

Lesson-8

WebView, Shared Preferences and JSON

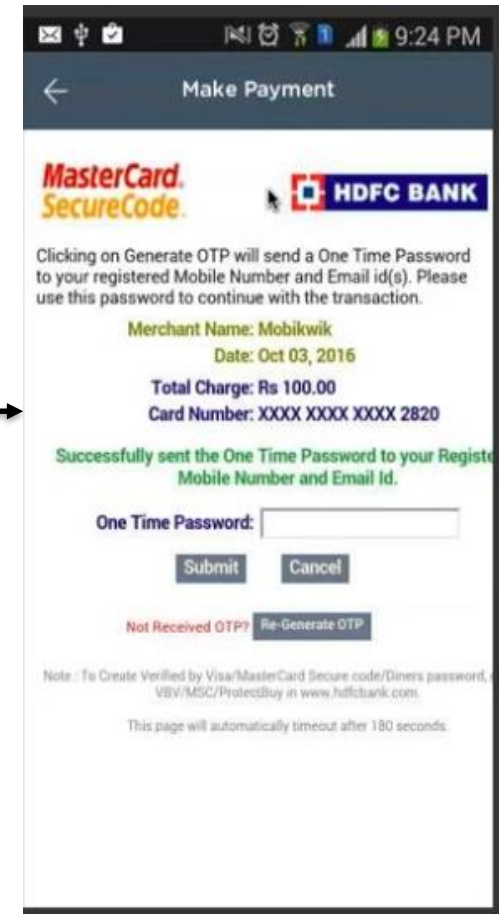
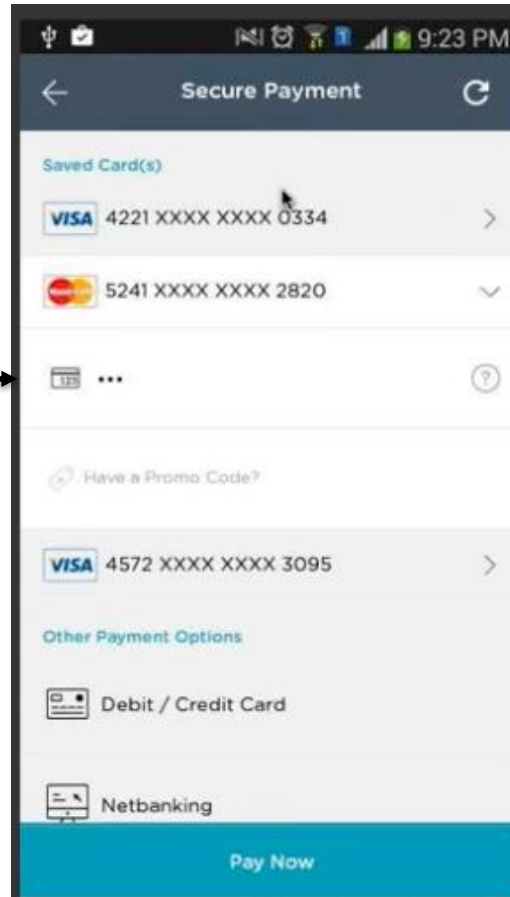
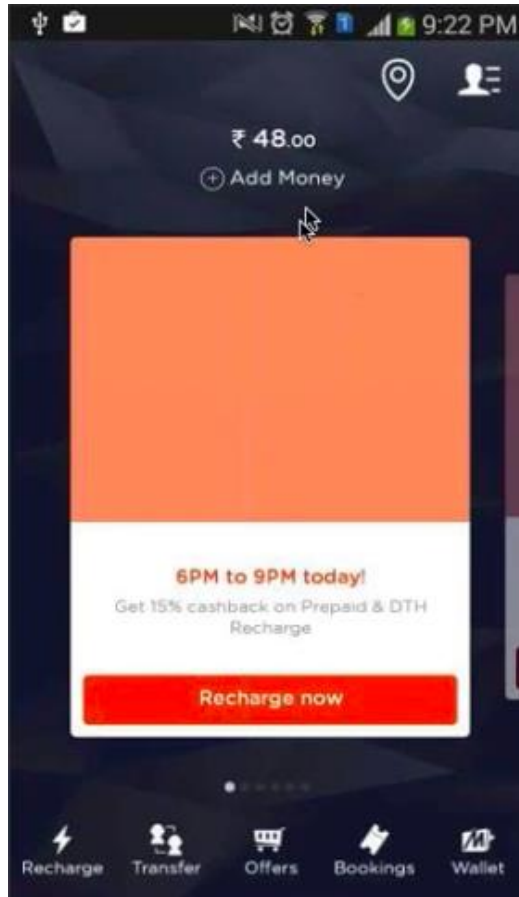
Contents

- **WebView**
 - Introduction
 - WebView Operations
 - Hands on Example
 - Methods in WebViewClient class
 - How create your own HTML Page
- **SharedPreferences**
 - Introduction
 - Applications of Shared Preferences
 - Steps for creating Shared Preferences
 - Retrieve, Update and Delete preferences
 - JSON, Gson
 - External Storage, Internal Storage
 - Retrofit to read data from Network JSON data

Introduction

- WebView is one of the UI component in Android, which is used to display webpages in your application.
- If you want to deliver a web application (or just a web page) as a part of a client application, you can do it using WebView
- The WebView class is an extension of Android's View class that allows you to display web pages as a part of your activity layout.
- Sample Scenario example : An Android app called MobikWik is a mobile recharge app, once you click on the Pay Now button, it will take the Bank Webpage.

Example



Working with WebView

Adding the Widget : To add a [WebView](#) to your Application, simply include the <WebView> element in your activity layout.

```
<WebView android:id="@+id/webview"/>
```

Loading Content Via URL : To load a web page in the [WebView](#), use [loadUrl\(\)](#).

```
WebView myWebView = (WebView) findViewById(R.id.webview);  
myWebView.loadUrl("http://www.example.com");
```

However, we also must make one change to AndroidManifest.xml, adding a line where we request permission to access the Internet:

```
<uses-permission android:name="android.permission.INTERNET" />
```

Reading Resource : <https://developer.android.com/guide/webapps/index.html>

WebView Operations

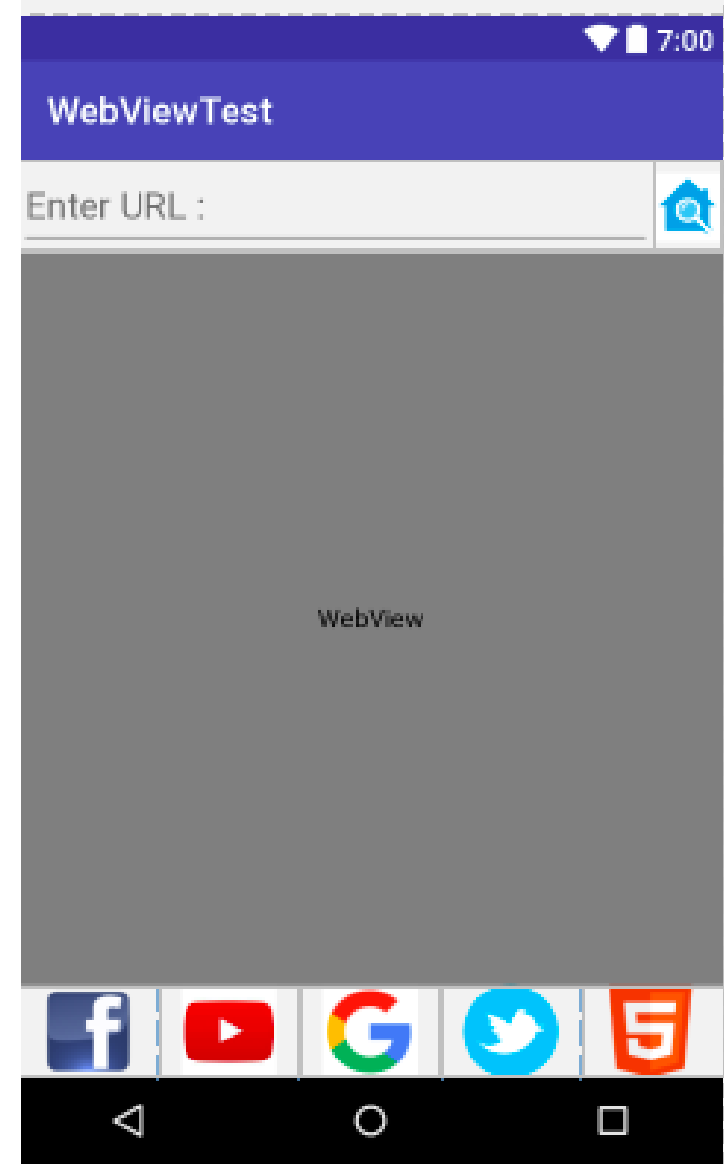
By using Webview we can perform 4 operations.

1. call browser
2. integrate browser
3. display .HTML files
4. we can provide a communication between
HTML UI & Android Activity.

Hands on example-1

Problem : Type the expected URL in the EditText UI and click the search button to load the webpage in the WebView component. Below the WebView there are five Images to open the specified URL mostly preferred by the users. Click that image to load the web page in the WebView component.

