
8 }
9
10 fun nomear(minhaFrase : String) { 2
11
12     println("Minha Frase é $minhaFrase")
13 }
```

Escopo das Variáveis

```
EscopoTest.kt x
1 package br.edu.infnet.mit_android_a2.ex01
2
3 var minhaFrase = "MIT Mobile"
4
5 ▶ fun main(args: Array<String>) {
6
7     nomear( minhaFrase: "Infnet")
8 }
9
10 fun nomear(minhaFrase : String) {
11
12     println("Minha Frase é $minhaFrase")
13 }

Run: EscopoTestKt x
▶ /snap/android-studio/94/android-studio/jre/b
■ Minha Frase é Infnet
📷
⚙️ Process finished with exit code 0
```

Kotlin: Orientação a Objetos

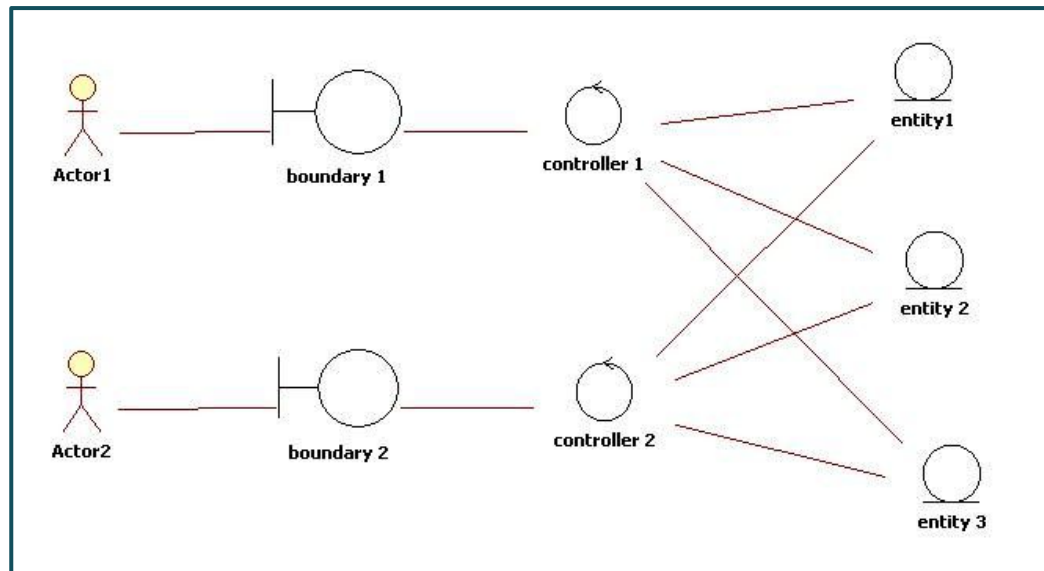
Classes

- Conceito de Classe:
 - Classe é a descrição de uma entidade existente no domínio do problema.
 - É uma “fábrica de objetos”.
 - Define a forma e a funcionalidade de objetos.
- Responsabilidades da Classe:
 - É o que a classe “sabe” e o que ela “faz”.
 - O que a classe “sabe” são as **propriedades** ou seus atributos.
 - O que a classe “faz” são os seus **métodos** ou funções.



Classes

- Classes de Entidade:
 - Cliente,
 - Pedido,
 - Item de Pedido,
 - Produto.
- Classes de Fronteira:
 - Botões,
 - Checkboxes,
 - Listas.
- Classes de Controle:
 - Data,
 - Conexão com BD,
 - Gerenciador de Impressão,
 - Leitura e Gravação de Arquivos.



Declaração e Uso de Classes

```
ContaCorrente.kt x TestaConta.kt x
3  class ContaCorrente(num: Int, tit: String, valor: Double) { 1
4
5      var numero = num
6      var titular = tit 2
7      var saldo = valor
8
9      fun depositar(valor: Float) {
10         saldo += valor
11     }
12
13     fun sacar(valor: Float) {
14         saldo -= valor
15     }
16
17     fun consultaSaldo(): String {
18
19         return "O saldo da conta $numero eh $saldo e o titular eh $titular"
20     }
21 }
22 }
```

Declaração e Uso de Classes



The screenshot displays the Android Studio interface. At the top, two tabs are open: 'ContaCorrente.kt' and 'TestaConta.kt'. The 'TestaConta.kt' tab is active, showing the following Kotlin code:

