CMR

INSTITUTE OF

TECHNOLOGY

USN | | | | | | | | | | |

Third Internal Test

Sub:	MOBILE APPLICATION DEVELOPMENT							Code:	**15cs661**
Date:	23 / 05 / 2018	Duration:	90 mins	Max Marks:	50	Sem:	VI	Branch:	**ISE,CSE**

Answer Any **FIVE FULL** Questions

		Marks	OBE	
			CO	RBT
1(a)	How to Schedule an alarm in android?	[5]	CO3	L3
(b)	What is job scheduler? What are the different components of it?	[5]	CO3	L2
2 (a)	What are the different Data Storage options in android? Explain? Differentiate between internal and external storage?	[10]	CO4	L3
3 (a)	Explain about SQLite database? Explain ACID properties?	[5]	CO4	L4
(b)	How to creating a shared preferences file and how to clear it?	[5]	CO4	L4
4 (a)	Explain App Architecture with a Content Provider? How to Implement a Content Provider?	[10]	CO4	L3

5 (a)	How to monitor the performance of your running app ? How can you make sure your app treats the user's data safely.	[10]	CO5	L4
6 (a)	Explain about Firebase? Explain any 4 applications of Firebase.	[10]	CO6	L4
7 (a)	What is an APK? How to publish your App in Google Play Store ? .	[10]	CO6	L3

1.How to Schedule an alarm in android?

Elapsed Real Time+ Real Time Clock (RTC)—2.5

Wakeup behavior+schedule-2.5

In Android, the AlarmManager lets us schedule work to happen at a specific time. If your app has work to do and you'd like it to do that work even when the app isn't running, then the AlarmManager might be a good solution.

- ✕ Fire intents at set times or intervals
- ✕ Goes off once or recurring
- ✕ App does not need to run for alarm to be active

Benefits of alarms

- ✕ App does not need to run for alarm to be active
- ✕ Device does not have to be awake
- ✕ Does not use resources until it goes off

- Use with BroadcastReceiver to start services and other operations

Measuring time

- Elapsed Real Time—time since system boot
    - Independent of time zone and locale
    - Use for intervals and relative time
    - Elapsed time includes time device was asleep
- Real Time Clock (RTC)—UTC (wall clock) time
    - When time of day at locale matter

Wakeup behavior

- Wakes up device CPU if screen is off
    - Use only for time critical operations
    - Can drain battery
- Does not wake up device
    - Fires next time device is awake
    - Is polite

What is AlarmManager

- AlarmManager provides access to system alarm services
- Schedules future operation
- When alarm goes off, registered Intent is broadcast
- Alarms are retained while device is asleep
- Firing alarms can wake device

```
AlarmManager alarmManager =    (AlarmManager) getSystemService(ALARM_SERVICE);
```

Scheduling Alarms

What you need to to schedule an alarm

1. Type of alarm
2. Time to trigger
3. Interval for repeating alarms
4. PendingIntent to deliver at the specified time

   (just like notifications)

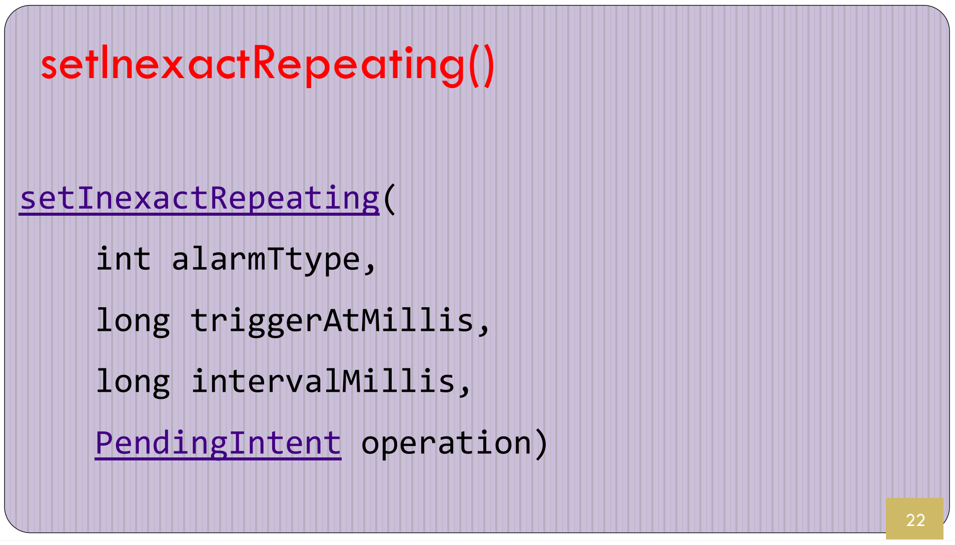
Schedule a single alarm

- set()—single, inexact alarm
- setWindow()—single inexact alarm in window of time
- setExact()—single exact alarm