Refer : Lesson7\WebViewTest



AndroidManifest.xml

Add the highlighted text in the AndroidManifest.xml to access internet user permission and usesCleartextTraffic:true to avoid url loading error

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    package="com.example.l7webviewtest">
    <uses-permission android:name="android.permission.INTERNET"></uses-permission>
    <application
        android:allowBackup="true"
        android:icon="@mipmap/ic_launcher"
        android:label="l7WebViewTest"
        android:roundIcon="@mipmap/ic_launcher_round"
        android:supportsRtl="true"
        android:usesCleartextTraffic="true"
        android:theme="@style/AppTheme">
        <activity android:name=".MainActivity">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <action android:name="android.intent.action.VIEW" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
</manifest>
```


MainActivity.java

*/*test method decide action based on the image view selected to load the Webpage in WebView UI*/*

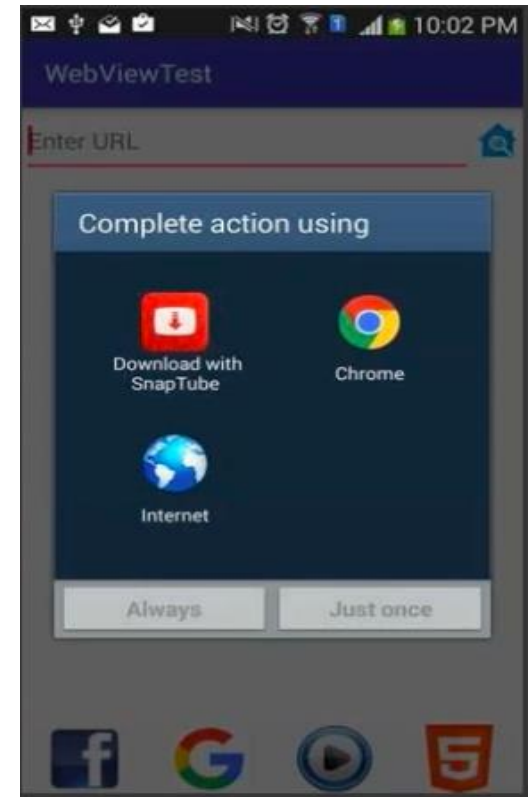
```
fun test(v:View) {  
    when (v.getId()) {  
        R.id.srch -> {  
            /* Load the EditText URL to the WebView UI, it is mendatory to  
mention http:// either here or at the runtime in EditText*/  
            wview.loadUrl("http://" + et1.getText().toString())  
        }  
        R.id.fb -> wview.loadUrl("http://facebook.com")  
        R.id.youtube -> wview.loadUrl("http://youtube.com")  
        R.id.google -> wview.loadUrl("http://google.com")  
        R.id.twitter -> wview.loadUrl("http://twitter.com")  
        R.id.html -> wview.loadUrl("file:///android_asset/login.html")  
    }  
}
```

WebViewClient

Once you run the app if you click the facebook, youtube, google and twitter image, it will produce the following screen, because just we loaded the URL, but we didn't integrate the browser.

Handling Page Navigation

- When the user clicks a link from a web page in your WebView, the default behavior is for Android to launch an application that handles URLs.
- Usually, the default web browser opens and loads the destination URL. However, you can override this behavior for your WebView, so links open within your WebView.
- You can then allow the user to navigate backward and forward through their web page history that's maintained by your WebView



WebViewClient

- To open links clicked by the user, simply provide a **WebViewClient** for your **WebView** using **setWebViewClient**.

- For example:

```
var myWebView = (WebView) findViewById(R.id.webview);  
myWebView.webViewClient = WebViewClient()
```

Supporting JavaScript

- Now, you may be tempted to replace the URL with something else that relies upon JavaScript. You will find that such pages do not work especially well by default.(see the image).
- That is because, by default, JavaScript is turned off in WebView widgets.
- If you want to enable JavaScript, call the below method on the WebView instance.

`wview.settings.javascriptEnabled = true;`

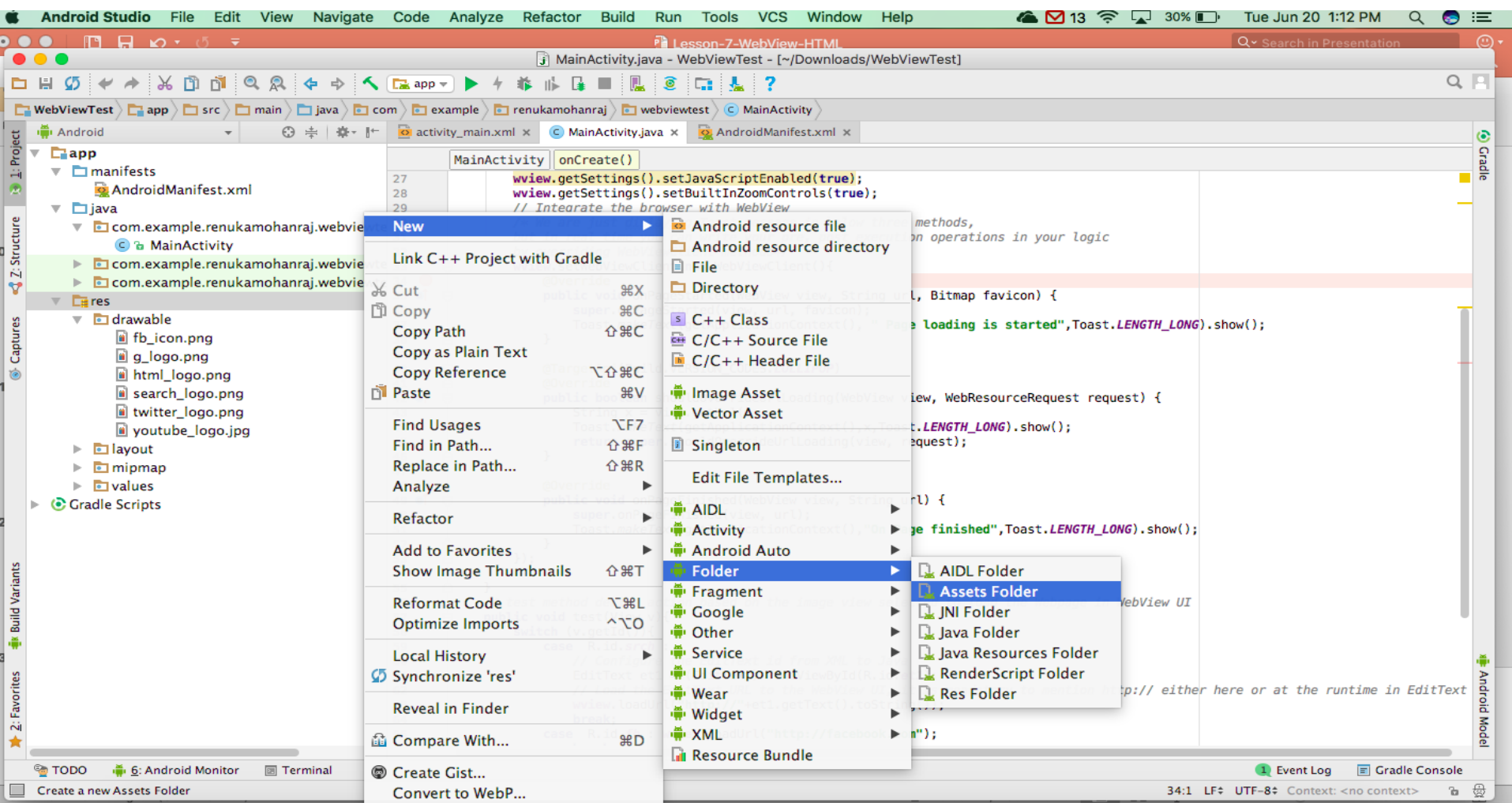
- At this point, any JavaScript referenced by your Web page should work normally.
- Sets whether the WebView should use its built-in zoom mechanisms.
- The built-in zoom mechanisms comprise on-screen zoom controls, which are displayed over the WebView's content, and the use of a pinch gesture to control zooming. To enable by giving

`wview.settings.builtInZoomControls = true;`



Creating own HTML Page

Step 1 : Create an assets folder in your res directory as per the given screen shot.



Creating own HTML Page

Step – 2 : Create an HTML under assets folder. By Right Click assets→New→File.
Then login.html as a file name.

```
<html>
<body bgcolor="#FF0000" text="FFFFFF">
<head>
</head>
<center>
  <h2> Welcome 2 MUM </h2>
  <table>
    <tr><td colspan='2' align="center">Login Form </td></tr>
    <tr><td>Enter UName : </td>
    <td> <input type="text" id="name"/> </td></tr>
    <tr><td>Enter Pass : </td>
    <td> <input type="password" id="pass"/> </td></tr>
    <tr><td colspan='2' align="center">
      <input type="button" value="Login"/> </td></tr>
  </table>
</center>
</body>
</html>
```



Welcome 2 MUM

Login Form

Enter UName :

Enter Pass :

Login

Creating own HTML Page

Step – 3 : Load login.html in MainActivity.java using **R.id.html ->**

```
wview.loadUrl("file:///android_asset/login.html")
```

Now the requirement is to get the username and password to the Android activity.

- Using JavaScript interface we can provide the communication between HTML UI & Android activity by calling

```
wview.addJavaScriptInterface(class_object, interface_name);
```

- Example :

```
wview.addJavaScriptInterface(this, "myinterface");
```

- Using the second parameter interface_name we can communicate with the specified class methods and variables from JavaScript.

JavaScript logic

Step 4 : Once the user click the login button, need of JavaScript logic in HTML.

