Royal University of Bhutan

Unit II:
AsyncTask

CTE308- AS2025

Tutor: Pema Galey
#17682761

OUTLINES

- Background Operations
- Threads
- AsyncTask
- Loader and AsysnTaskLoader
- Connection to the Internet
- Boradcast Receivers
- Services
- Notification
- Alarm Manager
- Job Scheduler

Introduction: Background Operations

- We have a good idea about user controls, UI Design, how we can transfer between different activities.
- Background Task deals with long running processes, which takes too long to run.
- It can crash your application or activity if it takes too long to run your application or fetch data.
- How can we deal with these kind of task?
  - Answer is Async Task

AsyncTask & Thread

Async Task

- ✓ AsyncTask enables proper and easy use of the **UI thread**.
- ✓ This class allows you to perform background operations and publish results on the UI thread without having to manipulate threads and/or handlers.

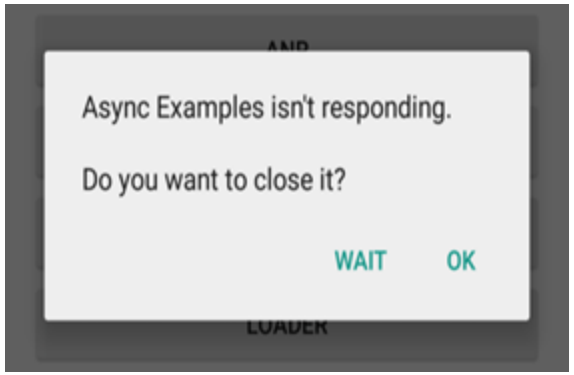
- ✓ **Thread:** Single unit task doing particular job.
- ✓ Single Thread vs Multiple Threads

The main Thread:

- Independent path of execution in a running program.
- Code is executed line by line.
- App runs on Java thread called "main" or "UI thread"
- Draws UI on the screen
- Responds to user actions by handling UI events

The Main Thread must be Fast

- Hardware updates screen every 16 milliseconds
- UI thread has 16 ms to do all its work
- If it takes too long, app stutters or hangs by generating ANR(Application Not Responding) dialog by the framework



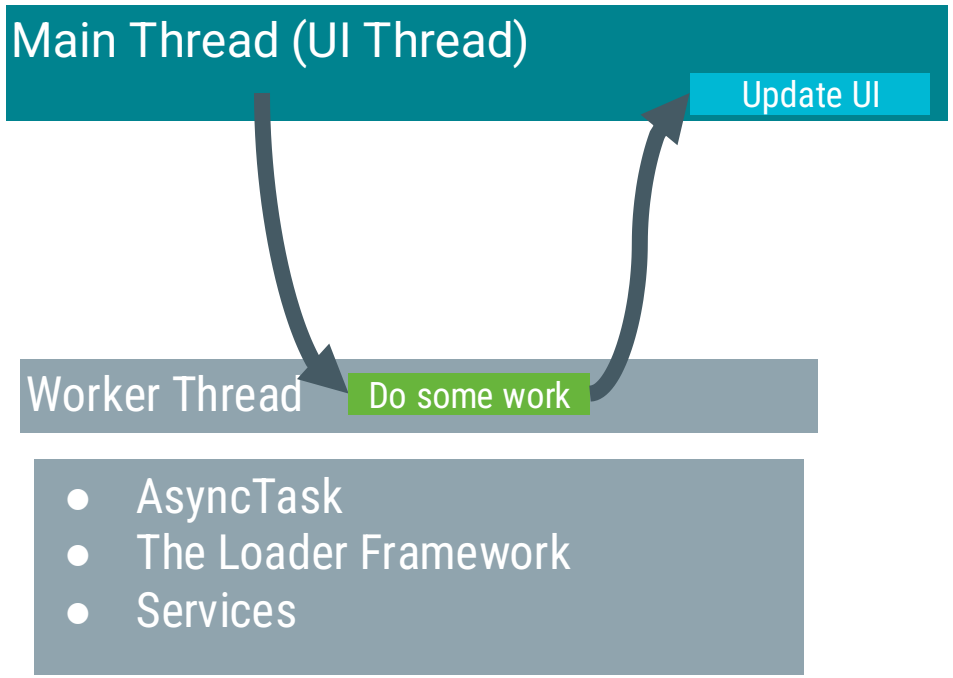
What is long running Tasks?

