INTRODUÇÃO À COMPUTAÇÃO MÓVEL

EXAME ANDROID

Written Test | Android module–2018-02-09(“Recurso”)

Q1.

The diagram shows a state machine that can be applied to the Android Activity component.

Consulta: https://developer.android.com/guide/components/activities/activity-lifecycle

a) Relate the transitions in the state machine with the moment(s)a given layout, specified in aXML file, is expected to be prepared (graphic elements are created) and disposed (views are discarded).

A: essa transição corresponde a quando a *activity* é criada pelo sistema e implementa o método onCreate(), e só deve acontecer uma vez durante todo o período que a *activity* durar. Sua implementação pode vincular dados a listas, associar a atividade a um ViewModel e instanciar algumas variáveis com escopo da classe.

C: após o método onCreate() concluir a execução, a *activity* insere o estado ‘iniciado’ e o sistema invoca o método onStart() que torna a *activity* visível ao utilizador, à medida que a aplicação prepara para inserir o primeiro plano e se tornar iterativa. É nesse método que a aplicação inicializa o código que mantém a UI.

E: após o método onStart () concluir a execução, a *activity* insere o estado ‘retomado’ e vem para o primeiro plano, então o sistema invoca o método onResume(). É nesse estado que a aplicação interage com o utilizador, e a aplicação permanece nesse estado até que algo afete o seu foco, como por exemplo ao receber uma chamada telefônica, se o utilizador navegar para outra *activity* ou o ecrã do dispositivo é desativado.

F: essa transição indica que a *activity* passa do primeiro plano (embora ainda possa estar visível em um modo de várias janelas) e fica parcialmente invisível. O sistema invoca o método onPause(), para pausar ou ajustar operações que não devem continuar e para liberar recursos do sistema. A conclusão do método onPause() não significa que a *activity* saia do estado de ‘pausa’.

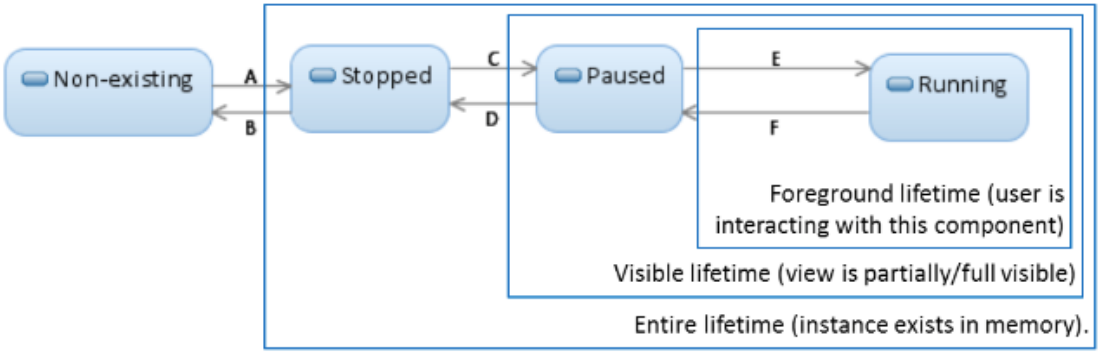
D: quando a *activity* não estiver mais visível ao utilizador, ela irá inserir o estado ‘interrompido’ e o sistema invocará o método onStop(). A aplicação libera ou ajusta recurso desnecessários enquanto não estiver visível ao utilizador, e realiza operações mais robustas que não foram realizadas no método onPause().

B: quando a *activity* deixa de operar, o sistema invoca o método onDestroy() antes de destruí-la, que libera todos os recursos ainda não liberados pelos métodos anteriores.

b) How can a programmer open Activity B in response to a user action in Activity A, passing parameters (e.g.: when user click a button in Activity A you should open Activity B, passing the content of a text field)?
Para iniciar a *activity* **B** e permitir a passagem de parâmetro, é preciso invocar o método startActivity (Intent) a partir da *activity* **A** (neste método pequenas quantidades de dados podem ser adicionados).

c) Which sequence of transactions (from the diagram) are associated with the previous programming case (with respect to Activity A and Activity B)?
Quando uma *activity* inicia outra, ambas passam por transições no ciclo de vida. A primeira atividade para de operar e entra em estado de 'pausa' ou 'interrompido' enquanto a outra *activity* é criada.

1. O método onPause() da *activity* A é executado
2. Os métodos onCreate(), onStart() e onResume() da *activity* B são executados em sequência (a *activity* B agora tem o foco do utilizador)
3. Em seguida, quando a *activity* A não estiver mais visível no ecrã, o método onStop() será executado



Q2.
Consider the following excerpt from an Android manifest.

