

Programação em Dispositivos Móveis

Teste Global de Época de Recurso, 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 incluído no APK resultante do procedimento de *build*
 - ☐ é usado para indicar que *activities* compõem a aplicação
 - ☐ é usado para caracterizar os requisitos da aplicação
 - todas as outras opções
2. Uma das seguintes afirmações é falsa. Indique qual.
 - ☐ A ativação de uma *activity* é realizada através um *intent* que pode ou não ser-lhe explicitamente dirigido.
 - A aplicação tem de incluir uma classe derivada de `android.app.Application`.
 - ☐ É possível existir mais do que uma *user task* por aplicação.
 - ☐ O uso da rede de comunicações requer a declaração da permissão correspondente no manifesto.
3. De acordo com o guia de arquitetura Android Jetpack, o *repository*
 - determina se obtém dados de uma API remota ou da base de dados local, dirigindo os pedidos para o componente adequado
 - ☐ representa os dados a serem armazenados na base de dados local
 - ☐ deve ser específico à *activity* porque fornece os dados necessários à mesma
 - ☐ nenhuma das outras opções
4. No âmbito do modelo de programação disponibilizado pela biblioteca Room, pode-se afirmar que
 - ☐ as classes anotadas com `@Entity` representam os dados a armazenar
 - ☐ as interfaces anotadas com `@Dao` caracterizam as operações de acesso a dados cuja implementação é fornecida durante a fase de *build*
 - ☐ a classe anotada com `@Database` tem de derivar de `androidx.room.RoomDatabase`
 - todas as outras opções

5. Considere a seguinte implementação de QuoteActivity que apresenta em quoteView o texto obtido a partir de um servidor remoto de “Citação do Dia”, operação desencadeada quando o botão fetchButton é premido.

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

class QuoteActivity : AppCompatActivity() {
    val viewModel by viewModels<QuoteViewModel>() // (A)
    val binding by lazy { ActivityQuoteBinding.inflate(layoutInflater) } // (B)

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

- 5.1. Para que a citação do dia seja correctamente apresentada, mesmo podendo ocorrer uma reconfiguração em qualquer altura,
- ☐ a implementação não poderia 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
- 5.2. Relativamente às linhas de código assinaladas com // (A) e // (B) pode-se afirmar que
- ☐ // (B) é equivalente a `val binding = ActivityQuoteBinding.inflate(layoutInflater)`
 - ☐ // (A) é equivalente a `val viewModel = viewModels<QuoteViewModel>()`
 - a chamada `ActivityQuoteBinding.inflate(...)` é realizada durante a execução de `onCreate`
 - ☐ nenhuma das outras opções
6. Numa aplicação que recorre a uma base de dados Cloud 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
 - retorna uma instância de ListenerRegistration que pode ser usada para cancelamento da subscrição
7. A execução do método `fun doWork(): Result` das classes derivadas de `androidx.work.Worker`
- ☐ é realizada na *UI thread* para garantir que a actualização das *views* é feita de forma *thread safe*
 - ☐ não tem garantida a conclusão da sua execução, por ser realizada numa *background thread*
 - dá relevância ao processo hospedeiro fazendo com que o mesmo não seja terminado precocemente
 - ☐ nenhuma das outras opções

8. Dadas as *activities* que se apresentam de seguida, cuja forma de activação é a usada por omissão:

```
const val TAG = "Tag"
abstract class BaseActivity(private val name: String) : AppCompatActivity() {
    override fun onCreate(b: Bundle?) {
        super.onCreate(b); 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") {
    private val binding by lazy { ActivityABinding.inflate(layoutInflater) }
    override fun onCreate(b: Bundle?) {
        super.onCreate(b)
        setContentView(binding.root)
        binding.gotoB.setOnClickListener {
            startActivity(Intent(this, ActivityB::class.java))
        }
    }
}

class ActivityB : BaseActivity("B") {
    private val binding by lazy { ActivityBBinding.inflate(layoutInflater) }
    override fun onCreate(b: Bundle?) {
        super.onCreate(b)
        setContentView(binding.root)
        binding.gotoA.setOnClickListener {
            startActivity(Intent(this, ActivityA::class.java))
        }
    }
}
```

