Royal University of Bhutan

Unit II:
Services and Storage

CTE308- AS2025

Tutor: Pema Galey

#17682761

OUTLINES

- Services for long tasks
- IntentService
- Other Background tasks (self-explanatory)

Introduction to Services

- **Services is an Advanced topic**
  - Services are complex
  - Many ways of configuring a service
  - This lesson has introductory information only
  - Explore and learn for yourself if you want to use services

Services for Long Tasks

- **What is a service?**
  - A service is an application component that can perform long-running operations in the background and does not provide a user interface.
  - Foreground vs background

Services

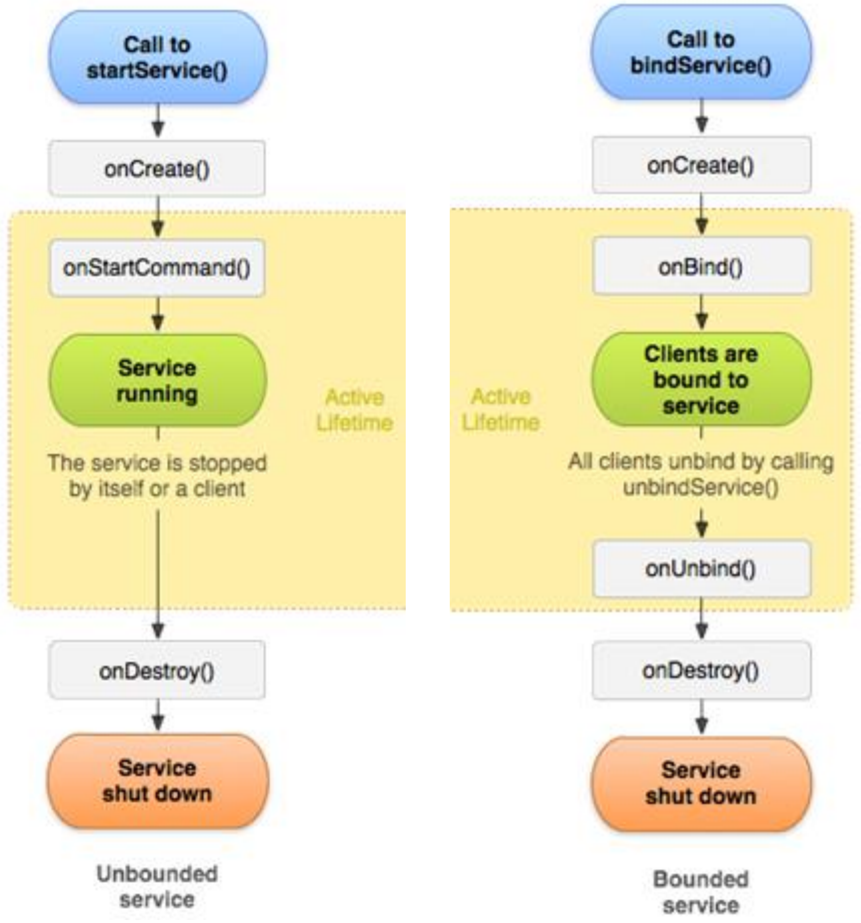
- **Services**
  - Network transactions
  - Play music
  - Perform file I/O
  - Interact with a content provider

Characteristics of Services

- Started with an Intent
- Can stay running when user switches applications
- Lifecycle—which you must manage
- Other apps can use the service—manage permissions
- Runs in the main thread of its hosting process

Forms of Services

- **Service Started**
  - started with startService()
  - Runs indefinitely until it stops itself
  - Usually does not update the UI



- **Service Bound**
  - offers a client-server interface with the service
  - clients send requests and get results
  - started with bindService()
  - Ends when all clients unbind

Services for Long Task

- **Services and Thread**
  - Although services are separated from the UI, they still run on the main thread by default (except IntentService)
  - Offload CPU-intensive work to a separate thread within the service
- **Updating the app**
  - If the service can't access the UI, how do you update the app to show the results?
  - Use a **broadcast receiver**!

