CMR Institute of Technology, Bangalore
Open Elective (Common to All Branches)
I – INTERNAL ASSESSMENT

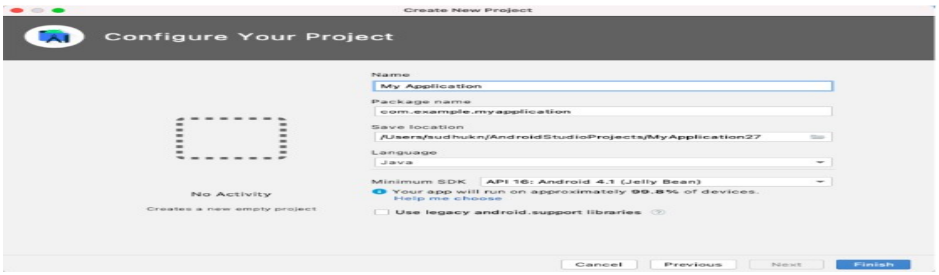
Semester: 6-CBCS 2018 Date: 21 May 2021
Subject: MOBILE APPLICATION DEVELOPMENT (18CS651/17CS661/15CS661)
Faculty: Dr. Sudhakar K N
Time: 01:00 PM - 02:30 PM Max Marks: 50

Scheme & Solution

ANSWER_ANY 5 Question(s)

Marks CO PO BT/CL

1a. Explain each of the fields as shown in figure and jusfy its relevance towards Android Project Configuration.



[4.0] 1 [1, 2, 3] [2]

Scheme: Each Field 1M x 4 = 4M

Solution

i. Specify the Name of your project.
ii. Specify the Package name. By default, this package name also becomes your application ID, which you can change later.
iii. Specify the Save location where you want to locally store your project.
iv. Select the Language you want Android Studio to use when creating sample code for your new project. Keep in mind, you are not limited to using only that language creating the project.
v. Select the Minimum API level you want your app to support. When you select a lower API level, your app can rely on fewer modern Android APIs. However, a larger percentage of Android devices are able to run your app. The opposite is true when selecting a higher API level. If you want to see more data to help you decide, click Help me choose.

1b. Define Android. Explain Android Architecture with a neat diagram.

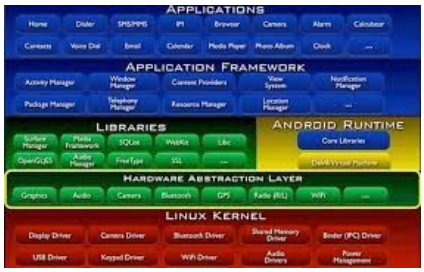
[6.0] 1 [1, 2, 3] [1]

Scheme: Definition 1M + Diagram 3M + Explanation 2M

Solution

Android: is a mobile operating system based on a modified version of the Linux kernel and other open source software, designed primarily for touchscreen mobile devices such as smartphones and tablets. Android is developed by a consortium of developers known as the Open Handset Alliance and commercially sponsored by Google. It was unveiled in November 2007, with the first commercial Android device, the HTC Dream, being launched in September 2008.

Android Architecture:



Description:
Linux kernel: the bottom of the layers is Linux - Linux 3.6 with approximately 115 patches. This provides a level of abstraction between the device hardware and it contains all the essential hardware drivers like camera, keypad, display etc. Also, the kernel handles all the things that Linux is really good at such as networking and a vast array of device drivers, which take the pain out of interfacing to peripheral hardware.

Libraries: top of Linux kernel there is a set of libraries including open-source Web browser engine WebKit, well known library libc, SQLite database which is a useful repository for storage and sharing of application data, libraries to play and record audio and video, SSL libraries responsible for Internet security etc.