```
1 package br.edu.infnet.mit_android_a3
2
3 fun main(args: Array<String>) {
4
5     var cc = ContaCorrente( num: 123, tit: "Machado de Assis", valor: 1234.56)
6     println(cc.consultaSaldo())
7 }
```

Below the code editor, the 'Run' tab is active, showing the execution output for 'TestaContaKt':

```
/snap/android-studio/94/android-studio/jre/bin/java ...
0 saldo da conta 123 eh 1234.56 e o titular eh Machado de Assis

Process finished with exit code 0
```

On the left side of the IDE, a vertical toolbar contains icons for running, debugging, and other development actions.

Propriedades de Classes

No Kotlin, temos três tipos de variáveis: pública, privada e protegida.

public → Todas as variáveis são públicas por padrão (implicitamente públicas) e são acessíveis a partir de todas as classes.

private → Essas variáveis são acessíveis apenas dentro de sua própria classe.

protected → Essas variáveis são uma versão das variáveis públicas restritas apenas às subclasses. Isso significa que apenas a classe atual e suas subclasses terão acesso ao campo ou método da classe protegida.

Construtores

Forma 1

```
ContaCorrente.kt x TestaConta.kt x
3 class ContaCorrente(num: Int, tit: String, valor: Double) {
4
5     var numero = num
6     var titular = tit
7     var saldo = valor
8
9     fun depositar(valor: Float) {
10
11         saldo += valor
12     }
13
14     fun sacar(valor: Float) {
15         saldo -= valor
16     }
17
18     fun consultaSaldo(): String {
19
20         return "O saldo da conta $numero eh $saldo e o titular eh $titular"
21     }
22 }
```

Construtores

Forma 2

```
ContaCorrente.kt x TestaConta.kt x
3 class ContaCorrente(var numero: Int, var titular: String, var saldo: Double) {
4
5     fun depositar(valor: Float) {
6
7         saldo += valor
8     }
9
10    fun sacar(valor: Float) {
11        saldo -= valor
12    }
13
14    fun consultaSaldo(): String {
15
16        return "O saldo da conta $numero eh $saldo e o titular eh $titular"
17    }
18 }
```

Construtores

Forma 3

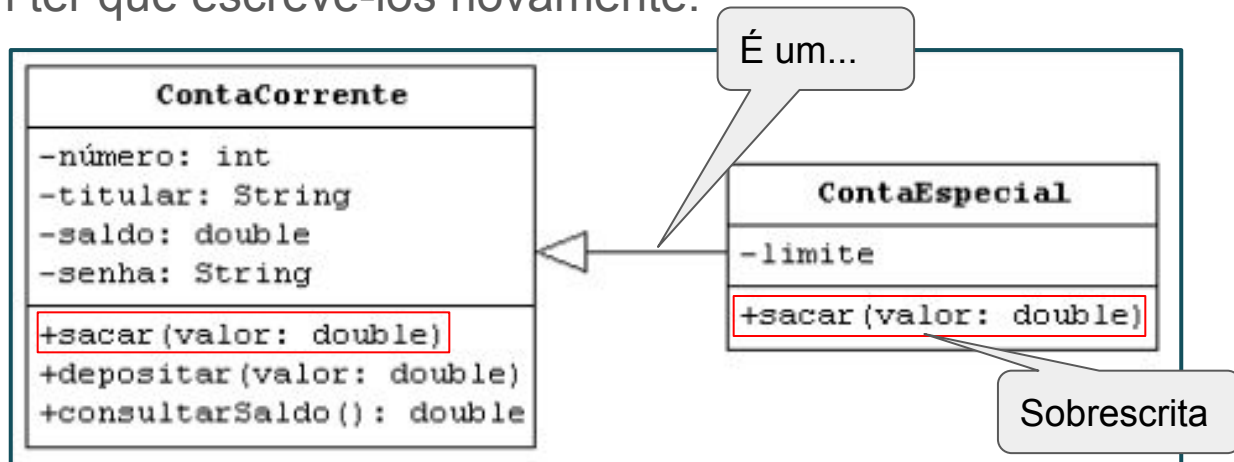
```
ContaCorrente.kt x TestaConta.kt x
3 class ContaCorrente {
4
5     var numero: Int
6     var titular: String
7     var saldo: Double
8
9     constructor(numero: Int, titular: String, saldo: Double) {
10
11         this.numero = numero
12         this.titular = titular
13         this.saldo = saldo
14     }
15
16     fun depositar(valor: Float) {
17
18         saldo += valor
19     }
20
21     fun sacar(valor: Float) {
```

1

Conceito de Herança

A ideia de herança é simples, mas poderosa. Quando você deseja criar uma nova classe e já existe uma classe que inclui alguns dos códigos de que você precisa, você pode derivar sua nova classe da classe existente (chamada superclasse).

Enquanto faz isso, você pode reutilizar as propriedades e métodos da classe existente sem ter que escrevê-los novamente.



Conceito de Herança

