4.

- Every Android application must have an AndroidManifest.xml file in the root directory of the application.
- This file describes the application components and the resources it needs for running:
    - First of all, it includes the application name and package name. The package name is considered the unique identifier of the application in the system. You cannot install two applications with the same package name. For example, if you have an application with the package name com.myapp installed on the phone, and you want to install another application with the same package name, it will not work, you will receive an installation error.
    - The Manifest file also describes every application component - activities, services, broadcast receivers, content providers. For each of them, it specifies the class and capabilities (for example, what intents they can receive).
    - It also specifies the main activity, the one that will start when we click on the icon found in the home screen.
    - Here we also specify the permissions needed by the application to call protected parts of the API and to interact with other applications. In addition, you must also specify the permissions needed by other applications for interacting with our own application.
    - The application may be linked with external libraries, which are also specified in the Manifest file.
    - We must also declare the minimum and target level of the Android API required by the app. For example, if we want to build our application for Android 4.4 but we want it to be backwards compatible with Android 4.2, the target API level will be 19 and the minimum API level will be 17. Each Android version has associated a certain API level.

5.

- An application may need a set of permissions to call methods from the API and to access data or resources.
- This security mechanism provides protection through sandboxing. This means that applications need to declare the permissions needed for them to run properly. This way, an application cannot do certain operations without having permissions to do them.
- The permissions are specified in the Manifest file with the tag <uses-permission>. Here you have an example, if an application wants to access the Internet, it needs the permission INTERNET. This way, the application will be able to use either Wi-Fi or 3G, you don't have special permissions for each of them.
- In addition, we can control who can access the components of our application: to start an activity, to start and bind a service, to send a broadcast message to our receiver, to access the data from our content provider.
    - Here we have an example on how to control who can start your activity. This is a custom permission, that was defined by the developer.
    - Standard permissions are not sufficient when dealing with content providers. This is why we can have specific read, write permissions per URIs.

6.

- Resources such as images, strings and layouts that are needed by the application are organized in the res/ directory. Each resource is placed in a subdirectory with a specific name, for example: drawable, layout, values, menu, xml, etc. In these subdirectories we have the default resources.
- Drawables are images, layouts are xml files that describe how UI elements are placed on the screen. Values include strings that are fixed portions of text (for example you can declare a set of strings for each supported language, so the application will use the set of strings for the configured

language). Menu includes descriptions of how menus look like in the app. Xml will include other xml files that can be used in the app.

- Different types of configurations (hardware or software) may need different resources.
    - For a device with a large screen (such as a tablet), we will make a different layout in order to make use of the available space.
    - On a device with a different language, we will have different values for the strings.
    - At runtime, the application will use the resources for a certain configuration, for example, english language and hdpi screen).
    - In order to have alternative resources, we will have subdirectories for each alternative configuration.
    - The name of the subdirectory will be composed of the resource name and the configuration name (for example, drawable-hdpi for highdensity screens - 240dpi).
- For each resource name, an unique ID will be generated in the gen/ directory.

7. This is an example of having different layouts for different sizes of screens, in order to make use of the available space.

8.

- Layout is a resource included in res/layouts/ which describes the interface with the user for an activity or a part of the interface.
- Practically, a layout will contain UI elements: buttons, lists, textboxes, and so on.
- A layout is described in an xml file and the filename will be the ID of that resource. You will be able to refer them in the code with R.layout.layoutname. R.java (from resources) is generated at build time, that can be found in gen/ folder and includes all resource IDs. You should not edit the R file by hand because it is generated based on the resources.
- The xml file containing a layout, can be edited directly or with other tools such as Android Studio.

9. This is an example of layout for an activity. We have this XML file: res/layout/main_activity.xml. It contains a linear layout consisting in a text area and a button. Below, you have the Java code that applies that layout to the activity.

10.

- Drawables are resources from res/drawables/, elements that can be displayed on the screen. These can be images or xml files.
- Xml files can be used for describing how an UI element will look like when the user interacts with it, how it is animated (for example, when it is pressed). These xmls make reference to images, for example, we can have an image, and when it is pressed, it is replaced with another image. Or a button that changes color when it is pressed.
- This improves the user's QoE through the visual feedback during the interaction with the graphical interface.

