Instituto Superior de Engenharia de Lisboa

LEIC

**Técnicas de Desenvolvimento de Software**

Teste Final (Época de Recurso), Verão de 2021/2022, 11 de Julho 2022

---

Nome: Número:

---

## Grupo I

Considerando os temas tratados nas aulas da unidade curricular, responda às perguntas seguintes assinalando de forma inequívoca a opção correta. Não responda arbitrariamente: cada resposta incorreta desconta 1/3 da cotação da pergunta ao total obtido no grupo. A cotação total do grupo é 7 valores e todas as perguntas têm a mesma cotação.

Considerando a seguinte declaração da função **`error`** e a declaração incompleta do valor **`value`**

```
fun error(msg: String?): Nothing {
   val value ...
   throw IllegalStateException("$msg: $value")
}
```

1. Qual das seguintes possíveis declarações de **`value`** provocaria erro de compilação

   - ☐ `val value: Int? = msg?.toInt()`
   - ☐ `val value: Any = msg ?: Unit`
   - ☐ `val value: String = msg ?: return`
   - ☐ `val value = msg ?: "null"`

2. Considerando a função **`f1`**, identifique a frase correta.

```
suspend fun f1(scope: CoroutineScope) {
   print(1)
   scope.launch { print(2); delay(50); print(3) }
   delay(100); print(4)
}
```

   - ☐ O modificador **`suspend`** poderia ser omitido sem provocar erro de compilação
   - ☐ O parâmetro **`scope`** é dispensável porque a função **`launch`** pode ser chamada sem receptor (*receiver*)
   - ☐ Com ou sem a chamada **`delay(100)`** a função escreve o mesmo
   - ☐ Com ou sem a chamada **`delay(50)`** a função escreve o mesmo

3. Identifique a frase correta quanto à utilização da herança.

   - ☐ Uma classe deriva obrigatoriamente de outra classe e implementa zero ou muitas interfaces.
   - ☐ Uma interface deriva de uma ou mais interfaces.
   - ☐ Uma classe derivada só pode redefinir (**`override`**) uma propriedade se ela for marcada com **`abstract`** na classe base.
   - ☐ Para efeitos de conversão de tipos, todos os tipos não *nullable* derivam do mesmo tipo *nullable*, mas **`Any`** não deriva de **`Any?`**.

4. Para acrescentar uma nova propriedade a uma classe já definida, podemos definir uma propriedade interna na classe ou uma propriedade extensão dessa classe. Neste contexto, identifique a frase correta.

- ☐ Se a nova propriedade for mutável (**`var`**) tem que ser definida como interna da classe.
- ☐ Tem que ser interna, para a propriedade ser definida também nas classes derivadas.
- ☐ As duas alternativas são equivalentes se a implementação só tiver que aceder às características públicas.
- ☐ Se a classe pertence a uma biblioteca externa só é possível definir como propriedade extensão, mas não haverá polimorfismo no seu acesso.

Considerando as seguintes declarações responda aos pontos 5 e 6.
Nota: Este tipo será usado no Grupo II.

```
class Grade private constructor(val value: Int) {
    companion object {
        const val MIN = 1; const val MAX = 20
        private val scale = (MIN..MAX).map { Grade(it) }
        operator fun invoke(value: Int) = scale[ value - MIN ]
    }
    override fun toString() = "$value values"
}
```

5. Identifique a frase correta

- ☐ Como o construtor de **`Grade`** é privado não é possível fazer: **`val g = Grade(10)`** fora da classe
- ☐ A avaliação da expressão **`Grade(22)`** lança uma exceção do tipo **`IllegalArgumentException`**
- ☐ O prefixo **`override`** na função **`toString`** é desnecessário porque **`Grade`** não é uma classe derivada
- ☐ A avaliação da expressão **`Grade(8).value==8`** dá **`true`**

6. Identifique a frase correta

- ☐ Para usar o operador **==** entre objetos do tipo **`Grade`** é necessário redefinir a função **`equals`**
- ☐ As expressões **`Grade.Companion.invoke(8)`** e **`Grade(8)`** produzem o mesmo resultado
- ☐ Entre dois objetos do tipo Grade os operadores **`===`** e **`==`** podem produzir resultados diferentes
- ☐ Seria equivalente se **`scale`** fosse iniciado com **`List(MAX-MIN+1){Grade(it)}`**

Considerando a seguinte declaração da classe **`Point`** e a sua propriedade **`module`**

```
data class Point(val x:Int, val y: Int) {
    val module ...
}
```