Android Libraries: category encompasses those Java-based libraries that are specific to Android development. Examples of libraries in this category include the application framework libraries in addition to those that facilitate user interface building, graphics drawing and database access. A summary of some key core Android libraries available to the Android developer is as follows −

- android.app − Provides access to the application model and is the cornerstone of all Android applications.
- android.content − Facilitates content access, publishing and messaging between applications and application components.
- android.database − Used to access data published by content providers and includes SQLite database management classes.
- android.opengl − A Java interface to the OpenGL ES 3D graphics rendering API.
- android.os − Provides applications with access to standard operating system services including messages, system services and inter-process communication.
- android.text − Used to render and manipulate text on a device display.
- android.view − The fundamental building blocks of application user interfaces.
- android.widget − A rich collection of pre-built user interface components such as buttons, labels, list views, layout managers, radio buttons etc.
- android.webkit − A set of classes intended to allow web-browsing capabilities to be built into applications.

Having covered the Java-based core libraries in the Android runtime, it is now time to turn our attention to the C/C++ based libraries contained in this layer of the Android software stack.

Android Runtime
This is the third section of the architecture and available on the second layer from the bottom. This section provides a key component called Dalvik Virtual Machine which is a kind of Java Virtual Machine specially designed and optimized for Android. The Dalvik VM makes use of Linux core features like memory management and multi-threading, which is intrinsic in the Java language. The Dalvik VM enables every Android application to run in its own process, with its own instance of the Dalvik virtual machine. Android runtime also provides a set of core libraries which enable Android application developers to write Android applications using standard Java programming language.

Application Framework
The Application Framework layer provides many higher-level services to applications in the form of Java classes. Application developers are allowed to make use of these services in their applications.
The Android framework includes the following key services −

- Activity Manager − Controls all aspects of the application lifecycle and activity stack.
- Content Providers − Allows applications to publish and share data with other applications.
- Resource Manager − Provides access to non-code embedded resources such as strings, color settings and user interface layouts.
- Notifications Manager − Allows applications to display alerts and notifications to the user.
- View System − An extensible set of views used to create application user interfaces.

Applications
You will find all the Android application at the top layer. You will write your application to be installed on this layer only. Examples of such applications are Contacts Books, Browser, Games etc.

2a. Elaborate on the resource files in the Android Studio System (contents of res directory).

[6.0] 1 [1, 2, 3] [1]

Scheme: RES directory structure 3M + Description 3M

Solution
Resources are the additional files and static content that your code uses, such as bitmaps, layout definitions, user interface strings,

animation instructions, and more.

You should always externalize app resources such as images and strings from your code, so that you can maintain them independently. You should also provide alternative resources for specific device configurations, by grouping them in specially-named resource directories. At runtime, Android uses the appropriate resource based on the current configuration. For example, you might want to provide a different UI layout depending on the screen size or different strings depending on the language settin~

```
MyProject/
    src/
        MyActivity.java
    res/
        drawable/
            graphic.png
        layout/
            main.xml
            info.xml
        mipmap/
            icon.png
        values/
            strings.xml
```

Directory	Resource Type
`animator/`	XML files that define property animations.
`anim/`	XML files that define tween animations. (Property animations can also be saved in this directory, but the `animator/` directory is preferred for property animations to distinguish between the two types.)
`color/`	XML files that define a state list of colors. See Color State List Resource
`drawable/`	Bitmap files (`.png`, `.9.png`, `.jpg`, `.gif`) or XML files that are compiled into the following drawable resource subtypes: • Bitmap files • Nine-Patches (re-sizable bitmaps) • State lists • Shapes • Animation drawables • Other drawables

Once you externalize your app resources, you can access them using resource IDs that are generated in your project's R class. Figures show above depicts how to group your resources in your Android project and provide alternative resources for specific device configurations, and then access them from your app code or other XML files.

2b. List at least 4 latest android versions with their names and supported API levels.