```
<tr><td colspan='2' align="center">  
  <input type="button" value="Login" onclick="login()"/>  
</td></tr>
```

- Write a JavaScript code inside the <head> tag. The reason behind this is as the Head gets loaded before the body. Any dynamic Javascript code that gets executed in the body on load will execute correctly. Also to speed up the process.
- The Document interface represents the entire HTML or XML document. Conceptually, it is the root of the document tree, and provides the primary access to the document's data.
- Element getElementById (String elementId) - Returns the Element that has an ID attribute with the given value. If no such element exists, this returns null .
- To retrieve the value invoke getElementById("ide").value

JavaScript logic

login.html

<head>

<script language="JavaScript">

function login(){

var name=document.getElementById('name').value;

var pass=document.getElementById('pass').value;

myinterface.displayMsg(name,pass);

}

</script>

</head>

MainActivity.java - add the below method

@JavascriptInterface // Annotation that allows exposing methods to JavaScript.

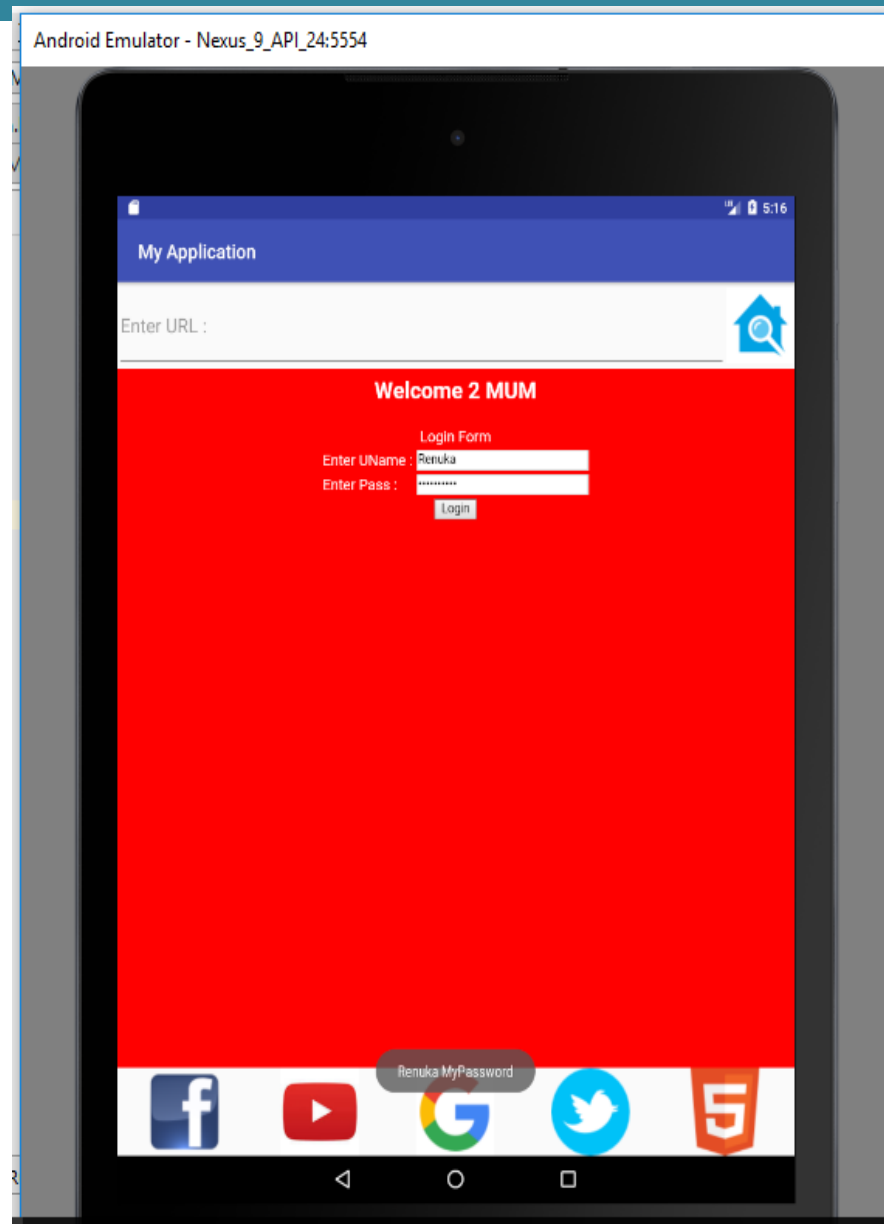
fun displayMsg(name: String, pass: String) {

Toast.makeText(applicationContext, "\$name \$pass",

Toast.LENGTH_LONG).show()

}

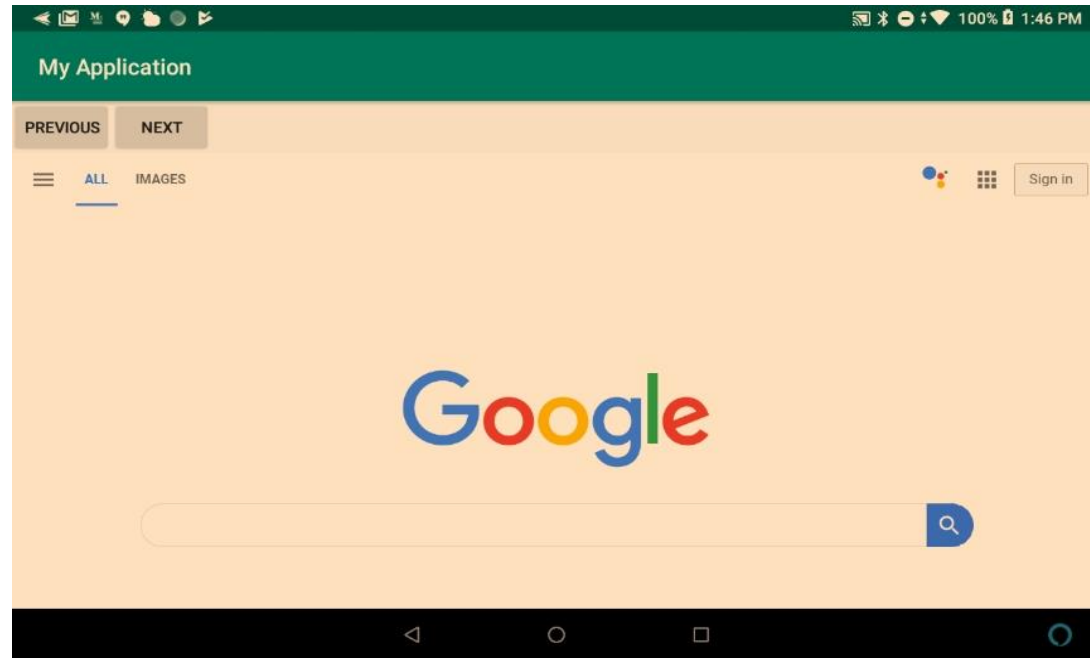
HTML Click Action Result Screen



Hands on Example-2 - Navigation

- The Screen load the url google.com.
- You can browse through this url.
- Click PREVIOUS and NEXT button to navigate.
- If there is no previous and next webpages get a Toast no page available.

Refer : MyApplication



Methods in WebViewClient class

Create an Inner class with inherit from WebViewClient class, override the below methods according to your application requirements.

1. void onPageStarted () - Notify the host application that a page has started loading.
2. boolean shouldOverrideUrlLoading() - Give the host application a chance to take over the control when a new url is about to be loaded in the current WebView.
3. void onPageFinished () - Notify the host application that a page has finished loading.

For more information WebViewClient refer :

<https://developer.android.com/reference/android/webkit/WebViewClient.html>

WebViewClient Methods

```
inner class MyWebClient : WebViewClient() {  
    // Page Loading started  
    override fun onPageStarted(view: WebView, url: String, favicon: Bitmap?) {  
        Toast.makeText(this@MainActivity, "Page Started",  
            Toast.LENGTH_LONG).show()  
    }  
    //Loading the Url  
    override fun shouldOverrideUrlLoading(view: WebView, url: String): Boolean {  
        view.loadUrl(url)  
        return true  
    }  
    // Page Loading finished  
    override fun onPageFinished(view: WebView, url: String) {  
        Toast.makeText(this@MainActivity,  
            "PageFinished", Toast.LENGTH_LONG).show()  
        super.onPageFinished(view, url)  
    }  
}
```

Navigating web page history

- When your WebView overrides URL loading, it automatically accumulates a history of visited web pages.
- You can navigate backward and forward through the history with `goBack()` and `goForward()`.
- The `canGoBack()` method returns true if there is web page history for the user to visit. Likewise, you can use `canGoForward()` to check whether there is a forward history. If you don't perform this check, then once the user reaches the end of the history, `goBack()` or `goForward()` does nothing.

MainActivity.java

```
override fun onCreate(savedInstanceState: Bundle?) {
    super.onCreate(savedInstanceState)
    setContentView(R.layout.activity_main)
    url = "http://google.com"
    webView.loadUrl(url)
    webView.webViewClient = MyWebClient()
    webView.settings.javaScriptEnabled = true
    webView.settings.builtInZoomControls = true
    btnprev!!.setOnClickListener {
        if (webView.canGoBack()) {
            webView.goBack() }
        else{
            Toast.makeText(this,"NO previous history available",Toast.LENGTH_LONG).show()
        }
    }
    btnnext!!.setOnClickListener {
        if (webView.canGoForward()) {
            webView.goForward() }