Consulta: https://developer.android.com/guide/topics/manifest/manifest-intro

a) What component-specific concerns, if any, should the developer take when implementing the PhoneCallStateReceiverclass?
O elemento <receiver> é a declaração de um componente receptor de broadcast, para permitir que a aplicação responda a *intents*, as chamadas de telefone e mensagens SMS para a aplicação em causa.
Para o sistema identificar os componentes que podem responder a um *intent*, compara-se o *intent* recebido com os <intent-filter> contidos no *receiver*.
O *intent-filter* precisa conter um ou mais sub-elementos <action>, para especificar as ações em um filtro.

b) Receivers are associated with an event-oriented programing style. Can this architectural-pattern be use to implement the programming feature describe in previous questions Q1.b) ? Provide a detailed answer.
TO DO

```
<receiver android:name=".PhoneCallStateReceiver">
  <intent-filter>
    <action android:name="android.intent.action.PHONE_STATE"/>
    <action android:name="android.intent.action.NEW_OUTGOING_CALL"/>
  </intent-filter>
</receiver>

<receiver android:name=".SMSReceiver">
  <intent-filter>
    <action android:name="android.provider.Telephony.SMS_RECEIVED" />
  </intent-filter>
</receiver>
```

Q3.
Observe the following code excerpt (the lines were number for reference).

Consulta: https://developer.android.com/guide/fragments

a) Describe the two ways to add a Fragment to an Activity. Which one is used in this example?
Um *fragment* pode ser adicionado a uma *activity*:
   1. de forma estática, inserido em um ficheiro XML de layout e assim permanece na *activity* da aplicação durante todo o tempo
   2. de forma dinâmica, o *layout* da *activity* deve incluir um container para um *fragment* que deve ser instanciado tal como um FragmentManager, para em seguida ser utilizado através das transações (como por exemplo *add*, *remove* e *replace*)

b) Explain the intended use of the three parameters in the call transaction.replace(line #3).
Neste exemplo o novo *fragment*, MediaBrowserFragment(), substitui qualquer *fragment* (se houver) que estiver no contâiner do *layout* com o identificador R.id.container, e o FRAGMENT_TAG é o tag name associado ao novo *fragment* para ser recuperado posteriormente.

c) Well-programmed Fragments would favor reusability. How can a Fragment J communicate with another Fragment H in the same Activity, while preserving its ability to be reused in future Activities (which may use J but not H)?
Para reutilizar os *fragments*, é preciso criá-los como componentes que serão associados a uma *activity* e conectados à lógica da aplicação.

Todas as comunicações entre *fragments* devem passar pela *activity* que os hospedam, um *fragment* comunica com a *activity*, e a *activity* comunica com o outro *fragment*.
Para que os *fragments* J e H se comuniquem, podem compartilhar um ViewModel utilizando o escopo da *activity* para fazer o processamento, em que ambos recebem a mesma instância do ViewModel.

Consulta: https://developer.android.com/guide/fragments/communicate

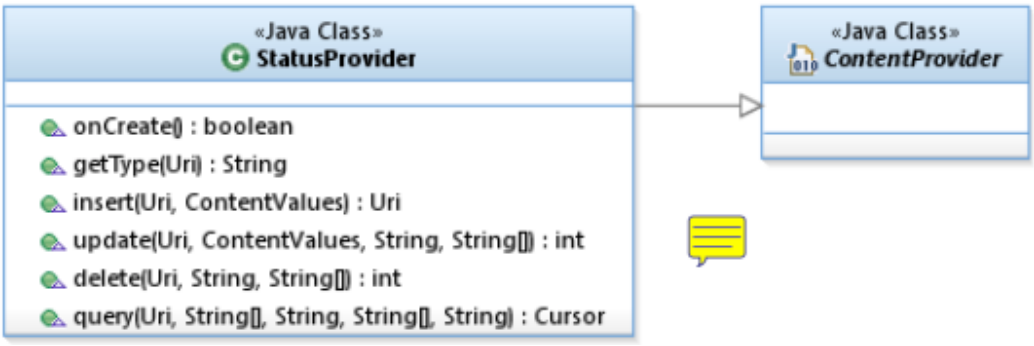
```
1.      MediaBrowserFragment fragment = new MediaBrowserFragment();
2.      FragmentTransaction transaction = getFragmentManager().beginTransaction();
3.      transaction.replace(R.id.container, fragment, FRAGMENT_TAG);
4.      if (mediaId != null) {
5.          transaction.addToBackStack(null);
6.      }
7.      transaction.commit();
```

Q4.
The StatusProvider class, presented in the diagram, implements a ContentProvider.