```
ContaCorrente.kt x ContaEspecial.kt x TestaConta.kt x
1 open class ContaCorrente(var numero: Int, var titular: String, var saldo: Double) {
4
5     fun depositar(valor: Double) {
6
7         if(valor > 0) {
8
9             saldo += valor
10        }
11    }
12
13    2 open fun sacar(valor: Double) {
14
15        if(valor > 0 && valor <= saldo) {
16
17            saldo -= valor
18        }
19    }
```

Conceito de Herança

```
ContaCorrente.kt x ContaEspecial.kt x TestaConta.kt x
3 class ContaEspecial(numero: Int, titular: String, saldo: Double, var limite: Double)
4 1 : ContaCorrente(numero, titular, saldo) {
5
6  override fun sacar(valor: Double) {
7
8  2   if(valor > 0 && valor <= saldo + limite)
9     saldo -= valor
10 }
11
12 override fun consultaSaldo(): String {
13 3   return super.consultaSaldo() + " com um limite de $limite"
14 }
15 }
```

Polimorfismo

- Polimorfismo é a capacidade de decidir, em tempo de execução, qual método deve ser chamado.
- Para que o polimorfismo seja possível, são necessários:
 - Uma hierarquia de classes onde existam métodos sobrescritos = Herança.
 - A possibilidade de se criar um objeto de uma subclasse e colocá-lo em uma referência de uma superclasse.
- Facilita o desenvolvimento de sistemas pois evita a necessidade de se conhecer, em tempo de compilação, qual objeto deve ser chamado.

Polimorfismo

```
TestaConta.kt x
3  fun main(args: Array<String>) {
4
5      1 var contas : Array<ContaCorrente> = arrayOf(
6          ContaCorrente( numero: 123, titular: "Machado de Assis", saldo: 1234.56),
7          ContaEspecial( numero: 456, titular: "Rachel de Queiroz", saldo: 2345.67, limite: 1000.0))
8      for(conta : ContaCorrente in contas) {
9
10         println(conta.consultaSaldo()) 2
11     }
12 }
```

Run: TestaContaKt x

