

Programação em Dispositivos Móveis

Teste Global de Época de Recurso, Inverno de 2021/2022

Nome:

Número:

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 do 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 correta. Não responda arbitrariamente: cada resposta incorreta desconta 1/3 da cotação da pergunta ao total obtido na prova.

1. Uma das utilidades do ficheiro de manifesto de uma aplicação Android é:
 - ☐ definir o idioma em que a aplicação é apresentada
 - ☐ definir se a aplicação utiliza Java ou Kotlin
 - ☐ definir as dimensões com que cada *activity* é apresentada
 - ☐ nenhuma das outras opções
2. Num dispositivo Android, ao ser mudada a definição global da língua do sistema:
 - ☐ são destruídas todas as instâncias de Activity existentes para que assumam a nova configuração na sua reconstrução
 - ☐ são automaticamente alterados os valores das *labels* e as dimensões dos controlos gráficos existentes sem que isso implique a destruição das instâncias de Activity existentes
 - ☐ são destruídas e reconstruídas todas as aplicações ativas para que assumam a nova configuração
 - ☐ é de novo executado o método *onCreate* nas instâncias de Activity já existentes
3. A inclusão de instâncias de tipos definidos pela aplicação em instâncias de *SavedStateHandle* é possível se:
 - ☐ esses tipos cumprirem o contrato *Parcelable*
 - ☐ esses tipos forem anotados com *@Parcelize*, independentemente da sua definição
 - ☐ a sua definição apenas incluir tipos primitivos
 - ☐ nenhuma das outras opções
4. Para a correta resolução de um *intent* explícito é imprescindível:
 - ☐ a criação do *intent* com o nome completo da classe do componente de destino
 - ☐ a definição de, pelo menos, um *intent-filter* no manifesto da aplicação de destino
 - ☐ que a aplicação de destino esteja ativa com o componente no estado *STARTED* ou *RESUMED*
 - ☐ todas as outras opções

5. Dada a aplicação Android composta pelas *activities* apresentadas de seguida:

```
class ActivityAViewModel(private val state: SavedStateHandle): ViewModel() {
    var vmCounter1: Int = 0
    var vmCounter2: Int
        get() = state.get("Counter2") ?: 0
        set(value) { state.set("Counter2", value) }
}
class ActivityA : AppCompatActivity() {
    private val binding by lazy { ActivityABinding.inflate(layoutInflater) }
    private val viewModel: ActivityAViewModel by viewModels()
    private var acCounter: Int = 0
    private fun updateCounts() {
        acCounter += 1; viewModel.vmCounter1 += 1; viewModel.vmCounter2 += 1;
        binding.acCounter.text = acCounter.toString()
        binding.vmCounter1.text = viewModel.vmCounter1.toString()
        binding.vmCounter2.text = viewModel.vmCounter2.toString()
    }
    override fun onCreate(s: Bundle?) {
        super.onCreate(s)
        setContentView(binding.root)
        updateCounts()
        binding.hitMe.setOnClickListener { updateCounts() }
    }
}
```

- 5.1. Para a sequência de acontecimentos: “ActivityA é lançada pela primeira vez” → “utilizador prime botão hitMe” → “utilizador selecciona outra *user task*” → “utilizador volta à *user task* da aplicação”, os valores apresentados nas caixas de texto são, respectivamente:
- ☐ acCounter → 1; vmCounter1 → 1; vmCounter2 → 1
 - ☐ acCounter → 2; vmCounter1 → 2; vmCounter2 → 2
 - ☐ acCounter → 3; vmCounter1 → 3; vmCounter2 → 3
 - ☐ acCounter → 1; vmCounter1 → 1; vmCounter2 → 3
- 5.2. Para a sequência de acontecimentos: “ActivityA é lançada pela primeira vez” → “utilizador prime botão hitMe” → “ocorre uma reconfiguração (e.g. ecrã do dispositivo é rodado)”, os valores apresentados nas caixas de texto são, respectivamente:
- ☐ acCounter → 1; vmCounter1 → 1; vmCounter2 → 1
 - ☐ acCounter → 2; vmCounter1 → 2; vmCounter2 → 2
 - ☐ acCounter → 1; vmCounter1 → 3; vmCounter2 → 3
 - ☐ acCounter → 1; vmCounter1 → 1; vmCounter2 → 3
- 5.3. Para a sequência de acontecimentos: “ActivityA é lançada pela primeira vez” → “utilizador prime botão hitMe” → “utilizador selecciona outra *user task*” → “processo hospedeiro é terminado” → “utilizador volta à *user task* da aplicação”, os valores apresentados nas caixas de texto são, respectivamente:
- ☐ acCounter → 1; vmCounter1 → 1; vmCounter2 → 1
 - ☐ acCounter → 2; vmCounter1 → 2; vmCounter2 → 2
 - ☐ acCounter → 1; vmCounter1 → 3; vmCounter2 → 3
 - ☐ acCounter → 1; vmCounter1 → 1; vmCounter2 → 3
- 5.4. No contexto do código apresentado, podemos afirmar que:
- ☐ a aplicação faz uso do suporte para *view binding*
 - ☐ a propriedade vmCounter1 poderia ter sido definida no construtor (veja-se abaixo) sem que isso implicasse mais nenhuma alteração à solução
// class ActivityAViewModel (var vmCounter1: Int = 0, private val state: SavedStatehandle)
 - ☐ existe um ficheiro de *layout* com o nome activity_a_binding.xml
 - ☐ todas as outras opções

