MAJLIS ARTS AND SCIENCE COLLEGE

PG DEPARTMENT OF COMPUTER SCIENCE

(Affiliated to the University of Calicut, approved by the Government of Kerala)

Majlis Nagar, Puramannur-P.O 676552, Malappuram Dt, Kerala, 0494 2643970, 9539111174

Sixth Semester Online Study Camp.

Study Camp. 6 BCA , BSc

WhatsApp group

Scan QR Code to Join Study Camp WhatsApp Group

*** UNIT WISE REVISION**
*** PREVIOUS YEAR QUESTION PAPER DISCUSSION**
*** ASSIGNMENTS**

masc.majliscomplex.org

1. How to work with the Android file system

Android provides many kinds of storage for applications to store their data. These storage places are shared preferences, internal and external storage, SQLite storage, and storage via network connection.

In this chapter we are going to look at the internal storage. Internal storage is the storage of the private data on the device memory.

By default, these files are private and are accessed by only your application and get deleted, when user delete your application.

Writing file

In order to use internal storage to write some data in the file, call the openFileOutput() method with the name of the file and the mode. The mode could be private, public e.t.c. Its syntax is given below –

```
FileOutputStream fOut = openFileOutput("file name here",MODE_WORLD_READABLE);
```

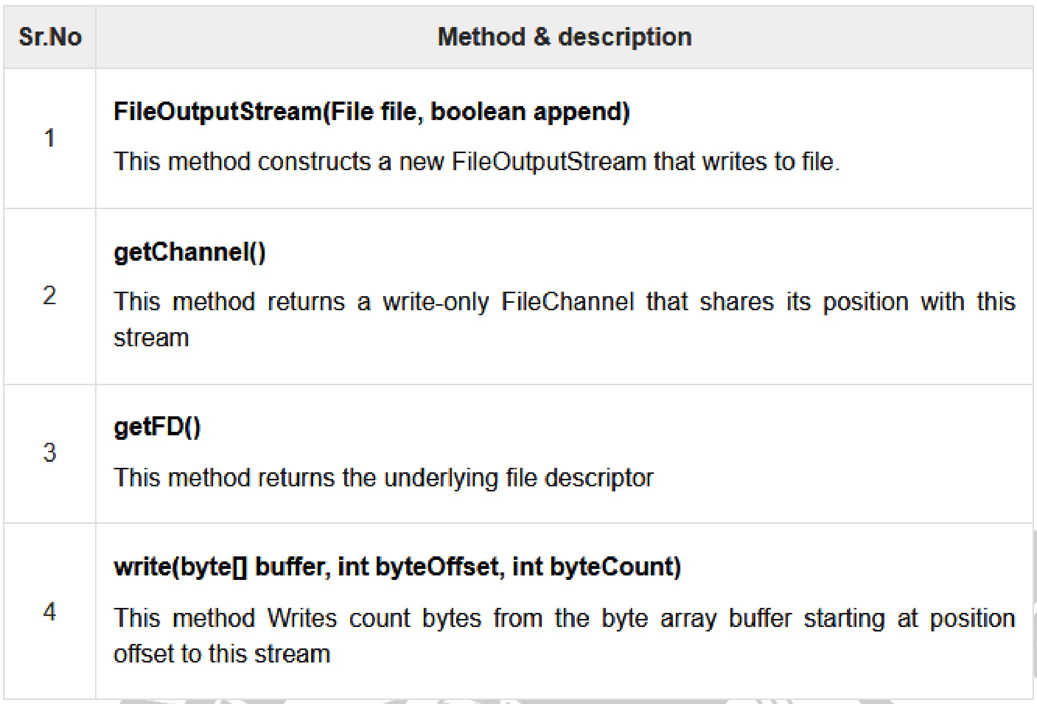
The method openFileOutput() returns an instance of FileOutputStream. So, you receive it in the object of FileInputStream. After that you can call write method to write data on the file. Its syntax is given below –

```
String str = "data";

fOut.write(str.getBytes());

fOut.close();
```