- **What is long running tasks?**
  - Network operations
  - Long calculations
  - Downloading/uploading files
  - Processing images
  - Loading data

- **Note:** Do not put the above tasks in UI Thread.

Background Task

- Execute long running tasks on a background thread



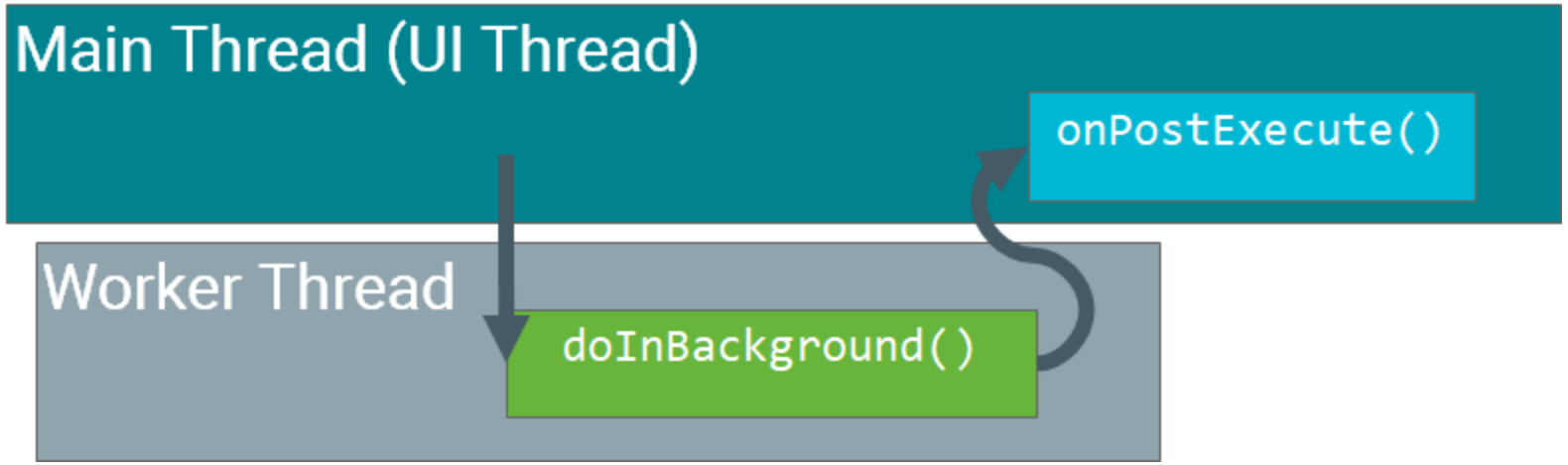
Two rules of Android Threads

- Do not block the UI thread
  - Complete all work in less than 16 ms for each screen
  - Run slow non-UI work on a non-UI thread
- Do not access the Android UI toolkit from outside the UI thread
  - Do UI work only on the UI thread

AsyncTask

What is AsyncTask?