6. Considerando a API *Work Manager* e dada a seguinte definição de *Someworker*:

```
fun syncFetchAndSave() {  
    // Synchronously fetches data from a remote API and stores it in a Local DB  
    // Throws an Exception if the operation failed  
}  
  
class Someworker(app: Context, params: WorkerParameters) : Worker(app, params) {  
    override fun doWork(): Result =  
        try { syncFetchAndSave(); Result.success() }  
        catch (e: Exception) { Result.failure() }  
}
```

Após análise da implementação conclui-se que

- ☐ a implementação está correcta
 - ☐ para que esteja correcta, o método `doWork()` tem que retornar sempre `Result.success()`
 - ☐ para que esteja correcta, a chamada a `syncFetchAndSave()` tem que ser executada numa *thread* alternativa
 - ☐ nenhuma das opções anteriores
7. No âmbito do modelo de programação disponibilizado pela biblioteca *Room*, pode-se afirmar que os acessos às propriedades das *entities*
- ☐ têm de ser realizados na *thread* de UI para que os resultados possam ser afixados nos controlos gráficos
 - ☐ têm de ser realizados fora da *thread* de UI para a não bloquear
 - ☐ independentemente da *thread* usada, só podem ser realizados pelos DAOs
 - ☐ não têm restrições específicas quanto às *threads* utilizadas
8. Para uma instância de `RecyclerView.Adapter`, o número de chamadas a `onCreateViewHolder` 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
9. Numa aplicação que recorre a uma base de dados *Firestore*, a subscrição a notificações de atualizações dos dados:
- ☐ não pode ser realizada na *main thread* porque é nessa *thread* que as notificações são realizadas
 - ☐ não pode ser realizada na *main thread* porque a *thread* invocante fica bloqueada até que haja notificação
 - ☐ retorna uma instância de `LiveData` que será usada para registar o *listener* das notificações
 - ☐ nenhuma das outras opções
10. Considerando uma *custom view* definida através de classe derivada de `View`, pode-se afirmar que
- ☐ a chamada a `onDraw()` é realizada pela *framework* quando for oportuno desenhar a *view*
 - ☐ a chamada a `repaint()` é realizada pela aplicação quando for necessário redesenhar a *view*
 - ☐ a implementação de `onDraw()` é responsável por especificar o aspecto da *view* no ecrã
 - ☐ todas as outras opções

11. Considere as seguintes definições:

```
interface OneRetrofitService { @GET("/") fun getData(): Call<String> }

class OneViewModel(application: Application): AndroidViewModel(application) {
    private val oneService by lazy { getApplication<OneApplication>().oneService }
    val result: MutableLiveData<String> = MutableLiveData()
    fun fetchData(resultHolder: MutableLiveData<String>? = null) {
        oneService.getData().enqueue(object : Callback<String> {
            override fun onResponse(call: Call<String>, response: Response<String>) {
                (resultHolder ?: result).value = response.body() ?: ""
            }
            override fun onFailure(call: Call<String>, t: Throwable) { /* ... */ }
        })
    }
    fun otherFetchData() { result.value = oneService.getData().execute().body() ?: "" }
}

class OneActivity : AppCompatActivity() {
    private val binding by lazy { ActivityOneBinding.inflate(layoutInflater)}
    private val viewModel: OneViewModel by viewModels()
    private val liveData = MutableLiveData<String>("")
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(binding.root)
        liveData.observe(this) { binding.textView2.text = it }
        viewModel.result.observe(this) { binding.textView.text = it }           //(1)
        binding.fetchData.setOnClickListener { viewModel.fetchData() }         //(2)
    }
}
```

11.1. Para a sequência de acontecimentos: “OneActivity é lançada pela primeira vez” → “utilizador prime o botão fetchData” → “ecrã do dispositivo é rodado”, e admitindo que os dados são obtidos com sucesso a partir da API remota, podemos afirmar que no final da sequência:

- ☐ os dados PODEM ou NÃO ser apresentados, dependendo do tempo que demora a serem obtidos
- ☐ os dados PODEM ou NÃO ser apresentados, independentemente do tempo que demora a serem obtidos
- ☐ os dados são SEMPRE apresentados, independentemente do tempo que demora a serem obtidos
- ☐ os dados NUNCA são apresentados, independentemente do tempo que demora a serem obtidos

11.2. Substitua o conteúdo do *listener* da linha //(2) pela expressão `viewModel.fetchData(liveData)`. Para a sequência de acontecimentos: “OneActivity é lançada pela primeira vez” → “utilizador prime o botão fetchData” → “ecrã do dispositivo é rodado”, e admitindo que os dados são obtidos com sucesso a partir da API remota, podemos afirmar que no final da sequência:

- ☐ os dados PODEM ou NÃO ser apresentados, dependendo do tempo que demora a serem obtidos
- ☐ os dados PODEM ou NÃO ser apresentados, independentemente do tempo que demora a serem obtidos
- ☐ os dados são SEMPRE apresentados, independentemente do tempo que demora a serem obtidos
- ☐ os dados NUNCA são apresentados, independentemente do tempo que demora a serem obtidos

- 11.3. Substitua o conteúdo do *listener* da linha //(2) pela expressão `viewModel.otherFetchData()`. Após a substituição, podemos afirmar que:
- ☐ a implementação está correta
 - ☐ para que a implementação esteja correta o conteúdo do *listener* da linha //(1) tem também de ser alterado para `runOnUiThread { binding.textView.text = it }`
 - ☐ para que a implementação esteja correta o conteúdo do *listener* da linha //(1) tem também de ser alterado para `Thread { binding.textView.text = it }.start()`
 - ☐ nenhuma das anteriores

Duração: 40 minutos
ISEL, 17 de Fevereiro de 2021