

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. Considere a seguinte implementação parcial de ActivityA cujo objectivo é apresentar o texto obtido a partir de um servidor remoto.

```
class ActivityA : AppCompatActivity() {
    private fun updateUI(msg: String?) { textView.text = msg }
    private fun fetchDataFromServer(): String? { /* Implemented with synchronous I/O */ }
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_a)
        // (A) TODO: fetch data and display it in the UI
    }
}
```

- 1.1. Para que a implementação esteja conforme o modelo de *threading* Android, uma possível implementação da linha assinalada com (A) é:
 - ☐ updateUI(fetchDataFromServer())
 - ☐ Thread { runOnUiThread { updateUI(fetchDataFromServer()) } }.start()
 - ☒ Thread { val m = fetchDataFromServer(); runOnUiThread { updateUI(m) } }.start()
 - ☐ Thread { updateUI(fetchDataFromServer()) }.start()
- 1.2. Por observação da implementação parcial apresentada, e sabendo que não existem erros de compilação, conclui-se que
 - ☐ existe um controlo gráfico com o id textView no layout usado
 - ☐ o layout usado está definido num ficheiro XML com o nome activity_a.xml
 - ☐ o procedimento de build inclui suporte para kotlin extensions
 - ☒ Todas as anteriores

2. Considere a ActivityA cujo objectivo é apresentar o texto obtido a partir de um servidor remoto:

```
class ViewModelA(var data: String? = null) : ViewModel() {
    fun fetchDataFromServer(ctx: Context, cb: (String?) -> Unit) {
        Volley.newRequestQueue(ctx).add(StringRequest(
            Request.Method.GET, "http://some.server.com/",
            { result -> data = result; cb(data) }, { error -> data = "error"; cb(data) }
        ))
    }
}

class ActivityA : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_a); textView.text = ""
        val model = ViewModelProviders.of(this).get(ViewModelA::class.java)
        if (model.data != null) textView.text = model.data
        else model.fetchDataFromServer(this) {
            // (A) TODO: Appends the received text to the contents of textView
        }
    }
}
```

2.1. Para a sequência de acontecimentos: “ActivityA é lançada pela primeira vez” → “ecrã do dispositivo é rodado”, o número de chamadas ao construtor de ViewModelA é:

- ☒ 1
- ☐ 2
- ☐ 3
- ☐ Nenhuma das anteriores

2.2. No final da sequência: “ActivityA é lançada pela primeira vez” → “Pedido HTTP é realizado” → “É recebida a resposta HTTP com o texto ‘Success’” → “ecrã do dispositivo é rodado”, a activity ...

- ☐ ... apresenta a *string* vazia
- ☒ ... apresenta a *string* ‘Success’
- ☐ ... apresenta a *string* ‘SuccessSuccess’
- ☐ ... apresenta a *string* ‘error’

2.3. No final da sequência: “ActivityA é lançada pela primeira vez” → “Pedido HTTP é realizado” → “ecrã do dispositivo é rodado” → “É recebida a resposta HTTP com o texto ‘Success’”, a activity ...

- ☒ ... apresenta a *string* vazia
- ☐ ... apresenta a *string* ‘Success’
- ☐ ... apresenta a *string* ‘SuccessSuccess’
- ☐ ... apresenta a *string* ‘error’

2.4. Para que a implementação esteja conforme o modelo de *threading* Android, a implementação correcta da linha assinalada com (A) é:

- ☒ textView.text = "\${textView.text}\$it"
- ☐ Thread { runOnUiThread { textView.text = "\${textView.text}\$it" } }.start() }
- ☐ Thread { textView.text = "\${textView.text}\$it" }.start() }
- ☐ Nenhuma das anteriores

3. No modelo de programação disponibilizado pela biblioteca Room, as Entities ...

- ☐ são as classes definidas pelo programador e que contém a implementação do código relativo aos acessos à base de dados
- ☐ são as interfaces definidas pelo programador que caracterizam as operações de acesso a dados e cuja implementação é gerada em tempo de *build*
- ☒ são as classes definidas pelo programador e que representam os dados a armazenar na base de dados
- ☐ Todas as anteriores