More power saving options AlarmManager API 23+

Schedule a repeating alarm

- setInexactRepeating()
  - repeating, inexact alarm
- setRepeating()
  - Prior to API 19, creates a repeating, exact alarm
  - After API 19, same as setInexactRepeating()



setInexactRepeating()

```
setInexactRepeating(
    int alarmTtype,
    long triggerAtMillis,
    long intervalMillis,
    PendingIntent operation)
```

22

1.b. What is job scheduler? What are the different components of it?

Job schedule+components(2.5+2.5)

- Used for intelligent scheduling of background tasks
- Based on conditions, not a time schedule
- Much more efficient than AlarmManager
- Batches tasks together to minimize battery drain
- API 21+ (**no support library**)

Job Scheduler components

- 1.JobService—Service class where the task is initiated
- 2.JobInfo—Builder pattern to set the conditions for the task- eg:only in wifi.
- 3.JobScheduler— Schedule and cancel tasks, launch service

JobService

- JobService subclass
- Override
  - onStartJob()
  - onStopJob()

- **Runs on the main thread**
    - onStartJob()

- Implement work to be done here
- Called by system when conditions are met
- Runs on main thread
- **Off-load heavy work to another thread**
- **FALSE—**Job finished
- **TRUE** Work has been off-loaded
- Must call **jobFinished()** from the worker thread

onStopJob()

Called if system has determined execution of job must stop

... because requirements specified no longer met

For example, no longer on WiFi, device not idle anymore

Before jobFinished(JobParameters, boolean)

Return TRUE to reschedule

Register your JobService

```
<service
  android:name=". MyJobService "
  android:permission=
     "android.permission.BIND_JOB_SERVICE"/>
```

`JobInfo`

A unit of work is encapsulated by a `JobInfo` object. This object specifies the scheduling criteria. The job scheduler allows to consider the state of the device, e.g., if it is idle or if network is available at the moment. Use the `JobInfo.Builder` class to configure how the scheduled task should run. You can schedule the task to run under specific conditions, such as:

- Device is charging
- Device is connected to an unmetered network
- Device is idle
- Start before a certain deadline

Eg: **setMinimumLatency(long minLatencyMillis)**

setOverrideDeadline(long maxExecutionDelayMillis)

setPeriodic(long intervalMillis)

setPersisted(boolean)

```
JobInfo.Builder builder = new JobInfo.Builder(
      JOB_ID, new ComponentName(getPackageName(),
      NotificationJobService.class.getName()))
   .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
   .setRequiresDeviceIdle(true)
   .setRequiresCharging(true);
JobInfo myJobInfo = builder.build();
```

Scheduling the job

1. Obtain a JobScheduler object form the system
2. Call schedule() on JobScheduler, with JobInfo object

```
mScheduler =(JobScheduler)getSystemService(JOB_SCHEDULER_SERVICE);

mScheduler.schedule(myJobInfo);
```

2. What are the different Data Storage options in android? Explain? Differentiate between internal and external storage?
   5 categories*2)

- Files
- Internal Storage
- External Storage
- SQLite Database
- Other Storage Options

Shared Preferences—Private primitive data in key-value pairs

Internal Storage—Private data on device memory

External Storage—Public data on device or external storage-sd card

SQLite Databases—Structured data in a private database

Content Providers—Store privately and make available publicly

- Network Connection—On the web with your own server
- Cloud Backup—Back up app and user data in the cloud
- Firebase Realtime Database—Store and sync data with NoSQL cloud database across clients in realtime-videos or images

Internal storage

- Always available
- Uses device's filesystem
- Only your app can access files, unless explicitly set to be readable or writable
- On app uninstall, system removes all app's files from internal storage
- App always has permission to read/write
- Permanent storage directory—getFilesDir()

File file = new File(context.getFilesDir(), filename);

Use standard java.io file operators or streams
to interact with files

External storage

- Not always available, can be removed
- Uses device's file system or physically external storage like SD card
- World-readable, so any app can read
- On uninstall, system does not remove files private to app

Internal is best when

- you want to be sure that neither the user nor other apps can access your files

External is best for files that

- don't require access restrictions and for
- you want to share with other apps
- you allow the user to access with a computer
- Set permissions in Android Manifest
- Write permission includes read permission

- If there is not enough space, throws IOException

- If you know the size of the file, check against space

    - getFreeSpace()

    - getTotalSpace().