7. Qual das seguintes declarações da propriedade **`module`** está incorreta:

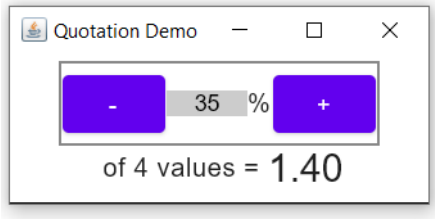
- ☐ `val module by lazy { sqrt(x.toDouble()*x + y*y) }`
- ☐ `val module = sqrt(x.toDouble()*x + y*y).also { return it }`
- ☐ `val module: Double = sqrt(x.toDouble()*x + y*y)`
- ☐ `val module get() = sqrt(x.toDouble()*x + y*y)`

## Grupo II

Pretende-se desenvolver um componente *Composable* para editar a avaliação, em percentagem, de cada pergunta de um teste. Para tal, e com o objetivo de não permitir a representação de estados inválidos (*Make Illegal States Unrepresentable*), foi declarado o tipo **`Eval`** que limita os valores das percentagem de **`MIN`** até **`MAX`** em múltiplos de **`DELTA`** e a única forma de instanciar objetos é utilizando a função **`of`**.

```
class Eval private constructor(val quotation: Int) {
   companion object {
       const val DELTA = 5; const val MIN = 0; const val MAX = 100
       val default = Eval(MIN)
       fun of(q: Int) = if (q in MIN..MAX && q % DELTA == 0) Eval(q) else null
   }
   /* ... equals ... */
   fun inc(): Eval = if (quotation==MAX) this else Eval(quotation+DELTA)
}
```

A figura seguinte apresenta a janela de uma aplicação de demonstração do componente que também usa o tipo **`Grade`** das perguntas 5 e 6 do Grupo I. O componente tem o aspeto apresentado na parte superior da janela (com rebordo) e será implementado com a função composable: **`EvalEdit(eval: Eval, set: (Eval)->Unit)`**
Na parte inferior da janela aparece a cotação parcial de uma pergunta com 4 valores, correspondente à percentagem editada no componente.
A função **`main`** da aplicação de demonstração, assim como a função auxiliar **`format`**, são as seguintes.




```
fun main() = singleWindowApplication(
    title = "Quotation Demo",
    state = WindowState(width= 280.dp, height= 130.dp),
) { MaterialTheme {
     DemoEvalEdit( Grade(4) ) { partial: Double ->
        println("Partial = ${partial.format(2)}")
     }
} }
fun Double.format(digs: Int) = "%.${digs}f".format(this)
```

1. [1] Implemente a função extensão **`Eval.dec():Eval`**, que faz o inverso da função interna **`inc()`**, ou seja, retorna um **`Eval`** com a cotação válida imediatamente inferior ou o próprio **`Eval`**, caso não seja possível.

2. [1] Para permitir que objetos do tipo **`Eval`** sejam comparados corretamente com o operador **==**, implemente a função interna **`equals`**, em falta na classe **`Eval`**, definindo também a sua assinatura completa.

3. [1] Implemente a função extensão **`partialValue`** que retorna o valor parcial de uma pergunta. Por exemplo, a chamada **`Grade(4).partialValue(Eval.of(35)?: Eval.default)`** retorna um **`Double`** com o valor **1.40**.

4. [2] Para facilitar a construção do componente **`EvalEdit`**, implemente o componente com a assinatura: **`SymbolButton(symbol: Char, enabled: Boolean, onClick: ()->Unit)`** para ser usado na criação dos botões **+** e **-**.

5. [4] Implemente o componente *Composable* **`EvalEdit`** já descrito. A função recebe como parâmetros o objeto **`Eval`** atual e a função que será chamada para atualizar para outro valor. Os botões **-** e **+** devem passar para o valor válido anterior e seguinte e cada um deles deve ficar *disable* quando o valor corrente for o máximo ou o mínimo. A zona editável, com fundo cinzento, usa um **`TextField`** com *40dp* de largura e só deve permitir editar valores válidos para **`Eval`**. Note que este componente não mantém o estado do **`Eval`** atual.

6. [4] Implemente o *Composable* **`DemoEvalEdit`**, usado na função **`main`** e que usa o componente **`EvalEdit`**. A função recebe como parâmetros o valor (**`Grade`**) da pergunta e a função que será chamada cada vez que a cotação for alterada. Note que este componente é que deve manter o estado do **`Eval`** atual.

Duração: 90 minutos
ISEL, 11 de Julho de 2022