Foreground services

- Runs in the background but requires that the user is actively aware it exists—e.g. music player using music service
  - Higher priority than background services since user will notice its absence—unlikely to be killed by the system
  - Must provide a notification which the user cannot dismiss while the service is running

Creating a service

- <service android:name=".ExampleService" />
- Manage permissions
- Subclass **IntentService** or **Service** class
- Implement lifecycle methods
- Start service from activity
- Make sure service is stoppable

Stopping a service

- A started service must manage its own lifecycle
- If not stopped, will keep running and consuming resources
- The service must stop itself by calling stopSelf()
- Another component can stop it by calling stopService()
- Bound service is destroyed when all clients unbound
- IntentService is destroyed after onHandleIntent() returns

IntentService

- **IntentService**
  - Simple service with simplified lifecycle
  - Uses worker threads to fulfill requests
  - Stops itself when done
  - Ideal for one long task on a single background thread

- **Limitation:**
  - cannot interact with the UI
  - can only run one request at a time
  - cannot be interrupted

IntentService Implementation

```
public class HelloIntentService extends IntentService {
    public HelloIntentService() { super("HelloIntentService");}
    @Override
  protected void onHandleIntent(Intent intent) {
       try {
       // Do some work
       } catch (InterruptedException e) {
       Thread.currentThread().interrupt();
       }
  } // When this method returns, IntentService stops the service, as appropriate.
}
```

Other Background Tasks

- Notification
- Alarm
- Job Scheduler

Storing Data

Outline for Storage

- Introduction
- Storage Types
- File Systems
- SharedPreferences

Introduction to Storage

- **Data Storage Option**

1. **Shared Preferences**
   - Private primitive data in key-value pairs.
2. **Internal Storage**
   - Private data on device memory
3. **External Storage**
   - Public data on device or external storage
4. **SQLite Databases**
   - Structured data in a private database
5. **Content Providers**
   - Store privately and make available publicly

Storing Data Beyond Android

- **Network Connection**
  - On the web with your own server.
- **Cloud Backup**
  - Back up app and user data in the cloud.
- **Firebase Real time Database**
  - Store and sync data with NoSQL cloud database across clients in real time.

Android File System

- All Android storage have files system to read and write with file API.
- Android devices have two file storage area:
  - **Internal Storage**
  - **External Storage**

Android Device File Storage Area

- **Internal Storage**
  - Most devices offer built in non-volatile memory.
  - Private Directories;
  - Always available.
  - Uses device's file system.
  - Only your app can access files, unless explicitly set to be readable or writable.
  - On app uninstall, system removes all app's files from internal storage.

Android Device File Storage Area

- **External Storage**
  - Removable storage medium such as micro SD Card.
  - Public Directories.
  - Not always available, can be removed.
  - Uses device's file system or physically external storage like SD card.
  - World-readable, so any app can read.
  - On uninstall, system does not remove files private to app.

When to use Internal/External Storage?

- Internal is best when you want to be sure that neither the user nor other apps can access your files.
- External is best for files that:

(a)don’t require access restrictions,

(b)you want to share with other apps, and

(c)allow the user to access with a computer.

More insight of Internal Storage

- Uses private directories just for your app
- App always has permission to read/write
- Permanent storage directory—getFilesDir()
- Temporary storage directory—getCacheDir()

```
File file=new File (context.getFilesDir(), filename)
```

- Use standard java.io file operators or streams to interact with files.

Shared Preferences

- **What is Shared Preferences (SP)?**

- Simplest mechanism to store data in an Android.
- Don’t have to worry about creating files and using file API
- Need to create XML file and Data will be managed automatically
  - Store data into SP and get the data from SP.

Shared Preferences Operations

- Read and write small amounts of primitive data as key/value pairs to a file on the device storage.
- Data is stored in XML file in the directory data/data/package_name/shared_pref folder.
- No need to worry about creating XML file, setting permission and accessing, everything managed by the system at the time of creating XML.
    - Therefore, it is the simplest

Key	Value
Name	Pema
password	abc123
location	CST
Department	IT

What data store in Shared Preferences?

- Primitive data
  - Boolean
  - Integer
  - String
  - String array
- If you want to store like table and order data with id, use database.
- Shared Preferences are not for such data and large data.

How to Access Shared Preferences?