4. Para uma instância de RecyclerView.Adapter, o número de instâncias do *view holder* correspondente é...
- ☒ menor ou igual ao número de elementos da coleção a ser apresentada
 - ☐ maior ou igual ao número de elementos da coleção a ser apresentada
 - ☐ sempre igual ao número de chamadas a `onBindViewHolder`
 - ☐ Nenhuma das anteriores
5. Dada uma aplicação Android composta pelas *activities* apresentadas de seguida:

```
const val TAG: String = "TAG"
abstract class BaseActivity(private val name: String) : AppCompatActivity() {
    override fun onCreate(s: Bundle?) { super.onCreate(s); Log.v(TAG, "$name onCreate") }
    override fun onStart() { super.onStart(); Log.v(TAG, "$name onStart") }
    override fun onStop() { super.onStop(); Log.v(TAG, "$name onStop") }
    override fun onDestroy() { super.onDestroy(); Log.v(TAG, "$name onDestroy()") }
}
class ActivityA : BaseActivity("A") {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_a)
        button.setOnClickListener { startActivity(Intent(this, ActivityB::class.java)) }
    }
}
class ActivityB : BaseActivity("B") {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_b)
    }
}
```

- 5.1. Para a sequência de acontecimentos: “ActivityA é lançada pela primeira vez” → “utilizador prime botão *button*”, o número de vezes que as *strings* “onCreate”, “onStart”, “onStop” e “onDestroy” surgem em *log* são, respectivamente:
- ☒ 2, 2, 1 e 0
 - ☐ 2, 2, 1 e 1
 - ☐ 2, 2, 2 e 0
 - ☐ 2, 3, 2 e 1
- 5.2. Para a sequência de acontecimentos: “ActivityA é lançada pela primeira vez” → “utilizador prime botão *button*” → “utilizador selecciona outra *user task*”, o número de vezes que as *strings* “onCreate”, “onStart”, “onStop” e “onDestroy” surgem em *log* são, respectivamente:
- ☐ 2, 2, 1 e 0
 - ☐ 2, 2, 1 e 1
 - ☒ 2, 2, 2 e 0
 - ☐ 2, 3, 2 e 1
- 5.3. Para a sequência de acontecimentos: “ActivityA é lançada pela primeira vez” → “utilizador prime botão *button*” → “utilizador volta para a *activity* inicial (prime *back*)”, o número de vezes que as *strings* “onCreate”, “onStart”, “onStop” e “onDestroy” surgem em *log* são, respectivamente:
- ☐ 2, 2, 1 e 0
 - ☐ 2, 2, 1 e 1
 - ☐ 2, 2, 2 e 0
 - ☒ 2, 3, 2 e 1

6. De acordo com o guia de arquitectura Android Jetpack ...
- ☐ cabe ao *ViewModel* determinar se obtém dados de uma API remota ou da base de dados local, dirigindo os pedidos para o componente adequado
 - ☒ cabe ao *Repository* determinar se obtém dados de uma API remota ou da base de dados local, dirigindo os pedidos para o componente adequado
 - ☐ cabe à *Activity* determinar se obtém dados de uma API remota ou da base de dados local, dirigindo os pedidos para o componente adequado
 - ☐ cabe ao *Repository* aceder às instâncias de *ViewModel* para consultar ou modificar dados
7. No âmbito da *framework WorkManager*, considere uma dada classe derivada de `androidx.work.Worker`. A implementação da *framework* garante que ...
- ☒ o método `doWork` é chamado pelo menos uma vez numa *thread* de *background*
 - ☐ o método `doWork` é chamado no máximo uma vez na UI *thread*
 - ☐ o método `doWork` nunca é chamado numa *thread* de *background*
 - ☐ o método `doWork` é chamado uma ou mais vezes na UI *thread*
8. A execução de uma tarefa por via de uma *AsyncTask* a partir de uma *activity* ...
- ☐ exige que essa *AsyncTask* seja declarada no manifesto da aplicação
 - ☒ não garante que a execução da tarefa seja concluída se essa *activity* deixar de estar visível
 - ☐ é equivalente à execução dessa tarefa por via de um *Foreground Service*
 - ☐ não é permitida, porque vai resultar na utilização de uma *worker thread*
9. Considere o método `fun onSaveInstanceState(outState: Bundle)` de uma dada *activity*. A colocação de dados na instância de *Bundle* recebida ...
- ☒ dá garantias de que esses dados sobrevivem à terminação do processo hospedeiro
 - ☐ dá garantias de que esses dados sobrevivem ao *reboot* do dispositivo
 - ☐ tem de ser realizada numa *background thread*
 - ☐ Todas as anteriores
10. Considere a *framework SharedPreferences*. A API inclui a interface `SharedPreferences.Editor`, que contém métodos para escrita e remoção de valores. A escrita de valores através dessa interface ...
- ☐ dá garantias de que esses dados sobrevivem à terminação do processo hospedeiro
 - ☐ dá garantias de que esses dados sobrevivem ao *reboot* do dispositivo
 - ☐ pode ser realizada na UI *thread*
 - ☒ Todas as anteriores

Duração: 1 hora
ISEL, 3 de Fevereiro de 2020