Consulta: https://developer.android.com/guide/topics/providers/content-provider-basics#java

a) Several methods in the StatusProvider make use of a URI as the first parameter. What is the meaning/content of this parameter?
O Uri de conteúdo é um URI que identifica dados em um provedor. URIs de conteúdo contêm um caminho que aponta para a autoridade do provedor e uma tabela a ser acessada.

b) What kind of data storage mechanism is provided in the base implementation of the ContentProvider component? Which choices does the programmer have in this context? Provide a detailed answer.
O ContentProvider gerencia o acesso a um repositório central de dados na aplicação para diversos componentes e APIs diferentes, na forma de uma ou mais tabelas similares às que são encontradas em um banco de dados relacional.
O provedor recebe solicitações de dados de clientes, realiza a ação solicitada e retorna os resultados. Os métodos fornecem as funções básicas “CRUD” do armazenamento persistente.
A interação entre um cliente e um provedor permite recuperar dados através de consultas, inserir dados, atualizar valores em uma linha e excluir linhas na tabela.



Q5.

The following excerpt shows parts of the manifest file for a given app.

a) How could the component "SettingsActivity" start the component "StorageService"? Explain your assumptions/options and show the relevant code.
O elemento <activity> declara uma *activity* para a aplicação, e as *activities* não declaradas no ficheiro Manifest não ficam visíveis ao sistema e nunca são executadas. O elemento <service> é utilizado para declarar um componente para realizar algum trabalho sem uma interface de usuário.
Um serviço pode ser iniciado de uma *activity* através do método startService() ou startForegroundService() e passa o *Intent* a ele que especifica qual serviço iniciar.

```
Intent(this, StorageService::class.java).also  {
                intent -> startService(intent)
}
```

Consulta: https://developer.android.com/guide/components/services

b) Which is the title that is likely to be shown (on the top bar) when of the PhoneInfoActivity is active and visible?
O atributo <android:label=…> define o *label* a ser exibido na tela quando a *activity* for apresentada para o utilizador. Se esse atributo não for definido para uma *activity*, o *label* definido para a aplicação é que será exibido.
O título que será exibido quando a *activity* PhoneInfoActivity estiver ativa é a *string* definida no ficheiro de *Strings* '@string', com a identificação "title_activity_phone_info".

Consulta: https://developer.android.com/guide/topics/manifest/activity-element

c) Which components will be started in a different (separate) process?
??? (não tenho certeza se a pergunta é sobre isso)
O componente <receiver>, GlobalBroadcastReceiver, declara ações para receber transmissões do sistema.

Consulta: https://developer.android.com/reference/android/content/Intent

```
 […]
        <uses-permission android:name="android.permission.BLUETOOTH" />
        <uses-permission android:name="android.permission.INTERNET" />
[…]
        <service
            android:name=".services.StorageService"
            android:enabled="true"
            android:exported="true"/>
        <activity
            android:name=".SettingsActivity"
            android:label="@string/title_activity_settings"/>
        <receiver
            android:name=".br.GlobalBroadcastReceiver"
            android:enabled="true"
            android:exported="true">
            <intent-filter>
                <action android:name="android.intent.action.BOOT_COMPLETED" />
                <action android:name="android.bluetooth.adapter.action.STATE_CHANGED" />
                <action android:name="android.intent.action.BATTERY_CHANGED" />
            </intent-filter>
        </receiver>
        <activity
            android:name=".PhoneInfoActivity"
            android:label="@string/title_activity_phone_info"
        </activity>
        <service
            android:name=".services.CheckUpService"
            android:enabled="true"
            android:exported="true"/>
        <activity
            android:name=".TargetActivity"
            android:label="@string/app_name"
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
[…]
```

Q6.

The following statementswere taken from developer.android.com: “There are simply two rules to Android's single thread model: (1) do not block the UI thread; (2) Do not access the Android UI toolkit from outside the UI thread”.

Consulta: https://developer.android.com/guide/components/processes-and-threads

a) What is the "UI thread"?

Quando uma aplicação é executada, o sistema cria um *thread* de execução para ele, que é chamada de “principal” Esse *thread* é encarregado de despachar eventos para os *widgets* adequados da interface do utilizador. Quase sempre, é também o *thread* em que a aplicação interage com componentes do *kit* de ferramentas da UI do Android. Portanto, o *thread* principal também pode ser chamado de *thread* de UI.

