Android Development with Kotlin

JSON and AlertDialog in Android

JSON

JSON stands for JavaScript Object Notation. JSON is an open standard file format, and data interchange format, that uses human-readable text to store and transmit data objects consisting of attribute–value pairs and array data types (or any other serializable value).

A JSON object with two properties, city and country is defined as follows:

```
{
    "city" : "New Delhi",
    "country" : "India"
}
```

The city and country are keys and the data i.e. New Delhi and India are the String values for these keys. A JSON object can also store an array of JSON objects, this can be achieved as follows:

```
{
     "cities": [
          {
               "city" : "New Delhi",
               "country" : "India"
          },
          {
               "city" : "New Delhi",
               "country" : "India"
          },
          {
               "city" : "New Delhi",
               "country" : "India"
          },
          {
               "city" : "New Delhi",
               "country" : "India"
          }
     ]
}
```

Here the “cities” key stores an array of city objects each with the properties of “city” and “country”. JSON objects do not have any fixed syntax, therefore, if you add/leave a property in any one of the objects in an object array, it will not prompt a syntax error but, it cannot be parsed either and will lead to an exception when doing so.

To parse a JSON object in Android, you need the Gson library by Google and a class that has the same structure as the JSON objects, in this case a data class with two properties, city and country. The class should be defined as follows:

```
data class City(val city: String, val country: String)
```

The data type and the property names should match the JSON object or else the JSON object won’t be parsed and will lead to run-time exceptions.

One of the main uses of JSON in Android can be to store data into SharedPreference, as SharedPreferences do not store collections of any other objects but strings. The Gson library can convert any object into a JSON formatted string and then, this string can be stored in the shared preference to allow data persistence within an application.

The following steps can be used to store data in a SharedPreference using the Gson library.

1. Add Gson library to the app: Module build.gradle file using the following statement

```
implementation 'com.google.code.gson:gson:2.8.2'
```

2. Create a Gson object

```
val gson = Gson()
```

3. Use the Gson object to parse the mutable list of City objects into a list of JSON objects (stored as Strings)

```
val cities = cityData.map { gson.toJson(it) }
```

4. Store the created list as a StringSet in the desired SharedPreference

```
val sharedPref = getSharedPreferences(SHARED_PREFERENCE_NAME,
                        Context.MODE_PRIVATE)
with(sharedPref.edit()) {
   putStringSet(DATA_KEY, cities.toSet())
   commit()
}
```

The data can also be retrieved from the SharedPreference using an object of the Gson class. This can be achieved using the following code segment:

```
val sharedPref = getSharedPreferences(SHARED_PREFERENCE_NAME,
                    Context.MODE_PRIVATE)
val cities = sharedPref.getStringSet(DATA_KEY, null)
val gson = Gson()
cities?.forEach{
    cityData.add(gson.fromJson(it, City::class.java))
}
```

Alert Dialogs

AlertDialog is a class is Android that can be used to display dialogs with an alert message and positive, negative and neutral buttons to register user input in response to the dialog. Dialogs can be useful in multiple ways:

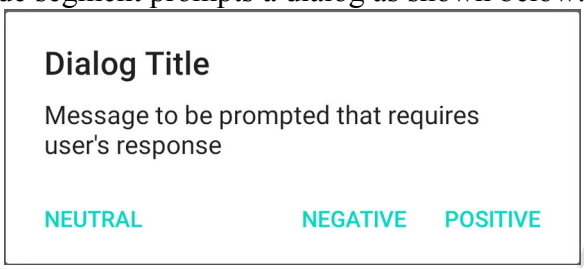
1. To record user's response to an encountered event.
2. To receive user's input to complete a task.
3. To confirm user's intentions/actions before proceeding further, etc.

The following code segment can be used to build an AlertDialog with only an alert message:

```
val builder = AlertDialog.Builder(this)
with(builder) {
    setTitle("Dialog Title")
    setMessage("Message to be prompted that requires user's response")
    //sets positive button
    setPositiveButton("Positive") { _, _ ->
          Toast.makeText(this@MainActivity, "Positive Action",
               Toast.LENGTH_SHORT).show()
    }
    //sets negative button
    setNegativeButton("Negative") { _, _ ->
          Toast.makeText(this@MainActivity, "Negative Action",
               Toast.LENGTH_SHORT).show()
    }
    //sets neutral action
```

```
        setNeutralButton("Neutral") { _, _ ->
              Toast.makeText(this@MainActivity, "Neutral Action",
                    Toast.LENGTH_SHORT).show()
        }
        //creates the dialog
        show()
    }
```

The above mentioned code segment prompts a dialog as shown below:



The dialog can also have a custom view attached to it, this can be achieved by creating an xml layout for the dialog and inflating it the layout dynamically. The inflated layout can then be set to the dialog using the following code segment.

```
val inflater = this.layoutInflater
val view: View = inflater.inflate(R.layout.dialog_add,null)
val builder = AlertDialog.Builder(this)
with(builder) {
    ...
    setView(view)
    ...
}
```

The remaining code stays the same for the Alert Dialog. The above mentioned changes will result in a dialog as shown below: