NAME: SAKSHI VINAYAK KITTURKAR
CSU ID: 2860273
HOMEWORK 12

Modify the demo app for the proximity sensor such that you will see the spectrum of colors (at least 5) while the object is getting closer and closer to the phone. For example, if the max range is 1.0, when the distance from phone is 0.9, the background color is brown, when the distance from phone is 0.7, the background color changes to red, when the distance from phone is 0.5, the background color changes to pink, when the distance from phone is 0.3, the background color changes to yellow, and when the distance from phone is 0.1, the background color changes to blue. If the stance from phone is larger than the max range, the background color is set to green as before.

= I have created an application for the proximity sensor to create a dynamic visual representation of the object's proximity. Implement a color spectrum with a minimum of five distinct colors to correspond to different proximity levels. For instance, when the object is at a distance of 0.9 (within the sensor's max range of 1.0), the background color should be brown. As the object moves closer, the background color transitions through red at 0.7, pink at 0.5, yellow at 0.3, and blue at 0.1. If the object is beyond the sensor's max range, maintain the original green background color. This modification enhances the app by visually indicating the proximity of the object through a dynamic and colorful spectrum as it approaches the phone.

In layout: MainActivity the code sets up a proximity sensor, reads its values, dynamically calculates distance thresholds, and changes the background color of a TextView based on the proximity level.

These are constants:
DEFAULT_COLOR: Default background color.
DISTANCE_THRESHOLDS: Thresholds for different proximity levels.
COLORS: Corresponding colors for different proximity levels.

```
private static final int DEFAULT_COLOR = Color.GREEN;
private static final float[] DISTANCE_THRESHOLDS = {0.9f, 0.7f, 0.5f, 0.3f, 0.1f};
private static final int[] COLORS = {Color.parseColor("#964B00"), Color.RED,
Color.parseColor("#FFC0CB"), Color.YELLOW, Color.BLUE};
```

This method is called when the activity is created. It initializes the UI components, including myLabel TextView.

```
protected void onCreate(Bundle savedInstanceState) {
    // ... (omitted for brevity)

    // Initialize UI components
    myLabel = (TextView) findViewById(R.id.label);
```

```
}
```

These methods manage sensor registration and unregistration when the activity is in the foreground or background.

```
@Override
protected void onResume() {
    // Register the proximity sensor listener when the activity resumes.
}

@Override
protected void onPause() {
    // Unregister the proximity sensor listener when the activity pauses.
}
```

This method is responsible for cleaning up resources when the activity is destroyed.

```
@Override
protected void onDestroy() {
    // Cleanup resources when the activity is destroyed.
}
```

In layout: ExampleInstrumentedTest this code is a basic Android instrumented test that verifies whether the package name of the application's context matches the expected package name. This type of test is useful for checking the behavior of Android-specific components on a real device or emulator.

This method is a simple test named useAppContext. It checks if the package name of the context obtained from the InstrumentationRegistry matches the expected package name (com.example.hw12). The assertEquals method is used to assert that the two values are equal.

```
@Test
public void useAppContext() {
    // Context of the app under test.
    Context appContext = InstrumentationRegistry.getInstrumentation().getTargetContext();
    assertEquals("com.example.hw12", appContext.getPackageName());
}
```

In layout: ExampleUnitTest this code is a simple JUnit test that checks the correctness of the addition operation. Local unit tests are essential for testing the logic and functionality of individual units (methods, functions) in isolation from the larger application context.

This method is a test named addition_isCorrect. It uses the assertEquals method to check if the result of the addition operation 2 + 2 is equal to the expected result, which is 4. If the actual result matches the expected result, the test passes; otherwise, it fails.

```
@Test
public void addition_isCorrect() {
    assertEquals(4, 2 + 2);
}
```

First Screen:

From the range 0.0 to 1.0 you will get brown color

From the range greater than 1.0 you will get green color