Instituto Superior de Engenharia de Lisboa

LEIC

Técnicas de Desenvolvimento de Software

Teste Final de Época Normal, Inverno de 2021/2022, 24 janeiro 2022

Nome: Número:

Grupo I

Considere os temas tratados nas aulas da disciplina e 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 8 valores e todas as perguntas têm a mesma cotação.

1. A expressão `16.dp` é usada para especificar uma dimensão de 16 *density-independent pixels* (DP) na biblioteca *Desktop Compose*. Sabendo que esta expressão é do tipo `Dp`, conclui-se que:
   - ◯ existe uma função de extensão ao tipo `Dp` com o nome `dp`
   - ◯ existe uma propriedade de extensão ao tipo `Int` com o nome `dp`
   - ◯ existe uma propriedade de extensão ao tipo `Dp` com o nome `dp`
   - ◯ nenhuma das outras opções

2. Dadas as seguintes definições:

```
fun log(prefix: String, message: String) { println("$prefix:$message") }
fun getLogger(prefix: String) = { msg: String -> log(prefix, msg) }
val mainLogger = getLogger("main")
```

   2.1. Os tipos de retorno das funções `log` e `getLogger` são, respectivamente:
   - ◯ `Unit` e `(String) -> Unit`
   - ◯ `Unit` e `() -> Unit`
   - ◯ `() -> Unit` e `(String) -> Unit`
   - ◯ `Unit` e `Unit`

   2.2. Uma das seguintes afirmações é falsa. Indique qual:
   - ◯ A expressão `::mainLogger` produz erro de compilação
   - ◯ A expressão `mainLogger(prefix = "prefix")` produz erro de compilação
   - ◯ A expressão `log("a_prefix", "a_message")` produz na consola a *string* "`log:a_message`"
   - ◯ A expressão `mainLogger("the_message")` produz na consola a *string* "`main:the_message`"

3. Considerando `T` um tipo qualquer em Kotlin
   - ◯ Existe conversão implícita de `T?` em `T`
   - ◯ Existe conversão implícita de `T` em `Any?`
   - ◯ Existe conversão implícita de `List<T?>` em `List<T>`
   - ◯ Não existe nenhuma das conversões anteriores

4. Qual das seguintes operações pode ser realizada diretamente na *thread* de UI de uma aplicação com *Desktop Compose*?

   ◯ Leitura de dados de base de dados com um custo típico esperado de 30ms
   ◯ Recolha de *user input* a partir da consola usando `readLine()`
   ◯ Cálculo computacional intensivo com duração máxima de 30 segundos
   ◯ Nenhuma das outras opções

5. Considere a execução do seguinte troço de código que faz comparações de igualdade e de identidade:

```
val a = SomeType(5)
val b = SomeType(5)
val c = a
println("${a == b} ${a === b} ${a === c}")
```

   ◯ Se `SomeType` for definido como `data class SomeType(val x: Int)`, será apresentado na consola a *string* "`true false true`"
   ◯ Se `SomeType` for definido como `class SomeType(val x: Int)`, será apresentado na consola a *string* "`false false false`"
   ◯ Se `SomeType` for definido como `fun SomeType(x: Int) = x`, será apresentado na consola a *string* "`false true true`"
   ◯ Todas as opções anteriores são verdadeiras

6. Considerando as seguintes declarações:

```
interface I {  fun f1() = 0 }
class A: I { override fun f1() = 1 }
object B: I { override fun f1() = 2 }

fun I.f2() = 'I'
fun A.f2() = 'A'

fun main() {
  val objs = listOf( A() , B )
  objs.forEach { print( it.f1() ) }
  objs.forEach { print( it.f2() ) }
}
```

   A execução da função main apresenta na consola a *string*:

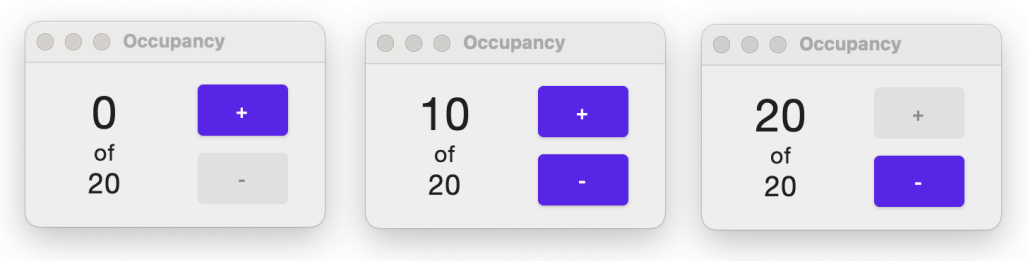
   ◯ "`12AI`"
   ◯ "`00AI`"
   ◯ "`12II`"
   ◯ Nenhuma das opções anteriores

Grupo II

Pretende-se desenvolver uma aplicação para apoiar a regulação da ocupação de recintos fechados. Os recintos têm lotação máxima que, quando atingida, impede a entrada de pessoas até que outras saiam. Sempre que alguém entra ou sai do recinto, a lotação do momento é atualizada. A função `main` da aplicação é a que se apresenta de seguida.

```
fun main() = application {
  MaterialTheme {
    val state = WindowState(width= 250.dp, height= Dp.Unspecified)
    Window(onCloseRequest= ::exitApplication, state= state, title= "Occupancy") {
      MainContent()
    }
  }
}
```

A figura que se segue apresenta a janela da aplicação nos seus estados possíveis (da esquerda para a direita): o recinto está vazio; o recinto ainda não está lotado; o recinto está lotado.



1. [2] Defina o tipo `Occupancy`, cujas instâncias **imutáveis** representam a lotação de recintos. As instâncias deste tipo contêm a lotação atual (`current`) e a lotação máxima (`capacity`) do recinto. Na definição de `Occupancy` certifique-se que é impossível existirem instâncias com lotações máximas negativas ou com lotações atuais negativas ou superiores à lotação máxima (i.e. é lançada `IllegalArgumentException`). Este tipo deve ter também as propriedades `isFull` e `isEmpty` cujos valores são calculados a partir de `current` e `capacity`.

2. [2] Estenda o tipo `Occupancy` com as operações `increment` e `decrement`, que produzem a ocupação atualizada quando alguém entra ou sai do recinto. Caso alguma das operações viole as invariantes do tipo, é lançada a excepção `IllegalStateException`. Valorizam-se as soluções que não alterem a definição de `Occupancy` da alínea anterior.

3. [2] Crie os testes automáticos para verificação da correcção da definição de `Occupancy` e das operações `increment` e `decrement`. Para que a resposta não se torne demasiado exaustiva, implemente apenas um teste de utilização válida e outro de utilização inválida do tipo `Occupancy` e da operação `increment`.

4. [2] Implemente o *Composable* `OccupancyView`, sem estado interno (*stateless*), para apresentação da lotação actual e máxima (com aspecto aproximado à parte esquerda das janelas da figura).

5. [3] Implemente o *Composable* `MainContent`, com estado interno (*stateful*), que representa o conteúdo da janela principal da aplicação. Note que o estado dos botões (parâmetro `enabled` do *Composable* `Button`) reflete o estado de ocupação do recinto. Por simplificação, admita que a capacidade do recinto está *hard-coded* na implementação de `MainContent`.

6. [1] Descreva as alterações que realizaria à aplicação para tornar a capacidade do recinto parametrizável. Na descrição indique a forma de parametrização da execução que escolheu.

Duração: 90 minutos
ISEL, 24 de Janeiro de 2022