```

> /snap/android-studio/94/android-studio/jre/bin/java ...
0 saldo da conta 123  eh 1234.56 e o titular eh Machado de Assis
0 saldo da conta 456  eh 2345.67 e o titular eh Rachel de Queiroz com um limite de 1000.0
```

Classe Abstrata

Uma classe é chamada de abstrata quando pelo menos um de seus membros é abstrato. É responsabilidade da classe derivada / filha fornecer a implementação dos membros abstratos da classe pai.

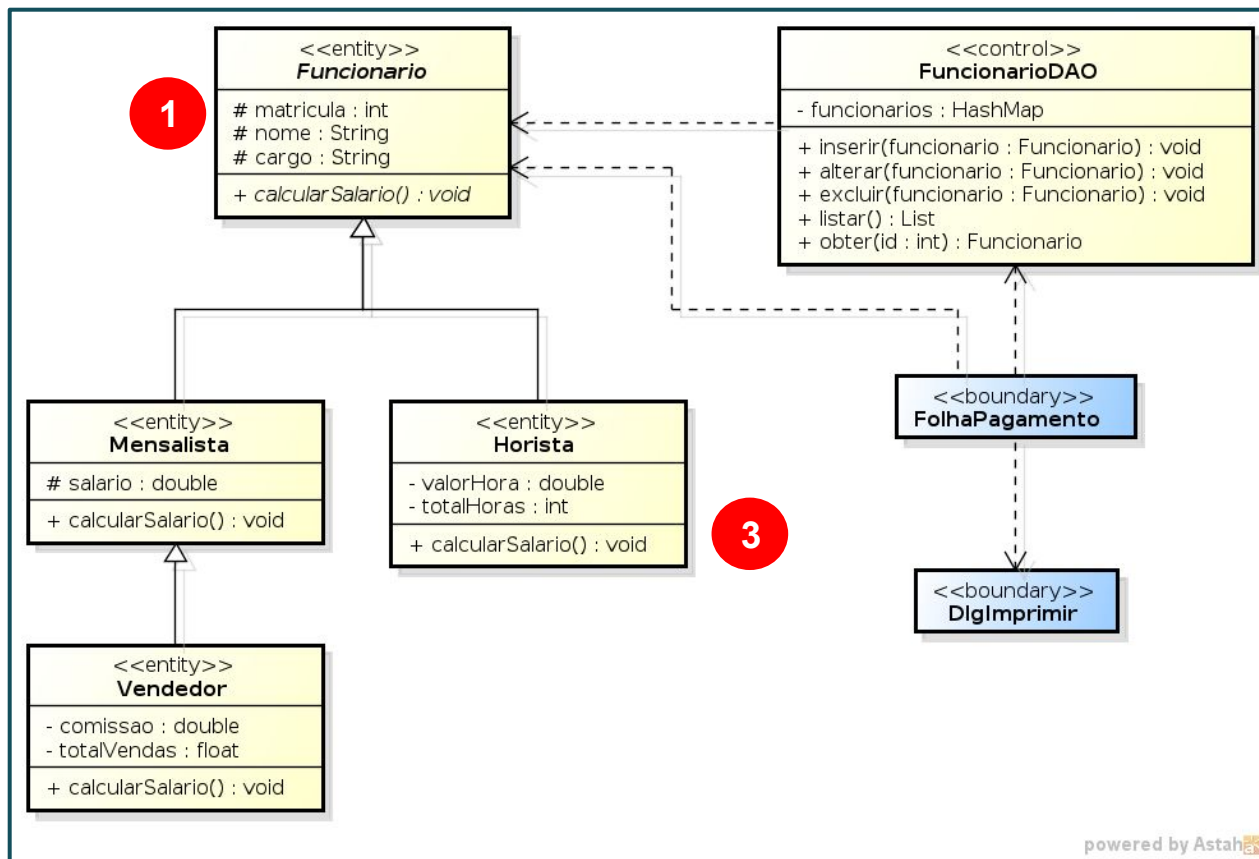
Uma classe abstrata precisa ser qualificada com a palavra-chave `abstract`.

Uma classe abstrata não pode ser instanciada e seu único propósito é ser herdada.

Não precisamos anotar uma classe ou função abstrata com a palavra-chave `open`.

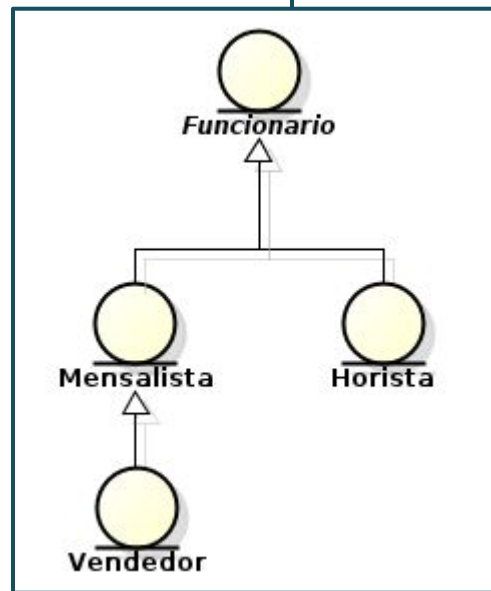


Classe Abstrata



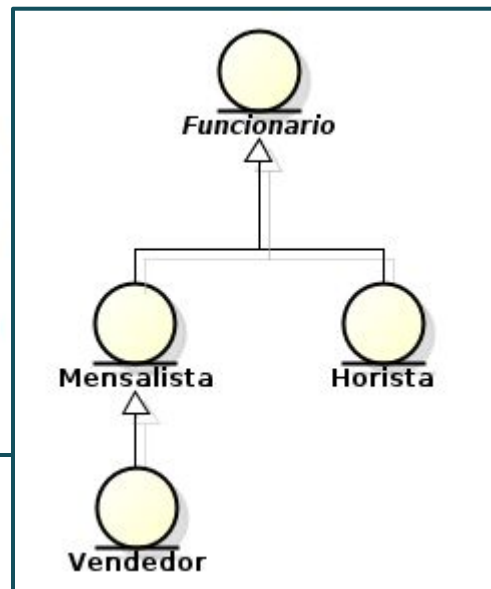
Classe Abstrata

