ANDROID

RISPOSTE ALLE DOMANDE

A01) Dopo aver descritto sinteticamente in cosa consista Android si dica quale vantaggio porta ART su Dalvik VM (JIT vs AOT).

Android è un sistema operativo per dispositivi mobili lanciato al pubblico nel 2008. Oltre ad essere basato sul kernel di Linux ha la particolarità di essere anche una piattaforma di sviluppo. Esso infatti sfrutta una particolare Java Virtual Machine nota come Dalvik Virtual Machine (DVM) utile sui dispositivi con poca memoria. La DVM è in grado di eseguire il bytecode in output dalla compilazione dell'app, rendendo il codice accessibile a tutti i dispositivi. Questa VM si basa sulla tecnologia JIN (Just in time) con cui il codice viene compilato ed eseguito in real time ad ogni esecuzione dell'app, ciò può comportare dei tempi di caricamento più lunghi. Per ovviare a questo problema è stato creato il compilatore AOT (ahead of time) che compila l'applicazione durante l'installazione. Questo garantisce un guadagno in termini di prestazioni a discapito dei tempi di installazione e della memoria del dispositivo.

A02) Dopo aver descritto brevemente in cosa consiste un BroadcastReceiver si dia uno skeleton di codice per una classe che ne realizzi uno e si indichino nel dettaglio quali modifiche vanno fatte al file AndroidManifest.xml

Un BroadcastReceiver è un componente Android che si occupa della sottoscrizione agli eventi di un'applicazione e di sistema. Tutte le applicazioni registrate ad un determinato evento riceveranno di conseguenza una notifica da parte di Android quando esso avviene (es ACTION_BOOT_COMPLETED). Possiamo quindi affermare che il BR si limita a rispondere ad un messaggio broadcast. Un'applicazione può inviare un broadcast per notificare ad altre applicazioni che è stato eseguito un evento particolare che può dare, ad esempio, accesso a risorse prima non disponibili.

La creazione del BR si basa su due fasi:

- Creazione: creazione del codice

```
public class MyBroadcastReceiver extends BroadcastReceiver{

    @Override
    public void onReceive(Context context, Intent intent) {
        Toast.makeText(context, "VIBRATE",
                Toast.LENGTH_LONG).show();
        // Vibrate the mobile phone
        Vibrator vibrator = (Vibrator) context.getSystemService(Context.VIBRATOR_SERVICE);
        vibrator.vibrate(300);
    }
}
```

- Registrazione: il BR viene registrato in AndroidManifest.xml

```
<application
   android:icon="@drawable/ic_launcher"
   android:label="@string/app_name"
   android:theme="@style/AppTheme" >
   <receiver android:name="MyReceiver">

      <intent-filter>
         <action android:name="android.intent.action.BOOT_COMPLETED">
         </action>
      </intent-filter>

   </receiver>
</application>
```

Android.intent.action.BOOT_COMPLETED: evento generato quando viene acceso il telefono.

A03) Dopo aver descritto brevemente in cosa consiste un Broadcast Receiver si dica quali sono le modifiche AndroidManifest.xml affinché possa essere attivato da un messaggio identificato dalla stringa "BLA BLA BLA"

Vedi "A02" per la descrizione del BR;

```
<receiver android:name="MyBroadcastReceiverXML" >
    <intent-filter>
        <action android:name="BLA BLA BLA" />
    </intent-filter>
</receiver>
```

A04) Dopo aver descritto brevemente in cosa consiste un Service si descriva come un Intent possa essere passato al servizio di Allarme

Un Service è un componente di un'applicazione Android che esegue normalmente delle operazioni lunghe in background, senza necessitare di interazioni con l'utente. Questo componente continua l'esecuzione anche dopo l'uscita dall'app. Un componente può inoltre collegarsi al Servizio in modo da poter interagire con esso e può addirittura eseguire quella che viene definita "Interprocess Communication (IPC)", cioè comunicazioni fra

processi. Un esempio di servizio è quello per la riproduzione musicale.