- If you do not know how much space is needed
    - try/catch IOException

SQLite Database

- Ideal for repeating or structured data, such as contacts
- Android provides SQL-like database

Shared Preferences

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage

Firebase Realtime Database

- Connected apps share data
- Hosted in the cloud
- Data is stored as JSON
- Data is synchronized in realtime to every connected client

Network Connection

- You can use the network (when it's available) to store and retrieve data on your own web-based services
- Use classes in the following packages
    - java.net.*
    - android.net.*

Backing up data

- Auto Backup for 6.0 (API level 23) and higher
- Automatically back up app data to the cloud
- No code required and free

3. (a) Explain about SQLite database? Explain ACID properties?
   ACID-2.5+sqlite explanation-2.5

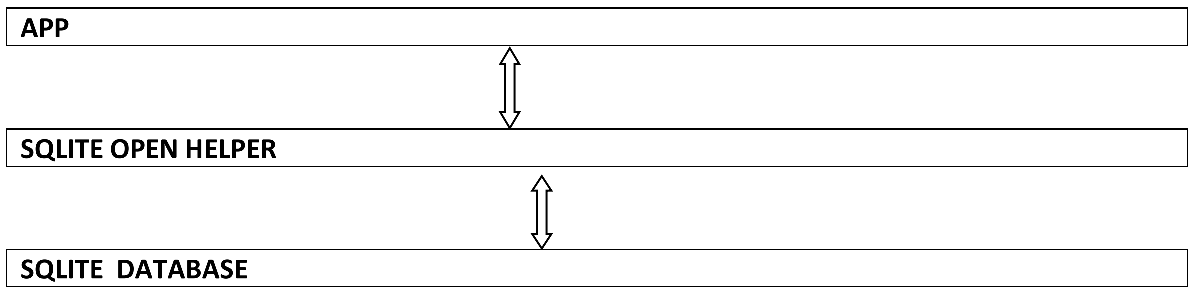
- Store data in tables of rows and columns (spreadsheet…)

- Field = intersection of a row and column
- Fields contain data, references to other fields, or references to other tables
- Rows are identified by unique IDs
- Column names are unique per table

Implements SQL database engine that is

- self-contained (requires no other components)
- serverless (requires no server backend)
- zero-configuration (does not need to be configured for your application)
- transactional (changes within a single transaction in SQLite either occur completely or not at all)

Components of SQLite database



SQLite database represented as an SQLiteDatabase object all interactions with database through SQLiteOpenHelper

1. Executes your requests
2. Manages your database
3. Separates data and interaction from app
4. Keeps complex apps manageable

A transaction is a sequence of operations performed as a single logical unit of work.
A logical unit of work must have four properties

- **Atomicity**—All or no modifications are performed
- **Consistency**—When transaction has completed, all data is in a consistent state
- **Isolation**—Modifications made by concurrent transactions must be isolated from the modifications made by any other concurrent transactions
- **Durability**—After a transaction has completed, its effects are permanently in place in the system

SQL basic operations include

1. Insert rows
2. Delete rows
3. Update values in rows
4. Retrieve rows that meet given criteria

1.Insert

long insert(String table, String nullColumnHack, ContentValues values)

First argument is the table name.Second argument is a String nullColumnHack. Workaround that allows you to insert empty rows .Use null .Third argument must be a ContentValues with values for the row Returns the id of the newly inserted item. Fourth argument are the arguments to the WHERE clause.

```
ContentValues values = new ContentValues();
// Inserts one row.
// Use a loop to insert multiple rows.
values.put(KEY_WORD, "Android");
values.put(KEY_DEFINITION, "Mobile operating system.");
db.insert(WORD_LIST_TABLE, null, values);
```

2.Delete

```
int delete (String table,
        String whereClause, String[] whereArgs)
```

- First argument is table name
- Second argument is WHERE clause
- Third argument are arguments to WHERE clause

3. update

```
int update(String table, ContentValues values,
    String whereClause, String[] whereArgs)
```

- First argument is table name
- Second argument must be ContentValues with new values for the row
- Third argument is WHERE clause
- Fourth argument are the arguments to the WHERE clause

4.Query()

1.SQLiteDatabase.rawQuery()

Use when data is under your control and supplied only by your app.

