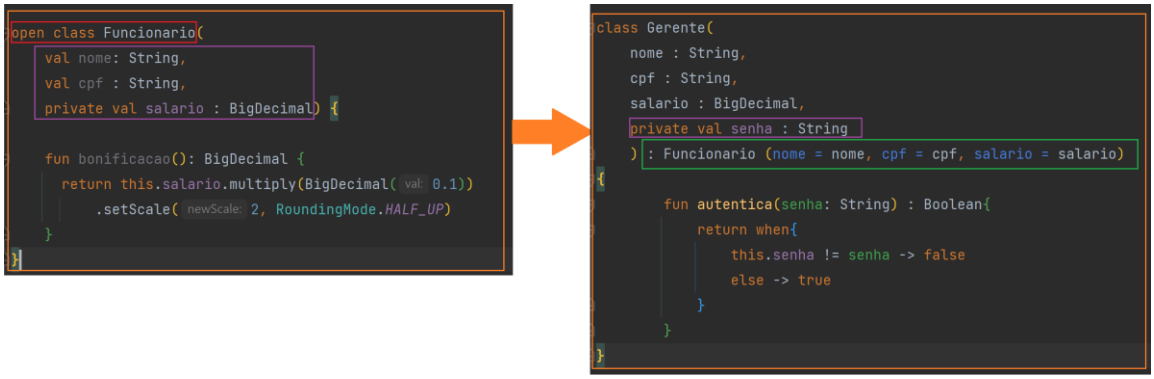
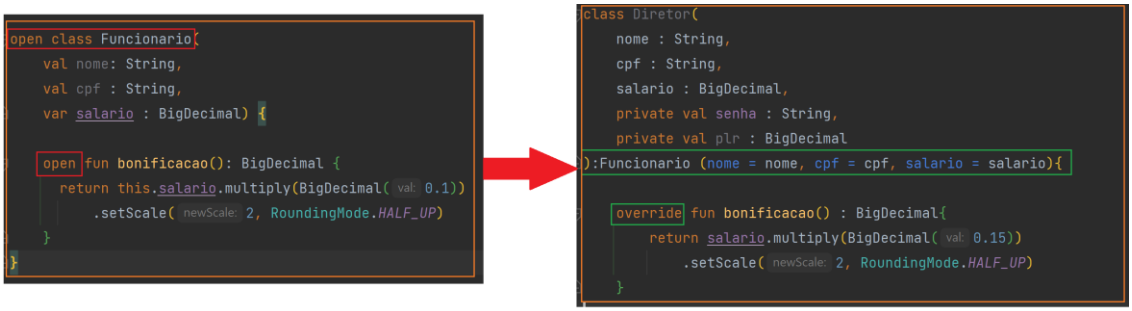
A herança em **Kotlin** é passada pela palavra chave **": Classe pai".** Além de que a **classe** que for ser herdada **PRECISA EXPLICITAMENTE** estar ciente que ela vai poder ser herdada, fazemos isso com a palavra chave **"open".**

Todas as propriedades da classe pai **precisam ser atribuídas** na **classe filho**:



Polimorfismo em **Kotlin** também precisa ser explicitamente especificado, então **métodos** que **podem ser sobrescritos** utilizam a **palavra chave "override"** enquanto na **classe pai é necessário usar a palavra reservada "open":**



Um dos **recursos específicos** de **Kotlin** nos permite **transformar pequenas funções que apenas devolvem um valor** em uma **propriedade** um pouco diferenciada.

Lembrando que **podemos definir o comportamento get()** de uma **propriedade** explicitamente, então toda vez que eu chamar essa propriedade diferenciada, o ***get*** dela vai se **comportar como sendo uma expressão.**

Por exemplo, a **função de bonificação devolve um valor baseado em uma porcentagem do** salário, se **transformarmos a função** em uma **propriedade**, o ***get()*** **sempre fará esse cálculo ao devolver essa propriedade:**

```
open class Funcionario(
    val nome: String,
    val cpf : String,
    var salario : BigDecimal

){
    open val bonificacao: BigDecimal get() = this.salario.multiply(BigDecimal( val: 0.1))
            .setScale( newScale: 2, RoundingMode.HALF_UP)
}
```

```
class Diretor(
    nome : String,
    cpf : String,
    salario : BigDecimal,
    private val senha : String,
    private val plr : BigDecimal
):Funcionario (nome = nome, cpf = cpf, salario = salario){

    override val bonificacao: BigDecimal
        get() {
            return salario.multiply(BigDecimal( val: 0.15))
                .setScale( newScale: 2, RoundingMode.HALF_UP)
        }
```

Na r**eutilização de código podemos usar a palavra chave "super"** para a**proveitar uma implementação já feita**. Por exemplo, todos os **funcionários irão receber uma bonificação base fixa de 10% do salário** e essa **implementação** existe na **classe funcionário**:

```
open class Funcionario(
    val nome: String,
    val cpf : String,
    var salario : BigDecimal

){
    open val bonificacao: BigDecimal get() = this.salario.multiply(BigDecimal( val: 0.1))
            .setScale( newScale: 2, RoundingMode.HALF_UP)
}
```

Agora todos os **herdeiros** de funcionário chamarão a bonificação padrão e adicionarão suas próprias regras de bonificação, assim **paramos de repetir código usando uma implementação já existente**:

```
class Gerente(
    nome : String,
    cpf : String,
    salario : BigDecimal,
    private val senha : String
):Funcionario (nome = nome, cpf = cpf, salario = salario){

    override val bonificacao: BigDecimal get() = super.bonificacao.plus(salario)
        .setScale( newScale: 2,RoundingMode.HALF_UP)
```

```
class Diretor(
    nome : String,
    cpf : String,
    salario : BigDecimal,
    private val senha : String,
    private val plr : BigDecimal
):Funcionario (nome = nome, cpf = cpf, salario = salario){

    override val bonificacao: BigDecimal
        get() {
            return super.bonificacao.plus(plr).plus(salario).setScale( newScale: 2, RoundingMode.HALF_UP)
        }
```

Também **dá pra tirar proveito do Polimorfismo em métodos mais abrangentes** através da **classe pai**.

Por exemplo, se eu tiver uma **calculadora de bonificação** dos **funcionários** pra calcular **o total das bonificações** sabendo que cada **filho (funcionário especifico) de funcionário** tem sua maneira de bonificar eu poderia simplesmente receber **Funcionário** como **parâmetro** dessa **função:**

```
class CalculadoraBonificacao {

    var total: BigDecimal = BigDecimal( val: 0.0)
        private set

    fun registra(funcionario: Funcionario){
        this.total += funcionario.bonificacao
    }
}
```

O ponto é que podemos usar **instâncias diferentes** para a mesma **referência:**

```
fun main(args: Array<String>) {

    val gerente = Gerente( nome: "Lauro", cpf: "88888", BigDecimal( val: 1000), senha: "laulo")
    val diretor = Diretor( nome: "Sevika", cpf: "66666", BigDecimal( val: 1000), senha: "lol",BigDecimal( val: 0.02))
    val funcionario = Funcionario( nome: "Vander", cpf: "77777", BigDecimal( val: 1000))

    val calculadora = CalculadoraBonificacao()

    calculadora.registra(gerente)
    calculadora.registra(diretor)
    calculadora.registra(funcionario)

    println(calculadora.total)
}
"C:\Program Files\Java\jdk-15\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition
2300.02
```