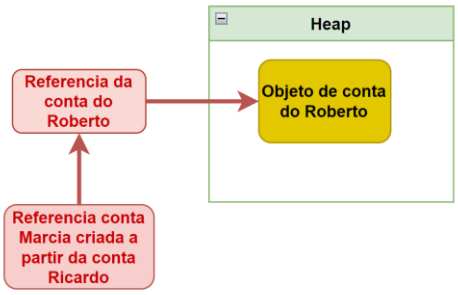
Em **Kotlin** os objetos funcionam bem semelhante a Java. Com relação a referência e cópia, **referência** é um **espaço reservado na memória** para um **Objeto,** se você tiver duas **referências *(váriaveis)*** a um mesmo **objeto instanciado** vai dar ruim, porque **não** são **objetos diferentes** e **sim duas referências** apontadas para o mesmo **objeto** em memória.

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

    val contaRoberto = Conta();
    val contaMarcia = contaRoberto;
}
```




**Atributos/Propriedades** e **Getters** e **Setters**

**Kotlin** tem como um dos principais pilares evitar *código boilerplate*, isso me lembra getters e setters dos **objetos** de Java. Obviamente **Kotlin** implementa getters e setters, só que ele faz isso no seu próprio idioma.

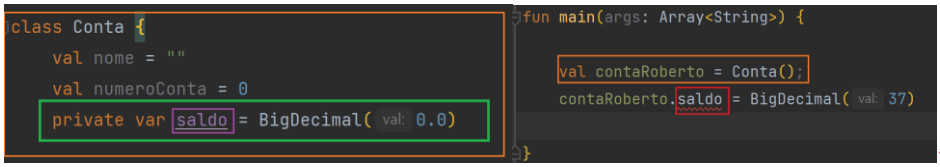
As **variáveis** de uma classe **Kotlin** são sempre acessadas por **getter** e **setters** nativamente ***(por baixo dos panos eles sempre são utilizados),*** na verdade, ***variáveis/atributos*** de classes **Kotlin** não são atributos/variáveis eles são **PROPRIEDADES** que por padrão já são encapsulados em **getters e setters:**

*****implementação por baixo dos panos ao acessar e modificar propriedades de classes Kotlin*****

```
class Conta {
    val nome = ""
    val numeroConta = 0
    private var saldo = BigDecimal( val: 0.0)

    fun deposita(valor: BigDecimal){
        this.saldo += valor
    }

    fun saca(valor: BigDecimal){
        when{
            saldo.compareTo(valor) == -1 -> println("Saldo insuficiente para saque")
            else -> this.saldo -= valor
        }
    }
}
```

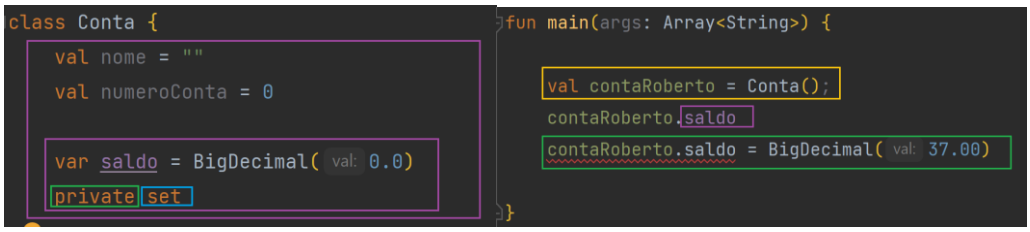
```
class Conta {
    val nome = ""
    val numeroConta = 0

    private var saldo = BigDecimal( val: 0.0)
        set(valor){ field = valor }
        get(){return field}
```

Então quando eu **privo** uma **propriedade** em uma classe **Kotlin** eu garanto **QUE SOMENTE MINHA CLASSE(que contém a propriedade obviamente)** consegue manipular aquela **propriedade:**

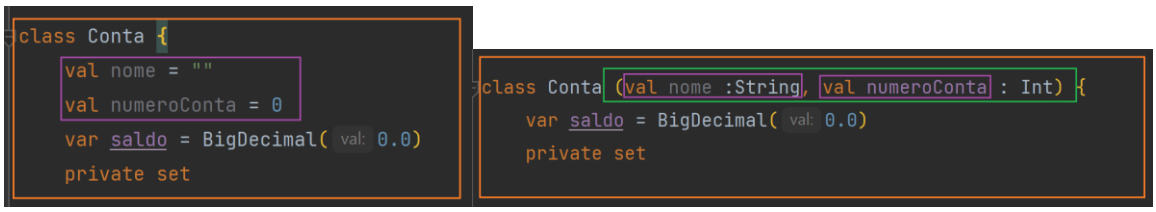



***é uma boa ideia pra falar a verdade, é um encapsulamento REALMENTE encapsulado, assim seriamos obrigado a alterar a propriedade somente dentro da própria classe, caso essa seja a necessidade****

Mas caso eu queira que a **visibilidade** da **Propriedade** seja **possível para classes externas**, porém ainda **sendo alterado somente pela própria classe dela**, eu posso colocar meus **modificadores de acesso** apenas onde eu **preciso**, nesse caso, no **setter**:




**Construtores** no **Kotlin** seguem aquela **ideia padrão de que o objeto deve ser inicializado já com os valores que o construtor dele exige**. Porém o **Kotlin** traz a ideia de um **construtor primário**, esse que fica declarado junto com o nome da classe, esse **construtor primário JÁ ENTENDE** que as propriedades que **ele** exige fazem parte da classe:




Caso a classe precise de um **construtor** que **faz mais funções do que só inicializar as propriedades**, você poderia **implementa**r na classe o **"constructor()"** ele é bem semelhante como funciona um em Java. Esse **constructor** declarado na classe é chamado de ***construtor secundário.***

Se eu quisesse que uma **propriedade** fosse ***"opcional"*** na classe **eu poderia inicializar ela** no **construtor**, dessa maneira eu **poderia criar o objeto** sem a necessidade de um ***valor ser passado***, mas se eu quisesse ***passar um valor*** então o **objeto** iria atribuir ***ele:***