```
Funcionario.kt x Mensalista.kt x
1 abstract class Funcionario(var matricula: Int, var nome: String, var cargo: String) {
2
3     protected var formatador = NumberFormat.getCurrencyInstance()
4
5     2 abstract fun calcularSalario()
6
7     protected fun mostrarCabecalho() {
8
9         println("-----")
10        println("    Infnet - Contracheque")
11        println("-----")
12        println("Matricula....." + matricula)
13        println("Nome....." + nome)
14        println("Cargo....." + cargo)
15    }
16
17    protected fun calcularImposto(valor: Double): Double {
18
19        var imposto = 0.0
20        if (valor > 1000 && valor <= 3000) {
21            imposto = valor * 0.05
22        }
23    }
24 }
```



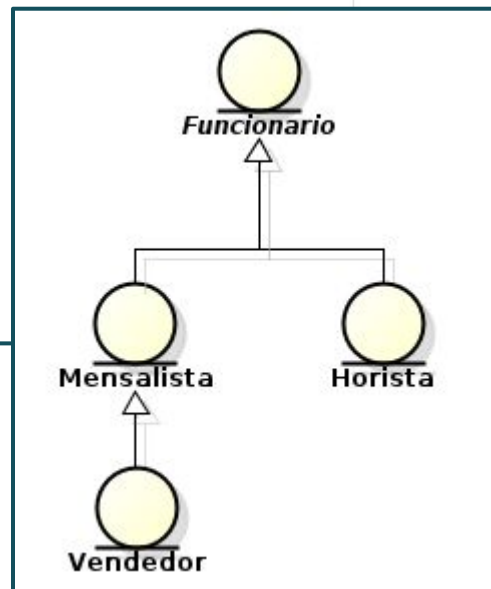
Classe Abstrata

```
Mensalista.kt x
3 1 open class Mensalista(matricula: Int, nome: String, cargo: String, var salario: Double)
4      : Funcionario(matricula, nome, cargo) {
5
6 2  override fun calcularSalario() {
7
8      mostrarCabecalho()
9      println("Sal. Bruto...." + formatador.format(salario))
10     val imposto:Double = calcularImposto(salario)
11     println("Imposto....." + formatador.format(imposto))
12     val liquido:Double = salario - imposto
13     println("Sal. Liquido.." + formatador.format(liquido))
14 }
15 }
```



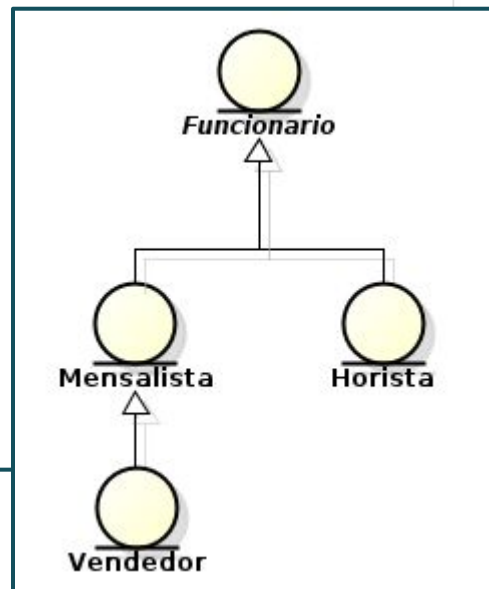
Classe Abstrata

