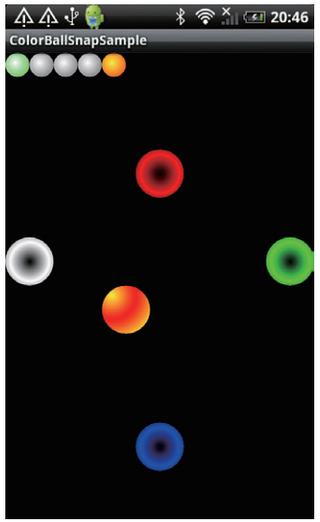
2-1 アプリケーションの概要

本章で使用するサンプルアプリは、「カラーボールゲームアプリ」です。まずここではアプリの概要について解説します。

なお、本章で扱う加速度センサーはエミュレータでは動作しないので、実機にインストールして動作を確認する必要があります。

図2.1.1　カラーボールゲームの画面



- ゲームを起動すると赤、青、緑、白の4つの穴と、中心にカラーボールが表示されます。
- 端末を左右に傾けると、加速度センサーが動作し、カラーボールが画面上を動き回ります。
- カラーボールは、それ自身と同じ色の穴に重なった時、色が変わります。
- ゲームプレイヤーがカラーボールを転がすことで、カラーボールの色が次々と変わります。

▶プロジェクト構成

```
ColorBallSnapSample（プロジェクト）
  src（ソースフォルダ）
    jp.co.techfun.colorballsnap（パッケージ）
      ColorBallSnapSampleActivity.java（ゲーム画面アクティビティ）
      DrawableView.java（画面の描画クラス）
  res（リソースフォルダ）
    drawable（画像格納用フォルダ）
      icon.png（アイコン画像）
    values（文字列定義ファイル格納用フォルダ）
      strings.xml（文字列定義ファイル）
```


2-2 センサー機能について

ここでは、センサーを扱う処理について解説します。本章のサンプルアプリでは、主に加速度センサーを扱いますが、他のセンサーの場合も基本は一緒なので、まずは原理を押さえるようにしましょう。

① センサー概要

標準的なAndroid端末には、さまざまなセンサーが搭載されています。センサーとは、温度や方位・方角、明るさ、傾き、加速度のような物理的な情報を電気信号に変換する測定機器のことです。

Androidでは、センサーで変換された電気信号を取得するためのセンサーAPIが提供されており、センサーを利用したアプリケーションを簡単に開発することができます。センサーを利用することができれば、工夫次第で生活を便利にするアプリケーションを作成することができるでしょう。

ただしAndroid端末の機種によっては、搭載されているセンサーが異なるので、センサーを扱うときにはその点に注意する必要があります。

② Androidのセンサー API

Androidでは、android.hardwareパッケージに、「SensorManager」、「Sensor」、「SensorEvent」、「SensorEventListener」という4つのセンサーAPIが提供されています。これらのAPIの働きは、次のようなものです。

- **SensorManager**

  端末に搭載されているセンサーにアクセスするためのクラスです。各種センサーのイベントリスナーの登録や解除、センサー情報の取得なども行います。このクラスに含まれる主なメソッドは、**表2.2.1**のとおりになります。

表2.2.1　SensorManagerクラスのメソッド

メソッド名	説明
registerListener(SensorEventListener listener, Sensor sensor, int rate, Handler handler)	イベントリスナーを登録するメソッド
registerListener(SensorEventListener listener, Sensor sensor, int rate)	
unregisterListener(SensorEventListener listener, Sensor sensor)	イベントリスナーを解除するメソッド
unregisterListener(SensorEventListener listener)	
getSensorList(int type)	使用可能なセンサーをリストで取得するメソッド

また、反応速度を表す定数があり、センサーのイベントリスナー登録時にこの定数を使用します(**表2.2.2**)。作成するアプリケーションにおけるセンサーの用途により、反応速度を適切なものに設定します。

表2.2.2　反応速度に関する定数

定数名	設定値	説明
SENSOR_DELAY_FASTEST	0	最高速の反応速度
SENSOR_DELAY_GAME	1	ゲーム向けの速い反応速度
SENSOR_DELAY_UI	2	ユーザインタフェース向けの遅い反応速度
SENSOR_DELAY_NORMAL	3	通常の反応速度