Ci sono tre tipi differenti di Service:

1. Scheduled: verrà eseguito secondo i criteri definiti nel rispettivo JobScheduler;
2. Started: è stato chiamato il metodo "*.startService()*". Una volta iniziato il Service può continuare all'infinito anche dopo la distruzione dell'app;
3. Bound: quando uno o più componenti vengono abbinati al Service tramite "*.bindService()*". Un Bound Service offre un'interfaccia client-server che permette lo scambio di informazioni. Il Service rimane attivo fino a quando l'ultimo componente è abbinato ad esso, dopodiché viene distrutto.

Le funzioni di callback più importanti sono:

1. onStartCommand(): invocata con il .startService(). Fa partire il Service e dovrai interromperlo da codice;
2. onBind(): invocato con .bindService().
3. onCreate(): eseguito dopo 1 o 2;
4. onDestroy(): eseguito prima della distruzione del Service;

Infine bisogna tener conto che il Service lavora sul Thread principale non su uno a parte o su un processo diverso, per evitare ciò va gestito a codice dall'utente.

Per passare un Intent al Service si può operare così:

```
public void startService(View view){
    String esempioStringa = "Stringa di prova";
    Intent intent = new Intent(this, AllarmeService.class);
    intent.putExtra("esempioStringa", esempioStringa);
    startService(intent);
}
```

t

A05) In cosa consiste un Intent, come viene creato e qual è l'eventuale relazione che esiste fra un Intent ed un PendingIntent?

Un Intent è una descrizione astratta di un'operazione che deve essere eseguita. Esso fornisce la possibilità di eseguire un collegamento tra i codici di applicazioni differenti anche dopo che sono state lanciate. In ogni caso il suo utilizzo primario è quello che viene sfruttato al lancio di una nuova Activity. L'Intent infatti può essere definito come la colla tra due Activity contenente la descrizione astratta di un'azione da eseguire. La sua struttura interna è formata da:

- ----------PRIMARI----------
- Action: l'azione da eseguire;
- Data: i dati su cui deve operare;
- ----------SECONDARI----------
- Category: informazioni aggiuntive dell'action;
- Type: il tipo di dati dell'intent;
- Component: specifica il componente su cui usare l'intent;
- Extras: è un Bundle contenente informazioni aggiuntive secondo la logica key-value;

L'Intent può essere espresso in due forme primarie:

- Intent implicito: non ha un componente specificato e richiede di conseguenza di avere abbastanza informazioni per comprendere quale sia il componente migliore da eseguire;
- Intent esplicito: ha un componente specificato che definisce la classe del componente da eseguire. Spesso viene utilizzato semplicemente per lanciare un'Activity, senza includere altre informazioni.

Un tipo particolare di Intent è il PendingIntent. Normalmente quando un Intent viene mandato ad un'altra applicazione, nel momento in cui essa lo esegue, lo fa con i suoi livelli di permessi. Per questo entra in gioco il PendingIntent. Esso permette di essere eseguito in un futuro non definito (quindi non necessariamente subito) e al momento dell'esecuzione essa avviene con i tuoi stessi permessi e la tua stessa identità. In questo modo un'altra applicazione può sfruttare il PendingIntent per eseguire un'azione a lui negata sotto la responsabilità

dell'applicazione che gliel'ha fornito. Essendo questo un sistema di "privilege escalation" bisogna stare molto attenti quando lo si utilizza specificando esplicitamente il componente a cui deve essere spedito così da impedire che sia utilizzato su altri componenti.

A06) Dopo aver descritto in cosa consista un Activity si descriva in che modo la si inserisce in AndroidManifest.xml

L'Activity è uno dei componenti basici di un'applicazione Android. Essa si occupa di fornire un'interfaccia a singola finestra con cui l'utente può interagire e scambiare informazioni. L'Activity si può presentare all'utente in modi differenti tra cui: "a schermo intero", fluttuante o inserita in un'altra activity.

