

Nome:

Número:

Grupo I

Considerando os temas tratados nas aulas da disciplina, 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.

1. Escolha a opção que completa corretamente a frase:

Uma função prefixada com `suspend` ...

- ☐ é executada automaticamente quando um determinado evento ocorre.
- ☐ é executada numa nova corrotina cada vez que é chamada.
- ☐ não bloqueia a execução da *thread* do chamador quando é chamada.
- ☐ só pode chamar funções que também estejam prefixadas com `suspend`.

2. Considerando `value` um valor de um tipo qualquer em Kotlin (e.g. `Int`, `Int?`, etc.), qual das seguintes declarações com iniciação poderá provocar erro de **compilação** para algum tipo de `value`:

- ☐ `val a: Any = value`
- ☐ `val a: Any? = value`
- ☐ `val a: Any = value as Any`
- ☐ `val a: Any? = value as Any?`

3. Dadas as seguintes definições:

```
open class A { open fun f() = "fA" }  
fun A.h() = "hA"
```

```
class B: A() { override fun f() = "fB" }  
fun B.h() = "hB"
```

Qual é a lista criada por: `ListOf(B(),A()).map{ it.f() + it.h() } + B().h() + A().f()`

- ☐ `[fB, hA, hA, fA]`
- ☐ `[fBhB, fAhA, hA, fA]`
- ☐ `[fA, hA, hB, fA]`
- ☐ `[fBhA, fAhA, hB, fA]`

4. Dadas as definições:

```
typealias Oper = (Int,Int) -> Int
fun eval(a: Int, b: Int, op: Oper) { println("Result = ${op(a,b)}") }
```

Qual é a alternativa que **está errada** ao chamar a função `eval`:

<input type="checkbox"/> <code>fun sum(a: Int, b: Int) = a + b</code> <code>eval(3, 10, ::sum)</code>	<input type="checkbox"/> <code>val sum = { a, b -> a + b }</code> <code>eval(3, 10, ::sum)</code>
<input type="checkbox"/> <code>val sum: Oper = { a: Int, b: Int -> a + b }</code> <code>eval(3, 10, sum)</code>	<input type="checkbox"/> <code>eval(3, 10) { a, b -> a + b }</code>

5. Dada a definição do tipo `Direction`, onde falta implementar a função `equals`:

```
class Direction private constructor(val dRow: Int, val dCol: Int) {
    companion object {
        val LEFT = Direction(0,-1); val UP = Direction(-1,0)
        val RIGHT = Direction(0,+1); val DOWN = Direction(+1,0)
        val values = listOf(LEFT, UP, RIGHT, DOWN)
    }
    val ordinal by lazy { values.indexOf(this) }
    override fun equals(other: Any?) ...
}
```

escolha a implementação que completa a função `equals` que esteja correcta e que seja mais eficiente:

- ☐ `= false`
- ☐ `= other is Direction && ordinal == other.ordinal`
- ☐ `= other is Direction && dRow==other.dRow && dCol==other.dCol`
- ☐ `= this === other.`

6. Dada a definição:

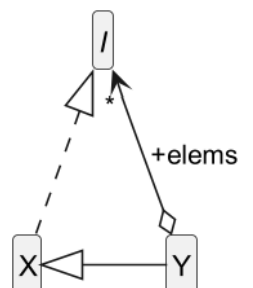
```
fun foo() { "ISEL".forEach{ if (it=='E') return; print(it) }; print("_OK") }
```

