Programming in Android

Session: 7

Data Handling and Content Providers

Objectives

- Explain the process of saving and loading preferences
- Explain persistence of data to file
- Explain external storage
- Explain internal storage
- Explain SQLite Database
- Explain the process of storing and retrieving data from the database
- Explain Content Providers
- Make and use content providers
- Explain Resources and assets

Introduction

- Android provides different ways of saving persistent application data
- The appropriate option can be chosen based on the need of the developer
- Some of the data storage options are as follows:
  - Shared Preferences
  - Internal Storage
  - External Storage
  - SQLite Databases

Persisting Data to Shared Preference

- Shared Preferences are to save simple application data
- Data can be saved and retrieved through the use of key/value pairs
- It enables the developer to save primitive data types such as boolean, float, long, string, and so on
- The SharedPreferences class is present in the android.content package

Creating and Retrieving SharedPreference Object

- Following Code Snippet demonstrates an example for creating a SharedPreference variable:

```
public class PreferencesActivity extends Activity {
SharedPreferences prefs;
String prefName = "MyPref";
…
}
```

- Following Code Snippet demonstrates an example for retrieving the SharedPreference object:

```
prefs = getSharedPreferences(prefName, MODE_PRIVATE);
```

Saving and Committing Changes Using SharedPreference

- Following Code Snippet demonstrates an example for retrieving the SharedPreference Editor:

```
prefs = getSharedPreferences(prefName, MODE_PRIVATE);
SharedPreferences.Editor editor = prefs.edit();
```

- Following Code Snippet demonstrates an example for saving data using the SharedPreferences Editor:

```
...
//---save some values using the SharedPreferences object---
editor.putFloat("temperature", 85);
editor.putBoolean("authenticated", true);
editor.putString("username", "Wei-Meng Lee");
...
```

- Following Code Snippet demonstrates an example for committing changes using the SharedPreferences Editor:

```
…
//---saves the values---
editor.commit();
…
```

Retrieving Data Using SharedPreference Code Snippet

- Following Code Snippet demonstrates an example for retrieving data using the SharedPreferences Editor:

```
…
prefs = getSharedPreferences(prefName, MODE_PRIVATE);
float temperature = prefs.getFloat("temperature", 50);
boolean authenticated = prefs.getBoolean("authenticated", false);
String username = prefs.getString("username", "");
…
```

External Storage

- External signifies that it is outside
- All Android-compatible devices support a shared ‘external storage’ which can be used to save files
- This can be a removable storage media (SD card) or USB storage
- Files which are saved to the external storage are readable by everyone and can be modified by the user

Advantages and Disadvantages of External Storage

- **Advantages**
  - External storage will be extending the internal storage space of the Android device so that more apps can be stored
  - External Storage stores all the data in the SD card of a particular Android device
  - The file which is stored in the SD card can be retrieved and the data which is stored in that particular SD card can be used in some other device
- **Disadvantages**
  - If the developer has put a few set of apps on SD card of the Android device, then on removal of the SD card, the apps are no longer available for use until and unless the developer replaces back the SD card
  - It will take long time for your device to boot-up or shut down
  - The apps placed on the SD card require frequent update to support the functionality

Internal Storage

- Files are saved directly on to the device's internal storage
- Files which are saved to internal storage are private to that particular application
- Once the user uninstalls the application, the files are removed
- Accessed using java.io classes

Writing Data to Internal Storage

- Invoke openFileOutput() method with the file name and the operating mode. This returns an object of FileOutputStream class
- Write to the file by invoking the write() method
- Close the stream invoking the close() method
- The code to achieve this is shown in the following Code Snippet:

```
…
String FILENAME = "android_file_save";// saving into file
String string = "Hello Internal Storage";
FileOutputStream fos = context.openFileOutput(FILENAME, Context.MODE_PRIVATE);
// write I/O using write() and close()
fos.write(string.getBytes());
fos.close();
…
```

Advantages and Disadvantages of Internal Storage

- **Advantages**
  - In this type of storage, the developer will be storing the apps or the data directly on to your device
  - All the data and apps saved in the device will be present in the device only
- **Disadvantages**
  - When the device is very low on internal storage space, then Android might delete the cache files to recover space
  - Once the internal memory is full then the device will send an alert informing about memory full and incoming SMS will be rejected

