

SENSORES ANDROID

Laboratorio de desarrollo y herramientas





INDICE

- Los Sensores.
- Integración con SonarQube.
- Pruebas Unitarias.



LOS SENSORES

-  Proximidad
-  Magnetómetro
-  Humedad



PROXIMIDAD

- Estos sensores se basan en un **LED infrarrojo** y también en un **receptor IR**. El funcionamiento se basa en emitir una luz infrarroja, y si algo la devuelve al receptor, este detecta así que hay algo muy cerca del dispositivo.



PROXIMIDAD

```
public class ProximidadActivity extends Activity implements SensorEventListener{
    LinearLayout fondo;
    TextView texto;
    Sensor s;
    SensorManager sensorM;
    List<Sensor> sensores;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_proximidad);
        fondo = (LinearLayout) findViewById(R.id.fondo);
        this.texto = (TextView) findViewById(R.id.proximidad);

        sensorM = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensores = sensorM.getSensorList(Sensor.TYPE_PROXIMITY);
        if (!sensores.isEmpty()){
            s = sensores.get(0);
            sensorM.registerListener(this,s,sensorM.SENSOR_DELAY_UI);
        }
        else
        {
            this.texto.setText("NO HAY SENSOR ACTIVO");
            this.texto.setBackgroundColor(Color.rgb(255, 0, 0));
        }
    }
}
```

```
        @Override
        public void onSensorChanged(SensorEvent evento) {
            float valor=Float.parseFloat(String.valueOf(evento.values[0]));
            if (valor <= 2.5) {
                int t_red =(int) (Math.random()*255+1);
                int t_green =(int) (Math.random()*255+1);
                int t_blue =(int) (Math.random()*255+1);
                int color = Color.rgb(t_red, t_green, t_blue);
                fondo.setBackgroundColor(color);
            }
            else{
                fondo.setBackgroundColor(Color.BLACK);
            }
        }

        @Override
        public void onAccuracyChanged(Sensor sensor, int accuracy) {
        }
    }
}
```



MAGNETOMETRO

- Se trata de un componente electrónico capaz de **medir y cuantificar la cantidad de fuerza magnética de un objeto**. Otro de los usos es, como **brújula**, detectando el polo norte magnético (que como curiosidad no coincide con el polo norte geográfico). Todos los dispositivos que estén dotados con brújula deberían de llevar incorporado un magnetómetro en su interior.



MAGNETOMETRO

```
public class MagnometroActivity extends Activity implements SensorEventListener {
    TextView magneticView;
    Sensor s;
    SensorManager sensorM;
    List<Sensor> sensores;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.content_magnometro);
        this.magneticView = (TextView) findViewById(R.id.magnetic);

        sensorM = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensores = sensorM.getSensorList(Sensor.TYPE_MAGNETIC_FIELD);
        if (!sensores.isEmpty()) {
            s = sensores.get(0);
            sensorM.registerListener(this, s, sensorM.SENSOR_DELAY_UI);
        } else {
            this.magneticView.setText("NO HAY SENSOR ACTIVO");
            this.magneticView.setBackgroundColor(Color.rgb(255, 0, 0));
        }
    }
}
```

```
@Override
public void onSensorChanged(SensorEvent evento) {
    this.magneticView.setText(String.format("Coordenada X"+
        ": %f\nCoordenada Y: %f\nCoordenada Z %f",
        new Object[] {evento.values[0], evento.values[1],
        evento.values[2]}));
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
}
```

MagnetometroActivity.java



HIUMEDAD

- Identifica los niveles de humedad en el ambiente circundante a través de un pequeño orificio situado en la base del dispositivo. Con esto se pretende que el usuario tenga mayor control sobre lo circundante, incluso puede ser útil consultarlo frecuentemente para saber en que niveles se encuentra más a gusto.



HUMEDAD

```
public class HumedadActivity extends Activity implements SensorEventListener {
    LinearLayout fondo;
    TextView texto;
    Sensor s;
    SensorManager sensorM;
    List<Sensor> sensores;

    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.content_humedad);
        fondo = (LinearLayout) findViewById(R.id.fondo);
        texto = (TextView) findViewById(R.id.humidity);

        sensorM = (SensorManager) getSystemService(SENSOR_SERVICE);
        sensores = sensorM.getSensorList(Sensor.TYPE_RELATIVE_HUMIDITY);
        if (!sensores.isEmpty()) {
            s = sensores.get(0);
            sensorM.registerListener(this, s, sensorM.SENSOR_DELAY_UI);
        }
        else
        {
            texto.setText("NO HAY SENSOR ACTIVO");
            texto.setBackgroundColor(Color.rgb(255,0,0));
        }
    }
}
```

HumedadActivity.java



HIJOS DE

```
@Override
public void onSensorChanged(SensorEvent evento) {
    float valor=Float.parseFloat(String.valueOf(evento.values[0]));
    int t_red =(int) (255); int t_green =(int) (255);
    int t_blue =(int) (255);
    int color = Color.rgb(t_red, t_green, t_blue);

    if(valor <= 25)
    {
        t_red = 255; t_green = 0; t_blue = 0;
        fondo.setBackgroundColor(color);
        texto.setText(""+valor);
    }
    else
    if(valor <= 50 && valor > 25)
    {
        t_red = 255;t_green = 0;t_blue = 85;
        fondo.setBackgroundColor(color);
        texto.setText("" + valor);
    }
    else
    if(valor <= 75 && valor > 50)
    {
        t_red = 85;t_green = 0;t_blue = 255;
        fondo.setBackgroundColor(color);
        texto.setText("" + valor);
    }
}
```

```
else
if(valor > 75 && valor <= 100)
{
    t_red = 0;t_green = 0;t_blue = 255;
    fondo.setBackgroundColor(color);
    texto.setText(""+valor);
}

@Override
public void onAccuracyChanged(Sensor sensor, int accuracy) {
}
```



INTEGRACION CON SONARQUBE

- Para la integración del proyecto con sonarqube, fue requerido modificar el fichero build.gradle del módulo app. Además deberemos incluir en el fichero gradle.properties algunas propiedades.