```
Vendedor.kt x
1 class Vendedor(matricula: Int, nome: String, salario: Double, var comissao: Double, var totalVendas: Double)
4 : Mensalista(matricula, nome, cargo: "Vendedor", salario) {
5
6 2 override fun calcularSalario() {
7
8     mostrarCabecalho()
9     val valorComissao: Double = comissao * totalVendas
10    println("Comissao....." + formatador.format(valorComissao))
11    val imposto: Double = calcularImposto( valor: salario + valorComissao)
12    println("Imposto....." + formatador.format(imposto))
13    val liquido: Double = salario + valorComissao - imposto
14    println("Sal. Liquido.." + formatador.format(liquido))
15 }
16 }
```



Classe Abstrata

```
Horista.kt x
1 class Horista(matricula: Int, nome: String, cargo: String, var valorHora: Double, var totalHoras: Int)
4 : Funcionario(matricula, nome, cargo) {
5
6 2 override fun calcularSalario() {
7
8     mostrarCabecalho()
9     println("Valor Hora...." + formatador.format(valorHora))
10    println("Total Horas..." + totalHoras)
11    val bruto:Double = valorHora * totalHoras
12    println("Sal. Bruto...." + formatador.format(bruto))
13    val imposto:Double = calcularImposto(bruto)
14    println("Imposto....." + formatador.format(imposto))
15    val liquido:Double = bruto - imposto
16    println("Sal. Liquido.." + formatador.format(liquido))
17 }
18 }
```



TestaFolha.kt

```

3 fun main(args: Array<String>) {
4
5     val funcs :Array<Funcionario> = arrayOf<Funcionario>(
6         Mensalista( matricula: 123, nome: "Machado de Assis", cargo: "Jornalista", salario: 2345.67),
7         Vendedor( matricula: 456, nome: "Rachel de Queros", salario: 1234.56, comissao: 0.05, totalVendas: 30000.0),
8         Horista( matricula: 678, nome: "Clarice Lispector", cargo: "Professor", valorHora: 70.0, totalHoras: 160))
9
10    for (funcionario :Funcionario in funcs) {
11
12        funcionario.calcularSalario()
13    }

```

Run: TestaFolhaKt

```

-----
Infnet - Contracheque
-----
Matricula.....123
Nome.....Machado de Assis
Cargo.....Jornalista
Sal. Bruto....R$ 2.345,67
Imposto.....R$ 117,28
Sal. Liquido..R$ 2.228,39
-----

```



Interface

Uma interface é um tipo de classe que consiste em métodos ou funções sem nenhuma implementação.

Você usa uma interface quando precisa associar duas classes com baixo acoplamento entre elas. Além disso, a interface garante a implementação de métodos.

A interface se torna superclasse e a classe que a implementa herda o seu tipo.



Interface

```
MainActivity.kt x activity_main.xml x
17 class MainActivity : AppCompatActivity(), View.OnClickListener,
18     DialogInterface.OnClickListener, DatePickerDialog.OnDateSetListener {
19
20     override fun onCreate(savedInstanceState: Bundle?) {...}
37
38     override fun onClick(view: View?) {...}
51
52     private fun alertDialogTest() {...}
62
63     override fun onClick(dialog: DialogInterface?, indiceBotao: Int) {...}
69
70     private fun progressBarTest() {...}
87
88     private fun datePickerTest() {...}
97
98     override fun onDateSet(datePicker: DatePicker?, ano: Int, mes: Int, dia: Int) {...}
102
103     private fun snackBarTest() {...}
111 }
```


Enum Class

Uma classe enum é um tipo de dados especial que permite que uma variável seja um conjunto de constantes predefinidas.

A variável deve ser igual a um dos valores predefinidos.

