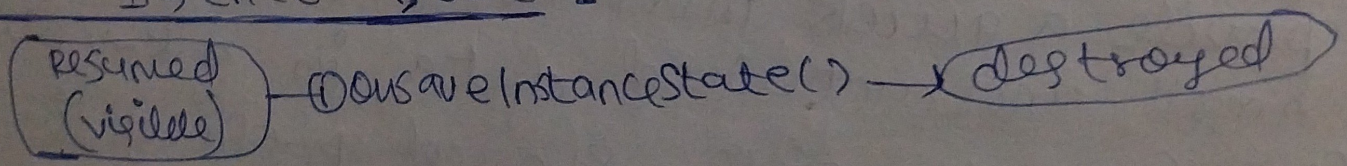
Module - V

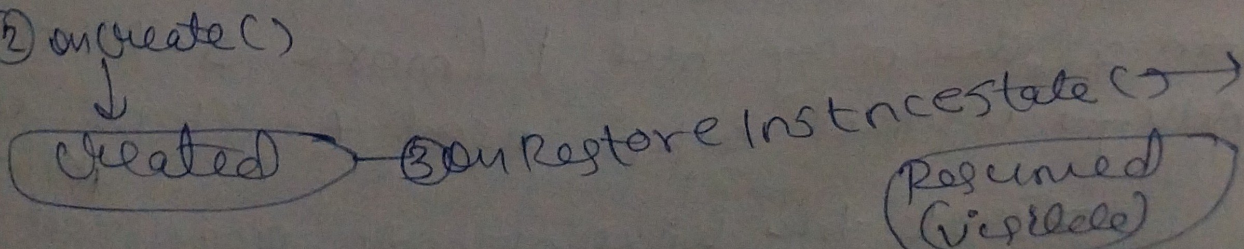
Persisting Data

* persistnt data denotes info that is frequently accessed & not likely to be modified
* (An) provides several options for saving data persistnt appli-
* Data storage options in (An) —

1) SharedPreferences → store primitive private data on key-value pairs.
2) Internal Storage → store private data in the device only
3) Ex. Storage → Store public data on Shared Ex. Storage.
4) SQLite DB → Store structured data in a private DB
5) N/w conn^{n} → Store data on the web with own n/w srvr.

Saved Instnce State :





* It is the very 1st scene when you are storing & retrieving the data in yr appli-
* onSaveInstanceState() called when an activity is destroyed which takes a bundle of parmtrs. This bundle is a collection of k-v pair in which we can store the state of every view in the bundle (txtview, Edittxt, checkbox)
* when the appli- is restarted then onRestore InstnceState() is called to which we provide this bundle & it automatically restores the activity to the state in which it was closed.
* Adv → state of activity is restored if the activity is closed
  → useful when device orient^n is changed.
* dis → data stored is not persistnt (data is lost if the device is restarted)

II Shared prefrnces =

* provide a quick intrface to save & retrieve small unstructured data in a k-v pair.
* (An) stores the appli- shared prefnces in an XML file with private access to the appli-
* An appli- may hve 1 / more S.prefnces.
* Adv → useful when need to store user prefnces
  → Just few lines of code required to manipulate data.
* to use S.P, call a method getSharedPreferences() that returns a SharedPrfnce instance pointing to the file.
* S/x → ~~public abstract SharedPrfnces~~
  SharedPrefnces SP = getSharedPrefnces(myprefncs, MODE_PRIVATE);
  (MODE_PRIVATE → context mode; myprefncs → name of SP.)

context mode
- MODE_PRIVATE → make the file private.
- " PUBLIC → " public
- " APPEND → used while reading data from SP file.

* To write data in SP use SharedPrefrences.Editor clss —

```
S.P.Editor editor = S.P.edit();
editor.putString(keyString, valueString);
editor.commit();
```

* To read data use S.P methods — getAll(), getBoolean(), getFloat(), getInt(), getLong() ~.

III Prefrnce Framwork & prefrnce Activity:

* (An) offers an XML-drivn framwork to create system-style prefrnce screens for yr appli-
* By using this framwork, you can create prfnce activities that are consistnt with those used in both native & othr 3rd-party appli~

* 2 adv —
  - users will be familier with the layout & use of yr settings screens.
  - you can integrate settings screens frm othr applin into yr applin's prefrnces.

* P. frmwrk consist of 4 parts —

1) P. screen Layout : xml file that defines the hierarchy of items displayed in yr P. screens.

2) P. Activity & P. Fragmnt : used to host P. screens.

3) P. Header defnition : xml file that defines the P. (F) for yr applin

4) S.P change listner : used to listen for changes to SP. using onSpChangeListener. cls.

V P. Layout in XML :

* P. defni" are stored in res/xml resource folder.
* It use a specialized set of controls designed specifically for P
* Each P. layout defined as a hierarchy beginning with a single P. screen elemnt —

```
<?xml ... ?>
<PrefnceScreen
    <checkboxPreference
        an:key="pre_sync"
        an:title= ~
        an:Summary=" ~ "
        an:defaultValue="true" />
an:key → Set a unique identifier for the P within the screen.
an:title → txt displayed as a title of P.
an:Summary → set Summary txt displayed below the title, providing more details about the P.
an:defaultValue → Set defaultValue for the checkbox.
    true → checkbox will be checked by default.
</PrefnceScreen>.
```

V Native P. control :

* checkbox.P → - used to set P to T/F
* EditText.P → Allow users to enter a String f value as a P
* List.P → Selecting this P will display a dialog box containing a list of values frm which to select.
* MultiSelectList.P.
* Ringtone.P → list of available ringtones for user selection.

VI P. Fragmnt :

* P(F) cls used to host the P. screens defined by P. Screen resources.
* To create a new P(F) —

public static class PrefsFragment extends PreferenceFragment.