SQLite Databases

- Android uses SQLite database to store relational data
- SQLite is a program provided by Android to execute services, such as creating tables and databases, amending rows, executing queries, and so on
- The SQLite database created for an Android application is private and cannot be accessed by other applications
- To create a database for your Android App, the developer needs to use the package android.database.sqlite

Creating an SQLite Database

- The developer needs to create a class which is a subclass of android.database.sqlite.SQLiteOpenHelper class
- This is because this class will enable the developer to create, open, close, and upgrade the database
- The developer needs to import android.database.sqliteSQLiteDataBase class
- Using this class instance and its properties the developer can execute the SQL statements to perform the CRUD operations such as create(insert), retrieve, update, and delete

Storing and Retrieving Data

- SQLite queries can be executed by using the query() method present in the SQLite database
- The query() method will return a Cursor object that points to all the retrieved rows
- It helps by caching the result of query
- It also provides functions with the help of which one can navigate to the desired row and obtain the data from the specified rows and columns

Adding, Modifying, Searching, and Removing Databases

- For Android, SQLite is ‘baked into’ the Android runtime
- Every Android database based application can create tables in SQLite databases
- Android provides classes for Database management in android.database and android.database.sqlite packages
- In this package the most important classes are as follows:
  - SQLiteOpenHelper
  - SQLiteDatabase
  - Cursor
  - SQLException

SQLiteOpenHelper 1-2

- The main functionality of the class is:
  - Open the database if it exists
  - Create it if it does not
  - Upgrade the version as required
- It provides a constructor to construct a helper class by subclassing this class and overriding the methods named onCreate() and onUpgrade()

SQLiteOpenHelper 2-2

Syntax:

```
SQLiteOpenHelper (Context context, String name,
SQLiteDatabase. CursorFactory factory, int version)
```

context – represent the context to create or open the database

name – represents the name of the database

factory – represents the factory class used for creating the cursor object

version – represent the number of the database

Methods

- **onCreate (SQLiteDatabase db)**
  The method is invoked when the database is created for the first time. This is where the table is created and populated. The database name is passed as an argument

- **onUpgrade (SQLiteDatabase db, int oldVersion, int newVersion)**
  The method is invoked when the database needs to be upgraded. The implementation should use this method to drop tables, add tables, or perform any other operations such as the need to upgrade to the new schema version

SQLiteDatabase 1-2

- The class has methods to create, delete, execute SQL commands, and other database management tasks

Methods

- **execSQL (String sql)**
  - Used to execute a single SQL statement that is not a SELECT statement or any other SQL statement that returns data and for creating tables. The sql statement to be executed is passed as an argument

- **long insert (String table, String nullColumnHack, ContentValues values)**
  - It is a convenience method used for inserting a row into the database

- **int update (String table, ContentValues values, String whereClause, String[ ] whereArgs)**
  - It is a convenience method used for updating rows in the database. The method returns the number of rows affected

SQLiteDatabase 2-2

Methods

- **int delete (String table, String whereClause, String[ ] whereArgs)**
  - It is a convenience method used for deleting rows in the database. The method returns the number of rows affected if a where clause is passed

- **Cursor query (String table, String[ ] columns, String selection, String[ ] selectionArgs, String groupBy, String having, String orderBy)**
  - The method queries the given table and returns a Cursor over the result set

Cursor 1-3

- This interface provides random read-write access methods to the result set returned by a database query
- Following table lists some of the methods present in Cursor class:

Methods	Description
`public int getColumnIndex(String colName)`	Returns the index of the indicated column as the result
`public String getColumnName(int colIndex)`	Returns the name of the specified column as the result
`public abstract String[] getColumnNames()`	Returns the column names as a string array
`public int getColumnIndexOrThrow (String colName)`	Returns the index of a column. The concept of throwing an exception arises when there is no existence of the column with the indicated name
`public abstract int getCount()`	Returns the count of number of rows
`public final int getPosition()`	Returns the present cursor position in the result set