Syntax: cursor rawQuery(String sql, String[] selectionArgs)

First parameter is SQLite query string and Second parameter contains the arguments.

2.SQLiteDatabase.query()

Use for all other queries.

Syntx:Cursor query (boolean distinct, String table,String[] columns, String selection,String[] selectionArgs, String groupBy,String having, String orderBy,String limit);

Cursors

Data type commonly used for results of queries. It is a Pointer into a row of structured data ...

Cursor class provides methods for moving cursor and getting data. SQLiteDatabase always presents results as Cursor.

```
// Store results of query in a cursor
Cursor cursor = db.rawQuery(...);



try {

while (cursor.moveToNext()) {

// Do something with data

}

} finally {

cursor.close();
}
```

SQLiteDatabase always presents results as Cursor include

1. getCount()—number of rows in cursor
2. getColumnNames()—string array with column names
3. getPosition()—current position of cursor
4. getString(int column), getInt(int column), ...
5. moveToFirst(), moveToNext(), ...
6. close() releases all resources and invalidates cursor

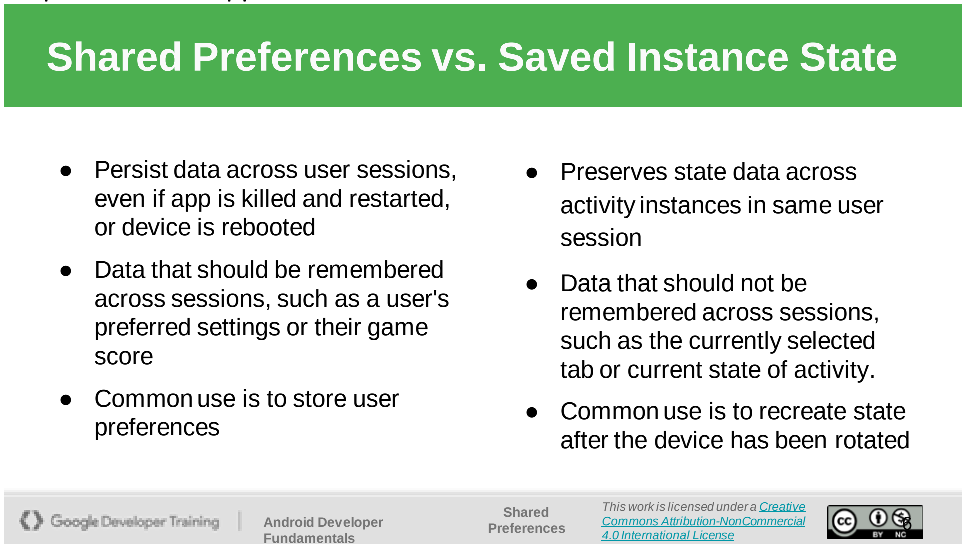
```
String query = "SELECT * FROM WORD_LIST_TABLE";

rawQuery(query, null);

query = "SELECT word, definition FROM
WORD_LIST_TABLE WHERE _id> ? ";

String[] selectionArgs = new String[]{"2"}

rawQuery(query, selectionArgs);
```

```
String table = "WORD_LIST_TABLE"

String[] columns = new String[]{"*"};

String selection = "word = ?"

String[] selectionArgs = new String[]{"alpha"};

String groupBy = null;

String having = null;

String orderBy = "word ASC"

String limit = "2,1"

query(table, columns, selection, selectionArgs,
groupBy, having, orderBy, limit);
```

3.bHow to creating a shared preferences file and how to clear it?

