HSBC

# Android App Development using Kotlin

Infosys®
Navigate your next

# Android App Development Using Kotlin



HSBC | Infosys®

# Activity Lifecycle

# What is an Activity lifecycle ?

- Each activity in Android does not control its own lifetime.
- Android runtime manages the lifecycle of each activity.
- The state of the activity will determine how the runtime treats an activity.
- Each activity is related to where the activity is positioned in the activity stack.
- All running activities are placed on this stack which works according to “last-in-first-out” principle.

# Activity lifecycle states

- An activity of an Android app can be in any of the following 4 states:

1. **Active**: An activity currently in focus, either displaying info or accepting input from the user

2. **Paused**: An activity currently in the view but out of focus. This happens when another activity overlaps the view of current activity and takes the focus away.

3. **Stopped**: An activity currently not visible. This activity is still present in the memory and can be restarted at any point of time

4. **Inactive**: An activity when taken out of memory becomes inactive.

# Demo on Activity lifecycle

# Context

- Provides global information about an environment of application.
- An Abstract class, whose Implementation is provided by Android System.
- Allows access to Application related resources and classes, application-level operations.
- Some Involved Operations: Launching Activities, Loading Resources, Displaying Toast messages, creating dynamic Views, etc.
- Methods:

  - ✓ getContext()
  - ✓ getBaseContext()
  - ✓ getApplicationContext()
  - ✓ this

# Intents

# Intent and its types

- Used  to pass messages and information to various components of an Android app
- Intents are instances of - **android.content.Intent**
- Used to trigger:
  - Services
  - Activities
  - Broadcast Receivers
- Also used to store data and get stored data in other component (via extras)
- Type - Internal and External

# Demo on Intents

# Intent Filters

- Used to declare a component's capability
- Used to find which components can handle an implicit intent at runtime.
- It is an Object of class IntentFilter
- It is in the Android Manifest file using <intent-filter> elements
- Fields of intent filter are :
  - Action
  - Data
  - Category

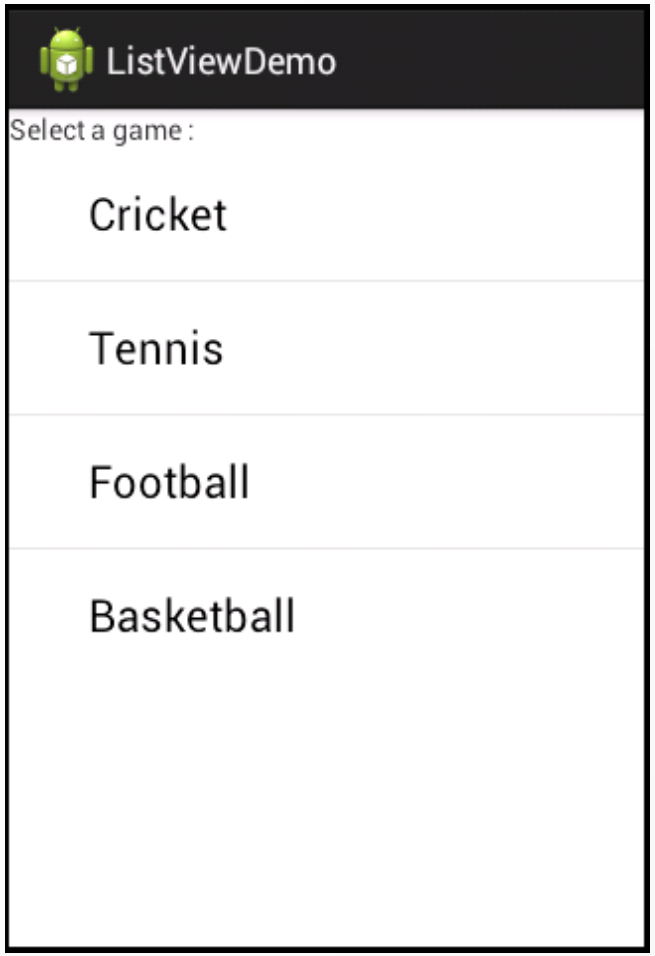
# Demo on Intent Filter

# ListView

# ListView

- Display List of Items in an activity
- Has “non-editable” TextViews
- It displays the list in a vertical manner.
- The activity code uses “android.widget.ListAdapter” class to display the list or to put the modified list
- Data can be populated into Listview in 2 ways

1. Statically using xml
2. Dynamically using Kotlin via Adapters



# Populating data into ListView using Adapters (Cont.)

Need of Adapters :

a. Adapters are used to populate more than one list of items into a ListView .