Cursor 2-3

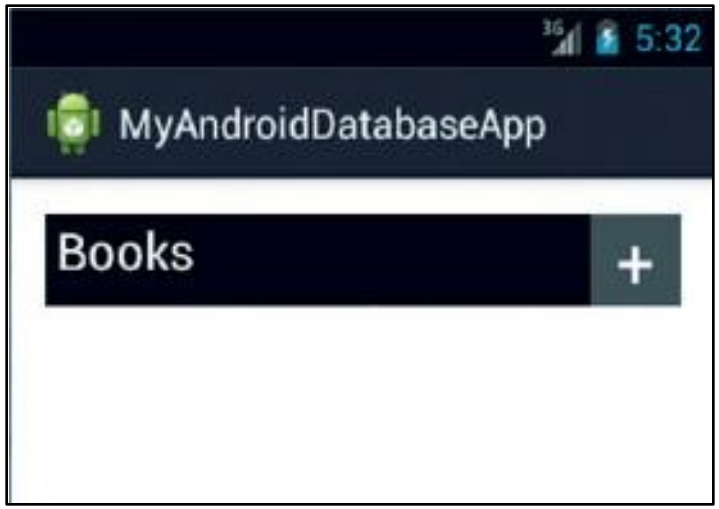
Methods	Description
`public final boolean moveToPosition (int pos)`	Moves the cursor towards the desired row in the result set and returns true, if the required record exists
`public final boolean moveToFirst()`	Moves the cursor to the first row in the queried result set and returns false, if there are no records in the result set
`public final boolean moveToNext()`	Moves the cursor to the subsequent rows in the result set and returns false, if the cursor is after the last record
`public final boolean moveToPrevious()`	Moves the cursor to the previous rows in the result set and returns false if the cursor is before the first record

Cursor 3-3

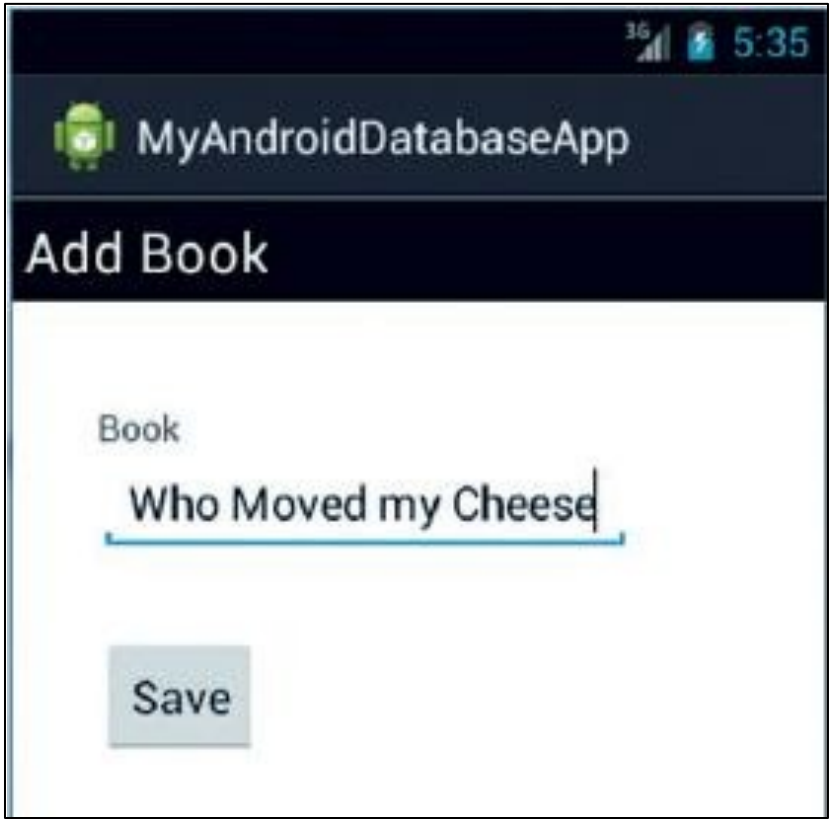
- The Cursor interface also has getXXX() style of methods to retrieve XXX type of values
- Some of the commonly used methods of this type are as follows:
  - getString(int columnindex)
  - getLong(int columnindex)
  - getInt(int columnindex)
  - getFloat(int columnindex)
  - getDouble(int columnindex)

SQLite Example Application 1-3

- Using the methods, an application for demonstrating usage of SQLite Databases is created as shown in the following figure:

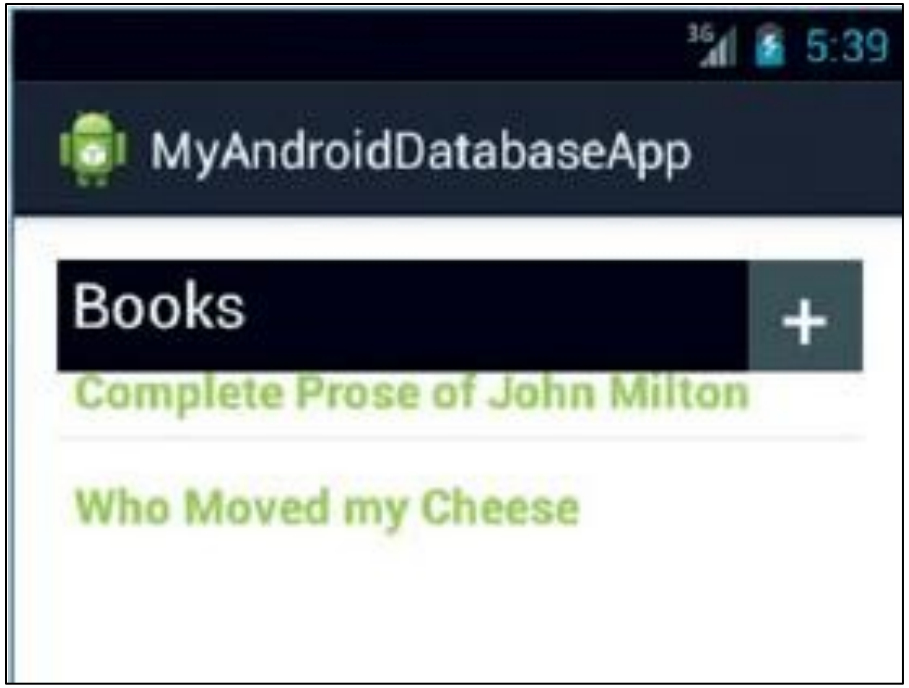


- Output when the plus symbol on the upper right corner is clicked as shown in the following figure:



SQLite Example Application 2-3

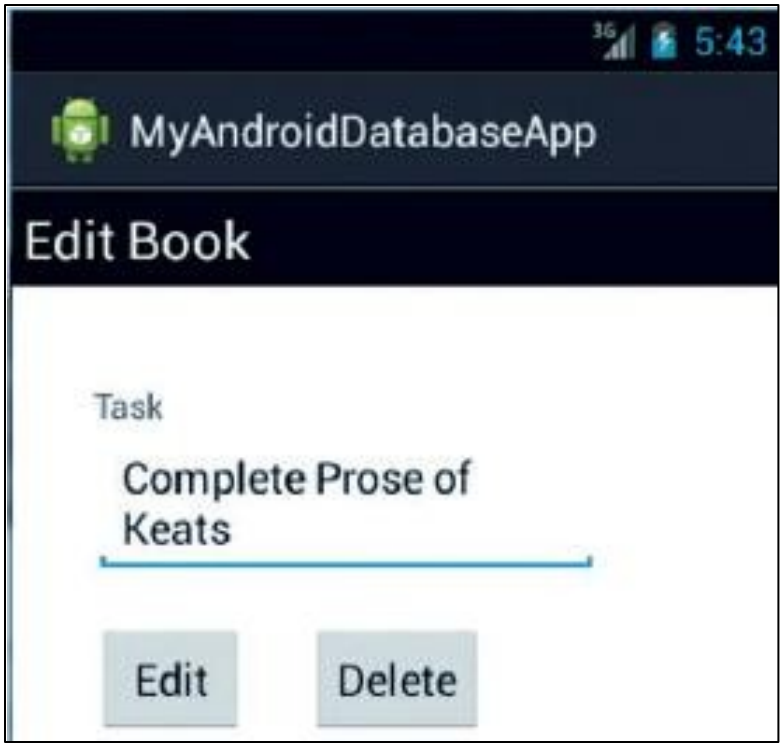
- Output when the user clicks Save and the book name gets added to the Book database as shown in the following figure:



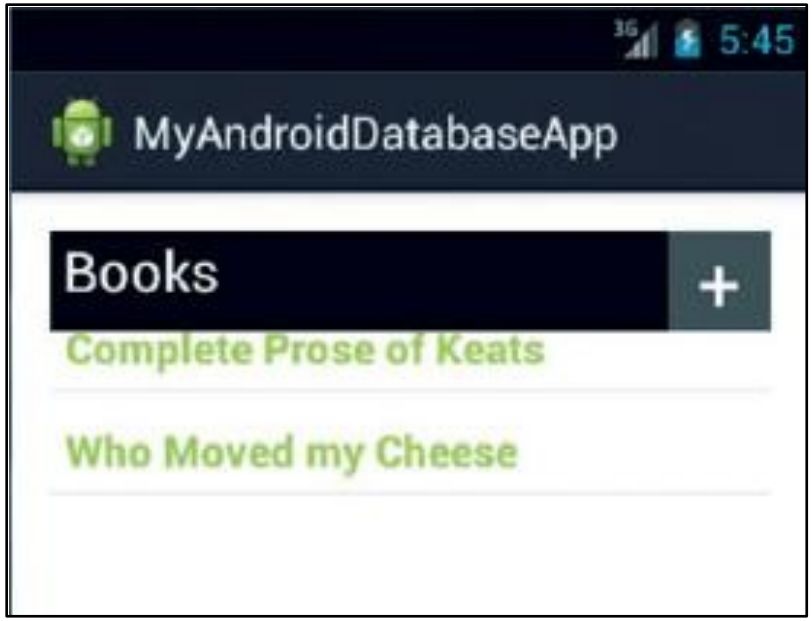
- Output when the book is selected to display the Edit Book pane as shown in the following figure:

SQLite Example Application 3-3

- Output after the book name has been edited in the Edit Book pane as shown in the following figure:



- Output when the user has clicked Edit as shown in the following figure:



Content Providers

- Content Providers provide access to data using a unified API
- Acts as a common gateway to applications for accessing the data
- Responsible for access management to the data
- Irrespective of the method the data is stored, data is almost always presented to the application in a table format
- A cursor is returned to navigate the data

Content URI

- Each content provider has a unique Content URI for the data associated with it
- This URI can be used by the applications to query data from the content provider
- An example content URI is shown in the following Code Snippet:

```
Content: //Domain/subdomain
```

Searching and Accessing Content Providers

- The first argument to the content provider is a URI to the content
- The second argument is the projection specification. It contains the list of the columns to retrieve from the table format
- The third argument is the selection statement. It specifies the selection criteria of the data. It is similar to the 'WHERE' clause in SQL or the 'IF' condition in Java
- The fourth optional argument is the arguments for the selection statement. This is used if placeholders are used in the selection statement. It can be null otherwise
- The final optional argument is sort order specification. It decides how the retrieved content is sorted before being presented to the application

Accessing Content Providers Example

- Following Code Snippet demonstrates an example for Searching and Accessing data from the MediaStore Content Provider as an example:

```
String selection = MediaStore.Audio.Media.IS_MUSIC + " != 0";

String[] projection = {
        MediaStore.Audio.Media._ID,
        MediaStore.Audio.Media.ARTIST,
        MediaStore.Audio.Media.TITLE,
        MediaStore.Audio.Media.DATA,
        MediaStore.Audio.Media.DISPLAY_NAME,
        MediaStore.Audio.Media.DURATION,
        MediaStore.Audio.Media.ALBUM_ID,
};

Cursor videoCursor = getContentResolver().query(
MediaStore.Video.Media.EXTERNAL_CONTENT_URI,projection,selection,   null,
null);
```

Adding, Modifying, and Removing Content

- **Inserting Content**:
  - The insert() method of the content resolver can be used
  - Data needs to be added as a key value pair
- **Updating Content**:
  - The Update() method of the content resolver can be used
  - For updating data a ContentValues object is created with the new values of the data
- **Removing Content**:
  - The delete() method of the content resolver can be used
  - The method accepts a URI along with the selection criteria of the entries to be deleted and the placeholder's values if required

Adding, Modifying, and Removing Content

- Following Code Snippet demonstrates an example for adding, modifying, and removing content:

```
//Insert
ContentValues insertValues = new ContentValues();
insertValues.put(CUSTOM.Domain.Column_name,”column value”);
…
Uri contentURI = uriBuilder.build();
getContentResolver().insert(uri,insertValues);
…

//Update
…
ContentValues updatedValues = new ContentValues();
updatedValues.put(CUSTOM.Domain.Column_name,”column value”);
…

String selectionCriteria = CUSTOM.Domain.ID + “ = 3 AND ”+CUSTOM.Domain.NAME+” = ?”;
String[] placeHolderValues ={“customer 3”};

getContentResolver().update(uri,updatedValues,selectionCriteria,placeHolderValues);
```

```
//Delete
String selectionCriteria = CUSTOM.Domain.ID + “ = 4 “;
getContentResolver().delete(uri,selectionCriteria,null);
// No placeholder values. Hence last argument is null
```

Using Native Content Providers

- Android provides several content providers to access from and add content to
- The native content providers of Android can be utilized by importing the package android.provider
- The set of native content providers are as follows:
  - Calendar
  - Contact
  - Media Store
  - Openable Columns
  - Sync States
  - Telephony




When to Create Content Providers?