(Creation-2.5+clear-2.5)

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage
- SharedPreference class provides APIs for reading, writing, and managing this data
- Save data in onPause()
  restore in onCreate()
- Small number of key/value pairs
- Data is private to the application



Shared Preferences vs. Saved Instance State

- Persist data across user sessions, even if app is killed and restarted, or device is rebooted
- Data that should be remembered across sessions, such as a user's preferred settings or their game score
- Common use is to store user preferences

- Preserves state data across activity instances in same user session
- Data that should not be remembered across sessions, such as the currently selected tab or current state of activity.
- Common use is to recreate state after the device has been rotated

- Need only one Shared Preferences file per app
- Name it with package name of your app—unique and easy to associate with app

```
private String sharedPrefFile =   "com.example.android.hellosharedprefs";
mPreferences =getSharedPreferences(sharedPrefFile,  MODE_PRIVATE);
```

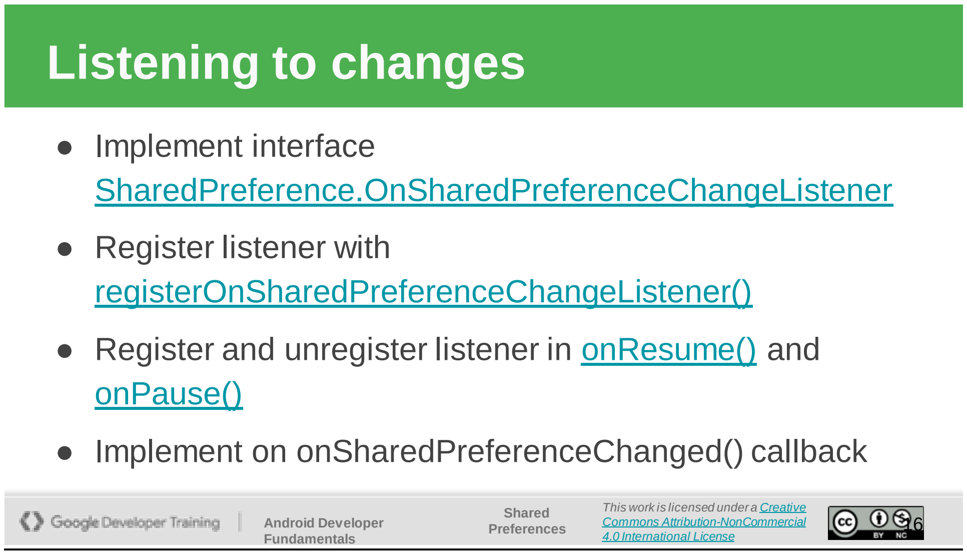
- SharedPreferences.Editor interface
- Takes care of all file operations
- put methods overwrite if key exists
- apply() saves asynchronously and safely

```
protected void onPause() {
  super.onPause();
  SharedPreferences.Editor preferencesEditor =
     mPreferences.edit();
  preferencesEditor.putInt("count", mCount);
  preferencesEditor.putInt("color", mCurrentColor);
  preferencesEditor.apply();
}
```

Clearing

- Call clear() on the SharedPreferences.Editor and apply changes

```
SharedPreferences.Editor preferencesEditor =
          mPreferences.edit();
preferencesEditor.clear();
preferencesEditor.apply();
```



Listening to changes

- Implement interface SharedPreference.OnSharedPreferenceChangeListener
- Register listener with registerOnSharedPreferenceChangeListener()
- Register and unregister listener in onResume() and onPause()
- Implement on onSharedPreferenceChanged() callback

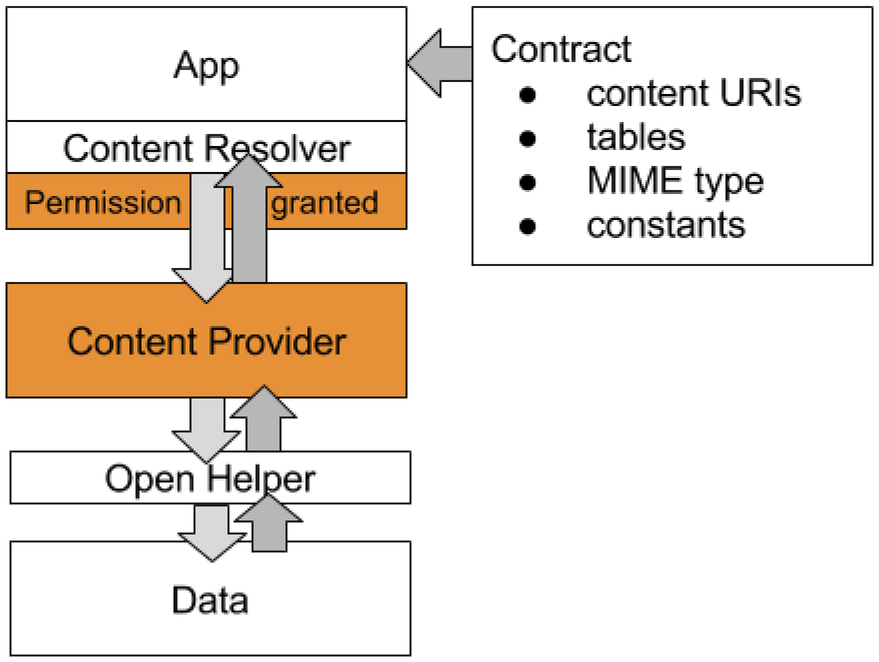
5. Explain App Architecture with a Content Provider? How to Implement a Content Provider? Diagram+explanation-8+implementation-2