- Class to implement basic background tasks.
- Implements override method 'doInBackground()' and 'onPostExecute()'



Override Two Methods

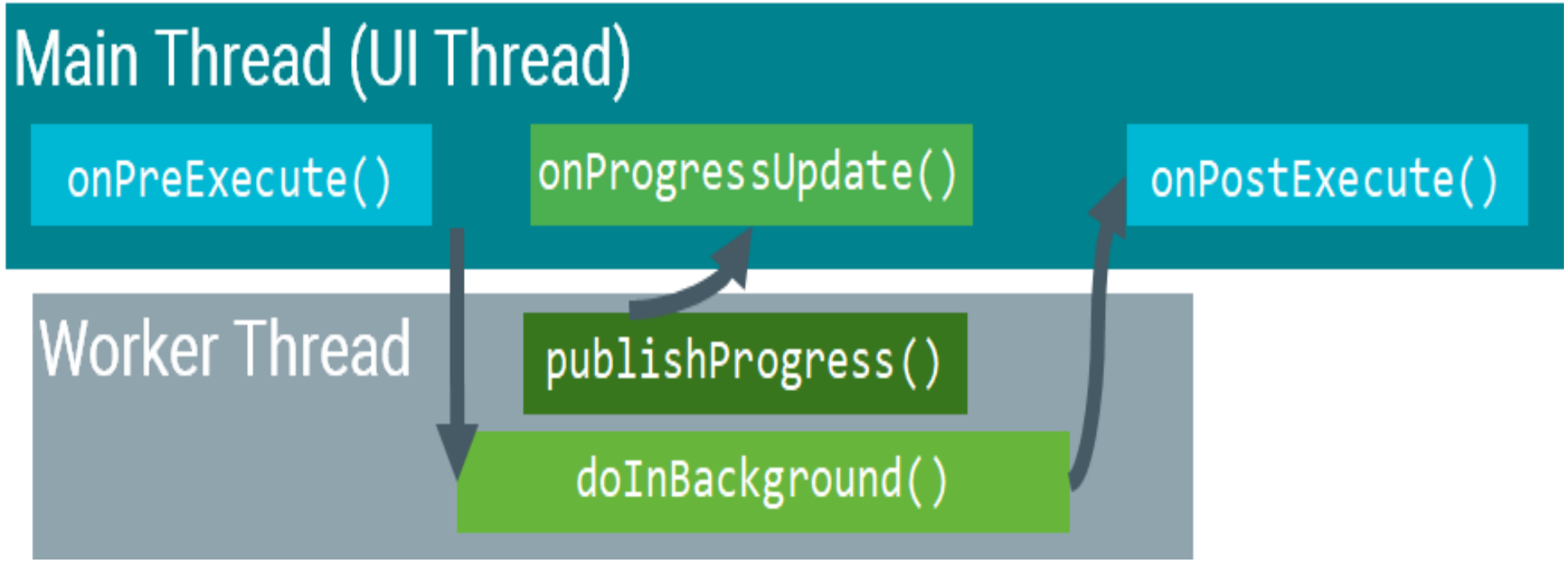
- `doInBackground()`—runs on a background thread
  - All the work to happen in the background

- `onPostExecute()`—runs on main thread when work done
  - Process results
  - Publish results to the UI

AsyncTask Helper methods

- `onPreExecute()`—
  - runs on a main thread
  - Sets up the task
- `onProgressUpdate()`
  - runs on main thread
  - receives calls from publishProgress() from background thread.

AsyncTask Helper methods



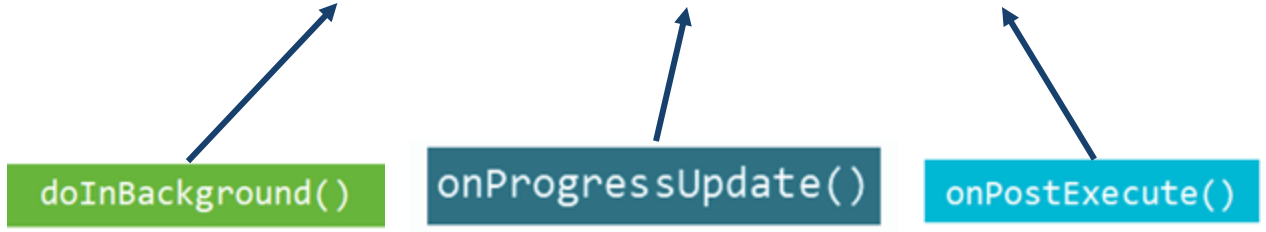
Creating AsyncTask

1) Subclass AsyncTask
2) Provide data type sent to `doInBackground()`
3) Provide data type of progress units for `onProgressUpdate()`
4) Provide data type of result for `onPostExecute()`

```
private class MyAsyncTask
        extends AsyncTask<URL, Integer, Bitmap> {...}
```