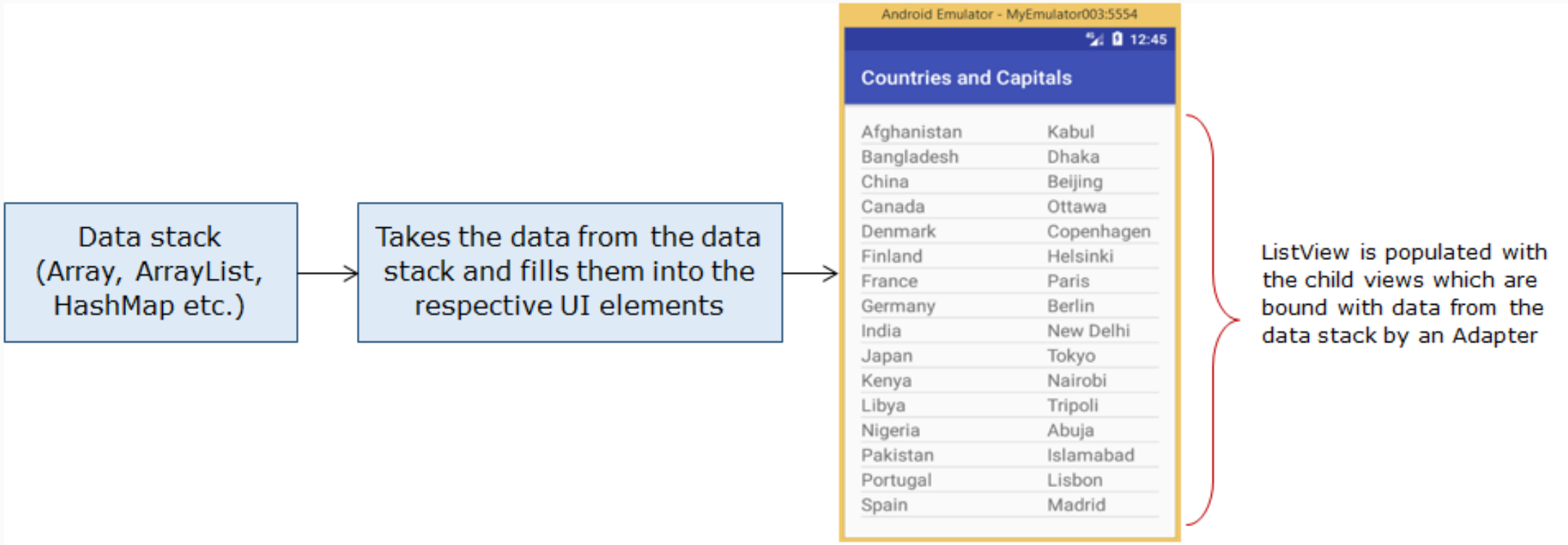
b. Adapter is an interface that acts as link/bridge between the UI elements and data stack.

The two main responsibilities of Adapter are:-

1. Creates a view for each child item of the ListView

2. Bound the data into the view from the data stack

# Populating data into ListView using Adapters



# Types of Adapters

Adapters are implemented in below APIs:

**Base Adapter :** This is the base class of an adapter ,used when there is need of customization in the ListView , such as displaying multiple UI elements for each items in the ListView.

**Array Adapter:** This kind of adapter is best used when there is a list of single items, that is backed by an array.

**Simple Adapter:** This kind of adapter is best suited to populate static data into the ListView.

# ListView Demo

# UI Customization-
# Customization  of ListView

# Customizing ListView with BaseAdapter

- **Steps:**

1. Create a ListView in the layout.xml file
2. Create the custom template for the ListView
3. Initialize the Dataset in the Activity
4. Create the Custom Adapter
5. Populate the data into ListView via Adapter

# CustomListView Demo

# Enhancing the Performance of ListView

# Enhancing the Performance of ListView

- To ensure smooth scrolling and better user experience, the performance of ListView needs to be enhanced.
- In order to do this, the main thread or the UI thread should be free from heavy processing.
- Ways:
  1. Holding View Objects in a View Holder
     - In place of repetitive calling of findViewById multiple times, a view holder design pattern can be implemented.
     - ViewHolder object stores each of the component views inside the tag field of layout.
     - This allows the accessing without the need to look up for the element repeatedly
  2. Using a background thread
     - This removes strain from the main thread to focus on drawing the UI
     - Implemented using AsyncTask

