補足資料
センサ値の取得と画面の描画について

実世界情報実験２モバイル端末プログラミング

柴田史久

はじめに

- 端末のセンサから得られる情報を使って，画面に描画する内容を変化させることを考える
  - 例１：端末を傾けるとボールが転がる
  - 例２：方位磁針を表示する

必要な機能について

- 画面の描画
  - SurfaceView（もしくはそれを継承したクラス）を使って描画する
  - 適切なタイミングでCanvasをロック／アンロックして描画する必要がある
- センサ値の取得
  - センサ値が変化した際に呼び出されるメソッドを準備し，そこで値を取得する

検討すべき点

- 画面を描画するタイミングをどうするのか
- センサ値をどのように扱うのか

実装パターン

- 実装パターン（１）
- 実装パターン（２－１）
- 実装パターン（２－２）
- 実装パターン（３）

実装パターン（１）

- すべての機能を１つのクラスに集約
- メリット
  - 簡単に実装できる

- デメリット
  - 機能の分離や拡張が難しい
  - センサ値の更新タイミングに描画タイミングが縛られる
  - センサ値の更新回数が多すぎると描画処理が間に合わず動作が遅くなる

実装パターン（１）詳細

- MainActivityにセンシング，描画機能を追加
  - SensorEventListenerインタフェースを実装
  - SurfaceHolder.Callbackインタフェースを実装
- 描画対象は既存のSurfaceViewをそのまま利用
- onSensorChangedの中で描画ルーチンをコール

AccBall

```
class MainActivity
  extends AppCompatActivity
  implements SensorEventListener,
             SurfaceHolder.Callback {

  protected void onCreate(…) {
    // （省略）センサマネージャの準備

    SurfaceView surfaceView
      = (SurfaceView)findViewById(…) ;
    sHolder = surfaceView.getHolder() ;
    sHolder.addCallback(this) ;
  }

  // SensorEventListenerインタフェースの実装
  public void onSensorChanged(…) {
    // センサ値を取得し，変数に格納した上で
    // 画面描画drawCanvasを呼び出す
  }
  public void onAccuracyChanged(…) ;

  // SurfaceHoler.Callbackインタフェースの実装
  public void surfaceCreated(…) {
    // センサマネージャを登録
  }
  public void surfaceChanged(…) ;
  public void surfaceDestroyed(…) ;

  private void drawCanvas() ;  // 画面の描画
}
```

実装パターン（２－１）

- 描画機能のみを分離
  - アクティビティ＋センシング
  - 描画

- メリット
  - 描画をスレッドで実行可能
  - 描画のタイミングを調整可能

- デメリット
  - センシングの結果を描画側に伝える仕組みが必要

実装パターン（２－１）詳細

- MainActivityにセンシング機能を追加
  - SensorEventListenerインタフェースを実装
  - onSensorChangedの中で変数にセンサ値をセット
  - 外部からのコールでセットしたセンサ値を提供するメソッドを準備
- 描画用のMySurfaceViewクラスを作成
  - SurfaceHolder.Callbackインタフェースを実装
  - surfaceCreatedで描画スレッドを起動
  - 描画のタイミングでセンサ値を取得

AnotherAccBall

```
class MainActivity
  extends AppCompatActivity
  implements SensorEventListener {

  protected void onCreate(…) {
    // センサマネージャの準備
  }

  // センサ値を取得し変数に格納
  public void onSensorChanged(…) ;

  public void onAccuracyChanged(…) ;

  /* 以下，自分で準備したメソッド */

  // センシングを開始
  public void startSensing() ;

  // センシングを停止
  public void stopSensing() ;

  // センサ値を返す
  public float getAccX() ;
  public float getAccY() ;
  public float getAccZ() ;
}
```

```
class MySurfaceView
  extends SurfaceView
  implements Runnable, SurfaceHolder.Callback {
  // MainActivityを保持する変数を準備
  MainActivity mainActivity ;

  // コンストラクタ．ここでMainActivityを渡す
  public MySurfaceView(Context context) {
    super(context) ;
    initialize(context) ;
  }
  // 他のコンストラクタは省略

  private void initialize(Context context) {
    mainActivity = (MainActivity)context ;
    surfaceHolder = this.getHolder() ;
    surfaceHolder.addCallback(this) ;
  }

  public void surfaceCreated(…) {
    // センシングを開始
  }
  public void surfaceChanged(…) ;
  public void surfaceDestroyed(…) {
    // センシングを停止
  }
  // スレッドの処理．ここで描画drawCanvasを呼び出す
  public void run() ;

  private void drawCanvas() ;  // 画面の描画
}
```

