Kotlin – Classes Abstratas e Interfaces

Uma classe abstrata, assim como os seus métodos abstratos, é open por padrão. Ao herdar de uma classe abstrata, todos os métodos abstratos da superclasse devem ser implementados na subclasse, a não ser que a mesma também seja abstrata. Caso contrário, um erro será emitido pelo compilador:

```
abstract class Veiculo(open val valor: Double) {

    abstract fun calculate(fee: Double): Double
}

class Bicicleta(valor: Double) : Veiculo(valor)

Class 'Bicicleta' is not abstract and does not implement abstract
base class member
public abstract fun calculate(fee: Double): Double defined in
Veiculo
```

Em Kotlin também é possível que uma classe abstrata herde de uma classe não abstrata. Quando isso for necessário, os métodos da superclasse não abstratos devem utilizar o modificador open.

Ainda que isso seja muito raro, há casos em que uma classe herda de uma outra que compartilha um membro com alguma interface, implementada pela subclasse. Quando isso acontecer, podemos usar uma forma qualificada de super para acessar a implementação da superclasse. Vejamos um exemplo que deixará isso mais claro:

```
open class Veiculo {
    open fun calculate() { /* Alguma implementação aqui */ }
}

interface Calculable {
    fun calculate() { /* Alguma implementação aqui */ }
}

class Bicicleta : Veiculo(), Calculable {
    override fun calculate() {
      super<Veiculo>.calculate()
      super<Calculable>.calculate()
    }
}
```

Via de regra, sempre que uma classe herdar membros de mesmo nome de supertipos diferentes, ela mesma deve fornecer uma implementação para esse membro. No exemplo acima, a classe `Bicicleta` fornece uma implementação para `calculate`, a qual invoca `calculate()` de Veiculo e `Calculable` a partir de super qualificado.

Por Hoje é só pessoal, sucesso nos estudos.