        else{
            Toast.makeText(this,"NO back history available",Toast.LENGTH_LONG).show() }
    }
}
```

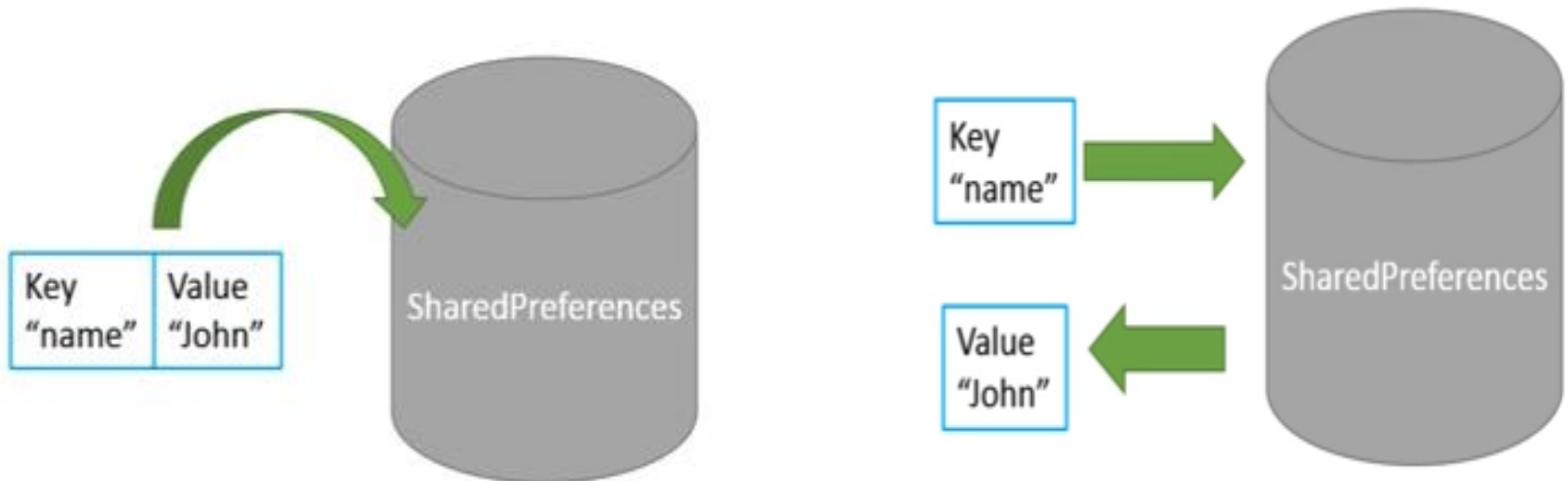
Shared Preferences

Introduction

- There are several options in android to save persistent application data. The solution you choose depend upon the requirement of the project such as, whether data should be private to your application or accessible to other application also. Following are the options which can be used for storing the data.
 - **Shared Preferences:** Store data as key-value pairs.
 - **Internal Storage:** Store data on the device memory. Data will not be accessed by other application.
 - **External Storage:** Store files on the shared external file system. This is usually for shared user files, such as photos. (SD Card)
 - **SQLite Databases:** Store structured data in a private database.
 - **Network Connection:** Store data on the Web with your own network server.
 - Refer about storage :
<https://developer.android.com/guide/topics/data/data-storage.html>

SharedPreferences

- Android provides SharedPreferences object to help the developer to save simple application data.
- Using the SharedPreferences object, the developer can save and retrieve the desired data using key/value pairs.



SharedPreferences

- The developer needs to specify a key for the data that is required to be saved, and then both the key and its value will be saved automatically to an XML file.
- It enables the developer to save primitive data types such as boolean, float, long, string, and so on.
- The SharedPreferences class is present in the `android.content` package and the developer needs to import the class to work with an object of the class.

Applications of SharedPreferences

- Last data user entered in your application
- Store the last updates of date and time
- Credentials – Remember user details like username and password
- Location catching – Identify the last location
- Store the login pattern
- Game's high score and current level.

Steps for creating SharedPreferences

Step 1 : Obtain the Shared Preference object by calling `getSharedPreferences()`.

```
val prefs = getSharedPreferences(prefName, MODE_PRIVATE)
```

Parameters :

`prefName` : name of the preference file as String

`mode` : operating file creation mode as int.

That is `MODE_PRIVATE` (takes 0 – Zero)

Step 2 : We need an editor object to make changes which is `SharedPreferences.Editor`.

- This is done by invoking the `edit()` method on the object of the `SharedPreferences` class as shown

```
val editor = prefs.edit()
```

Step 3 : Next, the developer can add key/value pairs to the Editor object by invoking its various methods such as `putBoolean()`, `putString()`, `putLong()`, `putInt()`, or `putFloat()`(which method to use depends on the type of data that is being saved) as shown below

```
editor.putString("username", "RenukaMohanraj")  
editor.putBoolean("authenticated", true)
```

We stored 2 key-value pairs. This is how your `MyPrefs.xml` should look s like :

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>  
<map>  
  <string name="username">RenukaMohanraj</string>  
  <boolean name="authenticated"> value="true" />  
</map>
```

Step 4 : Finally, the `apply()` method of the `Editor` class is invoked to commit the changes back to the `SharedPreferences` object so that the values can be saved to persistent storage.

```
editor.apply();
```

- If you want send as Object to Shared Preferences make use of JSON.

Retrieving Preferences

- Fetching the preferences is done directly on the SharedPreferences object.
- So SharedPreferences.Editor is not required.
- There are several get methods for this job like getBoolean(), getFloat(), getInt(), getLong(), and getString().
- All of them accept two arguments where the first is the name of the key while the second non-optional one is the default value to return if the preference does not exists (is undefined).

```
val prefs = getSharedPreferences("MyPrefs",  
                                Context.MODE_PRIVATE)  
val username = prefs.getString("username", "")  
val logged_in = prefs.getBoolean("logged_in", false)
```


Updating Preferences

- Updating the preferences is similar to the setting them that we just learnt a bit back.
- Get the SharedPreferences.Editor object, set values using the put*() methods and then commit your changes.

```
val pref = getSharedPreferences("MyPrefs",  
Context.MODE_PRIVATE)  
// We need an editor object to make changes  
val edit = prefs.edit();  
// Modify the Stored data  
edit.putString("username", "Renuka")  
edit.putBoolean("logged_in", false)  
// Commit the changes  
edit.apply()
```

Deleting Preferences & Contains check

- Deleting a particular preference is very simple. Just call the `remove()` method and commit your changes.

// Remove a key

```
prefs.remove("username")
```

// Commit changes


```
prefs.apply()
```

- You can check the exists or not by using the `contains` method on your `SharedPreferences` instance:

// Check the key exist or not

```
boolean hasUser = preference.contains("username")
```

Where Shared preference data goes?

- Internally SPF will maintain the data in an XML file, we can explore the SharedPreferences xml file using Android Studio's [Device File Explorer](#)
- Click View → Tool Windows → Device File Explorer or click the Device File Explorer  button in the tool window bar to open the Device File Explorer.
- The file will be saved under the directory `data/data/<application_package_name>/shared` preference folder.
- For more information to view the shared preference persisted data:
 - Refer the Lesson8 SPF Step by Step View Persisted data.pdf

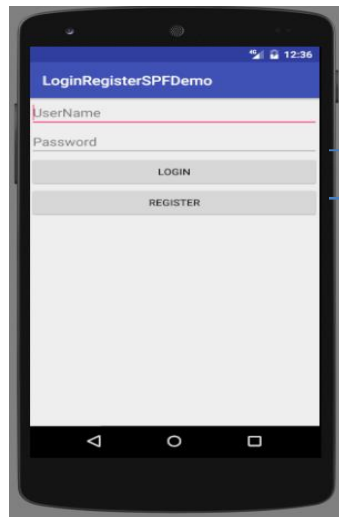
- By using SharedPreferences, we can not maintain huge amount of data , but for every value we must give a unique key its difficult to remember & assign new keys for every value , that's why SharedPreferences is preferred to maintain the limited amount of data.
- If you want to maintain huge amount of data, Android is preferred to use SQLite DB.

Hands-on-Examples

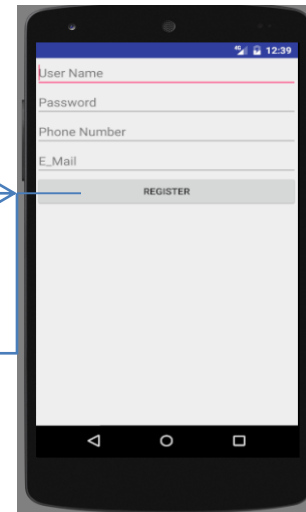
Problem Requirement : Design an application which preform, if the user already exists, they can directly click the Login button else by clicking Register button to register the information. Here we are using SharedPreferences to store the information of the last registered user.

Refer : SPFLoginRegister

activity_main.xml



activity_register.xml



MainActivity.kt

```
class MainActivity : AppCompatActivity() {  
    lateinit var spf:SharedPreferences  
    override fun onCreate(savedInstanceState: Bundle?) {  
        super.onCreate(savedInstanceState)  
        setContentView(R.layout.activity_main)  
        spf = getSharedPreferences("login", Context.MODE_PRIVATE)  
        // key, value pair. Here by default name is not found assign " no value"  
        val name = spf.getString("name", "no value")  
        val pwd = spf.getString("pass", "no value")  
        et1.setText(name)  
        et2.setText(pwd)  
    }  
}
```