Le Activity sono gestite dal sistema come uno stack, cioè la nuova activity viene posta in cima alle altre diventando l'applicazione attiva.

Il ciclo di vita di un'Activity è molto basilare e si basa su quattro stati essenziali:

1. Active/Running: l'applicazione è in primo piano;
2. Paused: quando non è in primo piano ma è ancora visibile;
3. Stopped: quando è oscurata da un'altra Activity. Può essere distrutta per guadagnare memoria;
4. Destoryed: quando viene terminata dall'utente o dal sistema.

Per inserire un'Activity in AndroidManifesr.xml bisogna utilizzare il tag <activity> e l'attributo "android:name" per assegnare il riferimento al nome della classe corretta.

```
<activity android:name=".MainActivity">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />
        <category android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

A07) Dopo aver descritto in cosa consista un Activity si dica quali sono i 3 stati in cui "essenzialmente" si può trovare un Activity descrivendoli brevemente.

Vedi "A06".

A08) Dopo aver descritto cosa sia un'Activity si descriva brevemente quali sono i 2 metodi che segnano l'ingresso e l'uscita nel ciclo in cui l'activity prende il focus.

Vedi "A06".

I metodi onPause() e onResume() sono due metodi di callback fondamentali nella gestione dell'activity. Essi infatti gestiscono l'entrata e l'uscita dall'interazione dell'utente con l'interfaccia. Il metodo onPause() viene richiamato quando l'Activity non è più in foreground, il metodo onResume() è l'opposto invece, cioè quando passa in foreground. Questi due metodi sono gli estremi di quello che viene definito "foreground lifetime", cioè il tempo in cui l'Activity è in primo piano. Questi due metodi vengono richiamati molto frequentemente quindi il codice in esso è preferibile sia leggero.

A09) Una volta fatto il pop dalla cima dello stack un Activity può restituire, tramite Intent, valori all'Activity che ora si trova in cima allo stack, come?

Per poter ottenere valori dall'Activity in cima allo stack bisogna iniziarla tramite il metodo "startActivityForResult()", passandogli l'Intent e un numero che identifica la chiamata. In questo modo al termine della nuova Activity sarà invocata la funzione di callback "onActivityResult()" che contiene tutte le informazioni che sono state passate all'Activity chiamante.

```
        MainActivity.this.startActivityForResult(quizIntent, 1);