[4.0] 1 [1, 2, 3] [1]

Scheme: 4 versions with API levels 1M x 4 = 4M

Solution

Android versions, name, and API level

Code name	Version numbers	API level	Release date
No codename	1.0	1	September 23, 2008
No codename	1.1	2	February 9, 2009
Cupcake	1.5	3	April 27, 2009
Donut	1.6	4	September 15, 2009
Eclair	2.0 - 2.1	5 - 7	October 26, 2009
Froyo	2.2 - 2.2.3	8	May 20, 2010
Gingerbread	2.3 - 2.3.7	9 - 10	December 6, 2010
Honeycomb	3.0 - 3.2.6	11 - 13	February 22, 2011
Ice Cream Sandwich	4.0 - 4.0.4	14 - 15	October 18, 2011
Jelly Bean	4.1 - 4.3.1	16 - 18	July 9, 2012
KitKat	4.4 - 4.4.4	19 - 20	October 31, 2013
Lollipop	5.0 - 5.1.1	21- 22	November 12, 2014
Marshmallow	6.0 - 6.0.1	23	October 5, 2015
Nougat	7.0	24	August 22, 2016
Nougat	7.1.0 - 7.1.2	25	October 4, 2016
Oreo	8.0	26	August 21, 2017
Oreo	8.1	27	December 5, 2017
Pie	9.0	28	August 6, 2018
Android 10	10.0	29	September 3, 2019
Android 11	11	30	September 8, 2020

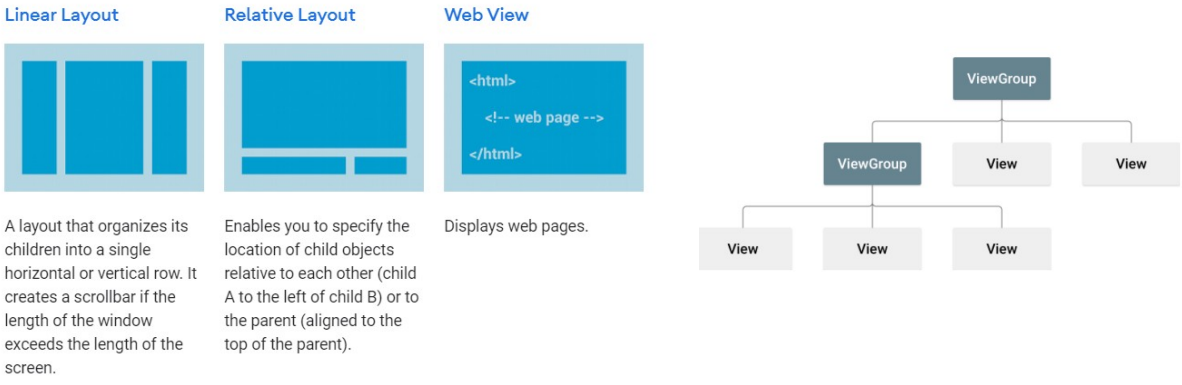
3a. List and explain different layout view groups.

[5.0] 1 [1, 2, 3] [1]

Scheme: 4 Layout Groups x 1M + Description 1M

Solution

Common Layouts:



A layout defines the structure for a user interface in your app, such as in an activity. All elements in the layout are built using a hierarchy of View and ViewGroup objects. A View usually draws something the user can see and interact with. Whereas a ViewGroup is an invisible container that defines the layout structure for View and other ViewGroup objects, as shown in figure 2. Each subclass of the ViewGroup class provides a unique way to display the views you nest within it. Figure 1 are some of the more common layout types that are built into the Android platform.

3b. List and explain Android App Components and Activity Components.

[5.0] 1 [1, 2, 3] [2]

Scheme: 1 Component 1M x 5 = 5M

Solution

Application components are the essential building blocks of an Android application. These components are loosely coupled by the application manifest file AndroidManifest.xml that describes each component of the application and how they interact.

There are following four main components that can be used within an Android application;

i. Activities: They dictate the UI and handle the user interaction to the smart phone screen.
ii. Services: They handle background processing associated with an application.
iii. Broadcast Receivers: They handle communication between Android OS and applications.
iv. Content Providers: They handle data and database management issues.