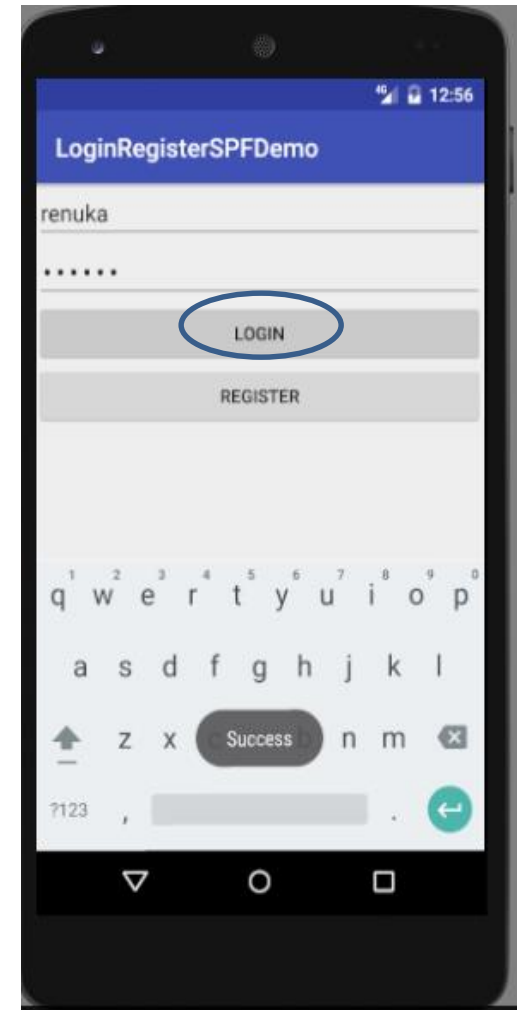
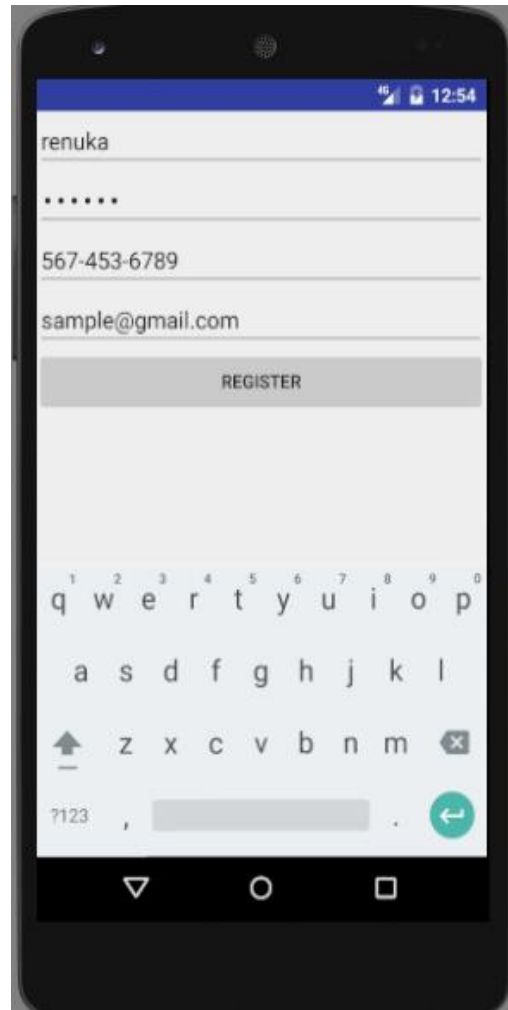
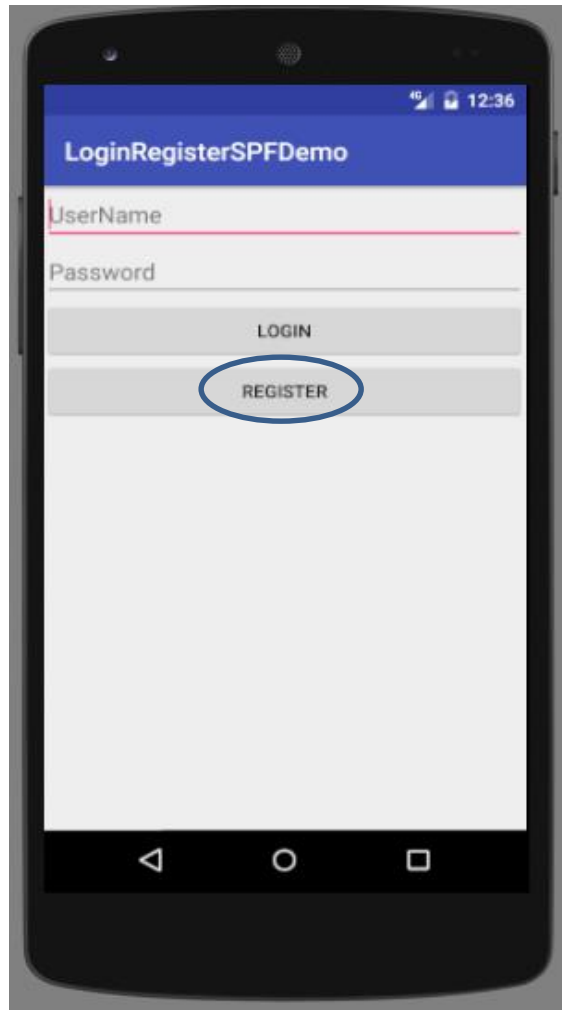
MainActivity.kt

```
// Check whether the user exist already in the SharedPreferences.
fun login(view: View) {
    // To get the SharedPreferences using its name
    // SharedPreferences spf = getSharedPreferences("login", Context.MODE_PRIVATE);
    val name = spf.getString("name", "no value") // key, value pair. Here by default name is
not found assign " no value"
    val pwd = spf.getString("pass", "no value")
    if ((et1.text.toString().equals(name,ignoreCase = true) &&
et2.getText().toString().equals(pwd))) {
        Toast.makeText(getApplicationContext(), "Success", Toast.LENGTH_LONG).show()
    }
    else {
        Toast.makeText(getApplicationContext(), "Fail to Login",
Toast.LENGTH_LONG).show() } }
fun register(view:View) {
    // We are navigating through to the RegisterActivty from ManinActivity using Intent
    val i = Intent(this, RegisterActivity::class.java)
    startActivity(i)
}
}
```

RegisterActivity.kt

```
class RegisterActivity : AppCompatActivity() {
    override fun onCreate(savedInstanceState: Bundle?) {
        super.onCreate(savedInstanceState)
        setContentView(R.layout.activity_register)
    }
    fun register(view: View) {
        // Create and Read the SharedPreferences
        val spf = getSharedPreferences("login", Context.MODE_PRIVATE)
        // To write a data using SharedPreferences Object
        val spe = spf.edit()
        // Using put method to write the data in SharedPreferences
        spe.putString("name", et1.text.toString())
        spe.putString("pass", et2.text.toString())
        spe.putString("phone", et3.text.toString())
        spe.putString("email", et4.text.toString())
        spe.apply()
        Toast.makeText(this, "Registered", Toast.LENGTH_LONG).show()
        // Once finished writing we need to go back to the main activity to show the Login
        finish() // automatically destroy the activity and give the visibility of Main Activity
    }
}
Register this activity in AndriodManifest.xml
<activity android:name=".RegisterActivity"></activity>
```


Run the Code



Demo Folder

- LoginRegisterSPF
- SharedPreferencesDemo

JSON

- JSON stands for JavaScript Object Notation. It is an independent data exchange format and is the best alternative for XML.
- A modifiable set of name/value mappings. Names are unique, non-null strings. Values may be any mix of JSONObjects, JSONArrays, Strings, Booleans, Integers, Longs, Doubles. Values may not be null.
- It is easy for machines to parse and generate" (for *more info* : www.json.org)
- JSON object represented using { } braces. JSON arrays represented using [] braces.

XML & JSON Format of Employee record

XML Format

```
<employees>
  <employee>
    <id> 123 </id>
    <name> Renuka </name>
    <desig> AP </desig>
    <dept> CS </dept>
  </employee>
  <employee>
    <id> 125 </id>
    <name> Mohanraj </name>
    <desig> GD </desig>
    <dept> CS </dept>
  </employee>
</employees>
```

JSON Format

```
{“employees”:[
  {
    “id” : 123,
    “name”: “Renuka”,
    “desig” : “AP”,
    “dept”:”CS”
  },
  {
    “id” : 125,
    “name”: “Mohanraj”,
    “desig” : “GD”,
    “dept”:”CS”
  }
]}
```

XML vs JSON

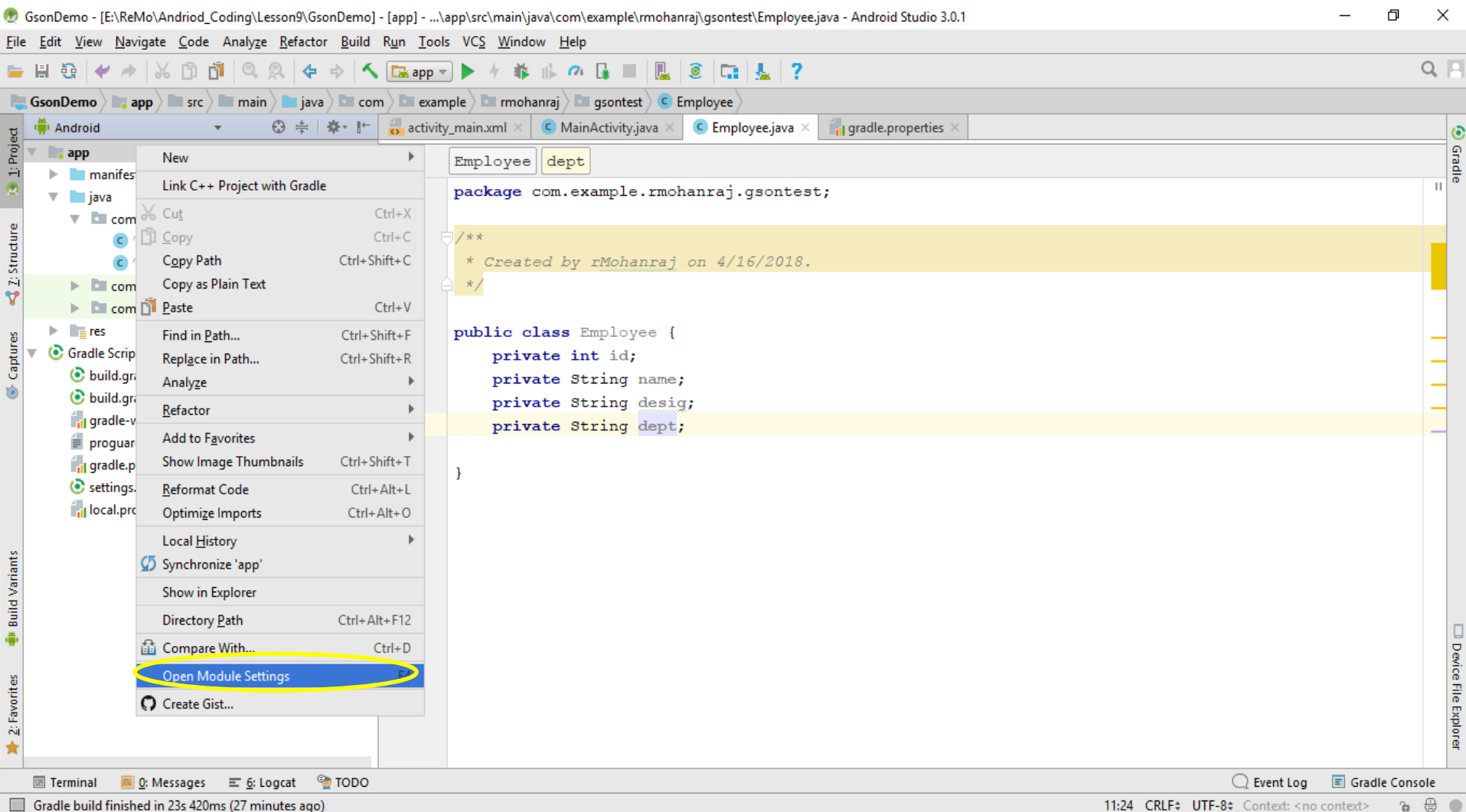
- Both are used to transfer data from one technology to another technology.
- JSON occupies less space and load faster than XML.
- Information is represented as a collection of key/value pairs, and that each key/value pair is grouped into an ordered list of objects. JSON can provide data type like Integer, String.
 - “id” : 123
 - “name”: “Renuka”
- JSON parsing is simple. Converting object to JSON and JSON to object.
- Android has a build-in support for JSON parsing.
- Third party libraries are available in the market for parsing such as GSON, Retrofit, Jackson & Jettison etc.
- In this course we will use GSONS and Retrofit.