    }


    @Override
    public void onActivityResult(int requestCode,int resultCode, Intent data) {
```

Nella nuova Activity invece possiamo gestire i risultati che vogliamo restituire all'Activity chiamante. Questo avviene tramite l'utilizzo della funzione "setResult()". A questa funzione dobbiamo passare il codice del result (es. RESULT_OK, RESUT_CANCELED, RESULT_FIRST_USER) e l'Intent con le informazioni che vogliamo restituire. Questi valori saranno poi passati ai vari campi della funzione "onActivityResult()" dell'Activity chiamante.

A10) Dopo aver descritto brevemente cosa sia un Fragment si dica quanti e chi sono i parametri del metodo onCreateView dandone breve descrizione.

Un Fragment è un pezzo di interfaccia dell'applicazione che può essere inserito nell'Activity. L'interazione con i Fragment è fatta attraverso FragmentManager.

Questo componente è strettamente collegato all'Activity in quanto non può essere usato senza. È un oggetto molto flessibile grazie al numero di composizioni che si possono fare a livello di interfaccia utilizzandone vari. Per questo viene definito un componente modulare, perché si può disporre a modulo all'interno dell'Activity. Esistono vari tipi di Fragment che si possono adattare alle varie esigenze del programmatore, così da poter risparmiare tempo nel corso della creazione dell'app.

Il ciclo di vita del Fragment è in realtà legato a quello dell'Activity, per quanto possa essere anticipato dall'interno del codice. In più tra le funzioni di callback ci sono numerose funzioni che servono a gestire l'interfaccia. La più importante è onCreateView() che viene richiamata dopo l'onCreate() e si occupa di assegnare al fragment il layout contenuto nel suo xml.

I campi di questa funzione sono tre, cioè:

1. LayoutInflater inflater: questo oggetto si occupa di inserire il layout.xml del Fragment all'interno dell'applicazione. Utilizza in particolare il metodo inflater per caricarlo.
2. ViewGroup container: se non è nullo, è il parent view a cui l'UI del Fragment è collegata.
3. Bundle savedInstanceState: se non è nullo, contiene lo stato salvato che deve essere riutilizzato per ricostruire il Fragment.

A11) Dopo aver descritto brevemente cosa sia un Fragment si descriva brevemente cosa sia un il LayoutInflater e si dica a quale service si riferisce.

Vedi “A10”.

Il LayoutInflater si occupa di istanziare il layout.xml nella sua View corrispondente. Non è mai utilizzato direttamente ma è preferibile ottenerlo dal Context essendo già configurato correttamente. Il LayoutInflater fa riferimento al system-service “LAYOUT_INFLATER_SERVICE” e si può utilizzare il metodo getSystemService() per ottenerlo.

A12) In cosa consiste un Intent? Qual è la differenza fa implicit ed explicit intent? Fare degli esempi.

Vedi “A05”.

A13) In cosa consiste un Intent? Dopo aver descritto brevemente cosa sia un Bundle si dicano quali sono i metodi che permettono di passare e recuperare “informazione” attraverso un bundle.

Vedi “A05”.

Il Bundle è un oggetto utilizzato per la mappatura di valori. I metodi principali sono:

1. put*TIPODIDATO*(String key, Paracelable value): permette di inserire nel Bundle un dato corrispondente ad una chiave specifica;
2. get*TIPODIDATO*(String key): ritorna il valore nel Bundle assegnato a quella key;

A14) Nel ciclo di vita di un fragment quando entra in gioco il metodo onAttach(...)? Qual è il ruolo che gioca nella “comunicazione” fra i fragment di una stessa activity?

Il metodo onAttach() è la funzione di callback che viene richiamata nel momento in cui il Fragment viene “attaccato” all’Activity e viene eseguito prima dell’onCreate().

Questo metodo è molto utile perché ci permette di mettere in relazione l’Activity con il Fragment. Uno dei metodi più utilizzati è quello di implementare all’Activity un Listener ad un evento che opera su un Fragment. Quando sul Fragment si verifica quell’evento allora si attiva una funzione di callback sull’Activity che va ad eseguire del codice presente sugli altri Fragment. In questo modo l’Activity si trova a giocare un ruolo da intermediario fra i vari Fragment gestendo le informazioni e distribuendole ai componenti.

A15) Per associare i fragment al layout di un'activity abbiamo a disposizione due vie: tag xml e programmatica: le si descriva sinteticamente.

Per associare un Fragment al suo Layout corretto si possono utilizzare due metodi:

- Tramite il tag XML.

```
<fragment android:name="com.example.android.fragments.NomeFragment"
          android:id="@+id/headlines_fragment"
          android:layout_weight="1"
          android:layout_width="0dp"
          android:layout_height="match_parent" />
```

  Come possiamo vedere dal codice, in questo modo possiamo definire all'interno dell'XML un Fragment generico che viene poi abbinato alla classe indicata. Il tag verrà poi sostituito dal Layout del Fragment stesso quando verrà richiamato il metodo onCreateView all'interno di esso.
- Programmaticamente: per inizializzare il Layout esso viene "agganciato " ad un componente già dichiarato all'interno dell'XML. Questo componente diventa quindi il container del Fragment e andando a modificare le sue proprietà, modificheremo indirettamente quelle del Fragment stesso.
  Per ottenere questa situazione si utilizza il FragmentTransaction. Questo oggetto permette di eseguire una serie di operazioni sui Fragment associati con il suo FragmentManager. È composto da una stack di operazioni che vengono eseguite al momento del .commit(). Per ottenere un FragmentTransaction si utilizza il metodo fragmentManager.beginTransaction(). Successivamente possiamo aggiungere un Fragment utilizzando il seguente codice:

```
ExampleFragment fragment = new ExampleFragment();
fragmentTransaction.add(R.id.fragment_container, fragment);
fragmentTransaction.commit();
```

  Con il commit il Fragment verrà aggiunto al container indicato nel metodo .add().

A16) Si supponga di avere un button con un'etichetta (testo). Si descriva almeno una strategia per far si che al click venga visualizzato un dialog mediante fragment in modo che questi riporti come testo la suddetta etichetta.

Dopo aver creato la classe del dialog fragment, bisogna aggiungere un onClickListener sul button. In questo modo, quando viene cliccato, possiamo, tramite il FragmentTransaction, avviare il Dialog passandogli l'etichetta contenuta nel pulsante. Verrà di conseguenza eseguito il metodo onCreateView() del Dialog Fragment possiamo prendere l'etichetta dal bundle e mostrarla a video.

A17) Nella gestione di una ListView, dopo aver descritto brevemente qual è il ruolo dell'ArrayAdapter, si illustri sinteticamente il pattern del ViewHolder.

L'ArrayAdapter è un oggetto che fa da intermediario tra la fonte di dati da caricare all'interno del ListView. Come dice il nome, esso si occupa di inserire il contenuto di un array o di un ArrayList all'interno della lista, utilizzando per creare ogni voce di essa un pattern definito in precedenza all'interno di un file XML (per esempio un TextView).
Per migliorare la performance si utilizza di solito un CustomAdapter a cui si applica un pattern ViewHolder, che aiuta a velocizzare il popolamento della ListView in modo più leggero. Questo avviene tramite la creazione

nell'Adapter di una classe statica, il ViewHolder appunto, che contiene una cache dei findByView() chiamati dall'applicazione. Tramite il metodo .getView() possiamo controllare se la View è una riga riciclata oppure se bisogna crearne una nuova, in questo modo possiamo gestire il salvataggio della View nel ViewHolder. Il metodo .getView() è ereditato dall'ArrayAdapter e prende tre parametri:

- int position: la posizione della View nell'adapter;
- View convertView: è una View inutilizzata che viene riusata per creare la nuova View.
- ViewGroup parent: il parent dove bisogna aggiungere la View.

Il funzionamento del pattern avviene così: se convertView è nullo, significa che bisogna creare una nuova View, di conseguenza ne viene creata una nuova che viene salvata all'interno di un nuovo ViewHolder. Il ViewHolder viene poi assegnato tramite il metodo .setTag() alla convertView; nel caso invece in cui convertView non sia nullo, allora si può accedere al suo ViewHolder, tramite il metodo .getTag(), e avere un accesso diretto alle View da modificare in quanto sono salvate all'interno di esso;

Per funzionare hanno quindi una rilevante importanza i metodi:

- getTag();
- setTag();

Essi permettono di gestire una "memoria" alle View potendo mantenere delle informazioni.

> *A19) Qual è il ruolo della classe SQLiteOpenHelper? Nella stringa MYDBCREATE c'è SQL per creare un DB, MYDBCREATE è elemento static per la classe MyHelper che estende SQLiteOpenHelper: si scriva per esteso per MyHelper il metodo onCreate specificando parametri e tipo di ritorno.*

È una classe di supporto per la gestione e la creazione di SQLiteDatabase. Si crea una sottoclasse che implementa metodi onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int) e opzionalmente onOpen(SQLiteDatabase). In pratica la classe si occupa di aprire il database nel caso esso esista, di crearlo nel caso non esista, e di aggiornarlo se necessario. Le transazione vengono utilizzate per assicurarsi che il database rimanga in uno stato di integrità.