```
Pag_3_30.kt x
1      package exemplo13
2
3      enum class Colleges {
4
5          ITCollege,
6          BusinessCollege,
7          ArtsCollege,
8          EngineeringCollege
9      }
10
11  fun main(args: Array<String>) {
12      var major = Colleges.EngineeringCollege
13
14      println(major)
15  }
```



Exercício

Implementar a hierarquia de classes Funcionario, Mensalista, Vendedor, Horista e um código de testes que “rode a folha de pagamento”. Use os exemplos de código anteriores e os que estão abaixo.

```
protected fun mostrarCabecalho() {  
  
    println("-----")  
    println("    Infnet - Contracheque")  
    println("-----")  
    println("Matricula....." + matricula)  
    println("Nome....." + nome)  
    println("Cargo....." + cargo)  
}
```

```
protected fun calcularImposto(valor: Double): Double {  
  
    var imposto = 0.0  
    if (valor > 1000 && valor <= 3000) {  
        imposto = valor * 0.05  
    } else if (valor > 3000 && valor <= 5000) {  
        imposto = valor * 0.2  
    } else if (valor > 5000) {  
        imposto = valor * 0.27  
    }  
    return imposto  
}
```


Laboratório

Flag Quiz



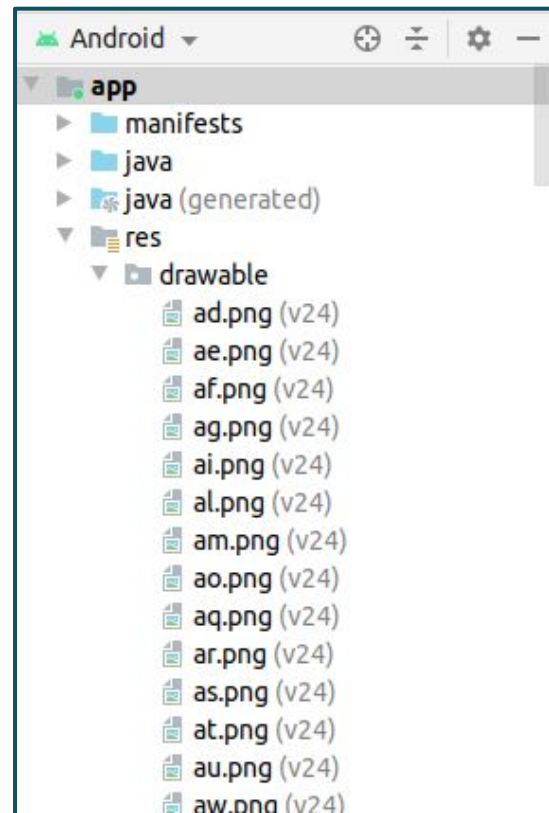
Flag Quiz

Na pasta **res** vamos carregar os arquivos de imagem referentes a cada um dos países, identificados pelo padrão **ISO 639** de duas letras.

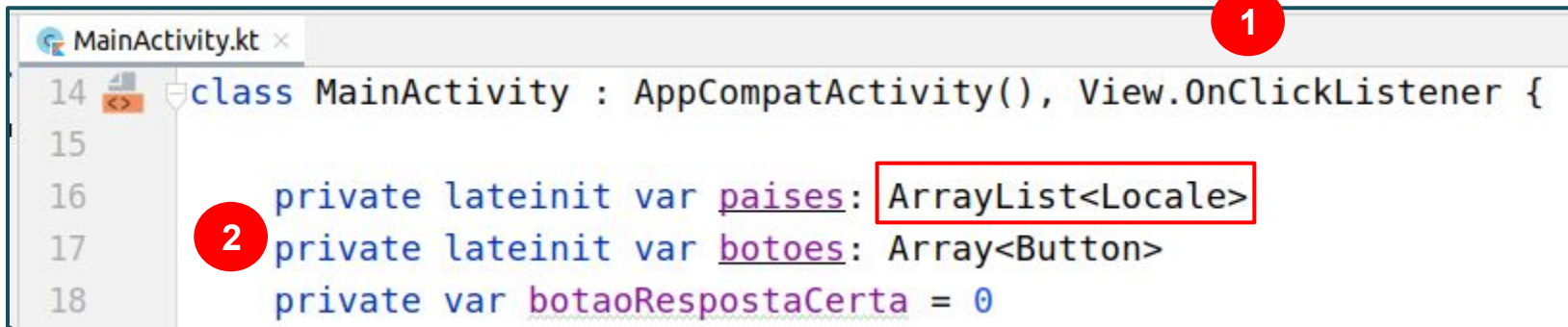
No Android temos a classe **Locale** que representa esses países e tem a identificação de duas letras.

O objetivo é poder gerar as perguntas de forma aleatória e carregar uma `ImageView` com essas imagens, dinamicamente.

Além disso, precisamos dos nomes dos países para usar nos botões com as possíveis respostas.



Flag Quiz



The screenshot shows a code editor window for MainActivity.kt. The code is as follows:

```
14 class MainActivity : AppCompatActivity(), View.OnClickListener {  
15  
16     private lateinit var países: ArrayList<Locale>  
17     private lateinit var botoes: Array<Button>  
18     private var botaoRespostaCerta = 0
```

Annotation 1 is a red circle with the number 1, pointing to the class declaration line 14.

Annotation 2 is a red circle with the number 2, pointing to the variable declaration line 16.

Flag Quiz

```
MainActivity.kt
20  override fun onCreate(savedInstanceState: Bundle?) {
21      super.onCreate(savedInstanceState)
22      setContentView(R.layout.activity_main)
23      //Carrega os países disponíveis
24      for (locale in Locale.getAvailableLocales()) {
25
26          1      if (locale.country.length == 2) {
27
28              paises.add(locale)
29          }
30      }
31      //Monta um array com os botões para facilitar trabalhar com o grupo
32      botoes = arrayOf(this.findViewById<Button>(R.id.btn0), this.findViewById<Button>(R.id.btn1),
33                      this.findViewById<Button>(R.id.btn2))
34      for (i in 0..2) {
35
36          2      botoes[i].setText("")
37              botoes[i].setOnClickListener(this)
38          }
39      this.sorteiaPais()
40  }
```

```
42 private fun sorteiaPais() {
43
44     1 var rodada = IntArray( size: 3)
45     //Sorteio de 3 países diferentes
46     for (i in 0..2) {
47         var numero = 0
48         var ok = false
49         while (!ok) {
50             numero = (Math.random() * paises.size - 1).roundToInt() 2
51             ok = true
52             for (j in 0..2) {
53                 if (rodada[j] == numero) {
54
55                     ok = false 3
56                     break;
57                 }
58             }
59             if (ok) {
60                 val uri = "@drawable/" + paises.get(numero).country.toLowerCase()
61                 4 val resourceId = this.resources.getIdentifier(uri, defType: "drawable", packageName)
62                 if (resourceId == 0) {
63                     ok = false
64                 }
65             }
66         }
67         rodada[i] = numero
68     }
```

Flag Quiz

```
MainActivity.kt x
69 //Sorteia que botão será o certo desta rodada
70 1 botaoRespostaCerta = (Math.random() * 2).roundToInt()
71 //Carrega a imagem da bandeira
72 val uri = "@drawable/" + paises.get(rodada[botaoRespostaCerta]).country.toLowerCase()
73 2 val imageResource = this.resources.getIdentifier(uri, defType: null, packageName)
74 var imgBandeira = this.findViewById<ImageView>(R.id.imgBandeira)
75 imgBandeira.setImageDrawable(resources.getDrawable(imageResource))
76 //Carrega os rótulos dos botões com os nomes dos países
77 3 for (i in 0..2) {
78     botoes[i].setText(paises[rodada[i]].displayCountry)
79 }
80
81 }
```


Flag Quiz

```
MainActivity.kt x
83  override fun onClick(view: View?) {
84
85      val button : Button = view as Button
86      val lblResposta = this.findViewById<TextView>(R.id.lblResposta)
87      if (botoes[botaoRespostaCerta] == button) {
88
89          lblResposta.setTextColor(Color.GREEN)
90          lblResposta.setText(button.text.toString() + " - CORRETO!")
91      } else {
92
93          lblResposta.setTextColor(Color.RED)
94          lblResposta.setText(button.text.toString() + " - ERRADO!")
95      }
96      this.sorteiaPais()
97  }
```




Exercício

Implementar a aplicação Flag Quiz apresentada nesta aula.

Solucionar os problemas de interface, como por exemplo o contraste de tela.