- A content provider is a component fetches data that the app requests from a repository
- The app doesn't need to know where or how the data is stored, formatted, or accessed

Content Provider is Good for

- Securely make data available to other apps

  Manage access permissions to app data



Data Repository

- Data, commonly in SQLite Database but can be any backend
- OpenHelper if you use SQLite database

Contract

Public class that exposes information about the content provider to other apps

Contract specifies

- URIs to query data
- Table structure of data
- MIME type of returned data
- Constants

Content Provider

- Public secure interface
- Permissions control access

- Subclass of ContentProvider
- query(), insert(), delete(), update() API

Content Resolver

- Send requests to content provider and return response
- A content resolver is a component that your app uses to send requests to a content provider
- Requests consist of a content URI and an SQL-like query
- The ContentResolver object provides query(), insert(), update(), and delete() methods

Implementing a Content Provider

1. Data—Commonly an SQLite database
2. Write contract—Information about the content provider
3. Subclass ContentProvider and implement methods
4. Get data using ContentResolver
5. Set permissions in Android Manifest

Data

- Represents data in tables
- Supports same operations as content provider
- Returns requested data as cursor
- OpenHelper class to simplify management

Contract

- Public class that documents content provider API
- Called Contract by convention
- Contains
    - Content URIs and URI scheme to query data
    - Table and column names for returned data
    - MIME types to help process returned data

    ○ Shared constants to make life easier

Content URIs format

scheme://authority/path/id

- **scheme** is always content:// for content URIs
- **authority** represents the domain, and for content providers customarily ends in .provider
- **path** is the path to the data
- **id** uniquely identifies the data set to search

3. Subclass ContentProvider

```
public class WordListContentProvider
        extends ContentProvider {}
```

4. ContentResolver

Must use a ContentResolver to send requests as queries to the content provider

Data is returned in a Cursor object, as rows and columns

Its methods

- ContentResolver.query()
- ContentResolver.insert()
- ContentResolver.delete()
- ContentResolver.update()

5.Permissions

By default, with no permissions set explicitly, any other app can access a content provider for reading and writing

Set read or write permissions in AndroidManifest

Use unique tags that include package name .

5.How to monitor the performance of your running app ? How can you make sure your app treats the user's data safely.

(performance monitor-6+security-4)

You have made your app as useful, interesting, and beautiful as possible. However, to make it stand out from the crowd, you should also make it as small, fast, and efficient as possible. Consider the impact your app might have on the device's battery, memory, and disc space.

