Instituto Superior de Engenharia de Lisboa

LEIC / LEIRT

Programação em Dispositivos Móveis

Teste Global de Época Normal, Inverno de 2020/2021 (via Moodle)

Nome: Número: Turma:

Código de Honra

A vida académica é o preâmbulo da vida profissional. A adesão às regras de conduta é uma responsabilidade social, ou seja, é responsabilidade de todos. A participação na comunidade académica pressupõe a adesão a um código de honra que exige respeito pelo trabalho próprio e pelo trabalho dos demais (colegas e docentes). Esse código de honra proíbe liminarmente o plágio, simplesmente porque é socialmente inaceitável.
Para que possa concluir a avaliação de PDM tem que subscrever de forma explícita, e sob compromisso de honra, a autoria das respostas que entregar. A ausência de assinatura implica que a prova não será aceite.

Eu, abaixo assinado, declaro por minha honra que as respostas abaixo são de minha exclusiva autoria. Mais declaro que durante a prova apenas usei elementos de consulta autorizados.

Assinatura:

Enunciado

Considere a plataforma Android estudada nas aulas da disciplina e responda às perguntas seguintes assinalando de forma inequívoca a opção mais correta. Não responda arbitrariamente: cada resposta incorreta desconta 1/3 da cotação da pergunta ao total obtido na prova.

1. O ficheiro de manifesto de uma aplicação Android
   - ◯ é usado para descrever o procedimento de *build*
   - ◯ é usado para descrever a UI da aplicação Android
   - ➢ é usado para caracterizar os requisitos da aplicação
   - ◯ todas as outras opções

2. Considere a seguinte implementação de `SomeActivity`, que apresenta numa `TextView` (com id `quoteView`) o texto obtido a partir de um servidor remoto de "Citação do Dia", acessível através do objecto `QuoteOfDay`.

```
object QuoteOfDay {
   fun fetchDailyQuote(completionCallback: (quote: String?) -> Unit) {
       /* Fetches quote and calls completionCallback using the main thread */
   }
}

class SomeActivity : AppCompatActivity() {

   val binding by lazy { ActivitySomeBinding.inflate(layoutInflater) }    // (A)

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       binding.fetchButton.setOnClickListener {                          // (A)
           QuoteOfDay.fetchDailyQuote {
               if (it != null) binding.quoteView.text = it               // (A)
           }
       }
   }
}
```

2.1. Para que a citação do dia seja correctamente apresentada, mesmo podendo ocorrer uma reconfiguração em qualquer altura,
- ➢ a implementação teria que fazer uso de um *view model*
- ◯ a implementação apresentada é suficiente
- ◯ a implementação teria que garantir que a chamada a `completionCallback` é realizada noutra *thread* que não seja a *main thread*
- ◯ nenhuma das outras opções

2.2. Por observação das linhas de código assinaladas com // (A) e sabendo que não há erros de compilação, conclui-se que
- ◯ a aplicação faz uso do suporte para *view binding*
- ◯ existe um controlo gráfico com o id `fetchButton` no *layout* usado
- ◯ o *layout* usado está definido num ficheiro com o nome `activity_some.xml`
- ➢ todas as outras opções

3. A activação de uma *activity* pode ser realizada
- ◯ chamando o construtor sem parâmetros
- ◯ criando um *intent* com o nome completo da classe do componente de destino
- ➢ passando um *intent* explícito a `startActivity`
- ◯ todas as outras opções

4. Dada a `ActivityA` que se apresenta de seguida:

```
class ViewModelA(var onCreateCounter: Int = 0, var onStartCounter: Int = 0) : ViewModel()

class ActivityA : AppCompatActivity() {

   val vModel: ViewModelA by viewModels()                                    // (3)
   val button: Button by lazy { findViewById(R.id.button) }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_a)
       button.setOnClickListener {
           startActivity(Intent(this, ActivityA::class.java))
       }
       Log.v("ActivityATag", "onCreateCounter = ${++vModel.onCreateCounter}" )    // (1)
   }

   override fun onStart() {
       super.onStart()
       Log.v("ActivityATag", "onStartCounter = ${++vModel.onStartCounter}")       // (2)
   }
}
```

4.1. Para a sequência de acontecimentos: "`ActivityA` é lançada pela primeira vez" → "ecrã do dispositivo é rodado", as últimas duas mensagens escritas em *log* nas linhas // (1) e // (2) contêm, respectivamente:
- ◯ onCreateCounter = 1 e onStartCounter = 1
- ◯ onCreateCounter = 1 e onStartCounter = 2
- ◯ onCreateCounter = 2 e onStartCounter = 1
- ➢ onCreateCounter = 2 e onStartCounter = 2

4.2. Para a sequência de acontecimentos: "`ActivityA` é lançada pela primeira vez" → "ecrã do dispositivo é rodado" → "botão `button` é premido", as duas últimas mensagens escritas em log nas linhas // (1) e // (2) contêm, respectivamente:
Nota: Assuma que o comportamento de activação da `ActivityA` na *user task* é o por omissão
- ➢ onCreateCounter = 1 e onStartCounter = 1
- ◯ onCreateCounter = 2 e onStartCounter = 2
- ◯ onCreateCounter = 3 e onStartCounter = 3
- ◯ nenhuma das outras

4.3. Se no construtor primário da classe `ViewModelA` não estivessem especificados valores por omissão, e considerando a forma de instanciação usada em // (3),
- ◯ seria produzido um erro de compilação na classe `ViewModelA` por falta do construtor sem parâmetros
- ➢ não seria encontrado um construtor adequado à criação da instância de *view model*
- ◯ as propriedades `onCreateCounter` e `onStartCounter` seriam iniciadas com valores arbitrários
- ◯ falharia a compilação no construtor de `ViewModelA` por falta do parâmetro de tipo `Application`