Google Gson (Gson)

- Gson is a Java library that can be used to convert Java Objects into their JSON representation.
- It can also be used to convert a JSON string to an equivalent Java object.
- It has extensive support for java generics.
- Provide simple toJson() and fromJson() methods to convert Java objects to JSON and vice-versa.
- To use Gson in Android add the below dependencies add directly in the gradle Module:app or follow the next few slides to add dependency.
implementation 'com.google.code.gson:gson:2.8.6'
- Square's Retrofit, including its Gson-based converter code, for retrieving JSON data from Web services.

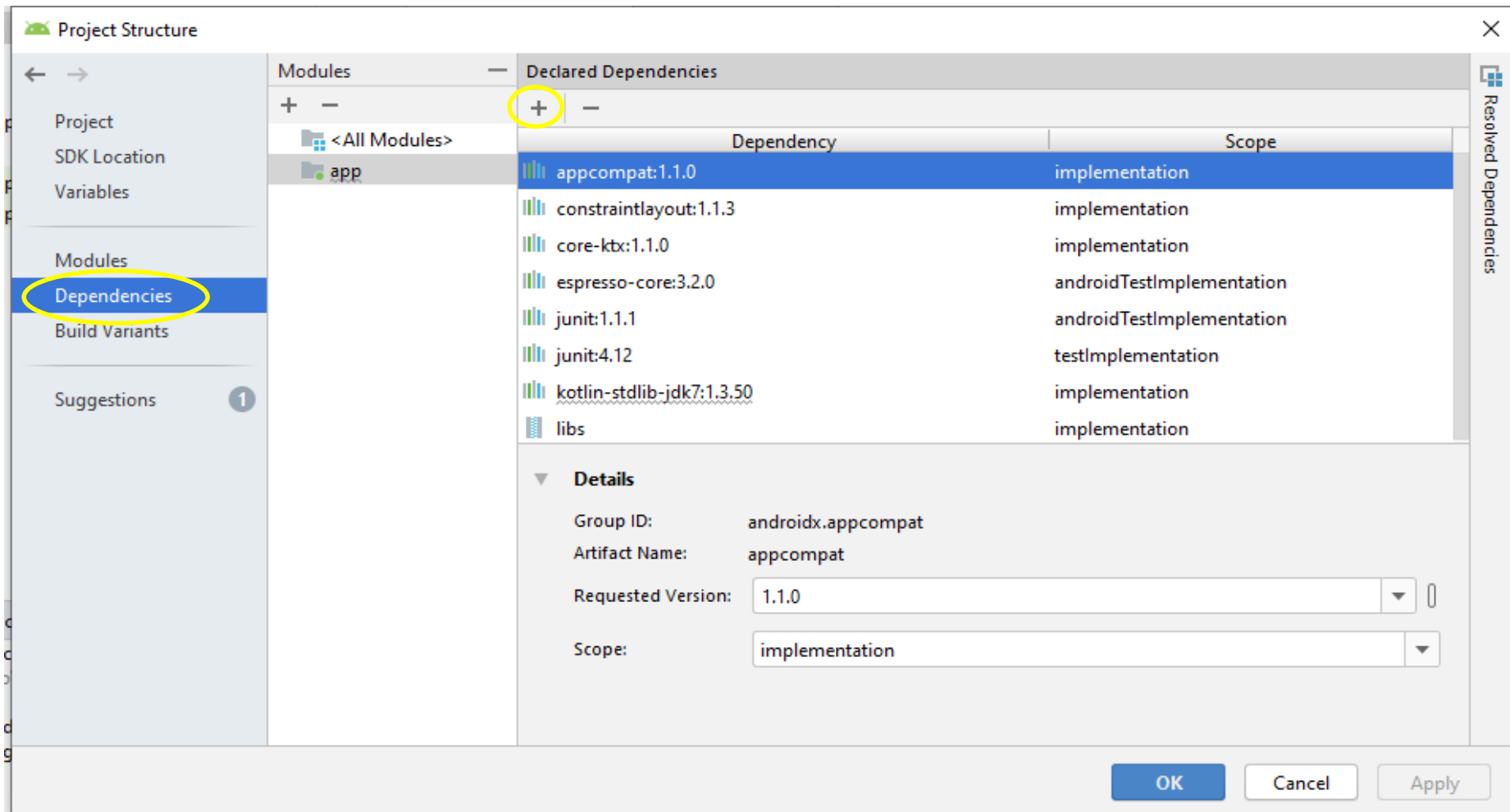
Add Gson dependencies

Step 1 : Right click app-> Open Module Settings



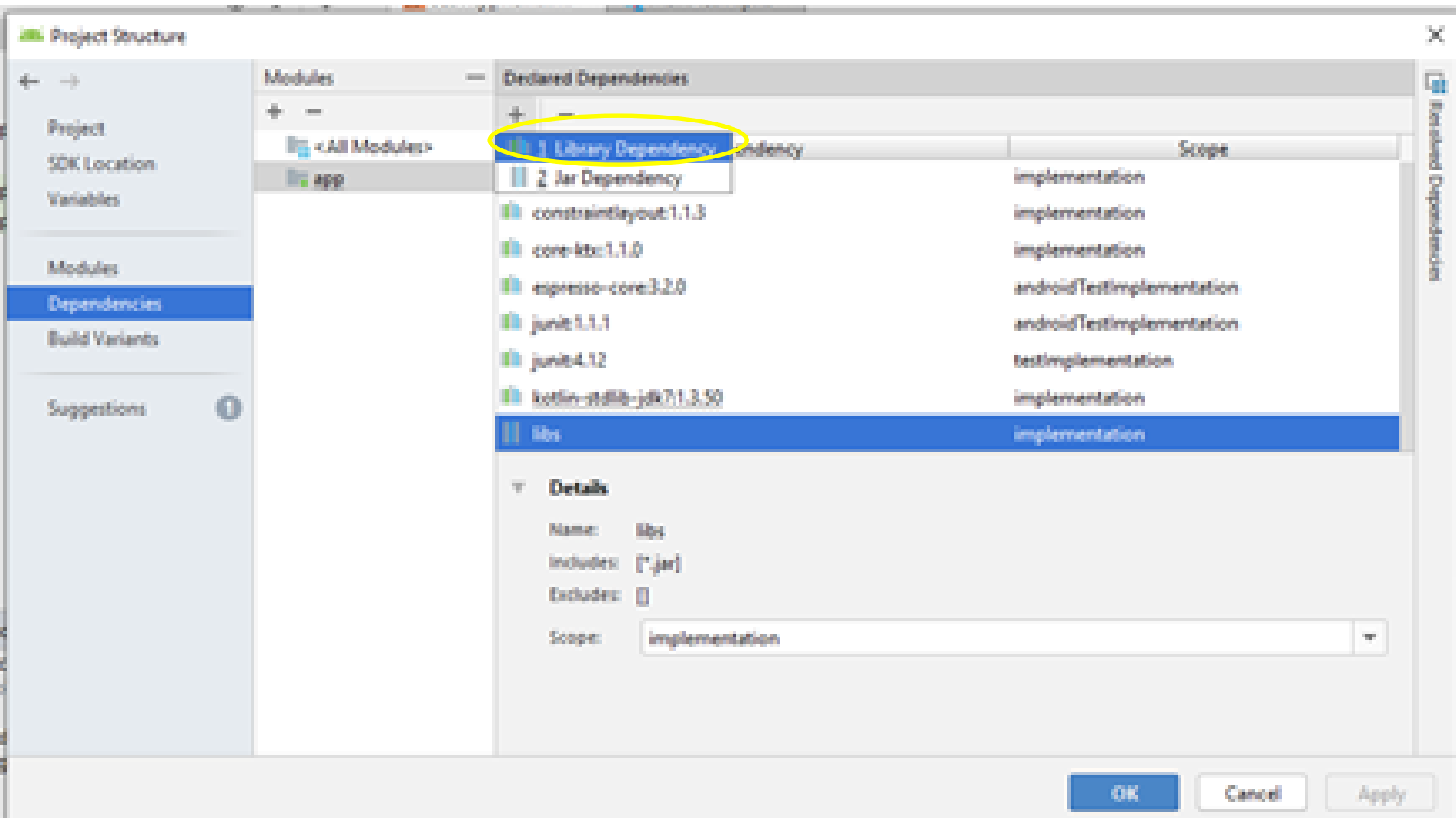
Add Gson dependencies

Step 2 : Choose the Dependencies Tab and click + icon to add dependency



Add Gson dependencies

Step 3 : Choose the Library Dependencies



Add Gson dependencies

Step 4 : Search gson dependency and select the highlighted dependency and click ok.

Add Library Dependency

Module "app"

Step 1.

Use the form below to find the library to add. This form uses the repositories specified in the project's build files (Google, JCenter, Android Repository, Google Repository)

gson

Search

Enter a search query or fully-qualified coordinates (e.g. guava or com.google.*guava or com.google.guava:guava:26.0)

| Group ID | Artifact Name | Repository | Versions |
|-------------------------------------|---------------|------------|----------|
| com.google.code.gson | gson | JCenter | 2.8.6 |
| app.raybritton.tokenstorage | gson | JCenter | 2.8.5 |
| at.stefanpeyer.challenge.serializer | gson | JCenter | 2.8.4 |
| com.github.link-kou | gson | JCenter | 2.8.3 |
| com.sdlite | gson | JCenter | 2.8.2 |
| com.solidfire.code.gson | gson | JCenter | 2.8.1 |
| nl.littlerobots.cupboard-tools | gson | JCenter | |

Library: com.google.code.gson:gson:2.8.6

Step 2.