- A content provider can be created when:
  - You want to provide complex data or files to other applications
  - To provide search suggestions from the application to the search feature of Android
- A content provider should NOT be created when:
  - Providing access to databases
  - To act as a mode of communication between applications

Considerations When Creating Content Providers

- Data Storage
  - Files
  - SQLite Databases
- Content URI
  - Authority
  - Path Structure
  - Content Id
- URI Patterns
- Manifest Changes

Implementing Custom Content Providers

- A concrete class extending the ContentProvider class needs to be created
- The following methods need to be overridden and implemented:
  - onCreate()
  - query()
  - update()
  - delete()
  - insert()
  - getType()

Content Provider MIME Types

- MIME is the abbreviation for Multipurpose Internet Mail Extensions
- MIME was added as an extension to the e-mail protocol
- Android supports the ability to return MIME media types or MIME message content from content providers
- Allows for handling data which cannot be handled using the table format
- Android also provides the ability to define custom MIME types

Resources

- **Resource Storage**:
  - Resources are not stored on the Android File System
  - They are packaged as part of the application itself
  - They cannot be accessed with the standard File class
  - Any sensitive or copyrighted content is advised to be encrypted

- **Locating Resources**:
  - The standard format for a resource identifier is shown in the following Code Snippet:

```
[<package_name>.]R.<resource_type>.<resource_name>
```

Procedure for Locating Resources

- Android follows the following procedure to locate Resources:
  - Locate the package that is being referred. If it is not mentioned, the application name is used such as com.aptech.app_name to locate the apk holding the resource
  - Identify the resource type being referred to
  - List the directories capable of holding the resource type
  - Eliminate the types not supported by the device. For example, depending on the resolution, the drawable-mdpi could be eliminated
  - Repeat steps 2-4 until all the qualifiers are eliminated, such as language, key support, and so on until a maximum of one directory remains
  - Search the directory for a file name matching the resource name without the extension
  - Return an integer identifier to the resource

Handling Runtime Configuration Changes

- Certain device types have a changing configuration during run time such as media centre devices running Android
- The developer may have intended the application to use a different set of resources for this changed configuration
- Developer can either handle the changes manually in code or let Android auto configure the application automatically
- To manually handle Configuration changes is shown in the following Code Snippet:

```
@Override
public void onConfigurationChanged(Configuration newConfig) {
    super.onConfigurationChanged(newConfig);

if (newConfig.orientation == Configuration.ORIENTATION_LANDSCAPE){
        setContentView(R.layout.horizontal_layout);     }
else{
       setContentView(R.layout.vertical_layout);   }
}
```

Reading Raw Data from Resources

- It is possible to read the raw data from a resource
- Assets are a better alternative for this purpose
- If resources are being used, then the developer is recommended to create a new directory called 'raw' in the /res directory
- The code to retrieve raw data from a resource with the file name picture.jpg stored in the raw directory is shown in the following Code Snippet:

```
InputStream iS = getResources().openRawResource("picture");
```

Assets 1-2

- Assets are functionally similar to resources, but serve a different purpose
- Assets can hold any type of data
- Assets are not given a resource id
- They are directly identified by their file names
- Assets are stored in their own directory inside the package apk
- No changes are made and the files in the asset folder are stored as is
- The AssetManager class is used to access the assets
- The API is similar to the File class API

Assets 2-2

- Application of Assets in Realtime Applications:
  - Store textures
  - Store models
  - Store blob data
  - Other pre-compiled data in games
- If the size exceeds beyond 100 MB, it is preferable to use external storage to store these assets
- The code to open an asset called image.png is shown in the following Code Snippet:

```
InputStream iS = context.getAssets().open("image.png");
```

Summary

- The different types of storages available in Android are Shared Preferences, Internal Storage, External Storage, and SQLite Database
- Android uses SQLite database to store relational data. SQLite queries can be executed by using the query() method present in the SQLiteDatabase. The query() method will return a Cursor object that points to all the retrieved rows
- Content Providers in Android act as resources to information
- Content Providers can be accessed using unique URIs
- Custom Content Providers can be created by extending the ContentProvider class
- Resources and assets are used to store static content for the application
- Assets do not have a resource id generated and they are packaged as-is