NAME: SAKSHI VINAYAK KITTURKAR
CSU ID: 2860273
HOMEWORK 10

Modify the Services app:
Use a ProgressBar view instead/in addition to the Toast to display the progress made by the service.

= I have built an app which uses a ProgressBar view and shows use what is the progress.

In layout: MainActivity.java this code demonstrates a simple Android application with a MainActivity that communicates with a background service (MyService) using broadcasts to update a progress bar in real-time. The progress bar is updated based on the progress value received from the service.

Defines a BroadcastReceiver named progressReceiver that listens for broadcasts with the action "progress_update." When it receives a broadcast, it extracts the "progress" extra from the intent and updates the ProgressBar with the received progress value.

```
private BroadcastReceiver progressReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if ("progress_update".equals(intent.getAction())) {
            int progress = intent.getIntExtra("progress", 0);
            // Update ProgressBar with the received progress value
            progressBar.setProgress(progress);
        }
    }
};
```

Sets the content view to the layout defined in activity_main.xml.
Initializes the ProgressBar and sets its progress to 0.
Registers the progressReceiver with a LocalBroadcastManager to listen for broadcasts with the action "progress_update."

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);
```

```
    progress = 0;
    progressBar = (ProgressBar) findViewById(R.id.progressbar);

    IntentFilter filter = new IntentFilter("progress_update");
    LocalBroadcastManager.getInstance(this).registerReceiver(progressReceiver, filter);
}
```

startService: Starts a background service (MyService) when a button is clicked.
stopService: Stops the background service when another button is clicked.

```
public void startService(View view) {
    startService(new Intent(getBaseContext(), MyService.class));
}

public void stopService(View view) {
    stopService(new Intent(getBaseContext(), MyService.class));
}
```

In layout: MyService This code represents an Android Service named MyService that performs a background task of downloading files from specified URLs.

Defines a method doSomethingRepeatedly that schedules a task (incrementing a counter and logging the result) to run at fixed intervals using the Timer.

```
private void doSomethingRepeatedly() {
    timer.scheduleAtFixedRate(new TimerTask() {
        public void run() {
            Log.d("MyService", String.valueOf(++counter));
        }
    }, 0, UPDATE_INTERVAL);
}
```

Declares an inner class DoBackgroundTask that extends AsyncTask. This class is responsible for downloading files in the background.
doInBackground: Performs the file download in the background, updating progress and returning the total bytes downloaded.
onProgressUpdate: Updates the progress of the download, sends a local broadcast with the progress, and displays a toast message.
onPostExecute: Displays a toast message with the total bytes downloaded.

```
private class DoBackgroundTask extends AsyncTask<URL, Integer, Long> {
    // ...
}
```

Simulates the file download by introducing a delay (5 seconds) and returning an arbitrary value representing the size of the downloaded file.

```
private int DownloadFile(URL url) {
    // Simulates taking some time to download a file
    try {
        Thread.sleep(5000);
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
    // Returns an arbitrary number representing the size of the file downloaded
    return 100;
}
```

Overrides the onStartCommand method, which is called when the service is started. It initializes the background task and starts file downloads.