MyAsyncTask class definition

```
private class MyAsyncTask
        extends AsyncTask<String, Integer, Bitmap> {...}
```



- ➔ **`String`**—could be query, URI for filename
- ➔ **`Integer`**—percentage completed, steps done
- ➔ **`Bitmap`**—an image to be displayed
- ➔ *Use **Void** if no data passed*

Override Methods Implementation

- `onPreExecute()`

```
protected void onPreExecute() {
    // display a progress bar
    // show a toast
}
```

- `doInBackground()`

```
protected Bitmap doInBackground(String... query) {
    // Get the bitmap
    return bitmap;
}
```

Override Methods Implementation

- `onProgressUpdate()`

```
protected void onProgressUpdate(Integer... progress) {
        setProgressPercent(progress[0]);
    }
```

- `onPostExecute()`

```
protected void onPostExecute(Bitmap result) {
  // Do something with the bitmap
}
```

Start Background work

```
public void loadImage (View view) {
    String query = mEditText.getText().toString();
    new MyAsyncTask(query).execute();
 }
```

Limitations of AsyncTask

- When device configuration changes, Activity is destroyed
- AsyncTask cannot connect to Activity anymore
- New AsyncTask created for every config change
- Old AsyncTasks stay around
- App may run out of memory or crash

When to use AsyncTask

- Short or interruptible tasks
- Tasks that do not need to report back to UI or user
- Lower priority tasks that can be left unfinished
- Use AsyncTaskLoader otherwise

Limitation of AsyncTask

- This class was **deprecated** in API level 30. Use the standard ***java.util.concurrent***
- **AsyncTask** is designed to be a helper class around Thread and Handler and does not constitute a generic threading framework.
- **AsyncTasks** should ideally be used for short operations
- If you need to keep threads running for long periods of time, it is highly recommended you use the various APIs provided by the ***java.util.concurrent*** package

Loaders

- Provides asynchronous loading of data
- Reconnects to Activity after configuration change
- Can monitor changes in data source and deliver new data
- Callbacks implemented in Activity
- Many types of loaders available (AsyncTaskLoader, CursorLoader)

Why use Loader?

- Execute tasks OFF the UI thread
- LoaderManager handles configuration changes for you
- Efficiently implemented by the framework
- Users don't have to wait for data to load

Why use LoaderManager?

- Manages loader functions via callbacks
- Can manage multiple loaders
  - loader for database data, for AsyncTask data, for internet data…

- **For more on Loader refer:**

- AsyncTaskLoader Reference
- LoaderManager Reference

Connect to the Internet

- Steps to connect to the Internet
    1. Add permissions to Android Manifest
    2. Check Network Connection
    3. Create Worker Thread
    4. Implement background task
        a. Create URI
        b. Make HTTP Connection
        c. Connect and GET Data
    a. Process results
        a. Parse Results

Connect to the Internet

- Internet Connection (Permission)
  - Adding permission to AndroidManifest.xml

- **Internet**

```
<uses-permission android:name="android.permission.INTERNET"/>
```

- **Check Network State**

```
<uses-permission
   android:name="android.permission.ACCESS_NETWORK_STATE"/>
```

Manage Network Connection

- ConnectivityManager
  - Answers queries about the state of network connectivity
  - Notifies applications when network connectivity changes

- NetworkInfo
  - Describes status of a network interface of a given type
  - Mobile or Wi-Fi