実装パターン（２－２）

- アクティビティから描画とセンシングを分離
  - アクティビティ
  - 描画＋センシング

- メリット
  - 描画をスレッドで実行可能
  - 描画のタイミングを調整可能

- デメリット
  - 機能の分離や拡張が難しい

実装パターン（２－２）詳細

- MainActivityは描画領域を配置するだけ
- 描画用のMySurfaceViewクラスにセンシング機能を追加
  - SensorEventListenerインタフェースを実装
  - onSensorChangedの中で変数にセンサ値をセット
  - SurfaceHolder.Callbackインタフェースを実装
  - surfaceCreatedで描画スレッドを起動
  - 描画のタイミングでセンサ値を利用

MoreoverAccBall

```
class MainActivity
  extends AppCompatActivity

  // 描画領域を配置するだけ
}
```

```
class MySurfaceView
  extends SurfaceView
  implements Runnable, SurfaceHolder.Callback,
             SensorEventListener {
  Context context ;

  // コンストラクタ
  public MySurfaceView(Context context) {
    super(context) ;
    initialize(context) ;
  }
  // 他のコンストラクタは省略
  // センサ値を取得し変数に格納
  public void onSensorChanged(…) ;
  public void onAccuracyChanged(…) ;

  private void initialize(Context context) {
    this.context = context ;
    surfaceHolder = this.getHolder() ;
    surfaceHolder.addCallback(this) ;
  }

  public void surfaceCreated(…) {
    // センサマネージャの登録とセンシングの開始
    // 描画スレッドの開始
  }
  public void surfaceChanged(…) ;
  public void surfaceDestroyed(…) {
    // 描画スレッドとセンシングを停止
  }
  // スレッドの処理．ここで描画drawCanvasを呼び出す
  public void run() ;

  private void drawCanvas() ;  // 画面の描画
}
```

実装パターン（３）

- すべての機能を分割
  - アクティビティ
  - センシング
  - 描画

- メリット
  - 描画をスレッドで実行可能
  - 描画のタイミングを調整可能
  - 各機能を拡張可能
  - 各機能の使いまわしが可能

- デメリット
  - ソースファイルが増える？

実装パターン（３）詳細

- MainActivityは描画領域を配置するだけ
- センシング用のInnerSensorListenerクラス
  - SensorEventListenerインタフェースを実装
  - onSensorChangedの中で変数にセンサ値をセット
  - 外部からのコールでセットしたセンサ値を提供するメソッドを準備
- 描画用のMySurfaceViewクラスを作成
  - SurfaceHolder.Callbackインタフェースを実装
  - surfaceCreatedで描画スレッドを起動
  - 描画のタイミングでセンサ値を取得

YetAnotherAccBall

```
class MainActivity
  extends AppCompatActivity

  // 描画領域を配置するだけ
}
```

```
class InnerSensorListener
  implements SensorEventListener

  public void onSensorChanged(…) {
    // センサ値を取得し，変数に格納
  }
  public void onAccuracyChanged(…) ;

  public synchronized void resume(…) {
    // センサマネージャの準備
  }
  public synchronized void pause() {
    // センサマネージャの登録解除
  }

  // センサ値を返す
  public synchronized float getAccX() ;
  public synchronized float getAccY() ;
  public synchronized float getAccZ() ;
}
```

```
class MySurfaceView extends SurfaceView
  implements Runnable, SurfaceHolder.Callback {

  InnerSensorListener innerSL ;
  // コンストラクタ
  public MySurfaceView(Context context) {
    super(context) ;
    initialize(context) ;
  }
  // 他のコンストラクタは省略

  private void initialize(Context context) {
    surfaceHolder = this.getHolder() ;
    surfaceHolder.addCallback(this) ;
    innerSL = new InnerSensorListener() ;
  }

  public void surfaceCreated(…) {
    innerSL.resume(…) ;  // センシングの開始
    // 描画スレッドの開始
  }
  public void surfaceChanged(…) ;
  public void surfaceDestroyed(…) {
    // 描画スレッドの停止
    innerSL.pause() ;  // センシングを停止
  }
  // スレッドの処理．ここで描画drawCanvasを呼び出す
  public void run() ;

  private void drawCanvas() ;  // 画面の描画
}
```