Assign a scope to the new dependency by selecting the configurations below.

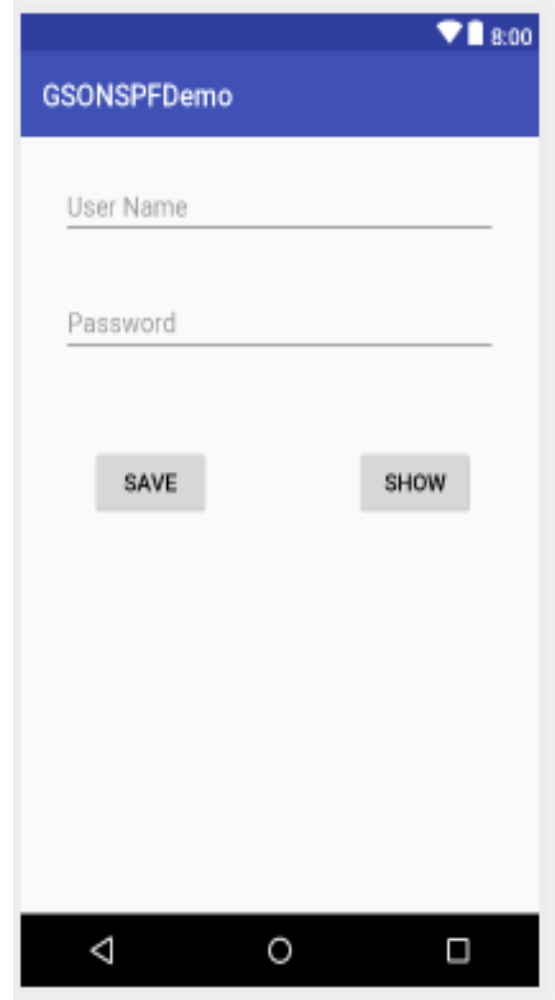
[Open Documentation](#)

implementation

OK Cancel

Hands on Example using Gson

- Here Username and Password stored as a User Object.
- Using Gson, object is stored and retrieved on Shared Preferences.
- Once the user select the SAVE button, data will be stored into Shared Preference.
- Once the user select the SHOW button, data will be retrieved from Shared Preference.
- Refer : Lesson9\GSONSPF



User.java

```
data class User(val uname:String, val  
password:String)
```

MainActivity.java

```
// This demonstrates to insert User object into Shared Preferences using GSON
fun save(view: View) {
    // Create Gson obeject to store into Shared Preferences
    val gson = Gson()
    val name = et1.getText().toString()
    val pass = et2.getText().toString()
    // Get the Input from the Edit text and make an User Object
    val ob = User(name, pass)
    if (ob != null){
        // Convert User object into String object using toJson() method
        val first = gson.toJson(ob)
        // Store the retrieved Sting into Shared Preferences
        val spf = getSharedPreferences("user", o) // o indicates MODE_PRIVATE
        val edit = spf.edit()
        edit.putString("data", first)
        edit.apply()
        Toast.makeText(getApplicationContext(), "Success", Toast.LENGTH_LONG).show()
    }
    else {
        Toast.makeText(getApplicationContext(), "Failure", Toast.LENGTH_LONG).show()
    }
    et1.setText("")
    et2.setText("")
}
```

```
fun show(view:View) {  
    // Create Gson object to retrieve data from Shared Preferences  
    val gson = Gson()  
    val spf = getSharedPreferences("user", 0) //0->MODE_PRIVATE  
    val res = spf.getString("data", "")  
    /* Convert the String object into User object using  
    fromJson() method, return a type of User*/  
    val opt = gson.fromJson(res, User::class.java)  
    if (opt != null){  
        et1.setText(opt.uname)  
        et2.setText(opt.password)  
    }  
    else{  
        Toast.makeText(getApplicationContext(), "Failure to  
retrieve", Toast.LENGTH_LONG).show()  
    }  
}
```

Hands on Example to store Multiple objects using Gson and store on SD card External Storage

- If the user click the INSERT LIST button, Entered employee data will be stored into `ArrayList<Employee>`.
- If the user click the WRITE GSON button, Employee List will be stored on File.
- If the user click the READ GSON button, employee data will be read from file and displayed one by one in Toast also displayed all data in the TextView.
- Ref : `GSONFileReadWrite`

The screenshot shows an Android application interface with four text input fields labeled "Enter ID", "Enter Name", "Enter Desig", and "Enter Dept". Below these fields are three buttons: "INSERT LIST", "WRITE GSON", and "READ GSON". A large, empty text view is positioned below the buttons, outlined with a blue border, intended for displaying the employee data.

Adding Dependency and Permission

- Add the below dependency in Gradle Scripts Module.app
implementation 'com.google.code.gson:gson:2.8.6'
- Add the below user permission in AndroidManifest.xml
**<uses-permission android:name=
"android.permission.WRITE_EXTERNAL_STORAGE">
</uses-permission>**

POJO/POKO Classes

- POJO/POKO, or *Plain Old Java/Kotlin Object*, is a normal Java /Kotlin class.
- To work with this Example, you must create two POKO classes.
 - 1.Employee(Structure of Individual Employee)
 2. Employees (Structure to store Multiple Employees)
- Data member of these classes annotated with *@SerializedName*
indicates this member should be serialized to JSON with the provided name value as its field name.

Employee class

```
import com.google.gson.annotations.SerializedName
// POJO, or Plain Old Java Object, is a normal Java object class
//(that is, not a JavaBean, EntityBean etc.)
// This class helps to give the structure of Individual Employee
class Employee {
    @SerializedName("id")
    var id:Int = 0
    @SerializedName("name")
    lateinit var name:String
    @SerializedName("desig")
    lateinit var desig:String
    @SerializedName("dept")
    lateinit var dept:String
    override fun toString():String {
        return ("Employee{" +
            "id=" + id +
            ", name= " + name +
            ", desig= " + desig +
            ", dept= " + dept +
            "} ")
    }
}
```

Employees class

```
import com.google.gson.annotations.SerializedName

class Employees {

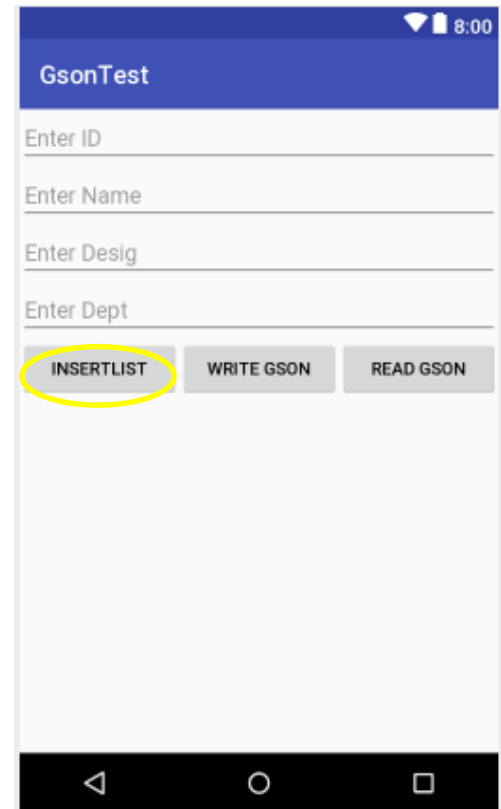
    @SerializedName("employees")

    lateinit var employees:ArrayList<Employee>

}
```

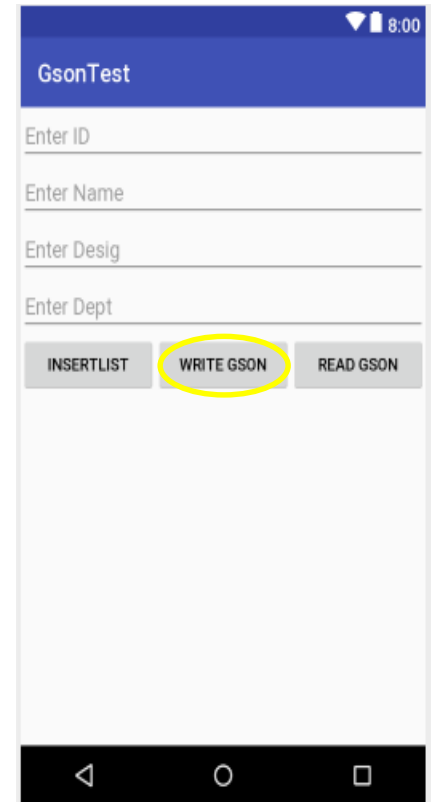
INSERT LIST BUTTON CLICK LOGIC using add function

```
fun add(v: View) { // Insert Employee into ArrayList
    val e = Employee()
    e.id = Integer.parseInt(et1.text.toString())
    e.name = et2.text.toString()
    e.desig = et3.text.toString()
    e.dept = et4.text.toString()
    // Add an Employee into the list
    list.add(e)
    Toast.makeText(applicationContext,
        "Added into List ${e.toString()}",
        Toast.LENGTH_LONG).show()
    et1.setText("")
    et2.setText("")
    et3.setText("")
    et4.setText("")
}
```