Check if network is available

```
ConnectivityManager connMgr = (ConnectivityManager)
            getSystemService(Context.CONNECTIVITY_SERVICE);
NetworkInfo networkInfo = connMgr.getActiveNetworkInfo();
if (networkInfo != null && networkInfo.isConnected()) {
    // Create background thread to connect and get data
    new DownloadWebpageTask().execute(stringUrl);
} else {
    textView.setText("No network connection available.");
}
```

Check for Wi-Fi & Mobile

```
NetworkInfo networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_WIFI);

boolean isWifiConn = networkInfo.isConnected();


networkInfo = connMgr.getNetworkInfo(ConnectivityManager.TYPE_MOBILE);

boolean isMobileConn = networkInfo.isConnected();
```

Broadcast Receiver

- Some Android application would do anything until a certain types of message is broadcasted by android OS or any other application.
- Suppose you want your application to react in some way when system events occurs.
  - Example, you may want to play music when ear phone is connected and stop when it is disconnected.
- **How can your app tell when this event occurs?**

Broadcast Receiver

- **How can your app tell when this event occurs?**
  - For this we have Broadcast Receiver.
- Broadcast Intent and Broadcast receiver helps to send and receive certain events.
- Use implicit intents to send broadcasts or start activities

Broadcast Receiver

- Listens for incoming intents sent by sendBroadcast()
  - In the background
- Intents can be sent
  - By the system, when an event occurs that might change the behavior of an app
  - By another application, including your own

Broadcast Receiver

- Broadcast receiver always responds.
  - ❖ Responds even when your app is closed.
  - ❖ Independent from any activity.
  - ❖ When a broadcast intent is received and delivered to onReceive(), it has 5 seconds to execute, and then the receiver is destroyed.

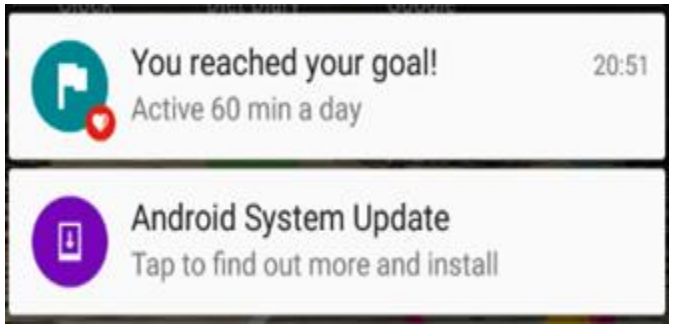
Services

- 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.

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

Notification

- What are Notifications?
- Message displayed to user outside regular app UI
  - Small icon
  - Title
  - Detail text



Alarm

What is an Alarm in Android?

- It is not an actual alarm clock.
- Schedules something to happen at a set time.
- Fire intents at set times or intervals.
- Goes off once or recurring.
- Can be based on a real-time clock or elapsed time.

Job Scheduler

Job Scheduler can we transfer data more efficiently.

- Used for intelligent scheduling of background tasks.
  - Based on conditions, not a time schedule.
- Much more efficient than AlarmManager
- Batches tasks together to minimize battery drain.
- API 21+ (no support library)

Job Scheduler Components

- Different Components needed for Scheduling Job.
  1. JobService: Service class where the task is initiated
  2. JobInfo: Builder pattern to set the conditions for the task
  3. JobScheduler: Schedule and cancel tasks, launch service

- Any Questions?

Class Work

- Develop a **Weather Forecast App** that provides users with real-time weather updates for their current location and allows them to check the weather in other cities. The app should be designed to meet the following requirements:
  - **Responsive Design:** The app should be fully responsive
  - **Background Task Handling:** Implement a background service that periodically fetches weather updates from a weather **API** (e.g., OpenWeatherMap) even when the app is not actively running.
  - **Push Notifications:** Set up push notifications to alert users of significant weather changes, such as temperature drops, storms, or extreme weather conditions.
  - Try to integrate with Firebase Cloud Messaging (FCM) to handle the push notifications, ensuring users receive timely alerts even when the app is not open.

Thank you!