UNIT V: Localization, Menus and shared preference

5.1 Localization

An android application can run on many devices in many different regions. In order to make your application more interactive, your application should handle text, numbers, files etc. in ways appropriate to the locales where your application will be used.

The way of changing string into different languages is called as localization

Localizing Strings

In order to localize the strings used in your application, make a new folder under **res** with name of **values-local** where local would be the replaced with the region.

For example, in the case of Marathi, the **values-mr** folder would be made under res

Once that folder is made, copy the **strings.xml** from default folder to the folder you have created. And change its contents as in Marathi.

5.2 Options menu

Options Menu is a primary collection of menu items for an activity and it is useful to implement actions that have a global impact on the app, such as Settings, Search, etc.

By using Options Menu, we can combine multiple actions and other options that are relevant to our current activity. We can define items for the options menu from either our Activity or Fragment class.

Create Android Options Menu in XML File

To define **options menu**, we need to create a new folder **menu** inside of our project resource directory (**res/menu/**) and add a new XML (**menu_example**) file to build the menu.

```
<? xml version="1.0" encoding="utf-8"?>
<menu xmlns:android="http://schemas.android.com/apk/res/android">

 <item android:id="@+id/mail"
      android:icon="@drawable/ic_mail"
      android:title="@string/mail" />

</menu>
```

Load Android Options Menu from an Activity

To specify the options menu for an activity, we need to override **onCreateOptionsMenu()** method and load the defined menu resource using **MenuInflater.inflate()**

```
@Override
public void onCreateOptionsMenu(ContextMenu menu, View v,
ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   MenuInflater inflater = getMenuInflater();
   inflater.inflate(R.menu.menu_example, menu);
}
```

Handle Android Options Menu Click Events

We can handle options menu item click events using the **onOptionsItemSelected()** event method.

```
@Override
public boolean onOptionsItemSelected(MenuItem item) {
   switch (item.getItemId()) {
      case R.id.mail:
         // do something
         return true;

      default:
         return super.onContextItemSelected(item);
   }
}
```

5.3 Context menu

Context Menu is like a floating menu and that appears when the user performs a long press or click on an element and it is useful to implement actions that affect the selected content or context frame.

The android Context Menu is more like the menu which displayed on right-click in Windows or Linux.

the Context Menu offers actions that affect a specific item or context frame in the UI and we can provide a context menu for any view. The context menu won't support any item shortcuts and item icons.

Create Android Context Menu in Activity

The views which we used to show the context menu on long-press, we need to register that views using **registerForContextMenu(View)** in our activity and we need to override **onCreateContextMenu()** in our activity or fragment.

When the registered view receives a long-click event, the system calls our **onCreateContextMenu()** method. By using the **onCreateContextMenu()** method

```
@Override
protected void onCreate(Bundle savedInstanceState) {
   super.onCreate(savedInstanceState);
   setContentView(R.layout.activity_main);
   Button btn = (Button) findViewById(R.id.btnShow);
   registerForContextMenu(btn);
}

@Override
public void onCreateContextMenu(ContextMenu menu, View v,
ContextMenu.ContextMenuInfo menuInfo) {
   super.onCreateContextMenu(menu, v, menuInfo);
   menu.setHeaderTitle("Context Menu");
   menu.add(0, v.getId(), 0, "Upload");
   menu.add(0, v.getId(), 0, "Search");
}
```

Handle Android Context Menu Click Events

we can handle a context menu item click events using the **onContextItemSelected()** method.

```
@Override
public boolean onContextItemSelected(MenuItem item) {
   if (item.getTitle() == "Save") {
      // do your coding
   }
   else {
      return  false;
   }
   return true;
}
```

5.4 Shared preferences

Shared Preferences are used to save and retrieve the primitive data types (integer, float, Boolean, string, long) data in the form of key-value pairs from a file within an apps file structure.

The **Shared Preferences** object will point to a file that contains key-value pairs and provides a simple read and write methods to save and retrieve the key-value pairs from a file.

The Shared Preferences file is managed by an android framework and it can be accessed anywhere within the app to read or write data into the file, but it's not possible to access the file from any other app so it's secured.

The Shared Preferences are useful to store the small collection of key-values such as user's login information, app preferences related to users, etc. to maintain the state of the app, next time when they login again to the app.

Handle Shared Preferences

We can save the preferences data either in single or multiple files based on our requirements.

In case if we use a single file to save the preferences, then we need to use **getPreferences()** method to get the values from Shared Preferences file

for multiple files, we need to call a **getSharedPreferences()** method and pass a file name as a parameter.

getPreferences()

This method is for activity level preferences and each activity will have its own preference file and by default

getSharedPreferences()

This method is useful to get the values from multiple shared preference files by passing the name as a parameter to identify the file. We can call this from any Context in our app.

```
SharedPreferences sharedPref = getPreferences(Context.MODE_PRIVATE);
SharedPreferences.Editor editor = sharedPref.edit();
editor.putString("keyname","string value");
editor.putInt("keyname","int value");
editor.commit();
```

5.5 Files access

Google Play restricts the use of high-risk or sensitive permissions, including special app access called files access.

This is only applicable to apps that target Android 11 (API level 30) and declare the MANAGE_EXTERNAL_STORAGE permission, which is added in Android 11.

If your app does not require access to the MANAGE_EXTERNAL_STORAGE permission, you must remove it from your app's manifest in order to successfully meet the policy review requirements.

If your app meets the policy requirements for acceptable use or is eligible for an exception, you will be required to declare this and any other high-risk permissions using the Permissions

Use of File Access

File management

App’s core purpose involves the access, editing, and management (including maintenance) of files and folders outside of its app-specific storage space.

Backup and restore apps

App must have a need to automatically access multiple directories outside of its app-specific storage space for the purpose of backup and restore

Anti-virus apps

App’s core purpose is to scan the device and provide anti-virus security features to the device user

Document management apps

Apps that must locate, access, and edit compatible file types outside of its app-specific or shared storage

Search

App’s core purpose is to search through files and folders across the device’s external storage

Disk/Folder Encryption and Locking

App’s core purpose is to encrypt files and folders.

5.6 Sending Email

Email is messages distributed by electronic means from one system user to one or more recipients via a network.

You must know Email functionality with intent, Intent is carrying data from one component to another component with-in the application or outside the application.

To send an email from your application, you don't have to implement an email client from the beginning, but you can use an existing one like the default Email app provided from Android, Gmail, Outlook, K-9 Mail etc.

we need to write an Activity that launches an email client, using an implicit Intent with the right action and data.

Intent Object - Action to send Email
You will use ACTION_SEND action to launch an email client installed on your Android device.

```
Intent emailIntent = new Intent(Intent.ACTION_SEND);
```

Intent Object - Data/Type to send Email
To send an email you need to specify mailto: as URI using setData() method and data type will be to text/plain using setType() method

```
emailIntent.setData(Uri.parse("mailto:"));
emailIntent.setType("text/plain");
```

Intent Object - Extra to send Email

Android has built-in support to add TO, SUBJECT, CC, TEXT etc.

Ex.

```
String[] TO = {""};
String[] CC = {""};

    Intent emailIntent = new Intent(Intent.ACTION_SEND);
    emailIntent.setData(Uri.parse("mailto:"));
    emailIntent.setType("text/plain");
    emailIntent.putExtra(Intent.EXTRA_EMAIL, TO);
    emailIntent.putExtra(Intent.EXTRA_CC, CC);
    emailIntent.putExtra(Intent.EXTRA_SUBJECT, "Your subject");
    emailIntent.putExtra(Intent.EXTRA_TEXT, "Email message goes here");
```

5.7 Sending SMS

SmsManager API

```
SmsManager smsManager = SmsManager.getDefault();
smsManager.sendTextMessage("phoneNo", null, "sms message", null, null);
```

Built-in SMS application

```
Intent sendIntent = new Intent(Intent.ACTION_VIEW);
sendIntent.putExtra("sms_body", "default content");
sendIntent.setType("vnd.android-dir/mms-sms");
startActivity(sendIntent);
```

both need **SEND_SMS permission**.

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

Apart from the above method, there are few other important functions available in SmsManager class.

ArrayList<String> divideMessage(String text)

This method divides a message text into several fragments, none bigger than the maximum SMS message size.

static SmsManager getDefault()

This method is used to get the default instance of the SmsManager

void sendDataMessage()

This method is used to send a data based SMS to a specific application port.

void sendMultipartTextMessage()

Send a multi-part text based SMS.

void sendTextMessage()

Send a text based SMS.

Thank You