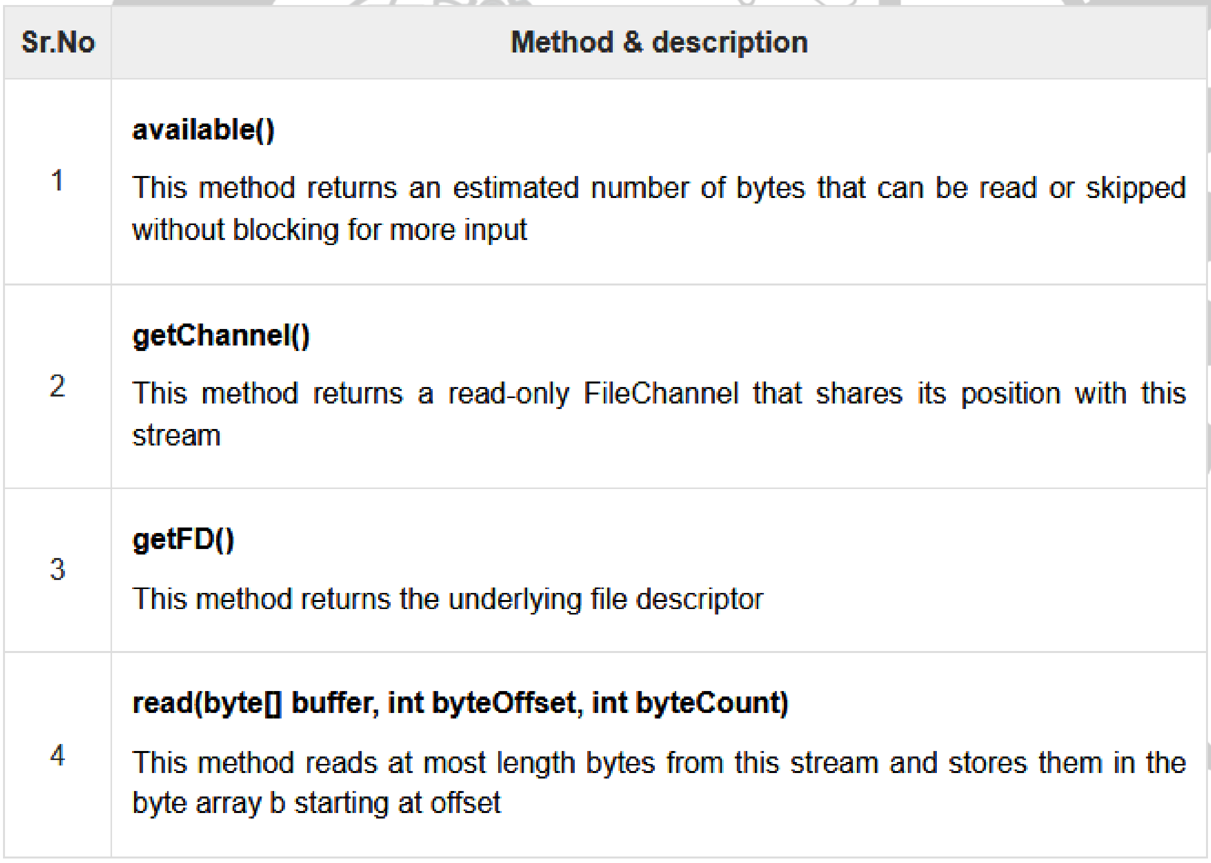
Reading file

In order to read from the file you just created, call the openFileInput() method with the name of the file. It returns an instance of FileInputStream. Its syntax is given below –

```
FileInputStream fin = openFileInput(file);
```



Sr.No	Method & description
1	**FileOutputStream(File file, boolean append)** This method constructs a new FileOutputStream that writes to file.
2	**getChannel()** This method returns a write-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**write(byte[] buffer, int byteOffset, int byteCount)** This method Writes count bytes from the byte array buffer starting at position offset to this stream



Sr.No	Method & description
1	**available()** This method returns an estimated number of bytes that can be read or skipped without blocking for more input
2	**getChannel()** This method returns a read-only FileChannel that shares its position with this stream
3	**getFD()** This method returns the underlying file descriptor
4	**read(byte[] buffer, int byteOffset, int byteCount)** This method reads at most length bytes from this stream and stores them in the byte array b starting at offset

2. Android - Shared Preferences

Android provides many ways of storing data of an application. One of this way is called Shared Preferences. Shared Preferences allow you to save and retrieve data in the form of key,value pair.

In order to use shared preferences, you have to call a method getSharedPreferences() that returns a SharedPreference instance pointing to the file that contains the values of preferences.

SharedPreferences sharedpreferences = getSharedPreferences(MyPREFERENCES, Context.MODE_PRIVATE);

epresents a top-level Preference that is the root of a Preference hierarchy. A PreferenceActivity points to an instance of this class to show the preferences. To instantiate this class, use PreferenceManager#createPreferenceScreen(Context).