12.

- An activity is an application component similar to a window, that represents the interface with the user. He will interact directly only with activities.
- For an activity to be displayed we need a layout that describes the UI elements and their placement on the screen, as we saw in the previous example.
- The graphical interface can be changed only from the main UI thread of the activity. Therefore, you should not perform intensive computational processing or potentially blocking operations on this thread. It is recommended to do these operations on different threads. An important rule is NOT to access the network in the UI thread, you must create a separate thread for the operations with the network.

- An application may include many activities, but only one main activity that will be started when we click on the icon placed in the launcher (or home screen).
- We can start an activity while being in another activity, the old one with be paused and saved on a stack and the new one started. When we press back, the current activity will be destroyed and the previous one will be resumed (it does not have to be re-created again). This way we have a stack of activities, that is called backstack.

13. For using a new activity in your application, you must declare it in the manifest file. You must add an activity element as child of the application element. The activity element can include many attributes, by the android:name is the only one that is mandatory. This is also the name of the class that implements the functionality of the Activity.

14.
- On this slide, we have a simplified representation of the lifecycle of an application. The methods you see here are callbacks, which are called by the system (by the ActivityManager), not by the application itself
- When an application is started, the method onCreate() is called (which is the entry point in the activity) and the Activity enters in the Created state. In this moment the activity gets its layout but it does not draw it on the screen yet.
- When onStart() is called, it will draw the layout on the screen and enter the Started state, in which the activity is visible on the screen.
- Then onResume() is called and the activity passes in the Resumed state, in which it is visible and accepts the user's input.
- When you receive a notification or a dialog box appears, onPause() will be called, and the activity will go in the Paused state, in which it will be partially visible and the user cannot interact with it. If you close the notification/dialog, onResume() will be called again and the activity will enter the Resumed state. The layout is not redrawn, the activity has focus again and receives the user's input.
- When from the current activity A you start another activity B, onPause() and onStop() from A will be called, and the activity A will remain in the Stopped state in backstack while the new activity B begins its lifecycle (activity A is not visible anymore). While in activity B, when you press back, this activity will be destroyed, and onRestart(), onStart() and onResume() will be called for activity A. Why do we go again through the Started state? because it is possible that the activity may need to be re-drawn.
- When in activity A, if the user presses back, onPause(), onStop() and onDestroy() will be called, in order to destroy that activity and come back to the previous activity.
- In the Stopped and Paused states, the activity may be killed by the Low Memory Killer, because another application with a higher priority needs memory. In this case, when the user re-enters in the activity, the methods: onCreate(), onStart() and onResume() will be called, so the activity will be re-created from scratch.

15-16. This a skeleton of an activity which includes every lifecycle method: onCreate(), etc.

17.
- Activities can be killed by the system if it needs memory for applications with a higher priority. In that moment, the state of the activity is lost. We have the possibility to save this state (including primitives, objects, UI) through the callback onSaveInstanceState(). In this callback, we need to save all information into a single Bundle. The state of the activity can be restored in onCreate or in onRestoreInstanceState(), which receive as argument the saved Bundle.
- However, you should save the activity state only when it is very important to do so, because you don't want to occupy the memory uselessly.

- When the process is killed, the activity threads will also be killed. These threads should be stopped graciously before the process is killed, especially if they are performing critical operations, such as writing in files. Therefore, it is recommended to signal threads to stop themselves in the method onPause().

18.

- Here we have represented the process of save and restore of the state. We observe that there is no need to do the restore operation if the process is not killed because the state of the activity is maintained during Paused and Stopped.

19.

- Fragments are UI elements that are contained by an Activity. They are like smaller activities inside an activity, which have their own lifecycle.
- They can be combined in different manners to build a multi-pane UI depending on the layout. For example, on a phone and on a tablet they can be arranged in different ways (in order to look better, not stretched), but the fragment code remains the same. For example, if you are in landscape mode on a tablet, you can have a different arrangement of the fragments than on a portrait mode on a phone.
- In addition, the fragments can be reused by other activities.

