4. In Android applications the User Interface (UI) thread is the main thread. This thread is very important because it is responsible with displaying/drawing and updating UI elements and handling/dispatching UI events (the user's interaction with the application). If another thread tries to update the UI elements, it will receive a CalledFromWrongThreadException. In addition, Services and Broadcast Receivers run by default on the main UI thread.

5.
When you perform CPU intensive or blocking operations (for example network or database operations), you will block the main thread, so UI events will not be dispatched. The user will not be able to interact properly with the application and will receive an Application Not Responding (ANR) dialog (after a few seconds).
There are two simple rules regarding the main UI thread:

1. Don't block the UI thread with CPU intensive or blocking operations
2. Call the Android UI toolkit API only from the main UI thread

6.
So if you want to perform blocking or CPU intensive operations, you should create new threads (called "worker" or "background" threads). For doing this, you have several options, for example to create a Thread instance and call start or to implement the Runnable interface.
However, when using the Thread and Runnable classes, you need to manually send the data back to the UI thread using a Handler.
Thread and Runnable are basic classes, but they are the basis of more powerful Android classes such as AsyncTask, IntentService and HandlerThread. They are also the basis of ThreadPoolExecutor, which manages threads and task queues.

7.
AsyncTask is a class that allows you to perform asynchronous operations on a separate thread. Practically, it executes the operations using a worker thread and then publishes the results to the UI thread. So you don't need to manage threads or use handlers to send the data back to the UI thread.
The class includes a method that runs on the worker thread and several methods that run on the UI thread, that can be used for publishing the results.

8.
The method doInBackground() from the AsyncTask class runs on a worker thread, while the methods onPreExecute(), onPostExecute(), and onProgressUpdate() run on the UI thread.
doInBackground() returns a value that is sent to the method onPostExecute() after the job is finished.
onProgressUpdate() can be executed when calling publishProgress() from doInBackground(), at any time, while the job is running.
The task is launched when calling execute().
The task can be canceled at any time, from any thread, using method cancel().

9.

Here you have an implementation example of an AsyncTask. It is used for downloading several files.
The class includes:

- doInBackground() method that executes the job on a worker thread
- onPostExecute(), that receives the result returned by doInBackground(), and runs on the UI thread (it displays a Dialog box with the result)
- onProgressUpdate() that runs on the UI thread and is executed when publishProgress() is called from the worker thread

In the box below, we have an example of how to launch the AsyncTask implemented above.

11.
Now let's see how we create applications that connect to the network. First of all, in order to perform network operations, we need to request the following permissions in the AndroidManifest.xml file:

- ACCESS_NETWORK_STATE - that allows the application to check the state of the network
- INTERNET - that allows the application to access resources over the Internet

And here you have an example on how to specify these permissions in the Manifest file.
Network operations may introduce delays. You should always perform network operations on a separate thread from the UI, in order to prevent poor user experience. AsyncTask provides the easiest way to execute operations on a worker thread.

12.
Before the application attempts to perform a network transfer, you should check the status of the network connection. In order to do this, obtain the ConnectivityManager, then call the method getActiveNetworkInfo(). This method returns a NetworkInfo object which can be used to call the method isConnected() in order to check for connectivity to the network.
Here you have an implementation example.

13.
You can access online content through several methods.
An option is to use Java sockets, which can be easy to use if you are implementing a very simple protocol, because you have to handle the application layer messages manually.
Another option is to use URLConnection, which is able to handle application layer messages automatically for different protocols, such as FTP, HTTP, HTTPS. This class is used for connecting to a specific URL with the purpose of reading or writing data. First of all, you need to build an URL object (e.g. new URL("ftp://example.com")) and then you can obtain the URLConnection object through the use of openConnection() method of the URL object.

14.
If we are dealing with HTTP or HTTPS, we can use more specific classes.
AndroidHttpClient is deprecated from Android 5.1, so we are not going to discuss it.
So the classes that are recommended for handling HTTP and HTTPS messages are HttpURLConnection and HttpsURLConnection. You can perform GET operations in order to download data.

It also provides transparent support for IPv6. If a host has both IPv4 and IPv6 addresses, it will try to connect to each address, until a connection is established.

15
First of all you should create URL object, then call openConnection() on that URL and cast the result to HttpURLConnection. Then, you can send a request through an OutputStream (getOutputStream()) and receive data through an InputStream (getInputStream()).
By default HttpURLConnection uses the GET method by default. For using the POST method, you have to call setDoOutput(true). For other methods use setRequestMethod(String).
You can also handle cookies by using CookieManager and HttpCookie.