This class can appear in two places:

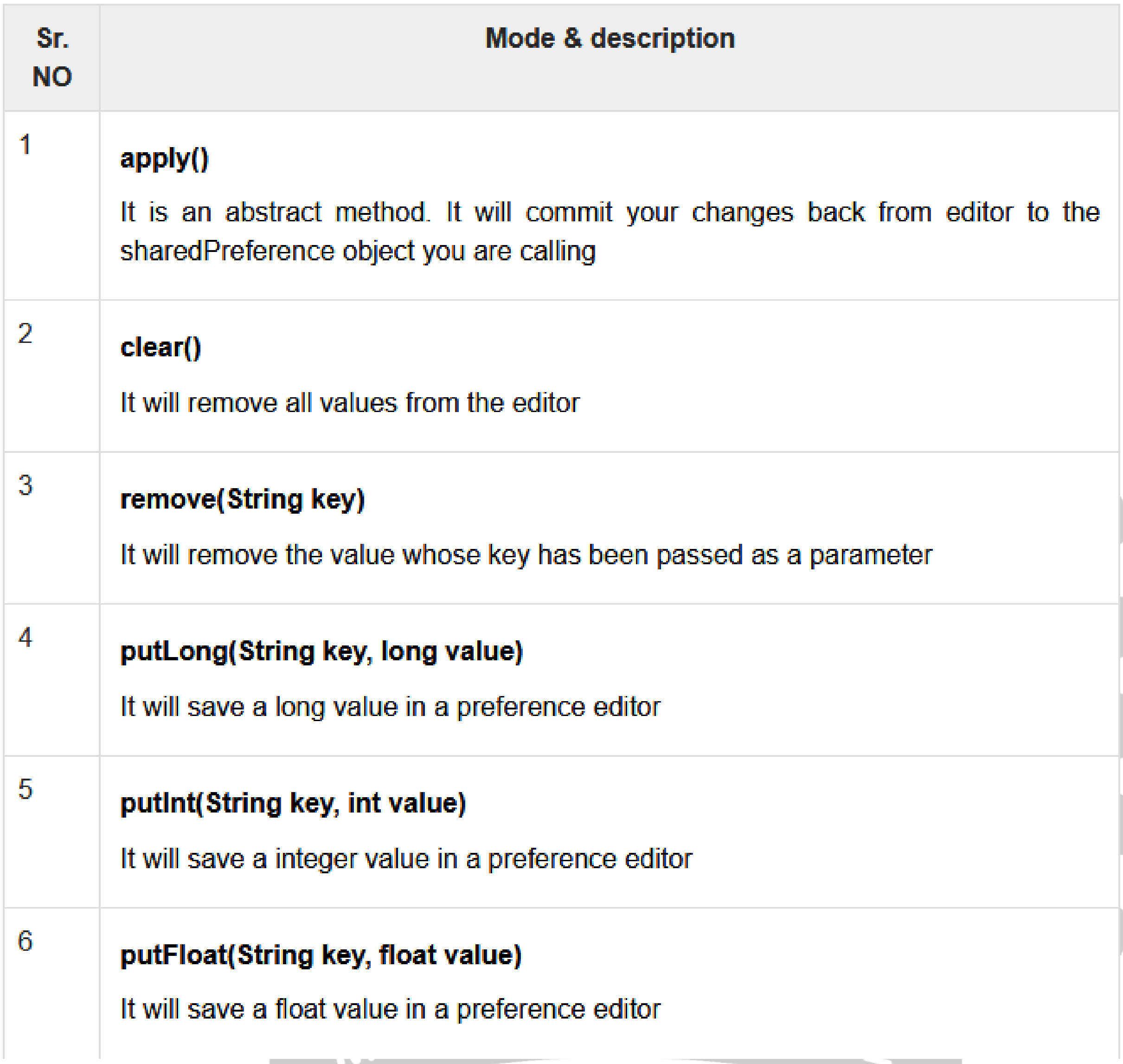
When a PreferenceActivity points to this, it is used as the root and is not shown (only the contained preferences are shown).

When it appears inside another preference hierarchy, it is shown and serves as the gateway to another screen of preferences (either by showing another screen of preferences as a Dialog or via a Context#startActivity(android.content.Intent) from the Preference#getIntent()). The children of this PreferenceScreen are NOT shown in the screen that this PreferenceScreen is shown in. Instead, a separate screen will be shown when this preference is clicked.