5. Para uma instância de `RecyclerView.Adapter`, o número de instâncias criadas do *view holder* associado ao *adapter* é ...
- ◯ igual ao número de elementos da coleção a ser apresentada
- ➢ menor ou igual ao número de elementos da coleção a ser apresentada
- ◯ sempre igual ao número de chamadas a `onBindViewHolder`
- ◯ nenhuma das outras opções

6. Dada a `ActivityA` apresentada de seguida ...

```
class TheApplication(var aCounter: Int = 0) : Application()

class ViewModelA(
       application: Application,
       private val viewState: SavedStateHandle
) : AndroidViewModel(application) {

   var counter1: Int
       get() = viewState["counter1"] ?: 0
       set(value) { viewState["counter1"] = value }

   var counter2: Int = 0
}

class ActivityA : AppCompatActivity() {

   val vModel: ViewModelA by viewModels()
   val button: Button by lazy { findViewById(R.id.button) }
   val app: TheApplication by lazy { application as TheApplication }

   override fun onCreate(savedInstanceState: Bundle?) {
       super.onCreate(savedInstanceState)
       setContentView(R.layout.activity_a)
       button.setOnClickListener {
           vModel.counter1 += 1
           vModel.counter2 += 1
           app.aCounter += 1
       }
   }

   override fun onStart() {
       super.onStart()
       Log.v("ActivityATag", "counter1=${vModel.counter1}; " +
               "counter2=${vModel.counter2}; aCounter=${app.aCounter}")  // (A)
   }
}
```

6.1. Para a sequência de acontecimentos: "`ActivityA` é lançada pela primeira vez" → "utilizador prime o botão *button*" → "ecrã do dispositivo é rodado", a última mensagem escrita em log na linha // (A) contém:
- ◯ counter1=0; counter2=0; aCounter=0
- ◯ counter1=0; counter2=0; aCounter=1
- ◯ counter1=1; counter2=0; aCounter=1
- ➢ counter1=1; counter2=1; aCounter=1

6.2. Para a sequência de acontecimentos: "`ActivityA` é lançada pela primeira vez" → "utilizador prime botão *button*" → "utilizador selecciona outra *user task*" → "processo hospedeiro é terminado" → "utilizador selecciona de novo a *user task* da `ActivityA`", a última mensagem escrita em log na linha // (A) contém:

- ◯ counter1=0; counter2=0; aCounter=0
- ➢ counter1=1; counter2=0; aCounter=0
- ◯ counter1=1; counter2=0; aCounter=1
- ◯ counter1=1; counter2=1; aCounter=0

6.3. Para a sequência de acontecimentos: "`ActivityA` é lançada pela primeira vez" → "utilizador prime o botão *button*" → "utilizador prime back" → "`ActivityA` volta a ser lançada através do *home screen*", a última mensagem escrita em log na linha // (A) contém:

- ◯ counter1=1; counter2=0; aCounter=0
- ➢ counter1=0; counter2=0; aCounter=1
- ◯ counter1=1; counter2=0; aCounter=1
- ◯ counter1=1; counter2=1; aCounter=1

7. A execução de uma tarefa, por via da API Work Manager, lançada a partir de uma *activity*:

- ➢ é garantida mesmo que essa *activity* deixe de estar visível
- ◯ está sujeita à existência de conectividade
- ◯ é equivalente à execução dessa tarefa por via de um Executor
- ◯ todas as outras opções

8. Considere a seguinte definição no âmbito do uso da *framework* Room.

```
@Dao
interface SomeDao {
   @Query("...")
   fun loadAll(): List<String>
   @Query("...")
   fun loadLast(count: Int): LiveData<List<String>>
   @Insert
   fun insertOne(one: String)
}
```

As execuções dos métodos:

- ◯ `loadAll` e `loadLast` são síncronas
- ◯ `loadAll` e `loadLast` são síncronas ou assíncronas conforme o valor da String passada em @Query
- ➢ `loadAll` e `insertOne` são síncronas
- ◯ são síncronas ou assíncronas conforme a implementação fornecida por quem desenvolve a aplicação

9. Numa aplicação que recorre a uma base de dados Cloud Firestore, para observar actualizações dos dados:

- ◯ deve-se recorrer a um `PeriodicWorkRequest` (da API Work Manager) para consultar periodicamente os dados
- ◯ regista-se o interesse invocando `observe` sobre a instância de `LiveData` retornada por `getAsLiveData`
- ➢ não existem operações que exponham diretamente `LiveData`
- ◯ executa-se `readData` em ciclo em *thread* auxiliar; readData retornará dados por cada actualização

10. Uma das seguintes afirmações é falsa. Indique qual.

- ◯ A necessidade de controlar o consumo de energia levou à introdução do `WorkManager`.
- ➢ A ideia de *user task* ajuda a impedir o uso de componentes de uma aplicação por outras.
- ◯ A introdução do `SavedStateHandle` ajuda a lidar com o controlo automático de recursos de memória.
- ◯ A existência de ViewModel ajuda a suportar a possibilidade de reconfigurar a língua do dispositivo.

11. A palavra *foreground* na designação de Foreground Service advém de:

- ◯ o serviço só poder ser iniciado a partir de uma *thread* que está em *foreground*.
- ◯ só se garantir a execução do serviço enquanto houver alguma *activity* da aplicação em *foreground*.

➢ a execução do serviço implicar a afixação de informação ao utilizador.
◯ a execução do serviço impedir outras aplicações de ficarem em *foreground*.

Duração: 40 minutos
ISEL, 30 de Janeiro de 2021