20. Here you have an example with two fragments displayed on different devices. Selecting an item from fragment A, updates the content of fragment B. On a tablet device you can combine both fragments in Activity A. However, on a handset device there is not enough space for both fragments, so Activity A includes only fragment A while Activity B includes the fragment B.

21.

- The lifecycle of a fragment is directly affected by the lifecycle of the parent activity. When the activity is paused, the fragments are also paused. When the activity is destroyed, the fragments are also destroyed.
- While an activity is in Resumed state, fragments can be added and removed. These operations are called fragment transactions.
- There is a fragment backstack associated to the activity. Each entry in the backstack contains fragment transaction that has occurred. When pressing back you can reverse the last fragment transaction.
- However, method onCreate() has a different functionality than the one for activities: you will not get the layout here and work with it, but in onCreateView(). This method is called when the fragment knows to what activity is being attached and what space is occupying in that activity. And its correspondent is onDestroyView(), this is where the threads will be stopped (not in onDestroy()).

22.

- The user interface is composed of a hierarchy of views (derived from class View: Button, TextView, Checkbox, etc.). Some examples are: buttons, lists, images, textboxes, and so on.
- Each View controls a rectangular space in the activity and provides user interaction.
- For the interaction with these objects, we have callbacks in which we can specify what actions to be executed, for example, when a button is pressed (here we use the onClick() callback).
- A ViewGroup may contain several View objects and other ViewGroups. In order to create more complex Views, you can extend the existent classes.
- In addition, you can use adapters for displaying complex types of data. Examples of adaptors are: ArrayAdapter, ListAdapter, CursorAdapter, and so on.

24.

- A service is an application component that can execute long-running background operations and does not provide a user interface.
- A service will continue to run even when the user has opened another application in foreground. For example, a music player, should be able to play music even when the application is in background.
- A service can perform network or file I/O operations, can interact with content providers, and so on.
- A service will usually run in the main UI thread of the process - it will not create a new thread and will not run in a different process by default. So, if we want to make CPU intensive or potentially blocking operations, it is better to create a new thread for the service or configure it to run in a separate process.
- A service can be started using an Intent, from the current or from other application.
- If we want to block the access of other applications to our service, we need to declare it private in the Manifest file.

25. Services must also be declared in the Manifest file. This is an example on how to do this: you must add a service element as child to the application element. You can have multiple attributes, for example the permission needed by other applications to start your service. You can also configure the service to run in a separate process. The attribute android:name is the only one that is mandatory. In this example, the service is configured to be private through the android:exported attribute.

26.

- There are two types of services: Started and Bound.
- A service is Started when an application component calls the method startService(). It will run indefinitely, even if the component that started it is destroyed. Such a service performs an operation and then it stops itself, without returning any result to the component that started it.
- A service is Bound when an application component binds to it using the method bindService(). Such a service will provide a client-server interface through which it can receive requests and send replies. AIDL will be used for generating the interface used between the client and the server.
    - This type of service will run only until the component that started it will call unbind. However, there is a corner case: if you bind a service from an activity and that activity is put in Stopped state, when you come back to that activity, the service may be null, because the system has needed memory and killed the service. So you should always check whether the service is not null, in order to avoid NullPointerExceptions.
    - We can have multiple components that bind to a service. In this case, the service will be destroyed only after all of them call unbind.
- A service can be both started and bound in the same time.

27.

- On this slide we have the lifecycle of a service in the two cases.
- In the first case, when a component calls startService(), the method onCreate() will be executed, then onStartCommand() (these are callbacks), then the service is up and running. After that, the service can stop itself using stopSelf() or can be stopped by another component using stopService(). Before the service is killed, the method onDestroy() will be called - here, we should perform a cleanup.
- In the second case, a service is started using bindService(), the method onCreate() will be executed, then onBind(), then the client is bound to the service and can communicate with it. After all clients make unbind, onUnbind() and onDestroy() are called.
- However, keep in mind that a service can be both started and bound. Even if the service has been started using startService(), it can also receive calls to onBind() (when clients call bindService()).

28-29. This is the skeleton of a service, including all callbacks: onCreate(), etc.

31.