The first parameter is the key and the second parameter is the MODE. Apart from private there are other modes available that are listed below –

Sr.No	Mode & description
1	**MODE_APPEND** This will append the new preferences with the already existing preferences
2	**MODE_ENABLE_WRITE_AHEAD_LOGGING** Database open flag. When it is set , it would enable write ahead logging by default
3	**MODE_MULTI_PROCESS** This method will check for modification of preferences even if the sharedpreference instance has already been loaded
4	**MODE_PRIVATE** By setting this mode, the file can only be accessed using calling application
5	**MODE_WORLD_READABLE** This mode allow other application to read the preferences
6	**MODE_WORLD_WRITEABLE** This mode allow other application to write the preferences

Apart from the putString method, there are methods available in the editor class that allows manipulation of data inside shared preferences. They are listed as follows –



Sr. NO	Mode & description
1	**apply()** It is an abstract method. It will commit your changes back from editor to the sharedPreference object you are calling
2	**clear()** It will remove all values from the editor
3	**remove(String key)** It will remove the value whose key has been passed as a parameter
4	**putLong(String key, long value)** It will save a long value in a preference editor
5	**putInt(String key, int value)** It will save a integer value in a preference editor
6	**putFloat(String key, float value)** It will save a float value in a preference editor

Example

This example demonstrates the use of the Shared Preferences. It displays a screen with some text fields, whose value are saved when the application is closed and brought back when it is opened again.

To experiment with this example, you need to run this on an actual device on after developing the application according to the steps below –

Steps	Description
1	You will use Android studio to create an Android application under a package com.example.sairamkrishna.myapplication.
2	Modify src/MainActivity.java file to add progress code to display the spinning progress dialog.
3	Modify res/layout/activity_main.xml file to add respective XML code.
4	Run the application and choose a running android device and install the application on it and verify the results.

3. Shared Preferences in Android

One of the most Interesting Data Storage option Android provides its users is Shared Preferences. Shared Preferences is the way in which one can store and retrieve small amounts of primitive data as key/value pairs to a file on the device storage such as String, int, float, Boolean that make up your preferences in an XML file inside the app on the device storage. Shared Preferences can be thought of as a dictionary or a key/value pair. For example, you might have a key being "username" and for the value, you might store the user's username. And then you could retrieve that by its key (here username). You can have a simple shared preference API that you can use to store preferences and pull them back as and when needed. Shared Preferences class provides APIs for reading, writing, and managing this data. A sample GIF is given below to get an idea about what we are going to do in this article. Note that we are going to implement this project using the Java language.

epresents a top-level Preference that is the root of a Preference hierarchy. A PreferenceActivity points to an instance of this class to show the preferences. To instantiate this class, use PreferenceManager#createPreferenceScreen(Context).

This class can appear in two places:

When a PreferenceActivity points to this, it is used as the root and is not shown (only the contained preferences are shown).

When it appears inside another preference hierarchy, it is shown and serves as the gateway to another screen of preferences (either by showing another screen of preferences as a Dialog or via a Context#startActivity(android.content.Intent) from the Preference#getIntent()). The children of this PreferenceScreen are NOT shown in the screen that this PreferenceScreen is shown in. Instead, a separate screen will be shown when this preference is clicked.

Here's an example XML layout of a PreferenceScreen:

```
<PreferenceScreen
       xmlns:android="http://schemas.android.com/apk/res/android"
       android:key="first_preferencescreen">
   <CheckBoxPreference
           android:key="wifi enabled"
           android:title="WiFi" />
   <PreferenceScreen
           android:key="second_preferencescreen"
           android:title="WiFi settings">
       <CheckBoxPreference
               android:key="prefer wifi"
               android:title="Prefer WiFi" />
       ... other preferences here ...
   </PreferenceScreen>
</PreferenceScreen>
```

4. Preference Framework

Android works with the Preference Framework which is considered a powerful framework in modern mobile technology. We also know, the most of the programs can be configured to suit our requirements for all purpose and we usually go out of our way to set up our favorite configuration for a given program and information. It permits our users to do the same in our Android application and gives our application an advantage in the usability area.

Common preferences that are built into the framework are comprise in the following preferences:

EditTextPreference: A preference that can store plain text in string format.

CheckBoxPreference: A preference that can store a boolean value.

RingtonePreference: A preference that also sanctions the user to store a preferred ringtone from those available in the device.