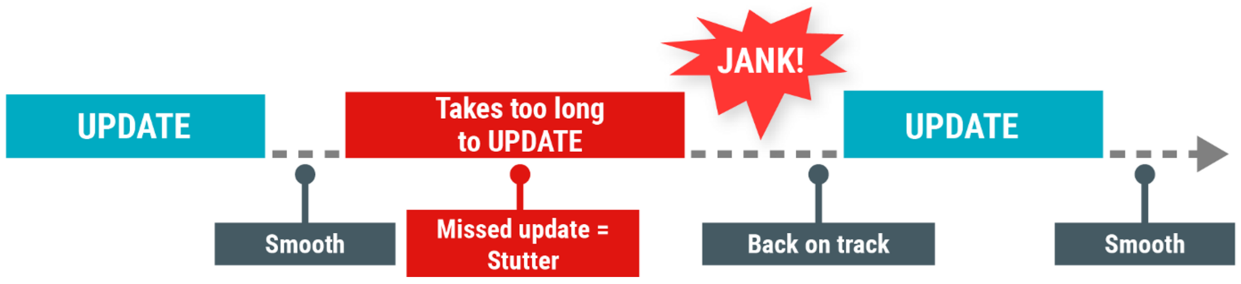
App performance can measure through

1. Speed
2. Responsiveness
3. Smoothness
4. Consistency
5. Resource-efficiency

speed

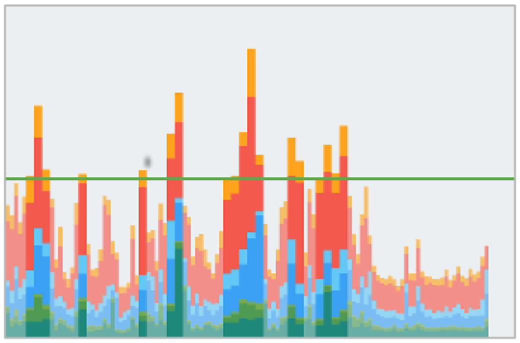
Keep long-running tasks off the main thread

The hardware that renders the display to the screen typically updates the screen every 16 milliseconds, so if the main thread is doing work that takes longer than 16 milliseconds, the app might skip frames, stutter or hang, all of which are likely to annoy your users.



Responsiveness

- You can check how well your app does at rendering screens within the 16 millisecond limit by using the **Profile GPU Rendering** tool on your Android device. If you spend time using the Profile GPU rendering tool on your app, it will help you identify which parts of the UI interaction are slower than might be expected, and then you can take action to improve the speed of your app's UI. One bar represents one screen rendered
- If a bar goes above the green line, it took > 16 ms to render
- Many bars above the line, or heavy spiking indicate problems
- User will see stuttering or inconsistent responsiveness



Smoothness and consistency

- Load data in background if it is taking more than 16ms to render
- Pre-fetch data before rendering
- Move work off the UI thread if it is time consuming
- Optimize UI—draw less and faster

- Eliminate overdraw and optimize view hierarchy

Simplify your UI

your layouts will draw faster and use less power and battery if you spend time designing them in the most efficient way.

Try to avoid:

- Deeply nested layouts
- Minimize overlapping views

Resource efficiency

- WiFi and mobile radio use lots of battery
    - Process Batch requests
    - Schedule to run when phone is being charged using job scheduler
- Large images consume lots of memory
    - Use smallest images possible
    - Always use compressed image formats. Use WebP when possible
- Getting and putting data on the internet uses up data plans
    - When possible download data when on WiFi

Security Issues Involve

- Android has built-in security features
- Significantly reduces the frequency and impact of application security issues

Your app's responsibility

- Keep user's private data safe
- Do not leak secret things
- Treat user's data with integrity
- Keep your own app and data safe

Handling user data

- Minimize access to sensitive or personal user data
- Do not store or transmit user data if possible
- But if you must, consider using a hash or non-reversible form of the data

For example, use hash of an email address as a primary key, so you do not store or transmit the email address

- Android logs are a shared resource, and are available to an application with the READ_LOGS permission
- Inappropriate logging of user information could leak user data to other applications
- Encrypt local files that contain sensitive data
- Store the key so it is not accessible by the app
  For example, use a KeyStore protected with a user password stored off the device

External storage

Do not store sensitive information on external storage

- Files created on external storage, such as SD Cards, are globally readable and writable
- External storage can be removed by the user
- External storage can be modified by any application
- Validate input on data from external storage

IP Networking

- Networking on Android is similar to other Linux environments
- Minimize network transactions that transmit private data
- Use HTTPS over HTTP wherever it's supported on the server
- Mobile devices often connect on networks that are not secured, such as public Wi-Fi hotspots avoid it
- You can implement authenticated, encrypted socket-level communication using the SSLSocket class

6. Explain about Firebase? Explain any 4 applications of Firebase.
   (application-8+general explanation-2)

Firebase can power your app's backend, including data storage, user authentication, static hosting, and more. Build cross-platform native mobile and web apps with our Android, iOS, and JavaScript SDKs.

Firebase is a platform that provides tools to help you

- develop your app
- grow your user base
- earn money from your app

Firebase Advantages

- It is simple and user friendly. No need for complicated configuration.
- The data is real-time, which means that every change will automatically update connected clients.
- Firebase offers simple control dashboard.
- There are a number of useful services to choose.