16.
Here you have a simple example of using HttpURLConnection. An URL object is created, then the HttpURLConnection is obtained and then the InputStream of that connection is obtained.

18.
The Android framework provides an API for controlling the Bluetooth adapter:

- Turn the BL adapter on and off
- Make the device discoverable through BL
- You can scan for other discoverable devices
- Pair devices
- Send data to other devices
- Receive data from other devices
- Handle multiple connections

19.
In order to use Bluetooth features in your application, you need to declare the BLUETOOTH permission. This is necessary for basic operations such as connecting to paired devices, sending data to another device or receiving data from a device.
The permission BLUETOOTH_ADMIN is necessary for changing the BL settings (for example turning the adapter on, off, and discoverable) and initiating device discovery and pair with them (with user's confirmation). If you are using this permission, you also have to request the BLUETOOTH permission.
The permission BLUETOOTH_PRIVILEDGED can be requested when you need to pair with devices without user's approval. However, this is not available to be used by third-party applications.

20.
Now lets see the BL API.
The BluetoothAdapter class represents the BL adapter of the device. To get an instance of this class, you need to call the static method getDefaultAdapter(). When this method returns null, the device does not support BL.
The class is used for performing any BL operation: to discover devices, to list the paired devices, to obtain an instance of BluetoothDevice (based on a known MAC address).

After obtaining the adapter object, you need to verify if BL is enabled. You can do this using the method isEnabled(). If BL is not enabled, you can enable it by sending an Intent.
You can also use the adapter object to obtain a BluetoothServerSocket, that is used for listening to incoming requests.

21.
The class BluetoothDevice represents a remote device. The method getBondedDevices() from BluetoothAdapter returns a set of BluetoothDevice objects, which represents the list of paired devices.
You can use a BluetoothDevice object to obtain information about that device (name, address, class and pairing status).
For initiating a connection to the remote device (as a client), obtain a BluetoothSocket from the BluetoothDevice by calling a method (createRfcommSocketToServiceRecord(UUID)).

22.
So a BluetoothSocket object can be used for initiating the connection with a remote device. For this you need to call the method connect().
This is similar to a TCP socket and will be used for sending and receiving data through the use of InputStream and OutputStream (to obtain them you need to call getInputStream() and getOutputStream()).

23.
A BluetoothServerSocket can be obtained from the BluetoothAdapter and can be used for listening for incoming connections (similar to a TCP server socket).
In order to wait for incoming connections, you need to call the method accept(). The call will block until a new connection is accepted and if successful, it returns a BluetoothSocket object.

24.
Bluetooth Low Energy (BLE) has been designed to consume less energy than the standard Bluetooth.
First of all, in the manifest file, besides the permissions that were presented on the previous slides, you need to make the device available only for BLE-enabled devices. So you need to specify this entry in the Manifest file:
<uses-feature android:name="android.hardware.bluetooth_le" android:required="true"/>
Than, at runtime, you need to verify whether BLE is supported on the device:
getPackageManager().hasSystemFeature(PackageManager.FEATURE_BLUETOOTH_LE).

25.
Scanning for BLE devices can be performed by using this method: BluetoothAdapter.startLeScan(BluetoothAdapter.LeScanCallback). You need to implement BluetoothAdapter.LeScanCallback in order to receive the result. More exactly, you need to override the method onLeScan() of LeScanCallback in order to receive the scanning result. The scanning results include the RSSI, which can be used to obtain an approximate distance to the other device. The scan records include the device type, device identifier and other attributes.

26.
Here you have an implementation example. LeScanCallback is implemented and we override onLeScan, where on the UI thread, we add the discovered device on a list and display it. In the lower box, we have an example on how to start and stop scanning.

28.
Near Field Communication (NFC) is a wireless technology for very short range (4 cm). It can be used to send small messages between an NFC tag and a mobile device or between two devices.
The transmitted data has a certain format (NFC Data Exchange Format - NDEF).
Android devices that have NFC hardware usually have 3 modes of operation:

- A reader/writer mode - that can be used when reading from or writing to NFC tags
- P2P mode - which is used for sending NFC messages to another mobile device
- Card emulation mode - in which the mobile device acts like an NFC card (for example you can use your phone at an NFC POS terminal).

29.
First of all, you need to request the NFC permission in the Manifest file:
<uses-permission android:name="android.permission.NFC" />
You should specify API level minimum 10:
<uses-sdk android:minSdkVersion="10"/>

30.
You also need to specify that the application is available only for devices that have NFC hardware:
<uses-feature android:name="android.hardware.nfc" android:required="true" />
And at runtime, you need to check whether NFC hardware is present: if the NfcManager.getDefaultAdapter() does not return null.