```
public void onCreate(SQLiteDatabase database){
    database.execSQL(MYDBCREATE);
}
```

Come parametro “database” passeremo l’oggetto in cui verrà gestito il database.
Tramite il metodo execSQL passeremo la stringa statica MYDBCREATE che verrà eseguita creando e salvando il database.

> *A20) Qual è il ruolo della classe SQLiteOpenHelper? Si descriva sinteticamente qual è il ruolo dei metodi onCreate(), onConfigure() e onUpgrade().*

È una classe di supporto per la gestione e la creazione di SQLiteDatabase. Si crea una sottoclasse che implementa metodi onCreate(SQLiteDatabase), onUpgrade(SQLiteDatabase, int, int) e opzionalmente onOpen(SQLiteDatabase). In pratica la classe si occupa di aprire il database nel caso esso esista, di crearlo nel caso non esista, e di aggiornarlo se necessario. Le transazione vengono utilizzate per assicurarsi che il database rimanga in uno stato di integrità.

onCreate: Chiamato quando il database viene creato per la prima volta, si occupa della creazione delle tabelle e della loro iniziale popolazione.

onConfigure: Chiamato quando viene configurata la connessione con il database per abilitare funzioni come e il supporto alle chiavi esterne. Questo metodo è chiamato prima di onCreate, onUpgrade e onOpen. Non dovrebbe modificare il database eccetto per la sua configurazione.

onUpgrade: Chiamato quando il database necessità di un aggiornamento Questo metodo dovrebbe essere utilizzato per l'aggiunta e l'eliminazione delle tabelle, o per compiere qualunque altra operazione che richieda un aggiornamento alla nuova versione del database.

A21) Si descriva brevemente la classe SQLiteDatabase. Come vengono usati i metodi void execSQL(String sql) e Cursor query(String table, String[] columns, String selection, String selectionArgs, String groupBy, String having, String orderBy, String limit)

La classe SQLiteDatabase mette a disposizione metodi per la gestione di un database SQLite. Contiene metodi per creare, eliminare, eseguire comandi SQL e compiere altre operazione di gestione del database.

execSQL: Metodo per l'esecuzione di una query che non ritorna un dato. Il tipo di return è void e riceve come parametro una stringa contenente un'istruzione SQL. Viene utilizzato in genere per la creazione della struttura del database.

query: Metodo che ha come return un oggetto Cursor relativo al ResultSet.

table: La tabella in cui verrà eseguita la query.
columns: Le colonne che verranno restituite dall'istruzione.
selection: Filtro che dichiara quali righe devono essere ritornate. Formattate come un WHERE. Se viene passato il valore null verranno ritornate tutte le righe.
selectionArgs: Valori che sostituiranno i "?" nella stringa selection.
groupBy: Filtro che dichiare come ragruppare le righe. Formattate come un GROUP BY. Se viene passato il valore null non sono raggruppati.
having: specifica quali gruppi di righe ritornare, usando una clausola SQL HAVING.
orderBy: specifica come ordinare le righe, secondo la clausola SQL ORDER BY.
limit: limita il numeri di righe che il cursore ritorna, secondo la clausola LIMIT.

A22) In un Adapter per una RecyclerView come viene implementato il pattern del ViewHolder?

Per implementare il RecyclerView bisogna in primis aggiungerlo al build.gradle perché sia compilato. In seguito possiamo estendere al nostro Adapter la classe RecyclerView.Adapter e estendere alla classe statica ViewHolder la classe RecyclerView.ViewHolder. Vengono quindi implementati due metodi all'Adapter che sono:

- onCreateViewHolder(): che svolge tutte le operazioni che servono per la creazione e la popolazione del ViewHolder;
- onBindViewHolder(): che popola l'oggetto prendendo le informazioni dal suo ViewHolder.