O sistema não cria um *thread* separado para cada instância de um componente. Todos os componentes executados no mesmo processo são instanciados no

thread de UI, e as chamadas do sistema para cada componente são despachadas a partir deste *thread*.

b) In the context of the stated "rules", explain a programming strategy to address the problem of getting a resource from a remote API (may be affected by latency), while keeping the possibility to update a progress bar in the UI (assume that you will get the resource in several segments).
Por causa do modelo de *thread* único descrito acima, é essencial para a capacidade de resposta da UI da aplicação que não bloqueie o *thread* de UI. Caso tenha de realizar operações que não sejam instantâneas, é preciso fazer isso em *threads* separados (*threads* de "segundo plano" ou "de trabalho").
AsyncTask permite o trabalho de forma assíncrona na interface do usuário. Ela realiza operações de bloqueio em um *thread* de trabalho e, em seguida, publica os resultados no *thread* de UI.
Para usá-la, é preciso atribuir a subclasse AsyncTask e implementar o método doInBackground(), que executa em um conjunto de *threads* de segundo plano. Para publicar atualizações no *thread* da UI enquanto a computação a computação em segundo plano ainda está em execução, cada chamada a este método acionará a execução do método onProgressUpdate(Progress...) que retorna os valores de progresso para atualizar a UI.

Q7.

Consider the following programming problems, related to the implementation of an application that allows the user to order parcel delivery services in his mobile device. The application allows the user to create a new order, define the package and the destination characteristics, and track the ongoing delivery. Describe in detail the implementation strategy/options that you would recommend for the following programming problems, related to this example, as if you were explaining to a junior programmer.You do not need to show code (but you may).

a) "The first screen should use a composition of three Fragments, if running in a tablet, or just the order creation Fragment, if running on a standard smartphone."
Para oferecer uma boa experiência ao utilizador é preciso oferecer uma UI adequada a diferentes tamanhos de tela, oferecendo recursos de *layout* alternativos para otimizar o *design* da UI para determinados equipamentos.
Ao projetar a aplicação para ser compatível com *tablets* e telemóveis, pode-se reutilizar os *fragments* em diferentes configurações de *layout* com base no espaço de tela disponível.
Quando aplicação está a ser executada em um dispositivo do tamanho de um *tablet*, pode incorporar os três *fragments* na *activity*. No entanto, em uma tela de tamanho de um telemóvel, não há espaço suficiente para os três *fragments*, então a *activity* inclui somente o *fragment* de criação de um novo pedido, e os *fragments* das outras operações devem ser inicializados em uma nova (ou mais)

activity. Portanto, a aplicação fica por ser compatível com *tablets* e telemóveis por meio da reutilização dos *fragments* em combinações diferentes.

Consulta: https://developer.android.com/guide/components/fragments

b) "When creating the new order, the app should assume the current location of the user as the default pick-up location."
É possível utilizar as APIs de localização para que a app solicite a última localização conhecida do dispositivo do utilizador e que geralmente é equivalente à última localização conhecida do dispositivo.
Para que a aplicação utilize os recursos de localização é preciso solicitar permissões de localização na app ao utilizador.
Ao criar um novo pedido, através da API Fused Location Provider, cria-se uma instância cliente desse provedor de localização na *activity* e depois poderá ver a última localização conhecida do dispositivo do utilizador através do método getLastLocation() que retorna as coordenadas de latitude e longitude de uma localização geográfica, para preenchimento da localização *default* de coleta do pedido.
*Também é possível ver a localização atual através do método getCurrentLocation() que recebe uma localização mais atualizada e precisa, no entanto pode fazer com que a computação ativa de localização ocorra no disposto.

Consulta: https://developer.android.com/training/location/retrieve-current

c) "The end-user should track the changing position of his/her package in a map and view regular updates of the actual location."
É possível solicitar atualizações regulares da localização de um dispositivo através do método requestLocationUpdates() no provedor de localização combinada, citado na resposta acima. O último local conhecido do dispositivo oferece uma base útil como ponto de partida, garantindo que a app tenha uma localização conhecida antes de iniciar as atualizações periódicas.
O provedor de localização combinada invoca o método de *callback* LocationCallback.onLocationResult() quando um novo resultado de localização estiver disponível com a precisão e a frequência de atualizações afetadas pelas permissões de localizações solicitadas e pelas opções definidas no objeto da solicitação.

Consulta: https://developer.android.com/training/location/request-updates