- 8.1. Para a sequência de acontecimentos: "ActivityA é lançada pela primeira vez" → "gotoB é premido", o número de vezes que as mensagens indicadas surgem em log é:
- ☐ "A onStop" → 0, "A onDestroy" → 0
 - ☐ "A onStop" → 0, "A onDestroy" → 1
 - ☒ "A onStop" → 1, "A onDestroy" → 0
 - ☐ "A onStop" → 1, "A onDestroy" → 1
- 8.2. Para a sequência de acontecimentos: "ActivityA é lançada pela primeira vez" → "gotoB é premido" → "gotoA é premido", o número de vezes que as mensagens indicadas surgem em log é:
- ☐ "A onCreate" → 1, "A onStart" → 1, "A onStop" → 1
 - ☒ "A onCreate" → 2, "A onStart" → 2, "A onStop" → 1
 - ☐ "A onCreate" → 1, "A onStart" → 2, "A onStop" → 1
 - ☐ "A onCreate" → 1, "A onStart" → 2, "A onStop" → 0
- 8.3. Para a sequência de acontecimentos: "ActivityA é lançada pela primeira vez" → "gotoB é premido" → "utilizador prime *back*" → "utilizador selecciona outra user task", o número de vezes que as mensagens indicadas surgem em log é:
- ☐ "A onStop" → 1, "A onDestroy" → 0, "B onStop" → 1, "B onDestroy" → 1
 - ☐ "A onStop" → 1, "A onDestroy" → 0, "B onStop" → 1, "B onDestroy" → 0
 - ☐ "A onStop" → 2, "A onDestroy" → 1, "B onStop" → 1, "B onDestroy" → 0
 - ☒ "A onStop" → 2, "A onDestroy" → 0, "B onStop" → 1, "B onDestroy" → 1

9. Ao estender a classe View para criar uma área de desenho visível numa *activity*, em relação à correta apresentação do desenho após rotação do dispositivo...
- ☐ os dados necessários para produzir o desenho são mantidos em campos da classe derivada de View, sendo reproduzido o desenho a partir dessa informação após a rotação.
 - ☐ os dados necessários para produzir o desenho são mantidos no *view model* da *activity* e o *view model*, após a rotação, invoca os métodos necessários da classe derivada de View para restaurar o desenho.
 - ☐ os dados necessários para produzir o desenho são mantidos pela classe base View, sendo o desenho restaurado automaticamente por esta após a rotação.
 - os dados necessários para produzir o desenho não são recuperáveis a partir de campos da *view* nem da *activity* e, após rotação, a *activity* invoca métodos da *view* para restaurar o desenho.
10. O método `onCleared` de `ViewModel` é invocado para assinalar o final do respetivo tempo de vida (salvo em caso de terminação abrupta do processo hospedeiro). Considere a *activity* `SomeActivity` e o seu *view model*, `SomeActivityViewModel`. Relativamente à relação entre o número de chamadas a `onDestroy` de `SomeActivity` e o número de chamadas a `onCleared` de `SomeViewModel`, pode-se afirmar que:
- ☐ o número de chamadas a `onDestroy` é **igual** ao número de chamadas a `onCleared`
 - o número de chamadas a `onDestroy` é **superior ou igual** ao número de chamadas a `onCleared`
 - ☐ o número de chamadas a `onDestroy` é **inferior ou igual** ao número de chamadas a `onCleared`
 - ☐ nenhuma das outras opções
11. Durante a execução do método `onBindViewHolder` de uma instância de `RecyclerView.Adapter`, pretende-se aplicar formatação específica (por exemplo, texto a negrito ou de cor diferente), dependente dos dados concretos a apresentar em cada posição. É possível fazê-lo?
- ☐ Não, uma vez que a formatação só pode ser definida em `onCreateViewHolder`, devido à reutilização das *views*.
 - ☐ Não, uma vez que nesta operação só se conhece a posição dos dados mas não o seu valor, o que permite a reutilização das *views*.
 - ☐ Sim, pode-se aplicar formatação apenas em parte das posições e manter a formatação resultante do *inflate* do *layout* nas outras.
 - Sim, desde que os aspectos sujeitos a formatação sejam definidos em todas as posições, mesmo nas que não precisam de formatação específica.
12. Qual das seguintes operações pode ser realizada diretamente na *main thread* de uma aplicação Android?
- ☐ Invocação de operação de uma API *web* por HTTP com um custo típico esperado de 30ms.
 - ☐ Leitura de dados de base de dados local, via Room, com um custo típico esperado de 30ms.
 - ☐ Cálculo computacional intensivo com duração máxima esperada de 30 segundos.
 - Nenhuma das outras opções.
13. A utilização de `SavedStateHandle` na definição de um `ViewModel` resolve o problema da preservação de estado quando um `ViewModel` precisa de ser reinstituído. Onde são mantidos os dados guardados por um `SavedStateHandle` na situação em que ocorre essa reinstalação?
- ☐ São mantidos em privado na memória do processo, associados ao objeto `Application`.
 - ☐ Ficam em base de dados relacional, com custos de I/O, até surgir a nova instância de `ViewModel`.
 - São serializados e persistidos fora da memória do processo, com custos na transferência de dados.
 - ☐ São mantidos como pares chave-valor numa instância de `LiveData`, evitando transferência de dados.

Duração: 40 minutos
ISEL, 5 de Março de 2021