Activities

An activity represents a single screen with a user interface,in-short Activity performs actions on the screen. For example, an email application might have one activity that shows a list of new emails, another activity to compose an email, and another activity for reading emails. If an application has more than one activity, then one of them should be marked as the activity that is presented when the application is launched.

An activity is implemented as a subclass of Activity class as follows −

```
public class MainActivity extends Activity {
}
```

Services

A service is a component that runs in the background to perform long-running operations. For example, a service might play music in the background while the user is in a different application, or it might fetch data over the network without blocking user interaction with an activity.

A service is implemented as a subclass of Service class as follows −

```
public class MyService extends Service {
}
```

Broadcast Receivers

Broadcast Receivers simply respond to broadcast messages from other applications or from the system. For example, applications can also initiate broadcasts to let other applications know that some data has been downloaded to the device and is available for them to use, so this is broadcast receiver who will intercept this communication and will initiate appropriate action.

A broadcast receiver is implemented as a subclass of BroadcastReceiver class and each message is broadcaster as an Intent object.

```
public class MyReceiver  extends  BroadcastReceiver {
   public void onReceive(context,intent){}
}
```

Content Providers

A content provider component supplies data from one application to others on request. Such requests are handled by the methods of the ContentResolver class. The data may be stored in the file system, the database or somewhere else entirely.

A content provider is implemented as a subclass of ContentProvider class and must implement a standard set of APIs that enable other applications to perform transactions.

```
public class MyContentProvider extends  ContentProvider {
   public void onCreate(){}
}
```

4. With a neat diagram explain the Acvity Life Cycle.

[10.0] 1 [1, 2, 3] [2]

Scheme: Diagram 6M + Description 4M = **10M**

To navigate transitions between stages of the activity lifecycle, the Activity class provides a core set of six callbacks: onCreate(), onStart(), onResume(), onPause(), onStop(), and onDestroy(). The system invokes each of these callbacks as an activity enters a new state.

As the user begins to leave the activity, the system calls methods to dismantle the activity. In some cases, this dismantlement is only partial; the activity still resides in memory (such as when the user switches to another app), and can still come back to the foreground.
If the user returns to that activity, the activity resumes from where the user left off. With a few exceptions, apps are restricted from starting activities when running in the background.

The system's likelihood of killing a given process—along with the activities in it—depends on the state of the activity at the time.
Activity state and ejection from memory provides more information on the relationship between state and vulnerability to ejection.

Depending on the complexity of your activity, you probably don't need to implement all the lifecycle methods. However, it's important that you understand each one and implement those that ensure your app behaves the way users expect.

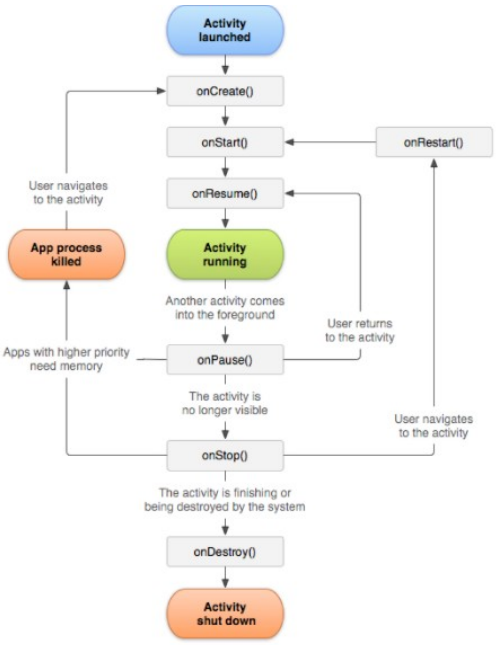


Figure 1. A simplified illustration of the activity lifecycle.

5a. Define Intent. Explain different types of Intents.