INTEGRACION CON SONARQUBE

Modificaciones en el archivo build.gradle

```
buildscript {  
    repositories {  
        maven {  
            url "https://plugins.gradle.org/m2/"  
        }  
    }  
    dependencies {  
        classpath "org.sonarqube.gradle:gradle-sonarqube-plugin:1.1"  
    }  
}  
  
apply plugin: "org.sonarqube"  
  
sonarqube {  
    properties {  
        property "sonar.projectName", "SensoresLDHGrupo3"  
        property "sonar.version", "1.0"  
  
        property "sonar.sources", "./src/main/java"  
        //property "sonar.tests", "./src/main/androidTest/java"  
        property "sonar.verbose", "false"  
    }  
}
```



INTEGRACION CON SONARQUAKE

```
systemProp.sonar.host.url=http://localhost:9000

#----- Security (when 'sonar.forceAuthentication' is set to 'true')
systemProp.sonar.login=admin
systemProp.sonar.password=admin
```

Modificaciones en el archivo gradle.properties



INTEGRACION CON SONARQUBE

- Por último ejecutaremos la orden “gradlew sonarqube” desde la terminal de Android studio y con esto bastará para que sonarqube (que deberá de estar lanzado ya) reconozca el nuevo proyecto y ejecute las pruebas.



INTEGRACION CON SONARQUAKE

SensoresLDHGrupo3

Version unspecified / 10 de enero del 2016 13:53

Overview

Componentes

Evidencias

Más ▾

Cuadro de mando

Evolución en el tiempo... ▾

Documentación

90.5%

APIs Públicas21

API Pub. No Doc.2

Comentarios

22.6%

Líneas De Comentario195

Líneas De Código

669

Java

Ficheros

10

Directorios1

Líneas1,245

Métodos

51

Clases10

Sentencias255

Accesores0

Duplicados

17.6%

Líneas219 ↗

Bloques13 ↗

Ficheros5 ↗

Calificación SQALE

A

Ratio De Deuda Técnica

6.8% ↗

Debt

2d 6h ↗

Evidencias

91 ↗

Bloqueante

0

Crítica

2 ↗

Mayor

29 ↗

Menor

54 ↗

Info

6 ↗

Índice De Interdependencia Entre Paquetes

0.0%

Ciclos> 0

Dependencias A Cortar

Entre Paquetes0

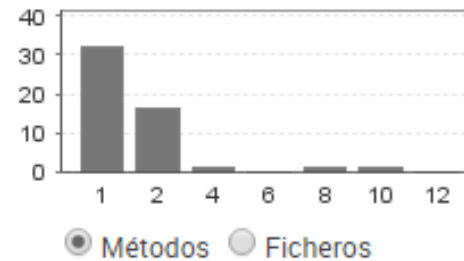
Entre Ficheros0

INTEGRACION CON SONARQUAKE

Complejidad


91

/Método /Clase /Fichero
[1.8](#) [9.1](#) [9.1](#)



Eventos Todos

| | | |
|-------------|---------|------------------------|
| 10/Ene/2016 | Versión | unspecified |
| 10/Ene/2016 | Perfil | Use 'Sonar way' (Java) |

 SensoresLDHGrupo3 Sensores_LDHGrupo3:app

Anadiendo Sonar con Android

Perfiles: [Sonar way](#) (Java)



PRUEBAS UNITARIAS

Archivo con los test implementados, creamos una instancia de cada sensor, y comprobamos que esta no es vacía.

```
public class ApplicationTest extends ApplicationTestCase<Application> {  
  
    ProximidadActivity proxi = new ProximidadActivity();  
    AcelerometroActivity acel = new AcelerometroActivity();  
    GiroscopioActivity giro = new GiroscopioActivity();  
    HumedadActivity hum = new HumedadActivity();  
    LuzActivity luz = new LuzActivity();  
    MagnometroActivity magno = new MagnometroActivity();  
    PodometroActivity podo = new PodometroActivity();  
    PulsometroActivity pulso = new PulsometroActivity();  
    TermometroActivity termo = new TermometroActivity();  
  
    public ApplicationTest() {  
        super(Application.class);  
    }  
}
```

```
public ApplicationTest() {  
    super(Application.class);  
}  
  
public void testProximidad() throws Exception {  
    assertNotNull("Existe Sensor", proxi.sensores != null);  
}  
  
public void testHumedad() throws Exception {  
    assertNotNull("Existe Sensor", hum.sensores != null);  
}  
  
public void testMagnometro() throws Exception {  
    assertNotNull("Existe Sensor", magno.sensores != null);  
}
```