VII P.Activity :

* P.Activity cls is used to host the PF hierarchy defined by a p.Headers res.
* To create a new P.A—

public class MyPActivity extends PActivity

⇒ Including static files as res :

* If yr appln requires ex. file res, you can include them in yr distribu" Pkge by placing them in res/raw fldr.
* To access these read-only file res, call the openRawResource method
* Ex—

```
Resources myres = getResources();
InputStream myfile = myres.openRawResource
        (R.raw.myfilename);
```

* Adding files to yr resources hierarchy is an excellnt alternative for large data.

I working with Filesystem :

* (An) supplies some basic file-managment tools to deal with system.
* Many of these are located within java.io.File pkge.

* deleteFile → Enables you to remove files created by the crnt appli-
* fileList → returns a string array that includes all the files created by the crnt appli-
* Files stored in the appli-folders should be specific to the paint appli- & typically not detected by the media-scanner.
* creating private Appli-files—

```
String filename = "temp.tmp";
FileOutputStream fos = openFileOutput(filename,
        Context.MODE_PRIVATE);
FileInputStream fis = openFileInput(filename);
```

[for public appln files instead of MODE_PRIVATE write MODE_WORLD_WRITEABLE].

* Storage Space—

Internal Storage	Ex. Storage
• located on the device itself • Faster in speed • More secure • limited capacity • Accessible by apps with appropriate permissions	- removable storage like SD card Slower less secure maybe larger than internal Storage. Accessible by apps & users.

=> SQLite :

* used to store appli- data using a managed structured approach.
* opensource SQL DB that stores data to a txt file on a device.
* offers a full SQLite relational Db library.
* Every appli- can create its own Dbs over which it has complete control.
* It is a well-regarded RDBMS & it is lightweight.
* reduces ex. dependencies, minimizes latency (comp-ൽ [illegible] data-ത് [illegible] time)
* (An) comes in with built in SQLite db implemⁿ.
* sqlite is serverless & stores data in 1 db file, offers only few datatypes, uses manifest typing instead of static types, uses cross-platform db files.

I Datatypes :

1) NULL → null value
2) INTEGER → Any no. which is no floating point no.
3) REAL → floating-point no.
4) TEXT → Any string & also single chars.
5) BLOB → A binary blob of data

The datatype of a value is associated with the value itself, not with its container. & uses a more general dynamic type systm.

* BLOB (binary large obj) is a collection of binary data stored as a value in the db. using BLOB, you can store doc, img. & othr multimedia files in the db.

II Db manipulation using SQLite :

* main pkge is an.database.sqlite that contains the clses to manage yr own dbs.
* creation :
  you just need to call this method - openOrCreateDatabase with yr dbname & mode as a paramtr.
  s/x →
  SQLiteDatabase mydb = openOrCreateDatabase("your dbname", MODE_PRIVATE, null);
* Table creation :
  mydb.execSQL("CREATE TABLE IF NOT EXISTS test table (usname txt, password text);");

III To perform ins, dlt, update opⁿ in 2 ways -

1) write parameterized queries (recommended)
   ↓
   queries which are performed using inbuilt (fns) to ins, dlt, read/update data
2) write raw queries.
   ↓
   simple sql queries like mysql, etc.. here user has to write query & pass to rawQuery(String sql, String[] selectArgs) or execSQL() method to perform oprs.

eg → raw query to ins data :

```
mydb.execSQL ("INSERT INTO testtable
              VALUES ("admin","admin");");
```

parametrized query :

```
public long insert (String tablename, String
nullColumnHack, ContentValues values)
```

(nullColumnHack → null; values → values that need to be inserted into the table.)

[values-ൽ ഏതു column-നും ഉള്ളത് indukl
nullColumnHack be null, if not അങ്ങനെയൊന്നു value ഇല്ലെങ്കിൽ
അതിന്റെ non nullColumnHack-ൽ കൊടുക്കണം]

IV update opr :

raw query

```
String updateq = "UPDATE mytable SET
    salary = 5000   WHERE id = '5'";
dbManager.RawQuery(updateq , null);
```

parametrized query —

```
public int update (String tablename, ContentValues
    contentvalues, String whereClause, String[]
    whereArgs)
```

(whereClause → salary = 5000; whereArgs → id = 5)

V Dlt query :

raw query —

```
String dltq = "DELETE mytable where id=5";
dbmanager.RawQuery(dltq , null);
```

parametrized q →

Same as update & here no content values.

```
public void dltitem(Item item) {
    SQLiteDatabase db = getWritableDatabase();
    String whereclause = "id=?";
    String whereArgs[] = {item.id.toString()};
    db.delete("items", whereClause, whereArgs);
}
```

[? → ഇതിന്റെ സ്ഥാനത്ത് ഇതിന്റെ actual value വരും when executing]

VI Fetching :

* by using an obj of the cursor cls.
* we will call a method of this cls → rawquery & it will return a rsltset with the cursor pointing to the table.
* we can move cursor forward & retrieve data.
* Sh →

```
cursor resltset = mydb.rawquery("select
        * from testtable", null);
resltset.moveToFirst();
String usname = rsltset.getString(0);
String password =     "      (1);
```

VII Hlper cls :

For managing all the opr related to the db & for version managmnt, a hlper cl

has been gvn → SQLiteopenHelper.
It automatically manages the creation & update of the db.

VIII (An) Db Cls :

1) SQLite Db : (An)'s java interface to its relational db SQLite.
2) cursor : It is a container which holds the rslt of a db query & also supts the commonly used MVC design pattn. It returns the values depending upon the crnt position of cursor indx.
3) SQLiteopenHelper
4) SQLiteQueryBuilder :
provides a high-level abstran to create sqlite queries to be used in (An) appln.