ListPreference: A preference that allows the user to select a preferred item from a list of items in the dialog box

5. What is PreferenceActivity Class?

The responsibility of the PreferenceActivity class is to show a hierarchy of the Preference objects as per lists, possibly on both sides of multiple screens. When the preferences are edited, they are stored by using an instance of the Shared- Preferences. The SharedPreferences class is an interface for accessing and modifying a preference data returned by the getSharedPreferences () from any Context object.

6. Laying out preferences:

Now working with the layouts in Android can sometimes be a time-consuming process. Actually, the Building layouts are almost like building a Web site with various tables all over the place. Sometimes it is very easy and simple way; sometimes it is complex. We are lucky that the layout of the Android preferences is much simpler than defining a layout for our application screens.

Android preference screens are broken into the following categories process:

Sl	Preference	Application
1	PreferenceScreen	Represents a top-level preference that is the root of the preference hierarchy. We can use a PreferenceScreen in two places for our job.
2	PreferenceActivity	The PreferenceScreen is not shown because it only shows the containing preferences within the PreferenceScreen definition.

7. Working with file system, SQLLite

SQLite is an open-source relational database i.e. used to perform database operations on android devices such as storing, manipulating or retrieving persistent data from the database.

It is embedded in android bydefault. So, there is no need to perform any database setup or administration task.

Here, we are going to see the example of sqlite to store and fetch the data. Data is displayed in the logcat. For displaying data on the spinner or listview

8. SQLiteDatabase class

It contains methods to be performed on sqlite database such as create, update, delete, select etc.

Methods of SQLiteDatabase class

There are many methods in SQLiteDatabase class. Some of them are as follows:

Method	Description
void execSQL(String sql)	executes the sql query not select query.
long insert(String table, String nullColumnHack, ContentValues values)	inserts a record on the database. The table specifies the table name, nullColumnHack doesn't allow completely null values. If second argument is null, android will store null values if values are empty. The third argument specifies the values to be stored.
int update(String table, ContentValues values, String whereClause, String[] whereArgs)	updates a row.
Cursor query(String table, String[] columns, String selection, String[] selectionArgs, String groupBy, String having, String orderBy)	returns a cursor over the resultset.

Example of android SQLite database

File: Contact.java

```
package example.javatpoint.com.sqlitetutorial;
public class Contact {
    int _id;
    String _name;
    String _phone_number;
    public Contact(){   }
    public Contact(int id, String name, String _phone_number){
        this._id = id;
        this._name = name;
        this._phone_number = _phone_number;
    }
```

9. Manipulation using SQLLite

```
// Creating Tables
   @Override
   public void onCreate(SQLiteDatabase db) {
      String CREATE_CONTACTS_TABLE = "CREATE TABLE " +
TABLE_CONTACTS + "("
            + KEY_ID + " INTEGER PRIMARY KEY," + KEY_NAME + " TEXT,"
            + KEY_PH_NO + " TEXT" + ")";
      db.execSQL(CREATE_CONTACTS_TABLE);

// Upgrading database
   @Override
   public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
      // Drop older table if existed
      db.execSQL("DROP TABLE IF EXISTS " + TABLE_CONTACTS);

      // Create tables again
      onCreate(db);
// Inserting Row
      db.insert(TABLE_CONTACTS, null, values);
      //2nd argument is String containing nullColumnHack
      db.close(); // Closing database connection
 // updating row
      return db.update(TABLE_CONTACTS, values, KEY_ID + " = ?",
            new String[] { String.valueOf(contact.getID()) });
```

10. SQLite Commands

SQLite commands are similar to SQL commands. There are three types of SQLite commands:

DDL: Data Definition Language

DML: Data Manipulation Language

DQL: Data Query Languag

Data Definition Language

There are three commands in this group:

CREATE: This command is used to create a table, a view of a table or other object in the database.

ALTER: It is used to modify an existing database object like a table.

DROP: The DROP command is used to delete an entire table, a view of a table or other object in the database.

Data Manipulation language

There are three commands in data manipulation language group:

INSERT: This command is used to create a record.

UPDATE: It is used to modify the records.

DELETE: It is used to delete records.

Data Query Language

SELECT: This command is used to retrieve certain records from one or more table.

SQLite dot Command

Following is a list of SQLite dot commands. These commands are not terminated by a semicolon (;).

.help command:

Check the list of dot commands by using the ".help" at anytime.