# Steps in enhancing the Performance of ListView

**Steps:**

1. Create a ListView in the layout.xml file
2. Create the custom template for the ListView
3. Initialize the Dataset in the Activity
4. Create the Custom Adapter implementing ViewHolder design pattern
5. Populate the data into ListView via Adapter

# ListView with ViewHolder Demo

# UI Enhancement- RecyclerView and CardView

# RecyclerView (Cont.)

ListView will have it's own limitations such as:

- ListView consumes more memory , it can't be used with huge dataset
- ListView supports only vertical orientation
- ListView does not support User Experience(look and feel)

To overcome the above limitations RecyclerView is used.

# RecyclerView (Cont.)

- RecyclerView is efficient and flexible than ListView
- Views can be Recycled and Scrolled efficiently.
- RecyclerView provides greater flexibility to customize and optimize the list-items with larger datasets in comparison with ListView.
- Supports both horizontal and vertical display and supports APIs such as Item Animator and ItemDecorator
- The view holder objects are managed by an adapter by extending RecyclerView. Adapter.

# RecyclerView

RecyclerView supports various API's such as

- *ViewHolder*:- Used to increase the speed of rendering data into RecyclerView
- *LayoutManager*:- Used to determine the size and position of items inside the RecyclerView
- *ItemDecoration*:- Used to add special drawing and decoration items inside RecyclerView
- *ItemAnimator*:- Used to implement custom animations for items inside RecyclerView

# CardView

- *CardView* represent the information, present in a RecyclerView, in a card format with a drop shadow and corner radius.
- The UI content is shown inside the CardView, which in turn becomes the row in the RecyclerView.
- Provide consistent look across various platform to data.
- Each card represents a UI layout, which is reused by the RecyclerView depending on the volume of the dataset.
- CardView provide a better UI as we can adjust the layout content, border radius, elevation etc.