escolha a opção com o *output* da execução de: `fun main() { foo() }`

- ☐ `IS_OK`
- ☐ `IS`
- ☐ `ISL_OK`
- ☐ `ISL`

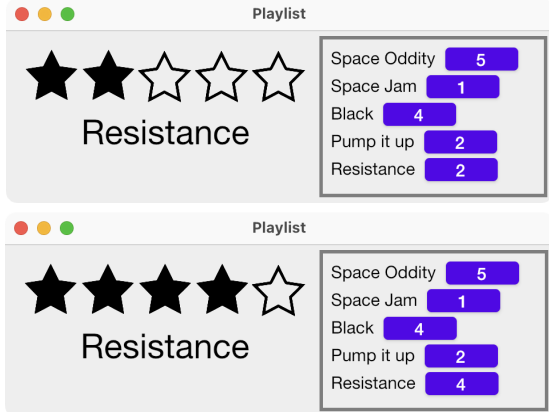
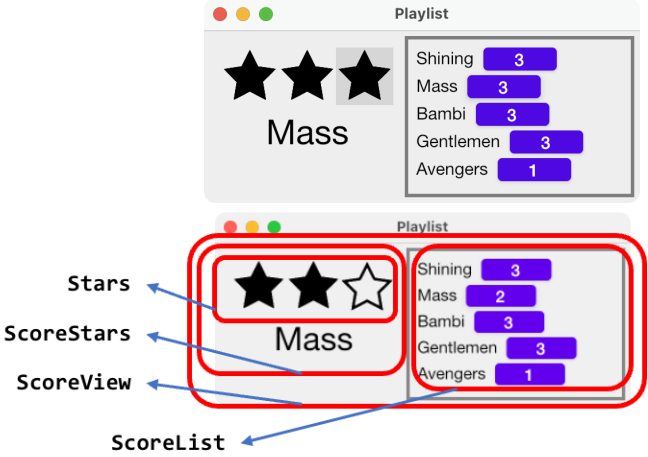
7. Para o diagrama de classes em UML apresentado, qual das declarações corresponde a um dos tipos representados no diagrama:

- ☐ `final class X : I()`
- ☐ `class Y(elems: I): X`
- ☐ `open class X : I`
- ☐ `interface I : Y`



Grupo II

Pretende-se desenvolver uma aplicação para classificação (i.e. *rating*) de itens de uma playlist (e.g. músicas, filmes, livros, etc). Na imagem seguinte apresentam-se 2 exemplos de utilização do componente *ScoreView*:

<pre>const val MAX_RATING = 5 fun main() { val musics: List<Score> = ... application { val winState = WindowState(...) Window(state = winState,...) { ScoreView(musics, MAX_RATING) } } }</pre>	<pre>const val MAX_RATING = 3 fun main() { val movies: List<Score> = ... application { val winState = WindowState(...) Window(state = winState,...) { ScoreView(movies, MAX_RATING) } } }</pre>
	

Clicando no botão de classificação de um item da lista (e.g. **2**) serão preenchidas o número de estrelas correspondente (neste caso: ★★☆☆☆).

Clicando sobre uma estrela a classificação deste item altera-se e é atualizado o valor apresentado na lista.

- [1.5] Tendo em atenção a função *main*, defina o tipo **imutável** *Score* que representa a informação de um item com as propriedades *name* e *rating*. Deve lançar a exceção *IllegalStateException* se o *rating* não estiver compreendido entre 1 e a constante *MAX_RATING*. (e.g. 5 para músicas e 3 para filmes).
- [1.5] Estenda o tipo *List<Score>* com a função **replace(score: Score)** que retorna uma nova lista substituindo o item pelo novo *Score* que tem nome igual.
- [3] Implemente o *Composable* **Stars**(*rating: Int*, *max: Int*, *onChange: (Int) -> Unit*) com o aspecto semelhante ao da imagem.
O click sobre uma estrela deve provocar a chamada à função *onChange* com o valor da posição dessa estrela. Assuma a existência de 2 recursos *star-full.png* e *star-empty.png* e da função *Composable ImageResource*(*resource: String*, *onClick: () -> Unit*) que apresenta a imagem corresponde ao *resource* e chama a função *onClick* quando se clicar sobre a imagem.
Nota: Este componente é *stateless*, ou seja, não mantém o estado do valor em edição.
- [2] Implemente o *Composable* **ScoreStars**(*score: Score?*, *max: Int*, *onChange: (Score) -> Unit*) com o aspecto semelhante ao da imagem e que usa o *Composable Stars* da alínea anterior.
O click sobre uma estrela deve provocar a chamada à função *onChange* com um novo objecto *Score*, correspondente ao valor da estrela clicada e mantendo o mesmo nome.
Se o *score* for *null* então não deve ser chamado o *onChange* e são apresentadas todas as estrelas vazias.
Nota: Este componente é *stateless*, ou seja, não mantém o estado do valor em edição.
- [2.5] Implemente o *Composable* **ScoreList**(*values: List<Score>*, *onSelect: (Score) -> Unit*) com o aspecto semelhante ao da imagem.
O click no botão associado a um item deve provocar a chamada à função *onSelect* com o objeto *Score* selecionado.
Nota: Este componente é *stateless*, ou seja, não mantém o estado do valor em edição.

6. [2.5] Implemente o *Composable* `ScoreView`(src: List<Score>, maxRating: Int) com o aspecto semelhante ao da imagem.

Este componente cria um componente `ScoreStars`, um componente `ScoreList` e mantém o estado de seleção e edição do `Score` selecionado (é *stateful*). Inicialmente nenhum dos scores está selecionado.

Duração: 90 minutos
ISEL, 30 de Janeiro de 2023