- An Intent is similar to a signal on Linux. It is used for sending a message or determine the execution of an action.
- An intent has 3 main components: the name of the target component, the action that must be executed and the data used in that action (for example, call a certain phone number).
- In order to specify which intents are accepted by a certain component, we will declare an intent filter in the Manifest file. The action and data type will be specified using the tags <action> and <data> included in the intent filter.

32.

- An intent can be used in the following cases: to start an activity, to start or bind to a service, to deliver a broadcast message.
- An activity can be started by passing an Intent to the startActivity() method. The Intent includes information about the activity that must be started and the necessary data.
- An activity can also be started by passing an Intent to startActivityForResult() method. The result will be received through a separate Intent.
- A service can be started by passing an Intent to startService() method. The Intent includes information about the target service and the necessary data.
- In a similar manner, you can bind to a service by passing an Intent to the bindService() method.
- A broadcast message can be sent by passing an Intent to one of the methods: sendBroadcast(), sendOrderedBroadcast(), or sendStickyBroadcast().

33.

- There are two types of intents: explicit and implicit.
- The explicit intent specifies the exact target component, the name of the class. Usually it is used for starting an activity or a service within the same application. There is no need to specify an intent filter when we are using only explicit intents, because these will be delivered even if there is no intent filter specified.
- The implicit intent does not specify the name of the target component, but only the action to be executed. The Android system will search for the most appropriate component which can perform that action. It will do this by searching for a match between the intent filters from the Manifest files and the intent that is sent.
- If there is more than one component, the user will be asked which one he would like to use (for example reading a pdf file). So in this case, we clearly need to specify intent filters into the Manifest file.

34.

- On this slide we can see how implicit intents are delivered.
- Activity A creates an Intent object with the associated action, and gives it as argument to the method startActivity(). The system searches for intent filters that match the action. When the appropriate activity has been found, it calls onCreate() and then it sends the Intent object to that activity.

35.

- This is an example of how an implicit intent is sent. Let’s say you have some content to share. You create an Intent, configure ACTION_SEND, add some text message (data type is text) and call startActivity() with that Intent.

- If there is no activity that would receive that kind of intent, your application may crash. So it is better to verify if there is any application that can receive that intent before sending it, through the use of resolveActivity() method. If the result is different from null, there is at least one application in the system that can receive that kind of intent.

36. This is an example of activity declaration in the Manifest file that includes an intent filter with action SEND and data type is text. This specifies that the activity will receive the intent from the previous slide.

38.

- A broadcast receiver is an application component that handles broadcast messages sent throughout the system.
- Broadcast messages are notifications or announcements. Most broadcast messages are generated by the system: for example, when the battery is low, when the screen is turned off, when an SMS is received, and so on.
- Even applications can generate broadcast messages to make announcements to other applications.
- Receivers do not have a graphical interface, but they can generate notifications in the status bar in order to make the user aware of an event.
- A receiver usually should not perform elaborate operations because it always runs in the main UI thread. So a receiver should delegate tasks to other components, for example a service, to do the complex operations, which are determined by the event.

39.

- Any broadcast message is sent using an Intent (using sendBroadcast() or sendOrderedBroadcast()).
- If we want to use broadcast messages only in our application, it is better to use LocalBroadcastManager, because it is more efficient (it does not go through the system), the data does not get out of the application and another application is not allowed to send those messages (so we have no security holes).
- A receiver can be declared statically using the tag <receiver> in the Manifest file, or dynamically using an anonymous instance that extends BroadcastReceiver class and the method Context.registerReceiver(). It is recommended to use the <receiver> tag in order to keep your code clean.

40. We can have two types of broadcast messages:

1) normal - these are completely asynchronous. The receivers for that message run in an undefined order, possibly even in the same time. These are sent by passing an Intent to the method sendBroadcast().
2) ordered - these are delivered to a receiver at a time (ordered receivers). The receiver executes and then chooses whether to propagate the results to the next receiver or to abort the broadcast (this means not sending the broadcast further to the next receiver). The receiver order is determined using “android: priority” in the intent filter. These broadcasts are sent by giving an intent to the method sendOrderedBroadcast().