Firebase Limitations

- Firebase free plan is limited to 50 Connections and 100 MB of storage.
-

Firebase Analytics

- Unlimited free reporting
- Audience segmentation

Define custom audiences based on device data, custom events, or user properties

Firebase Analytics:

1. Default reports

Get reports without adding code

- geographic
- demographic
- engagement
- revenue

2. **custom reports**

Add code to log events in your app to get more reports

- Predefined events such as:

  user adds an item to their cart
- Custom events such as:

  user achieves a level in a game
- In your app project, add dependency in app/build.gradle:

  compile 'com.google.firebase:firebase-analytics:n.n.n'
- No need to write code for default reports
- Write code for custom events if you want them

Firebase Notifications

In the Notifications lesson, you learned how ***your app*** can send notifications to the user.

The Firebase Console lets ***you*** send notifications to your users.

- You write notification messages in the Firebase Console
- Firebase Cloud Messaging delivers the notifications to the target audience

Firebase Database

You have already learned your app can

- save data in an SQLite database on the device
- use a ContentProvider to share data with other apps

How do you enable different users using different devices, to share and update data?

Use Firebase Database

Store and sync data with the Firebase cloud database

Data is synced across all clients, and remains available when your app goes offline

- Firebase Realtime Database is hosted in the cloud
- Data is stored as JSON
- Data is synchronized in realtime to every connected client.

```
FirebaseDatabase database = FirebaseDatabase.getInstance();
DatabaseReference myRef = database.getReference(path);
myRef.setValue("New value");
```

Firebase Cloud Test Lab

Test your app on real devices in a Google data center .

Test your devices on wide range of devices.

Set test targets:

- Devices
- API levels
- Orientations
- Locales

Finally will get the result as report with issues.

<u>Ways to monetize apps in Google Play</u>

- **Premium** model—users pay to download app
- **Freemium** model
  - downloading app is free
  - users pay for upgrades or in-app purchases
- **Subscriptions**—users pay recurring fee for app
- **Ads**—app is free but displays ads -*AdMob*

7. What is an APK? How to publish your App in Google Play Store ?

(Apk-3+steps to create-7)

Android **A**pplication **P**ackage file $\rightarrow$ .apk file

- It's like the executable
- Each Android application is compiled and packaged in a single file that includes all the app's code, resources, assets, and manifest file
- You need an APK to publish on Google Play

<u>Steps for Publishing</u>

1. Prepare app for release
2. Generate signed APK

3. Upload to Google Play
4. Run alpha and beta tests
5. Publish to the world

<u>1. Prepare app for release</u>

- Test, test, test!
  - Test your app on different devices and screen size
  - Test your app on older devices
  - Use support library for backwards compatibility

- Add an icon
- Make sure your app has the correct filters
  - Google Play search results only show apps that are compatible with the user's device.It may be based on Hardware features ,API level ,countries.

- Choose an Application ID
  - Application ID defines your application's identity
  - Must be unique across all apps from everyone!
  - If you change App ID and re-publish

- Specify API levels targets
  - **Set min and target API level**
- Clean up your app
  - Remove logging statements
  - Disable debugging
  - Clean up your project directories
  - Update URLs for servers and services
  - Reduce APK size
    - Remove unused resources
    - Reuse resources
    - Minimize resource use from libraries
    - Reduce native and Java code
    - Reduce space needs for images
    -
- Generate a signed APK for release
  - Remove unused private or proprietary data files

        - For example, delete unused drawable files, layout files, and values files from res/ folder
        - Review the assets and res/raw directories for raw asset files and static files to update or remove
        - Remove unused test directories

2. Generate signed APK

- Android apps must be digitally signed before users can install them
- Use Android Studio to generate and sign your APK

A public-key certificate contains:

- the public key of a public/private key pair,
- other metadata identifying the owner of the key
  such as name and location

The owner of the certificate holds the private key.

The public-key certificate serves as as a "fingerprint" that uniquely associates the APK to you and your corresponding private key.

3. Upload to Google Play

    - Create an account on Google Play developer console
    - Create an entry for your app
    - Add the required assets and information
    - Run alpha and beta tests
        - Alpha test during development
        - Use alpha tests for early experimental versions of your app that might contain incomplete or unstable functionality
        - Beta test with limited number of real users
        - Use beta tests for apps that should be complete and stable

    - Run pre-launch reports
        - Pre-launch reports identify crashes, display issues, and security vulnerabilities
        - During pre-launch check, test devices automatically launch and crawl your app for several minutes
        - The crawl performs basic actions every few seconds on your app, such as typing, tapping, and swiping

        -
    - Publish to the world!
        - Your app must meet core app quality requirements
        - Visual design and user interaction
        - Functionality
        - Performance and stability