Programming in Android

Session: 3

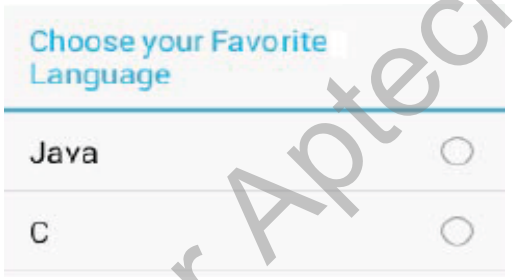
Android System Overview

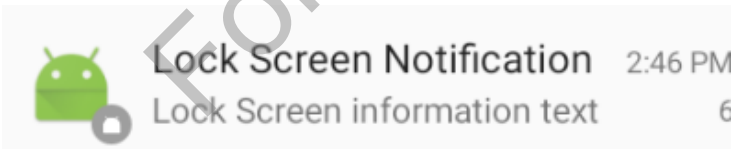
Objectives

- Explain Preferences
- Explain Shared Preferences Storage Structure
- Explain the Android File System
- Explain Notifications

Introduction

- To create a user friendly app, the developer needs to know –
  - Setting Preferences
  - Saving Files
  - Sending Notifications
  - Android Security Model
- Customizability and saving settings can be implemented by using Preferences
- Notifications are of several types such as simple, progress bar, lock screen, and so on





Shared Preferences 1-2

- Shared Preferences are used to store persistent data on the device
- The three commonly used preferences are -
  - CheckBoxPreference
  - ListPreference
  - EditTextPreference
- Preferences can be defined in XML
- Preference Manager class is used to save Preferences

Shared Preferences 2-2

- Following Code Snippet demonstrates an example for different types of preferences:

```
<?xml version="1.0"encoding="UTF-8"?>

<PreferenceScreen
xmlns:android="http://schemas.android.
com/apk/
res/android">

<PreferenceCategory
android:title="Checkbox Preference">

<CheckBoxPreference
..
/>
</PreferenceCategory>

<PreferenceCategory
android:title="EditTextPreference">

<EditTextPreference
..
.. "/>
</PreferenceCategory>

<PreferenceCategory
android:title="ListView Preference">

<ListPreference ..
..
../>
</PreferenceCategory>

</PreferenceScreen>
```

Preference Fragment

- Following Code Snippet demonstrates an example for creating a Preference Fragment:

```
package com.example.PreferenceFragment;
Import …;

public class PrefsFragment extends PreferenceFragment {
    @Override
    public void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);

addPreferencesFromResource(R.xml.preferences);
   }
}
```

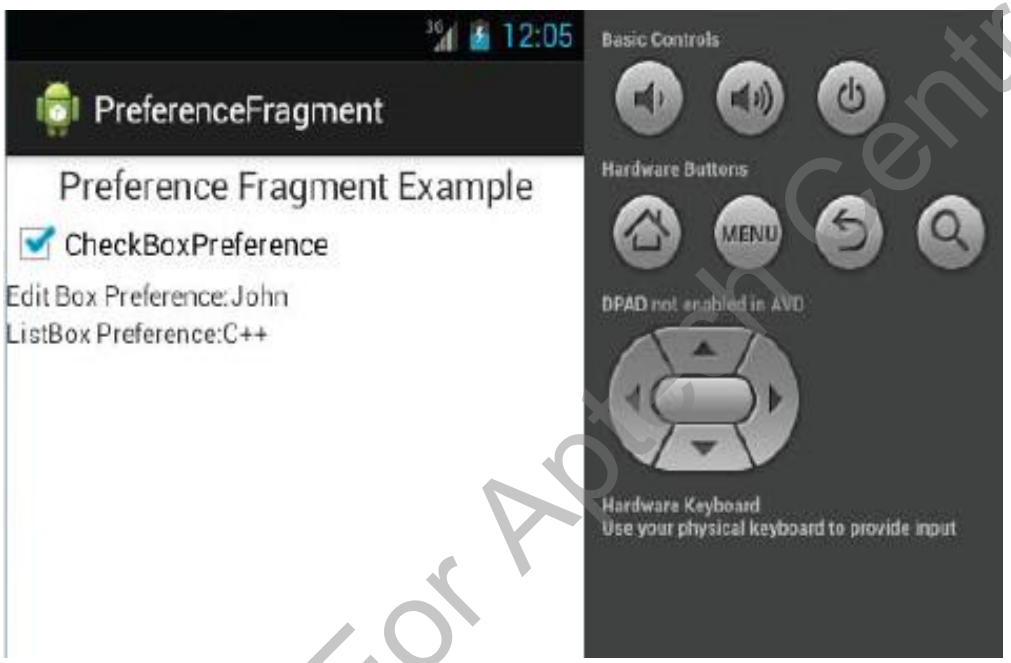
Preference Manager

- Following Code Snippet demonstrates using PreferenceManager class to get preference values and loading them into the activity:

```
…
SharedPreferences mySharedPreferences =
PreferenceManager.getDefaultSharedPreferences(this);

 boolean my_checkbox_preference = mySharedPreferences.
getBoolean("checkbox_preferencevalue", false);
prefCheckBoxvalue.setChecked(my_checkbox_preference);

String my_edittext_preference = mySharedPreferences.
getString("edittext_preferencevalue", "");
 prefEditTextvalue.setText("Edit Box Preference:"+my_
edittext_preference);

String my_edittext_preference1 = mySharedPreferences.
getString("lp_android_choice", "");
prefEditTextvalueone.setText("ListBox Preference:"+my_
edittext_preference1);
…
```

Preferences Example Application 1-3

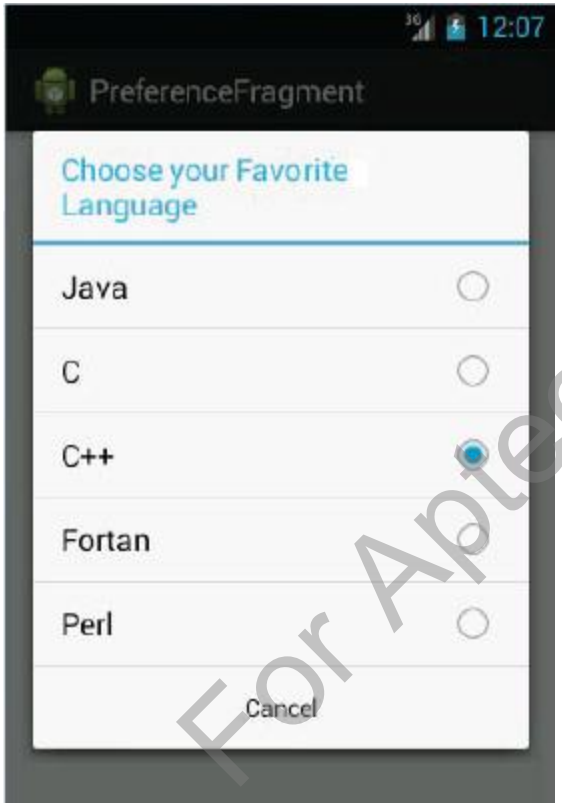
- Using the code, an application for demonstrating retrieving Shared Preferences is created as shown in the following figure:



- Click Menu and then Click Preference Example to display the output as shown in the following figure:

Preferences Example Application 2-3

- Click Choose your Favorite Language to display the output as shown in the following figure:



- Click title_edittext_preference, to display the output as shown in the following figure:

Preferences Example Application 3-3

- The application logic is as follows:

- Developer creates an XML file and specifies the preferences in the file
- The XML contains a list of preference objects and preference subclasses
- The preferences are loaded into the Activity Preference fragment
- The application allows the user to modify them

Saving Preferences

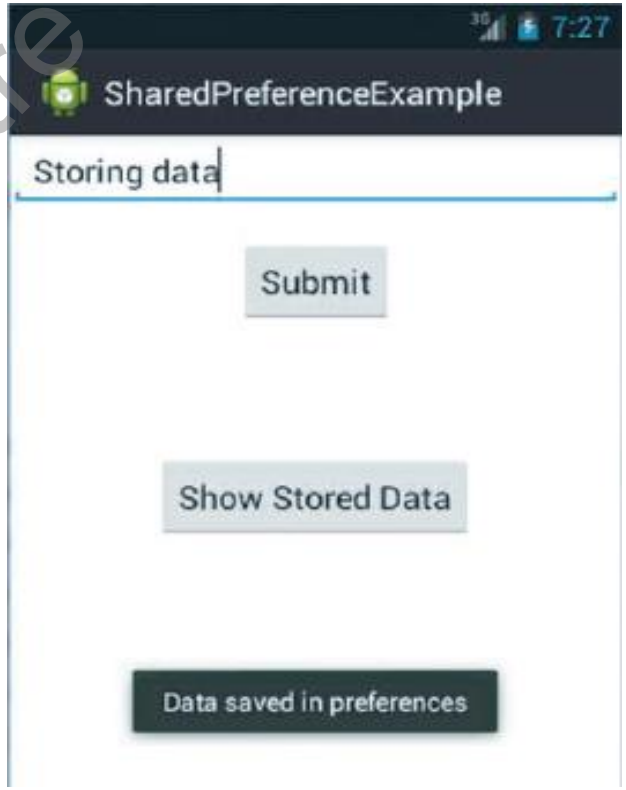
- Following Code Snippet demonstrates using PreferenceManager class to store custom Preference values:

```
SharedPreferences preferences =  PreferenceManager
.getDefaultSharedPreferences(MainActivity.this);

SharedPreferences.Editor editor =  = preferences.edit();


editor.putString("data", "value");

editor.commit();
```

Saving Preferences Example Application 1-2

- Using the explained code, an application for demonstrating saving Shared Preferences is created as shown in the following figure:

- On typing the content in the edit box and click Submit, the output will be as shown in the following figure:



Saving Preferences Example Application 2-2

- On clicking the Show Stored Data, the output will be as shown in the following figure:

- The application logic is as follows:

- Developer creates a Edit Text to accept text input from the user
- The user input is retrieved and stored using the Shared Preferences. Editor object
- When the user clicks the button, the value of the preference is retrieved
- The retrieved value is displayed to the user using a TextView

File System

- Two types of storage options supported -
  - Internal
  - External
- Several types of file systems are supported such as ext4, NTFS, and so on
- The developer can read and write files using the File APIs
- Each application has its own internal file system
- No permissions needed for internal file system
- Manifest file must explicitly specify permission to write to external storage

Internal File System 1-2

- No permissions needed
- Always available
- More secure
- Removed once, the application is deleted
- Accessed using java.io classes

Internal File System 2-2

- Following Code Snippet demonstrates an example for creating and deleting a directory on the internal file system:

```
File srDir=new File(getFilesDir(), "file_name");
srDir.mkdir();

file.delete()
```

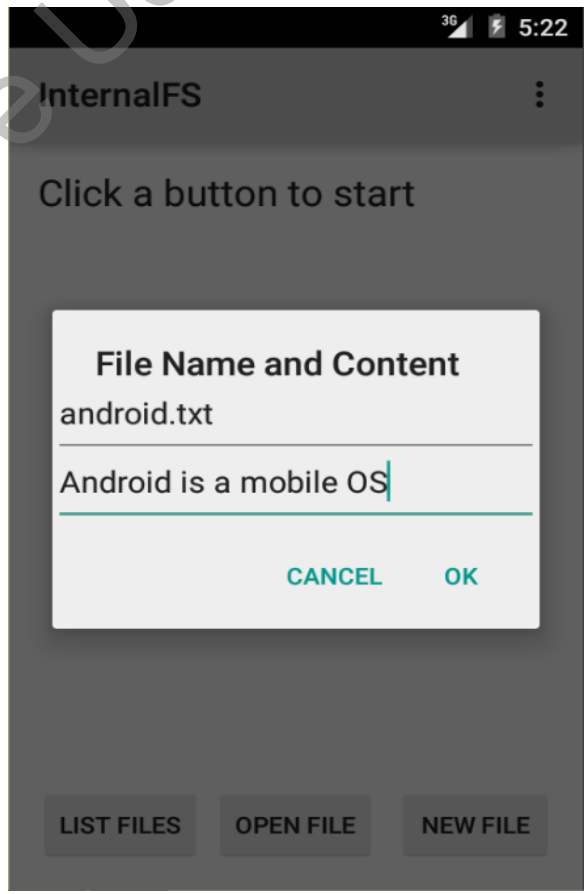
- Following Code Snippet demonstrates an example for creating, reading, writing, and deleting a file on the internal file system:

```
File file=new File(getAbsolutePath(), "android.txt");
file.createNewFile();

FileInputStream fis = openFileinput(file);
String text = fis.read();
fis.close();

FileOutputStream fos = openFileOutput(FILENAME, Context.MODE_ PRIVATE);
fos.write(string.getBytes());
fos.close();
file.close();
file.delete();
```

Internal File System Example Application 1-3

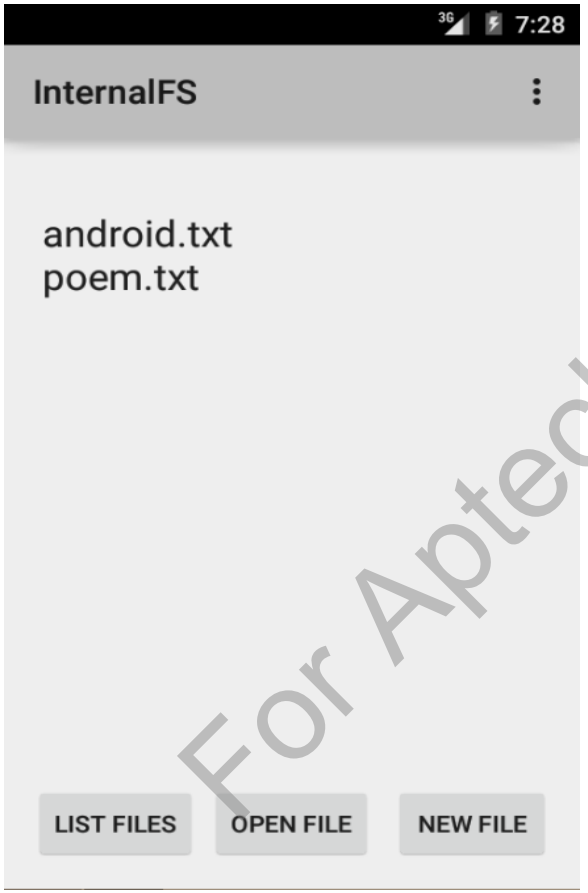
- Using the explained code, an application for accessing the Internal File System is created as shown in the following figure:

- Clicking the NEW FILE will create a popup for the file name and contents as shown in the following figure:

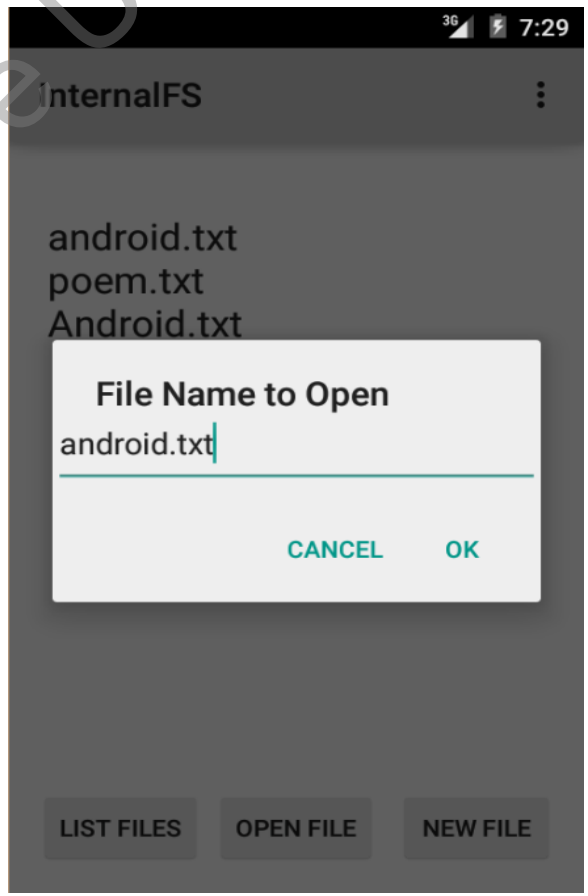


Internal File System Example Application 2-3

- After a file has been created, Clicking the LIST FILES will list the current files in the internal file system as shown in the following figure:

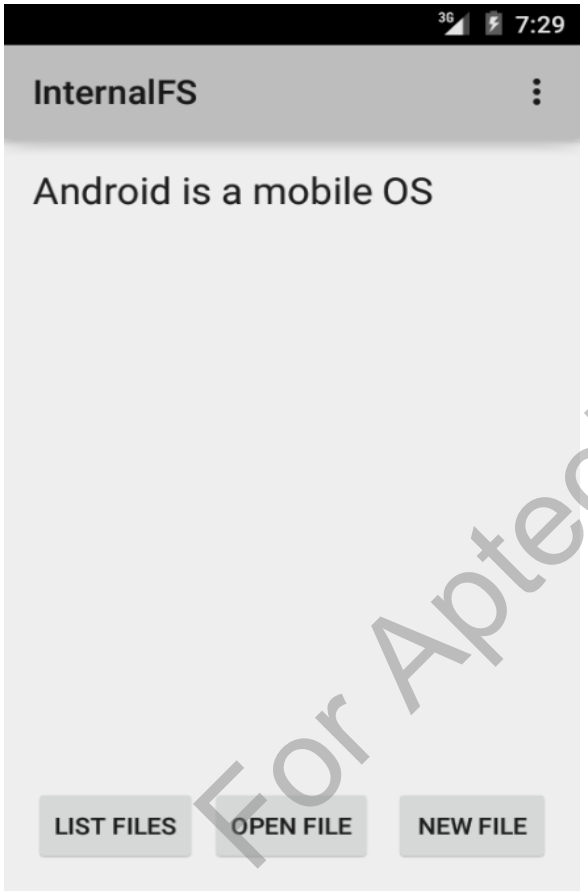


- Clicking the OPEN FILE button will open the dialog Box prompting for the file name to open as shown in the following figure:



Internal File System Example Application 3-3

- Clicking OK will display the file contents as shown in the following figure:



- The application logic is as follows:

- An activity with three button with obvious functionality are created
- Clicking LIST FILES will retrieve the files list and display it to the user using a TextView
- Clicking NEW FILE will create an Alert Box asking for the necessary inputs to create a file
- The File is then created using a File Object
- Clicking OPEN FILE will create an Alert Box asking for the File Name to open
- The file contents are read using a File Objects and displayed using a TextView

External File System 1-2

- Explicitly permissions needed
- Not reliable
- More volume of storage available
- Can persist even if the app is removed
- Data can be shared between applications

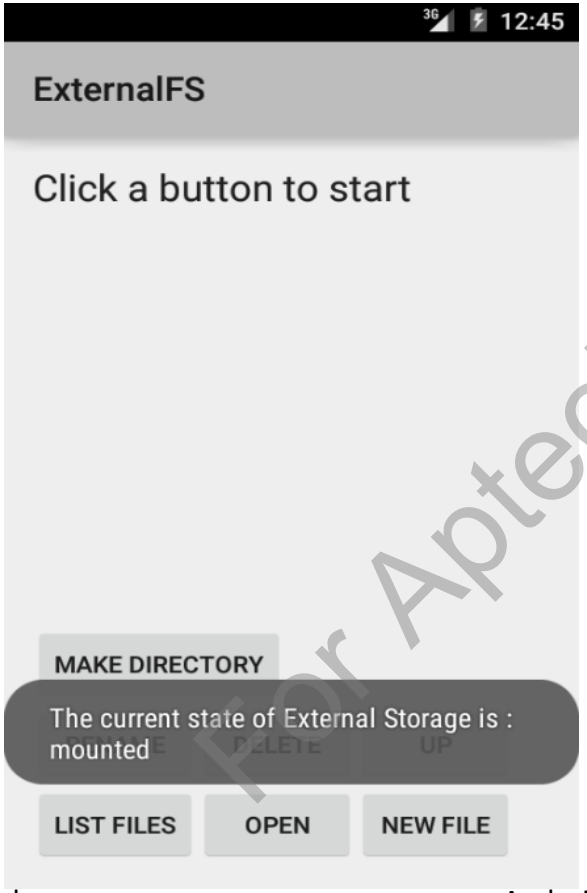
External File System 2-2

- The code to access files and directories on External File System is similar to that of the Internal File System
- Following Code Snippet demonstrates how to retrieve a File object or a path reference to the external storage:

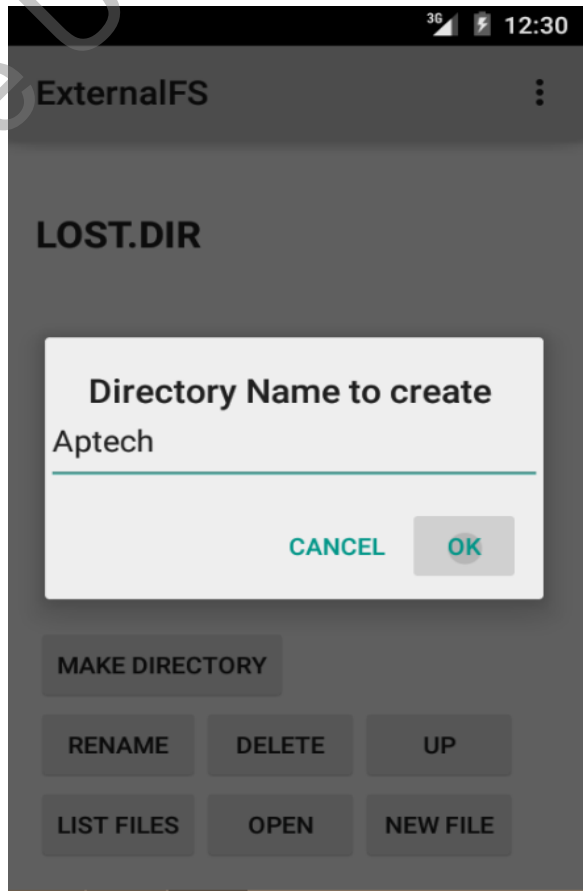
```
//Path
String path = Environment.getExternalStorageDirectory().getAbsolutePath();

//File Object
File file = Environment.getExternalStorageDirectory();
```

External File System Example Application 1-4

- Using the explained code, an application for demonstrating accessing the External File System is created as shown in the following figure:



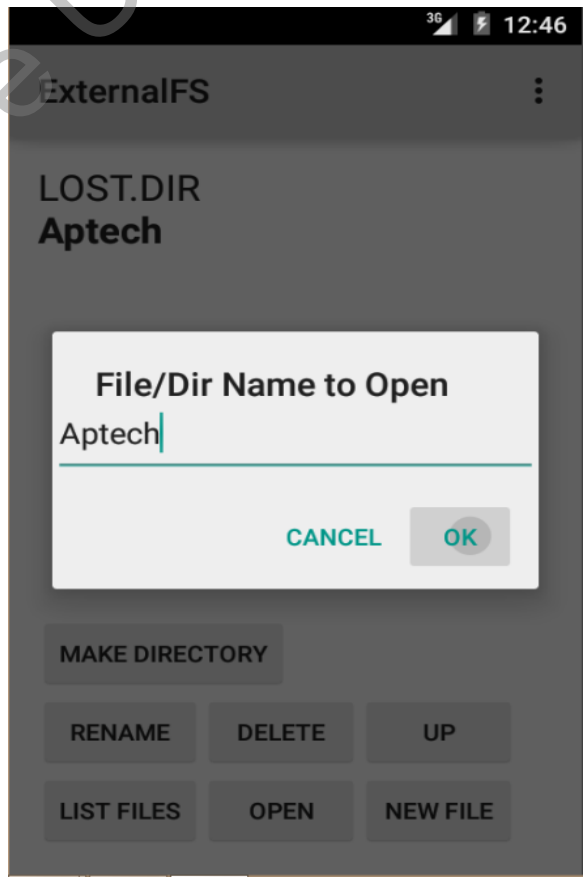
- Clicking MAKE DIRECTORY will create a prompt asking for directory name as shown in the following figure:



External File System Example Application 2-4

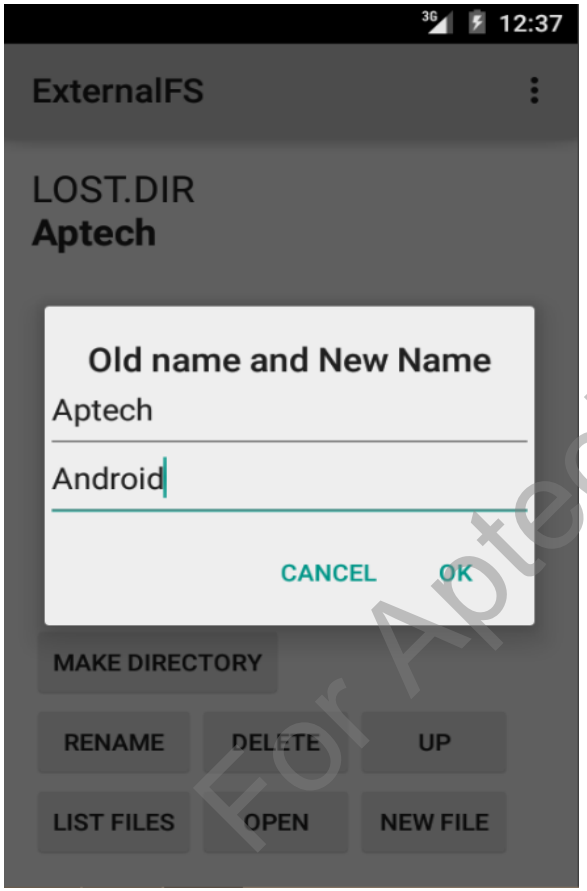
- The directory is created and the files and folders are listed
- The directories are shown in bold as shown in the following figure:

- Clicking OPEN will create prompt asking for the file/directory to open as shown in the following figure:

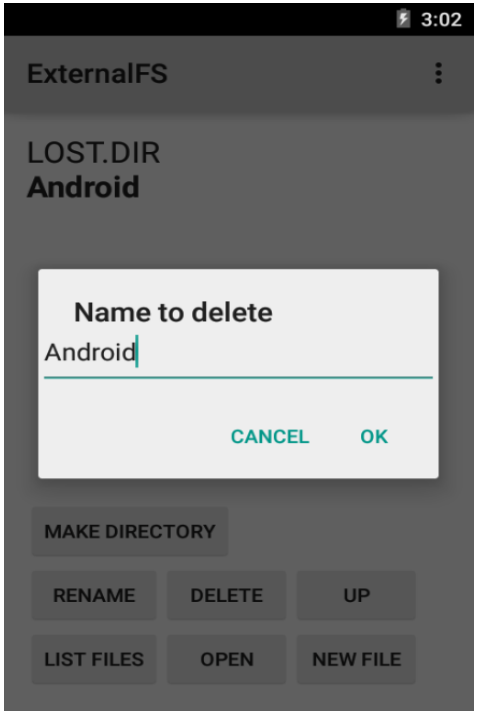


External File System Example Application 3-4

- Clicking RENAME will create prompt asking for necessary details as shown in the following figure:



- The LIST FILES and NEW FILE button function similar to the buttons of Internal File System Example
- Clicking DELETE will create prompt asking for directory to delete as shown in the following figure:



External File System Example Application 4-4

- The application logic is as follows:

- The functional code is similar to that of the Internal Storage Example
- The External Storage directory is retrieved using Environment.getExternalStorageDirectory() method

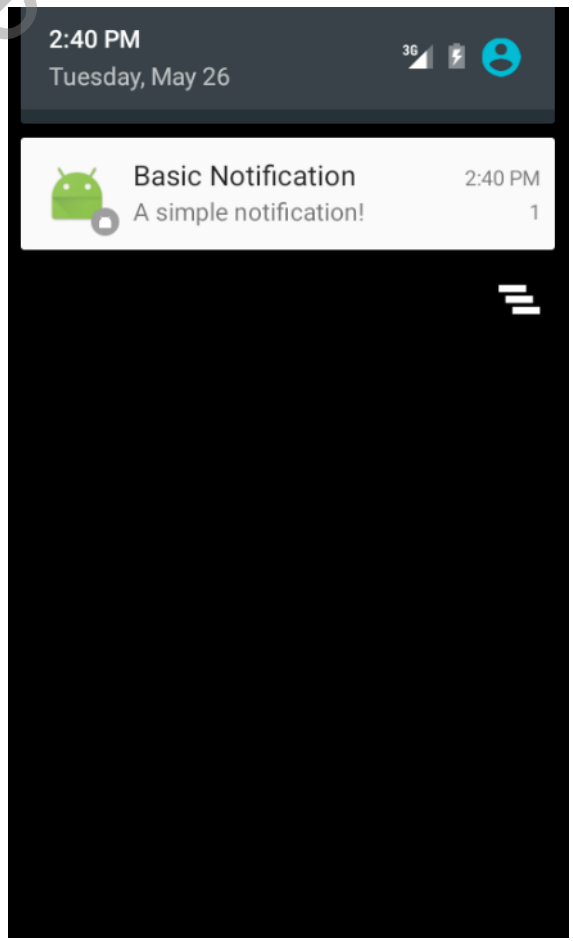
Notifications

- Notifications enable an app to inform the user about events
- Every notification needs to contain the following information:
  - Small Icon
  - Title
  - Detailed Text
- Several types of notifications:
  - Normal/Basic
  - Expanded
  - Progress
  - Heads Up
  - Lock Screen

Normal/Basic Notification

- Following Code Snippet demonstrates creating a Basic Notifications:

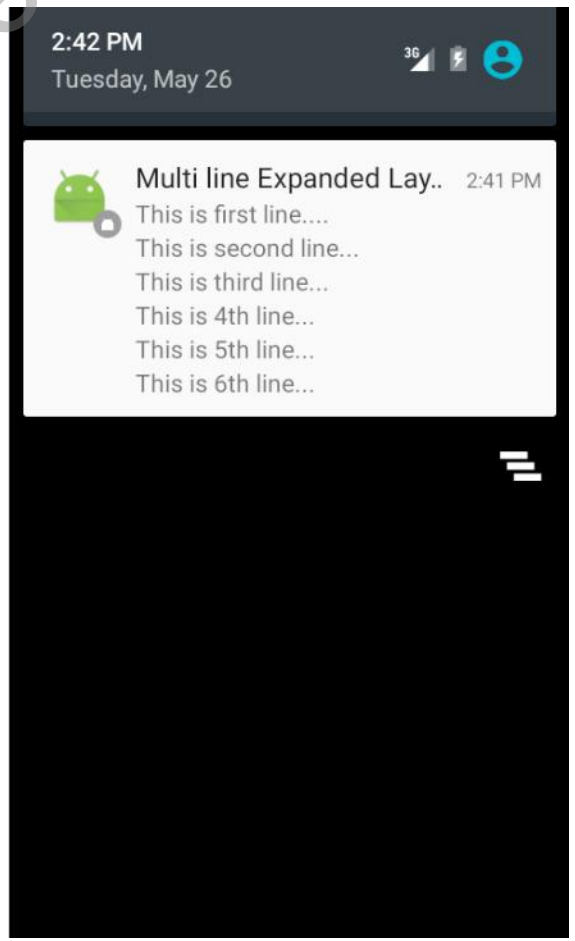
```
 Notification.Builder notification =new
Notification.Builder(this)

.setSmallIcon(R.drawable.ic_notification_small)
.setContentTitle("Basic Notification")
.setContentText("A simple notification!")
.setLargeIcon(BitmapFactory.decodeResource(getRes
ources(), R.mipmap.ic_notification_large));
…
NotificationManager mNotificationManager =
(NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(0,
notification.build());
```



Expanded Notification

- Following Code Snippet demonstrates creating an Expanded Notifications:

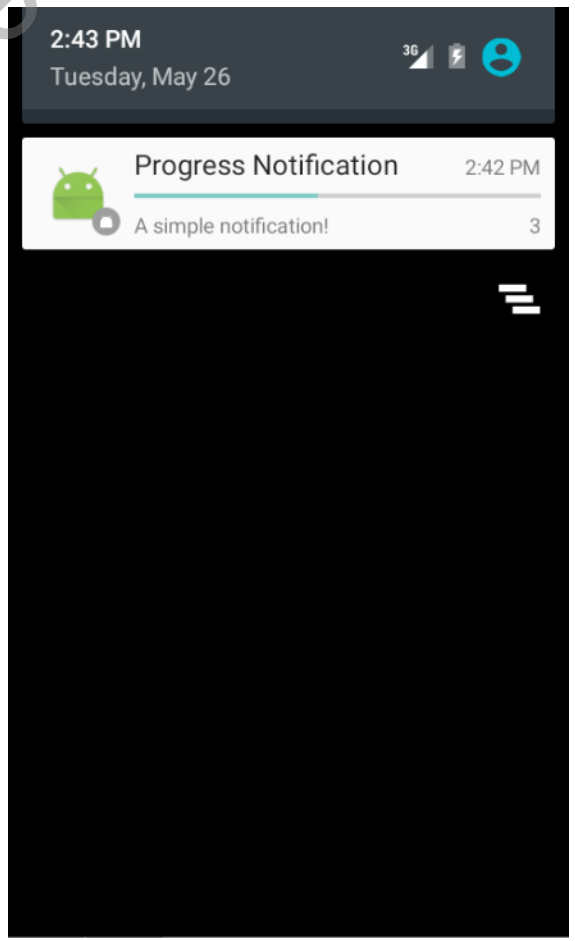
```
 Notification.Builder notification =new
Notification.Builder(this)
.setSmallIcon(R.drawable.ic_notification_small)
.setContentTitle("Expanded Layout Notification")
.setContentText("This notification can be
expanded")
.setLargeIcon(BitmapFactory.decodeResource(getRes
ources(), R.mipmap.ic_notification_large));

Notification.InboxStyle inboxStyle =new
Notification.InboxStyle();
inboxStyle.setBigContentTitle("Multi line
Expanded Layout ");

for( int i=0;i<5;i++)
{
        inboxStyle.addLine(" ");
}
notification.setStyle(inboxStyle);
mNotificationManager.notify(1,notification.build(
));
```



Progress Notification

- Following Code Snippet demonstrates creating a Progress Notifications:

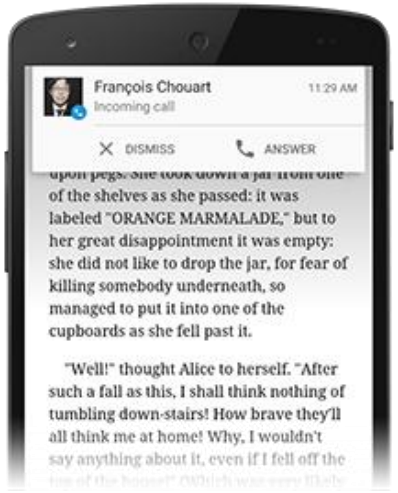
```
final Notification.Builder notification =new
Notification.Builder(this)
                  .set….
..
.. ;

 notification.setProgress(100, incr, false);

 mNotifyManager.notify(2, notification.build());
```



Heads Up Notification

- Heads Up Notifications are not guaranteed to be displayed Heads Up mode
- Several conditions need to be met
- Following Code Snippet demonstrates creating a Heads Up Notifications:

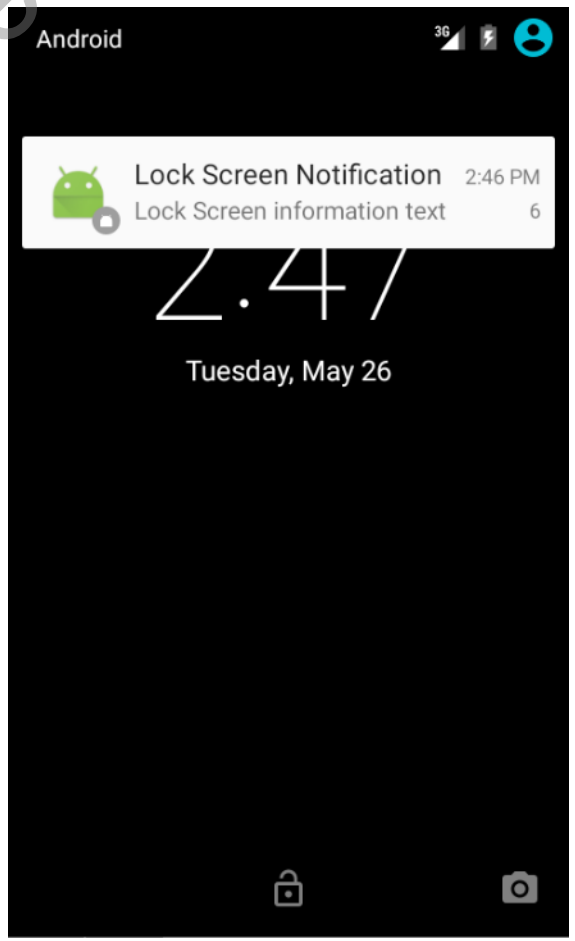
```
Notification.Builder notification =new
Notification.Builder(this)
        .set…

        .setPriority(Notification.PRIORITY_MAX);
        …;


NotificationManager mNotificationManager
=(NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(3,notification.build(
));
```



Lock Screen Notification

- Following Code Snippet demonstrates creating a Lock Screen Notifications:

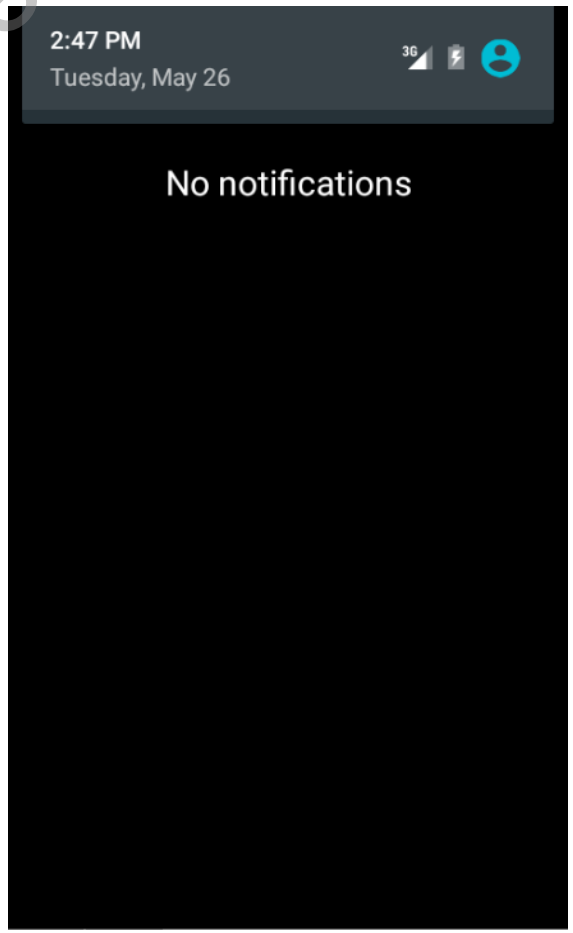
```
Notification.Builder notification =new
Notification.Builder(this)
.setSmallIcon(R.drawable.ic_notification_small)
.setVisibility(Notification.VISIBILITY_PUBLIC)
.set…;

NotificationManager mNotificationManager =
(NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);

mNotificationManager.notify(4,
notification.build());
```



Lock Screen Notification

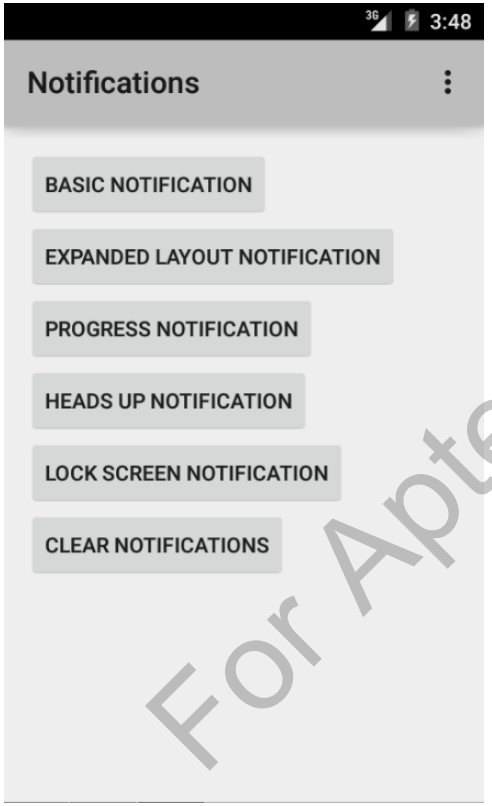
- Following Code Snippet demonstrates clearing all the notifications:

```
 NotificationManager mNotificationManager =
(NotificationManager)
getSystemService(Context.NOTIFICATION_SERVICE);
        mNotificationManager.cancelAll();
```

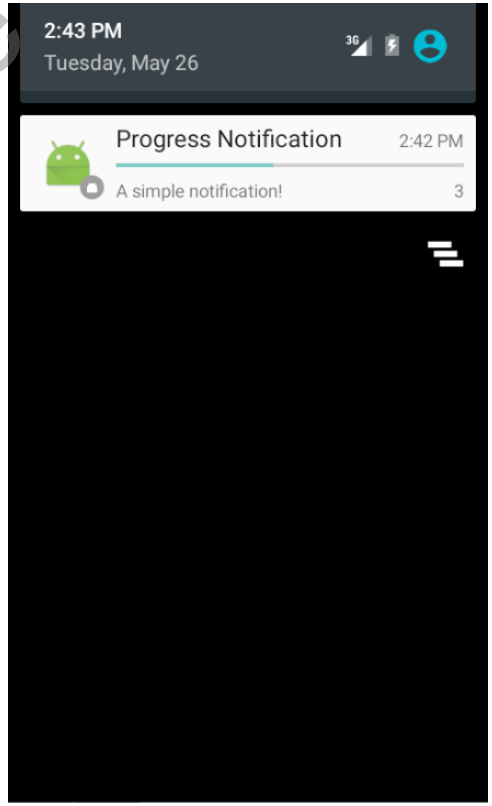


Notification Example Application 1-2

- Using the code, an application for demonstrating accessing the External File System is created as shown in the following figure:

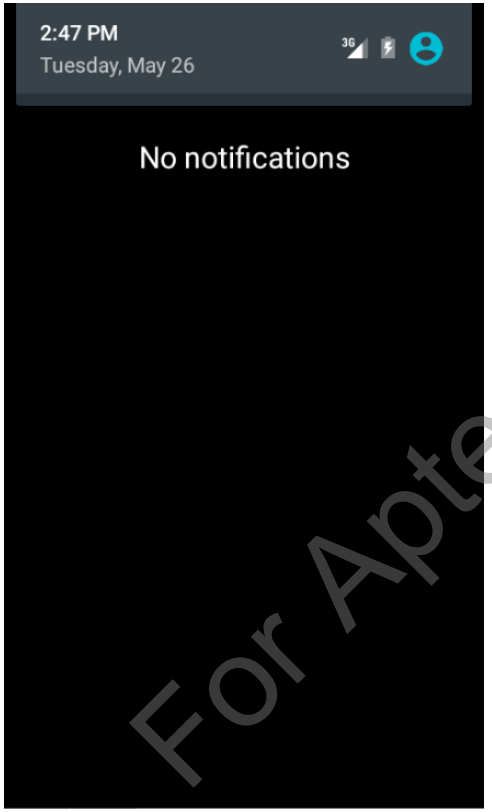


- By clicking the Notification button, the appropriate notification is displayed as shown in the following figure:



Notification Example Application 2-2

- By clicking the CLEAR NOTIFICATIONS button, all the notifications are cleared as shown in the following figure:



- The application logic is as follows:

- The developer creates an activity with the necessary buttons
- Whenever a button is clicked, the onClickListener will execute the code to create the appropriate notification
- When the CLEAR NOTIFICATIONS button is clicked, all the notifications are cleared using NotificationManager's cancelAll() method

Android Security Model 1-3

- Core of Security is the Linux Kernel
- **Features -**
  - Permissions tailored to user needs
  - Separating processes
  - Securing Inter-Process Communication (IPC)
  - Eliminating parts of the Kernel that are not secure
  - Separating resources of one user from another

- **Permissions -**
  - Apps can only access resources they have explicit permission for
  - Permissions are displayed during installation
  - Requires User consent
  - Each application gets a different User ID
  - Application resources are protected to it's User ID
  - OS data cannot be modified without root access

Android Security Model 2-3

- **Securing Processes & Resources**
  - Processes and resources of every application are isolated
  - Applications cannot have resources that are restricted from the OS
  - OS keeps tracks of the Process, Resources, and the respective Permissions
- **User Security Options**
  - Device Management
  - Secure Password
  - Data Encryption
  - Certification Based System
  - VPN
  - Protection Relating to Third-Party Applications

Android Security Model 3-3

- **Security Features in Lollipop**
  - Full Disk Encryption Enabled by Default
  - Improved Encryption Procedures
  - SELinux Access Control
  - Updates to Cryptography of HTTPS and SSL
  - Position Independent Execution Removed
  - Multi User Profiles
  - Additional Locking/Unlocking Options

Summary

- Preferences are a subclass that helps users to customize their applications. A developer can specify preferences using an XML file or using code
- Title, Intents, and sub screens are the different ways that helps to group settings
- The Android file system offers internal and external storage. Internal Storage is removed along with the application. External Storage can persist
- Android's notifications enable an app to inform the user about events
- Starting from Android Lollipop, Notifications can be made to be intractable on lock screen and act as substitutes for lock screen widgets
- The Android security model provides several security features, such as, permissions, the Application Sandbox, several user security options, Full Disk Encryption, and protection from third-party applications