# RecyclerView and CardView

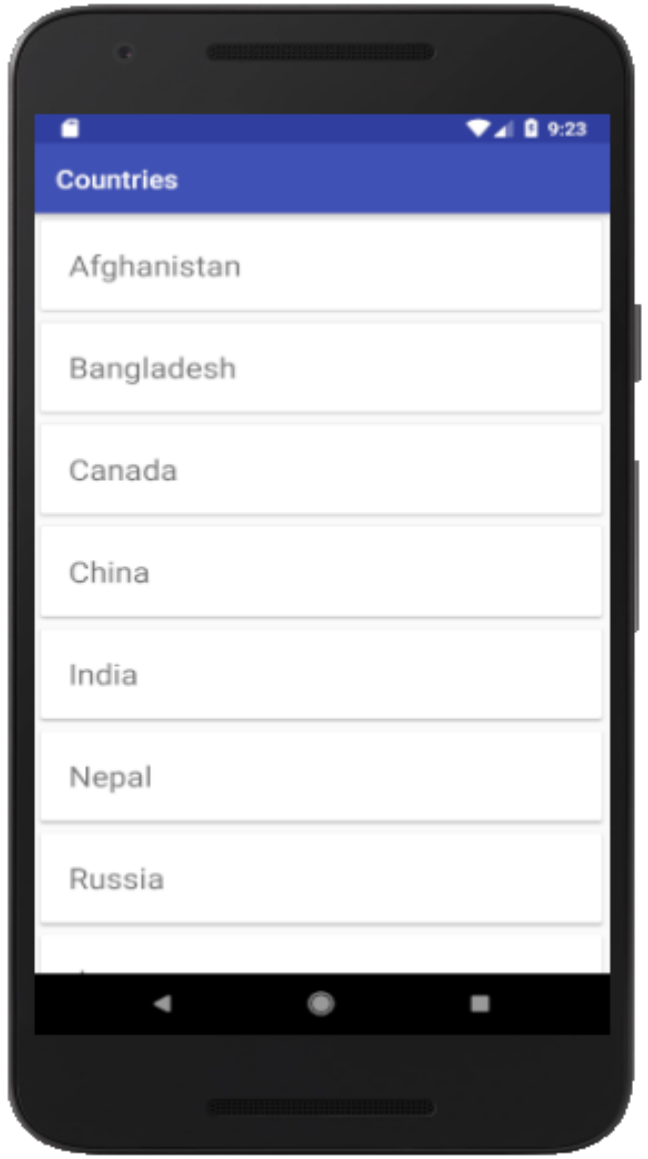
**Steps:**

1. Configure the layout .xml file with the RecyclerView and adding the dependency

2. Create a new layout .xml file as a template for the RecyclerView and adding the dependency for CardView

3. Create a Custom Adapter for filling the data into the RecyclerView

4. Populating the items into the RecyclerView via the Custom Adapter



# RecyclerView and CardView Demo

# Data Handling in Android Devices – Challenges & Solutions

# Data Storage

- Storage Mechanisms:
  - Files (unstructured data)
  - Key value pairs (semi-structured  data)
  - Database (structured data)

- Storage Location:
  - Internal storage
  - External storage (SD card)
  - Cloud

# Challenges

- Limited memory
- Synchronization challenges
- Need for Simple flat files
  - Everything can not be stored in the database
- Need for dealing with XML files
- Sharing of data between 2 applications on a single device.

# Storage Solutions in Android

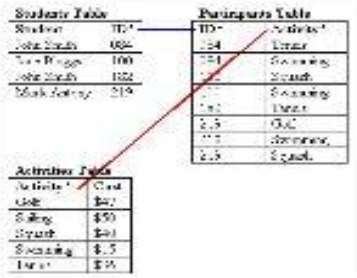
| Key | Value |
|---|---|
| fName | Tyrion |
| lName | Lanniste r |

Basic key-value pairs

→ Shared Preferences

Unstructured/binary data

→ Flat files



Relational data

→ SQLite database

# Transfer Mechanism

- To transfer the data between apps the below methods can be used.

Within device:

Inter App ⟶ Content Provider

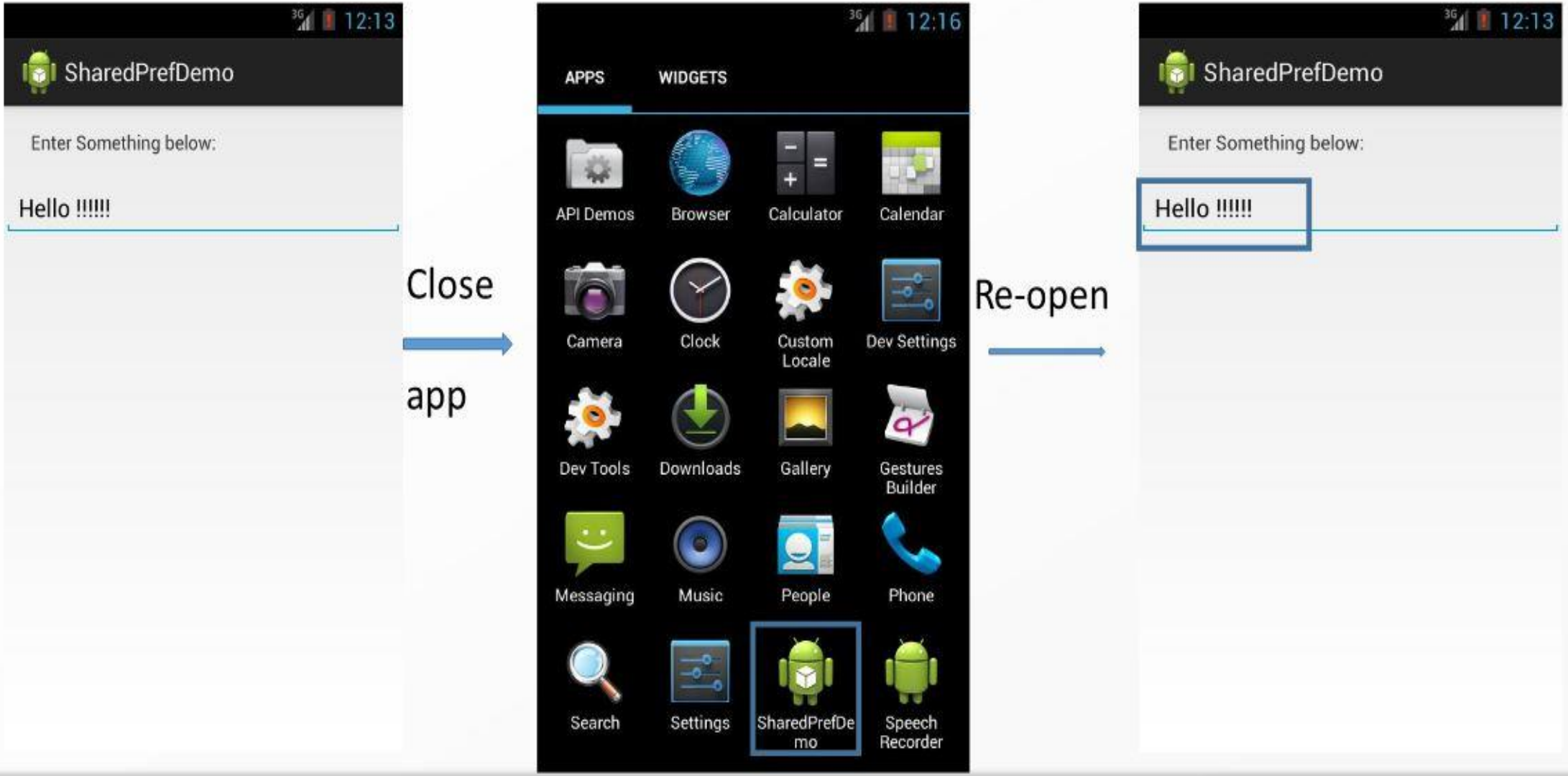
Outside Device:

With Network ⟶ Network API

Formats: JSON/XML/binary

# SharedPreferences API in Android

# Preference Demo



# How does Shared Preference work?

- Android uses XML files for storing small amounts of data in key-value format.
- It stores primitive data types like Boolean, int, long, float and string.
- Data types are easily edited using SharedPreference API.

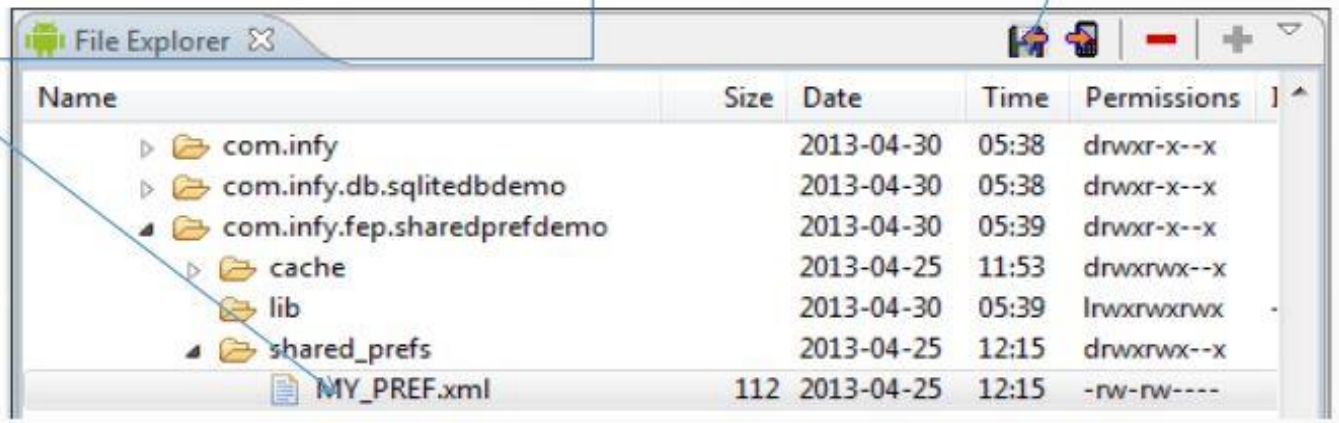
# Creating SharedPreferences

- Invoke getSharedPreferences method for getting SharedPreferences.
- Open/Create our SharedPreferences object, take out the already stored value.
- To edit the SharedPreferences, we open its Editor and save the data as a key-value pair.
- It is necessary to call commit() to save changes after editing the preferences file.

# Locating My_PREF.xml

Open File Explorer in DDMS perspective. Go to **data/data/package name/shared_prefs** to find the created xml file.

To look at its contents, select the file and click on 'Pull a file from device' button. Save it and open it with a browser to view its contents.




## Content of MY_PREF.xml

```
<?xml version="1.0" encoding="utf-8" standalone="yes" ?>
- <map>
    <string name="key1">Hello !!!!!!</string>
  </map>
```

## Demo on SharedPrefernces

# Thank You



HSBC | Infosys®