```
@Override
public int onStartCommand(Intent intent, int flags, int startId) {
    // ...
}
```

In layout: ExampleInstrumentedTest.java this code defines a simple instrumented test for an Android application. The test checks whether the package name of the app under test matches the expected package name and serves as an example of how to set up a basic instrumented test in Android.

Retrieves the context of the app under test using InstrumentationRegistry. It then checks if the package name of the app matches the expected package name ("com.example.hw10") using the assertEquals method.
This test essentially verifies that the application's package name matches the expected package name. It's a basic test to ensure that the app is correctly configured with the expected package name.

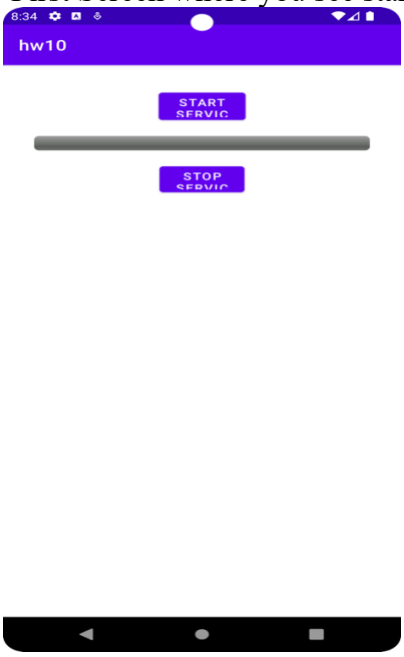
```
// Context of the app under test.
Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
assertEquals("com.example.hw10", appContext.getPackageName());
```

In layout: ExampleUnitTest.java this code defines a simple local unit test for an Android application. The test checks whether the addition operation produces the expected result. Local unit tests are typically executed on the development machine (host) and are independent of the Android runtime environment. This example serves as a basic illustration of how to set up a local unit test in Android using JUnit.

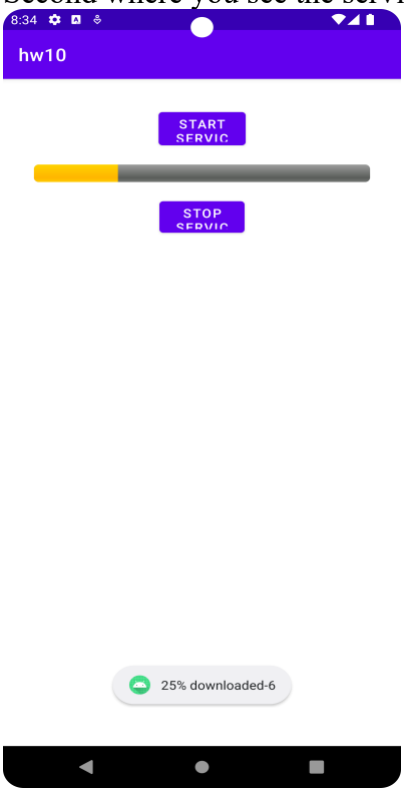
Declares a test method named addition_isCorrect. This method will contain the actual test logic for verifying the correctness of addition.

```
@Test
public void addition_isCorrect() {
    // ...
}
```

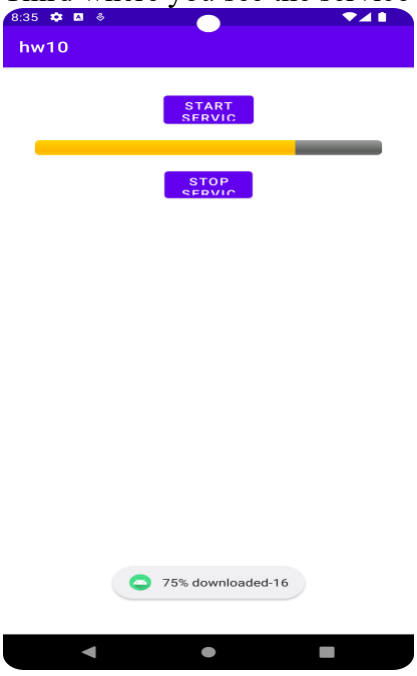
First Screen where you see start service and stop service and a ProgressBar



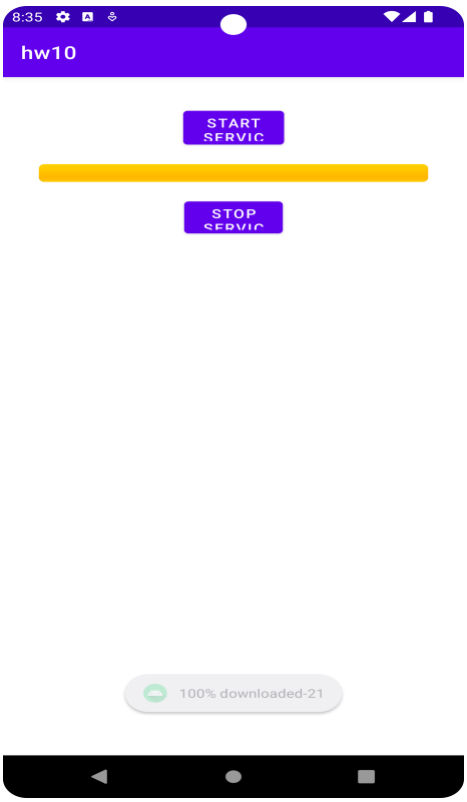
Second where you see the service is started and is at 25%



Third where you see the service is started and is at 75%



Fourth where you see the service is started and is at 100%



Fifth when we stop servicing.