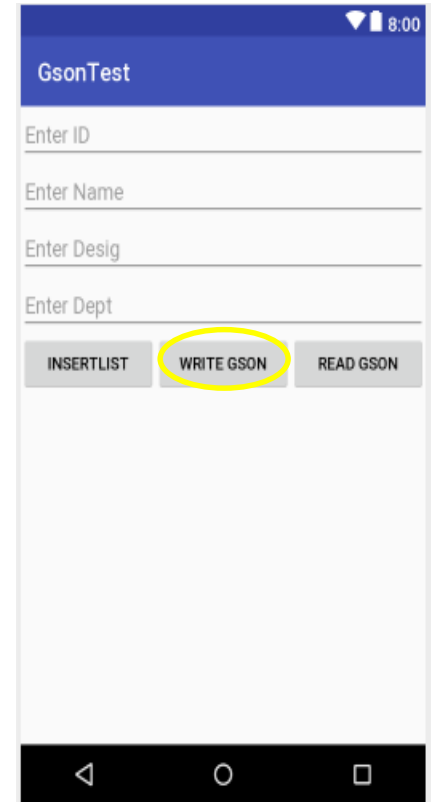
WRITE GSON BUTTON CLICK LOGIC

```
fun write(v:View) {  
    // Set the list of Employees  
    if (list.size > 0){  
        val emps = Employees()  
        emps.employees = list  
        // Conversion Employees list object to JSON using Gson  
        val gson = Gson()  
        val response = gson.toJson(emps)  
        // Writing the converted data into File using  
        // FileWriter in Your device external storage  
        val path = (Environment.getExternalStorageDirectory().absolutePath  
            + "/emps_gson.json")
```



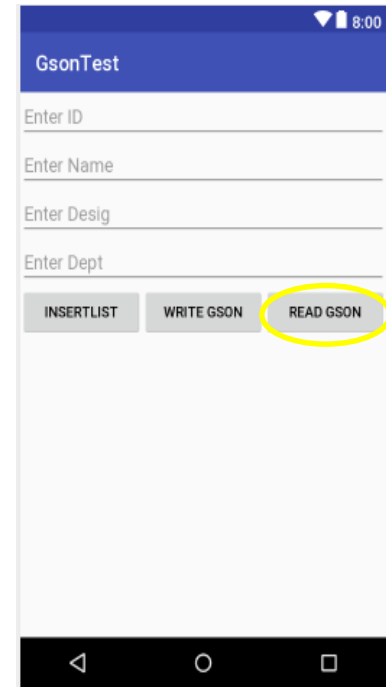
WRITE GSON BUTTON CLICK LOGIC

```
try{  
    val writer = FileWriter(path)  
    writer.write(response)  
    writer.flush() // will flush all the buffers in a chain of Writers  
    writer.close()  
    Toast.makeText(applicationContext,  
        "Write successfully on file",  
        Toast.LENGTH_LONG).show()  
}  
catch (e1: IOException) {  
    e1.printStackTrace()  
}  
}  
else  
    Toast.makeText(applicationContext, "Can't Write Empty List",  
        Toast.LENGTH_LONG).show()  
}
```



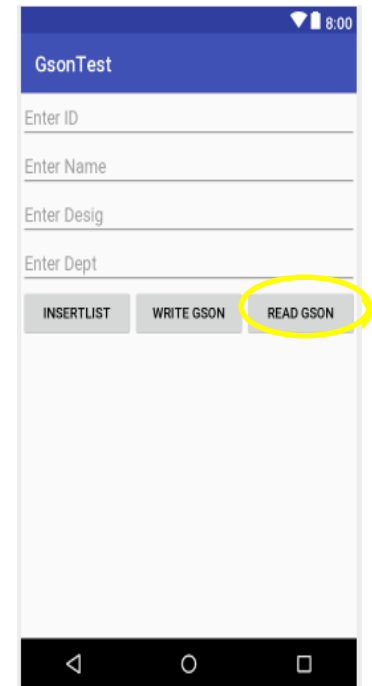
READ GSON BUTTON CLICK LOGIC

```
fun read(v:View) {  
    // Read the File using FileReader  
    val path = (Environment.getExternalStorageDirectory()  
        .absolutePath + "/emps_gson.json")  
    try {  
        val reader = FileReader(path)  
        val gson = Gson()  
        val emps = gson.fromJson(reader, Employees::class.java)  
        Toast.makeText(applicationContext, emps.toString(),  
            Toast.LENGTH_LONG).show()  
        val list = emps.employees  
        val builder = StringBuilder()
```



READ GSON BUTTON CLICK LOGIC

```
if (list.size > 0) {  
    for (e in list){  
        // Add the list into buffer and the result is added in the Textview  
        builder.append(e.toString())  
        builder.append("\n")  
        Toast.makeText(applicationContext, e.toString(),  
Toast.LENGTH_LONG).show()  
    }  
    result.text = builder.toString()  
}  
else  
    Toast.makeText(applicationContext, "No data",  
Toast.LENGTH_LONG).show()  
}  
catch (e: FileNotFoundException) {  
    e.printStackTrace()  
}  
}
```



Result Screen after clicking READ GSON

GSONFileReadWrite

Enter ID

Enter Name

Enter Desig

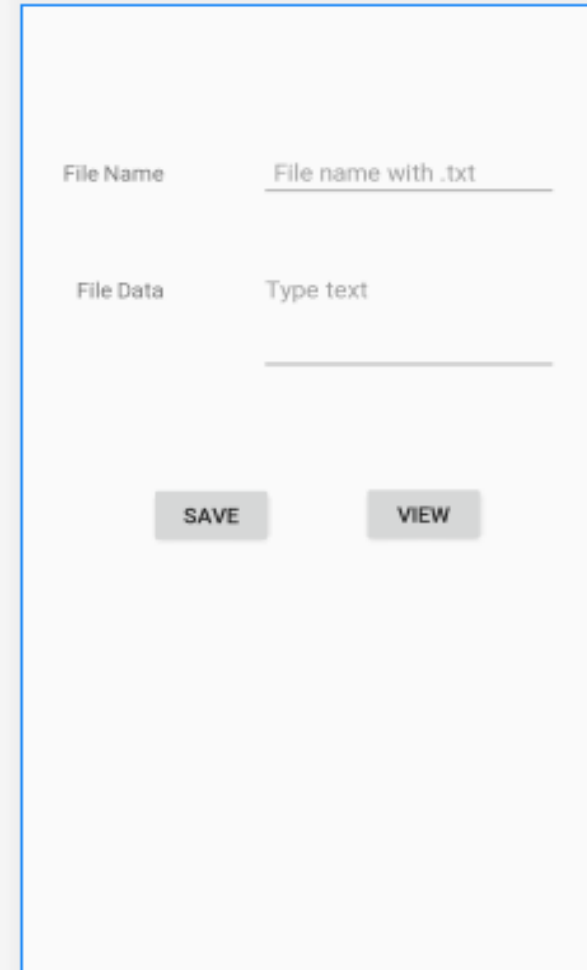
Enter Dept

INSERTLIST **WRITE GSON** **READ GSON**

```
Employee{id=111, name= Renuka, desig= Assoc. Prof, dept= Computer Science}  
Employee{id=222, name= Paul, desig= Professor, dept= Computer Science}  
Employee{id=3333, name= Joe, desig= AP, dept= MBA}
```

Hands on Example - File Read and Write Internal Storage

- Refer InternalStorageDemo
- Give a Filename and type the multiple lines of text data.
- Click the SAVE button to store on the device Internal storage.
- Give the filename to view and Click VIEW to retrieve the data from the file which displayed on the File data UI.
- To see the files from Internal storage,
- Click View→Tools Window→Device File Explorer
- Select the Device you installed
- Click data→data→goto the right package → files



The screenshot shows a mobile application interface with a light gray background. At the top, there is a label "File Name" followed by a text input field containing the text "File name with .txt". Below this, there is a label "File Data" followed by a text input field containing the text "Type text". At the bottom of the form, there are two buttons: "SAVE" on the left and "VIEW" on the right. Both buttons are gray with white text.

Retrofit

- **Two Web Services**
 - **SOAP** stands for Simple Object Access Protocol.
 - **REST** stands for REpresentational State Transfer
- Most popular Webservice is Retrofit. It's a REST Client for Java and Android. It makes it relatively easy to retrieve and upload JSON (or other structured data) via a REST based webservice. Developed by Square.
- In **Retrofit** you configure which converter is used for the data serialization.(like JSON, Gson)
- **Retrofit** automatically serializes and desterializes the JSON data as a type of POJO(Plain Old Java Object)
- Best Example for WebService : Finding places using Google Maps
- Read more about from : <https://square.github.io/retrofit/>

Hands on Example

- **Problem Requirement** : Read the JSON Data from this url: <https://api.myjson.com/bins/jmdmr> using Retrofit Library and show the data in the GridView
- Need to add the following dependencies in build.gradle (Module:app)
// Retrofit and the Gson converter dependency
implementation 'com.squareup.retrofit2:retrofit:2.5.0'
implementation 'com.squareup.retrofit2:converter-gson:2.5.0'

// Glide support for Image downloading from the third party
implementation 'com.github.bumptech.glide:glide:4.8.0'
annotationProcessor 'com.github.bumptech.glide:compiler:4.8.0'
// Add permission in your Manifest
<uses-permission
android:name="android.permission.INTERNET"></uses-permission>

Steps to follow

1. Add Dependencies to the app-module build.gradle file:
2. Design an UI to retrieve network data probably ListView, GridView and RecyclerView
3. Create Data Class
4. Create the API interface which we will use to make requests and get responses via retrofit.
5. Create a Client Class to get the response from the url using Retrofit object
6. Make a network call in your activity through `apiclient.apiinterface.call` methods
`var call = APIClient.ApiInterface().method()`

Steps to follow

7. Then call `enqueue()` and override the following two methods to retrieve the data into your view object.

```
override fun onResponse(call: Call<List<POJO Type>?>?, response:
                                                                    Response<List<POJO
Type>?>?) {
    if( response!!.isSuccessful){ // Check using non null !! operator
        /*The deserialized response body of a successful response and set into your
        adapter */
        }
    }
call.enqueue(object : Callback<List<Animals>> {

    override fun onFailure(call: Call<List<Animals>?>?, t: Throwable?) {
        // The localized description of this throwable, like getErrorMessage from
        throwable
        Toast.makeText(applicationContext,"An Error Occured
        ${t?.localizedMessage}",Toast.LENGTH_LONG).show()
        }
    })
```

Expected Outcome

- Read animals data from the given URL JSON data and displayed in the GridView.
- Refer Demo: RetrofitJSONURL

Workflow

- Client Request → send through Retrofit → Network Server
- Network Server → Send responses in JSON Format → Response converted using GSON into object format → retrieve as POJO/POKO Type