31.
In order to receive an Intent when a specific NFC tag is scanned, you need to specify an Intent Filter with the action android.nfc.action.NDEF_DISCOVERED.
Then, when receiving the Intent, you need to check if the action is NfcAdapter.ACTION_NDEF_DISCOVERED.
Finally, you can obtain the data using intent.getParcelableArrayExtra(NfcAdapter.EXTRA_NDEF_MESSAGES).

32.
And this is an example on how to retrieve the messages received through NFC.

33-34.
For sending NFC messages, you need to go through the following steps:

- You must have an Activity that implements NfcAdapter.CreateNdefMessageCallback and NfcAdapter.OnNdefPushCompleteCallback.

- In onCreate(), you should get an instance of the NfcAdapter and set the current activity as responsible for handling the callbacks of the adapter: NfcAdapter.setNdefPushMessageCallback() and NfcAdapter.setOnNdefPushCompleteCallback()
- You have to override the callback createNdefMessage() which is called when a new NFC tag is discovered. In this callback you can create the message that you want to transmit to the NFC tag.
- In addition, you can use the callback onNdefPushComplete() in order to notify the UI that a message has been sent.

35.
Here you have an example on how to implement an Activity that sends an NFC message when a new tag is discovered.

37.
The services provided by Google, such as Maps, Plus, Drive, Cloud Messaging, and so on, cannot be accessed through the framework API. In order to use them, you need to have the Google Play Services package installed on the device and other proprietary libraries.
Google Play Services APK includes several individual services and can be installed through Google Play.
The client libraries are delivered through the SDK Manager and can be included in the applications.

38.
The application includes the client library that communicates with the Google Play Services APK that in turn communicates with the external services.
The Google Play Services package is updated through Google Play, so the updates of the services are not dependant on the OEM system image updates.

39.
Google Cloud Messaging or Firebase Cloud Messaging (this is how the new version is called), is a service for sending messages between client applications that run on mobile devices and application servers.
It supports two types of messages:

- Downstream messages, that are push notifications, sent from the server to the client
- Upstream messages, that are sent from the client to the server

40.
Let’s see the steps that are required for implementing a GCM client.
In the Manifest file, you need to request the INTERNET permission and the permission com.google.android.c2dm.permission.RECEIVE.
You also need to declare and use a new permission that has the name: packageName + ".permission.C2D_MESSAGE". This is for preventing other applications from registering and receiving your application’s messages.
You need to configure the minimum SDK to level 8, because it would not run properly on devices with a lower SDK.

You need to implement a broadcast receiver called exactly com.google.android.gms.gcm.GcmReceiver. You must specify that the sender has the permission com.google.android.c2dm.permission.SEND.

41.
Then, you need to implement a service that extends GcmListenerService which is used for receiving downstream messages (onMessageReceived()), for determining the upstream send status.
You also need to implement a service that extends InstanceIDListenerService in order to handle the registration tokens. You will use the API to obtain a registration token.
Finally, you can send messages through the use of GoogleCloudMessaging.send()

42.
Another Google service is Maps. You can use the Maps API in your application in order to access the Google Maps servers, download maps, display maps and handle user interaction with maps.
The API also allows applications to add data to a map, such as markers, overlays, polylines and polygones.

43-44.
Let's see the steps required for using Maps into an Android application.
First of all, you need a Google Maps API key that should be added as meta-data in the Manifest file. The key is obtained by registering your application to the Google API Console.
Then, you must request a set of permissions in the Manifest file:

- The INTERNET permission, that is necessary for connecting to the Google servers through the Internet and downloading the map tiles.
- ACCESS_NETWORK_STATE is needed in order to check the status of the network connecting before downloading the data.
- WRITE_EXTERNAL_STORAGE is needed in order store the map tile data on the SD card (external storage).
- ACCESS_COARSE_LOCATION is necessary for obtaining the location through WiFi or the mobile network (or both). The location has an accuracy equivalent to a city block.
- ACCESS_FINE_LOCATION is needed for obtaining precise location through GPS, as well as WiFi and mobile.

45.
Then, you need to implement a Fragment in the Activity that includes the map. The fragment's name should be exactly com.google.android.gms.maps.MapFragment.
In the Activity, you must obtain an instance of that fragment and cast it to MapFragment.
Then, for rendering the map, you must implement the OnMapReadyCallback interface and then call getMapAsync() method of the fragment in order to register the callback.