[4.0] 1 [1, 2, 3] [1]

Scheme: Definition 2M + Types 2 M

Solution

An intent is an abstract description of an operation to be performed. It can be used with startActivity to launch an Activity, broadcastIntent to send it to any interested BroadcastReceiver components, and Context.startService(Intent) or Context.bindService(Intent, ServiceConnection, int) to communicate with a background Service.

An Intent provides a facility for performing late runtime binding between the code in different applications. Its most significant use is in the launching of activities, where it can be thought of as the glue between activities. It is basically a passive data structure holding an abstract description of an action to be performed.

There are two types of intents:

Explicit intents specify which application will satisfy the intent, by supplying either the target app's package name or a fully-qualified component class name. You'll typically use an explicit intent to start a component in your own app, because you know the class name of the activity or service you want to start. For example, you might start a new activity within your app in response to a user action, or start a service to download a file in the background.

An explicit intent is one that you use to launch a specific app component, such as a particular activity or service in your app. To create an explicit intent, define the component name for the Intent object—all other intent properties are optional.

For example, if you built a service in your app, named DownloadService, designed to download a file from the web, you can start it with the following code:

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

Implicit intents do not name a specific component, but instead declare a general action to perform, which allows a component from another app to handle it. For example, if you want to show the user a location on a map, you can use an implicit intent to request that another capable app show a specified location on a map.

An implicit intent specifies an action that can invoke any app on the device able to perform the action. Using an implicit intent is

useful when your app cannot perform the action, but other apps probably can and you'd like the user to pick which app to use.

For example, if you have content that you want the user to share with other people, create an intent with the ACTION_SEND action and add extras that specify the content to share. When you call startActivity() with that intent, the user can pick an app through which to share the content.

```
// Create the text message with a string.
Intent sendIntent = new Intent();
sendIntent.setAction(Intent.ACTION_SEND);
sendIntent.putExtra(Intent.EXTRA_TEXT, textMessage);
sendIntent.setType("text/plain");

// Try to invoke the intent.
try {
    startActivity(sendIntent);
} catch (ActivityNotFoundException e) {
    // Define what your app should do if no activity can handle the intent.
}
```

5b. Demonstrate the usage of Explicit Intent with Code Snippet.

[6.0] 1 [1, 2, 3] [3]

Scheme: Explanation 3M + Code snippet 3M

Solution

An explicit intent is one that you use to launch a specific app component, such as a particular activity or service in your app. To create an explicit intent, define the component name for the Intent object—all other intent properties are optional.

For example, if you built a service in your app, named DownloadService, designed to download a file from the web, you can start it with the following code:

```
// Executed in an Activity, so 'this' is the Context
// The fileUrl is a string URL, such as "http://www.example.com/image.png"
Intent downloadIntent = new Intent(this, DownloadService.class);
downloadIntent.setData(Uri.parse(fileUrl));
startService(downloadIntent);
```

6. List and explain the challenges of Android app development.

[10.0] 1 [1, 2, 3] [2]

Scheme: 5 challenges x 2M

Solution

The challenges of Android app development

While the Android platform provides rich functionality for app development, there are still a number of challenges you need to address, such as:

i. **Building for a multi-screen world:** Devices can come in different sizes and shapes affect the screen designs for UI elements in your application.

ii. **Getting performance right:** how fast it runs, how easily it connects to the network, how well it manages battery and memory usage—is affected by factors such as life, multimedia content, and Internet access. You must be aware of these and write code in such a way that the resource utilization is balanced and optimally. For example, you will have to balance the background services enabling them only when necessary; this will save battery life of the user's device.

iii. **Keeping your code and your users secure:** You need to take precautions to secure code and the user's experience when using your app. Use tools such as ProGuard provided in Android Studio), which detects and removes unused classes, fields, methods, and attributes, and encrypt all of your app's code and resources while the app.

iv. **Remaining compatible with older platform versions:** Consider how to add new Android platform version features to an app, while ensuring that the app can still run on devices with old platform versions.

v. **Understanding the market and the user.**