41. This is an example of broadcast receiver declaration in the Manifest file. It includes an intent filter for the BOOT_COMPLETED event. For this, we must request the RECEIVE_BOOT_COMPLETED permission.

42. And this is the implementation example of the receiver. The callback onReceive() will be called when the BOOT_COMPLETED event takes place. In the callback it starts a service.

44.

- A content provider is an application component that provides access to a repository of data. Usually, an application shares its data with another application using a content provider.
- There are some system content providers (the default ones), for example: contacts, dictionary, settings, and so on.
- In order to access a provider, our application needs specific permissions for reading or writing that data (the permissions are declared in the Manifest file). For example, if we want only to read data from the dictionary, we need the permission READ_USER_DICTIONARY.
- There are two methods to store the data:

1) Using files, such as audio, video, photos
2) Using structured data that includes tables with rows and columns (for example databases or arrays). The most used solution for storing data on Android is SQLite.

45.

- So we can use a provider to access the data of an application from another application, we will call them provider and client.
- The application that owns the data includes the provider and the application that accesses the data includes the client.
- We need a client object called ContentResolver. Through its methods, we can create, access, update and delete data.
- When we call its methods, the system will call the methods with the same name from the ContentProvider.

46.

- In order to identify data, we can use Content URIs.
- An URI has two components: authority is the name of the provider, and path is the name of the table. Here we have an example of URI: content://user_dictionary/words -> the provider is user_dictionary and the table is words.
- The ContentResolver will read this URI and first of all, it will identify the provider. It will search in a system table with all providers and will obtain access to the provider it was looking for. The Resolver will make a query to the provider for that exact path. The provider will use the path in order to identify the table and will perform the requested operation on that table.

47. These are some examples of operations that are performed on the user dictionary content provider. We can see here three operations: a query, an insert and an update. For each operation, we will give the content URI of the Words table as argument. The projection specifies the columns that will be included for each selected row. Here we specify the criteria for selecting the rows. And finally the sort order for the returned rows.

49.

- Next, we will go through some tools, that will be used in the lab.
- Android SDK Manager is used for downloading and managing SDK packages, samples, system images for the emulator and tools (platform tools, build tools). Here you can see a screenshot of the SDK Manager. You can choose to install or uninstall different packages.

50.

- Android Virtual Device Manager can be used for creating and managing virtual devices that can be used by the emulator (here you have a screenshot of the AVD Manager). The emulator is used for running virtual devices for testing your applications directly on the computer, without the need of a physical device.

51.

- Dalvik Debug Monitor Server (ddms) is basically a debugging tool that provides port-forwarding services, screen capture on the device, thread and heap information on the device, logcat, process, and radio state information, incoming call and SMS spoofing, location data spoofing, and more.
- Android Debug Bridge (adb) is used for the communication between the computer and the virtual or real device.
- dx is used for translating and aggregating multiple class files into a single dex file.

52.

- aidl allows for a service to be accessed from another application. The client and the server should have the same definition of the interface between the two. Aidl is used for generating interfaces and stubs for the Binder.
- aapt is used for creating and managing archives compatible with zip (apk). In addition, it is able to compile the resources into binary assets (for example xml files).
- dexdump is a disassembler that can be used for obtaining the class files from the dex file. Practically it is the opposite of dx.

53.

- Let's see some details about adb.
- Adb has 3 main components: a client and a server that run on the PC and a daemon that runs on the virtual or real device.
- In the command line interface, when we use adb and a command, the client will send the command to the server, which will execute it by communicating with the daemon on the device.
- You can use adb for performing several actions: copying the files from the PC to the device and vice versa (using adb push and pull), installing applications on the device (with adb install), displaying debug messages (with adb logcat), obtaining a shell on the device (with adb shell) and many others.

54.

- The Android Emulator is based on QEMU. This provides the ability of working with many components such as the display, keyboard, network, gps, audio, video, and so on.
- You can increase the virtualization speed, by using an Android image for x86 and using KVM on Linux, and HAXM on Windows (because in this case, the kernel cannot be modified). This way, there is no need for the instructions to be translated (from instructions for the target device into instructions for the host), but they can run directly on the host processor.
- In addition, instead of using the virtual GPU, you can use the host's GPU in order to obtain a better performance.