- If you have only one preference file, call
  - `getPreferences (int mode)`
- If you have several files, call
  - `getSharedPreferences(String name, int mode)`

  - **Mode:**
    - **MODE_PRIVATE**: Only your app can access the file
    - **MODE_WORLD_READABLE**: All apps can read the file
    - **MODE_WORLD_WRITABLE**: All apps can write to the file
    - **MODE_MULTI_PROCESS**: Multiple processes can modify the same shared preferences file

Uses of SharedPreferences

- **First time:** Check if user is using your app.
- **Updates:** Check when your app was last updated.
- **Credentials:** Remember user details.
- **Settings:** Remember using settings.
- **Location:** location in the app.

Shared Preferences

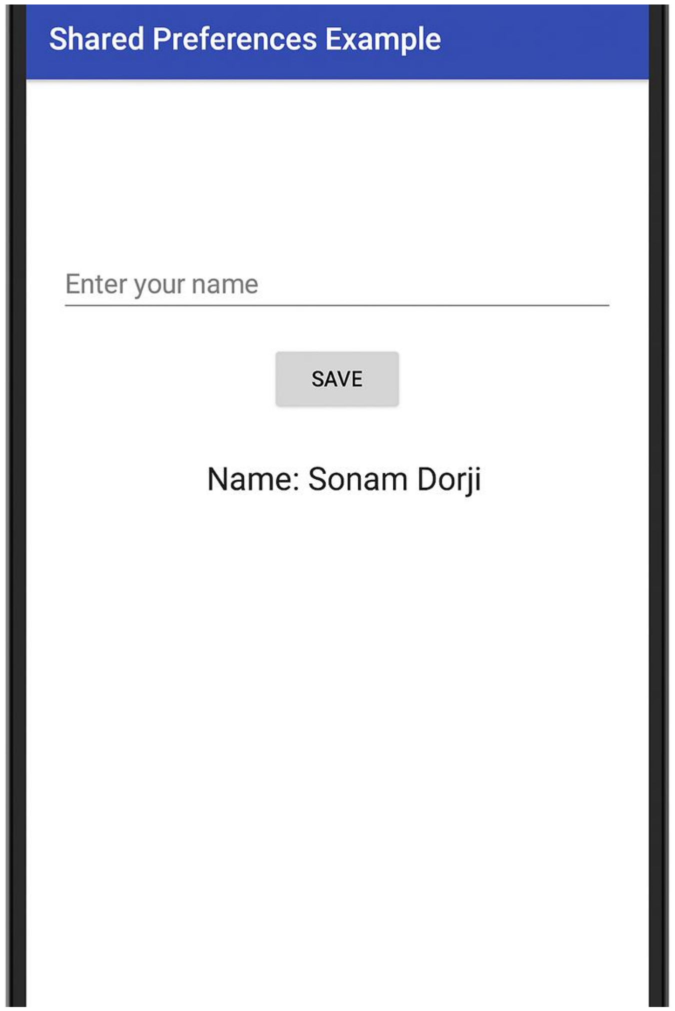
```
import android.content.Context;
import android.content.SharedPreferences;
public class PreferenceHelper {
    private static final String PREF_NAME = "UserPrefs";
    private static final String KEY_USERNAME = "username";
    private static final String KEY_PASSWORD = "password";

    private SharedPreferences sharedPreferences;
    private SharedPreferences.Editor editor;

    public PreferenceHelper(
   Context contexsharedPreferences =
context.getSharedPreferences(PREF_NAME, Context.MODE_PRIVATE);
        editor = sharedPreferences.edit();
    }
    // Save user data
    public void saveUser(String username, String password) {
        editor.putString(KEY_USERNAME, username);
        editor.putString(KEY_PASSWORD, password);
        editor.apply(); // or commit()
    }
    // Retrieve username
    public String getUsername() {
        return sharedPreferences.getString(KEY_USERNAME, null);
    }
    // Retrieve password
    public String getPassword() {
        return sharedPreferences.getString(KEY_PASSWORD, null);
    }
    // Clear all data
    public void clearUser() {
        editor.clear();
        editor.apply();
    }
}
```

Class Work

- Design a simple Android application that uses **SharedPreferences** to store and retrieve a user's name.
- **Requirements:**
  - Create an activity with the following UI elements:
    - An EditText for entering the user's name.
    - A Button to **save** the name into SharedPreferences.
    - A TextView to **display** the saved name.
  - When the app is reopened, the last saved name should be automatically displayed in the TextView.
  - Use **SharedPreferences** methods (*putString()* and *getString()*) to implement this functionality.
- Replicate this concept for the Note-Taking application



Thank you!