・Sensor

各種センサーに関する情報(センサーの種類、バージョン、名前、開発ベンダー名など)を管理するクラスです。主なメソッドは、**表2.2.3**のとおりです。

表2.2.3　Sensorクラスのメソッド

メソッド名	説明
getMaximumRange()	センサーが検出できる最大値を取得するメソッド
getName()	センサーの名前を取得するメソッド
getPower()	センサーからの電気信号をmA(ミリアンペア)で取得するメソッド
getResolution()	センサーの精度を取得するメソッド
getType()	センサーの種類を取得するメソッド
getVendor()	センサーのベンダー名を取得するメソッド
getVersion()	センサーのバージョンを取得するメソッド

Sensorクラスには**表2.2.4**に挙げるセンサーの種類を表す定数があります。ただし前述のように、全てのAndroid端末が、ここであげた全てのセンサー機能を持っているわけではありません。

表2.2.4　センサーの種類に関する定数

定数名	設定値	説明
TYPE_ACCELEROMETER	1	加速度センサー
TYPE_MAGNETIC_FIELD	2	磁界(磁気)センサー
TYPE_ORIENTATION	3	方位(傾き)センサー
TYPE_GYROSCOPE	4	ジャイロスコープセンサー
TYPE_LIGHT	5	照度センサー
TYPE_PRESSURE	6	圧力センサー
TYPE_TEMPERATURE	7	温度センサー
TYPE_PROXIMITY	8	近接センサー
TYPE_GRAVITY	9	重力センサー
TYPE_LINEAR_ACCELERATION	10	直線加速度センサー
TYPE_ROTATION_VECTOR	11	回転角度センサー
TYPE_ALL	-1	この定数で指定したセンサーの全ての値が取得できる。例えば、TYPE_ACCELEROMETERを指定すると加速度センサーで検知した値が取得でき、TYPE_ALLで指定するとその機種が持っている全てのセンサーの検知する情報を取得することができる。

・SensorEvent

センサーが取得できる情報を1つにまとめて管理するクラスです。このクラスで管理できる項目は、全てパブリックフィールドで宣言されています。主なフィールドは、**表2.2.5**のとおりです。

表2.2.5　SensorEventクラスのフィールド

フィールド定義	説明
int accuracy	センサーの精度を格納するフィールド
Sensor sensor	Sensorオブジェクトを保持するフィールド
long timestamp	タイムスタンプ(ナノ秒)を格納するフィールド
float[] values	センサーの値を格納するフィールド

・SensorEventListener

センサー関連のイベントを取得するためのイベントリスナーです。SensorManagerのメソッドでこのイベントリスナーの登録や解除をすると、センサーからの情報を取得できます。主なメソッドは、**表2.2.6**のとおりです。

表2.2.6　SensorEventListenerのメソッド

メソッド名	説明
onAccuracyChanged(Sensor sensor, int accuracy)	センサーの精度が変わったときに呼び出されるメソッド
onSensorChanged(SensorEvent event)	センサーの値が変わったときに呼び出されるメソッド

③ センサー処理

センサーを扱った時の処理の流れは次のとおりです。

① センサーを管理するSensorManagerを取得する。
② SensorManagerに利用したいセンサーとイベントリスナーを登録する。
③ センサーの精度や値が変わったときに呼び出されるコールバックメソッドでセンサー情報を取得する。

続いて、それぞれの手順を詳述します。

④ SensorManager取得

センサーを利用するにはまず、SensorManagerのインスタンスを**getSystemServiceメソッド**で取得します(**リスト2.2.1**)。引数には、android.contentパッケージのContextクラスの定数**SENSOR_SERVICE**を指定しましょう。

サンプルアプリでは、onCreateメソッドでSensorManagerのインスタンス取得処理を行っています。

リスト2.2.1　ColorBallSnapSampleActivity.java (15行目～ 37行目)

```
// SensorManagerインスタンス
private SensorManager sensorManager;

(省略)                  :                  :                  :

// onCreateメソッド(画面初期表示イベント)
@Override
public void onCreate(Bundle savedInstanceState) {

    (省略)                  :                  :                  :

    // SensorManagerのインスタンス取得
    sensorManager = (SensorManager) this.getSystemService(SENSOR_SERVICE);

}
```


⑤ イベントリスナー登録

次に、SensorManagerの**getSensorListメソッド**でセンサーオブジェクトをリスト形式で取得します(**リスト2.2.2**)。引数には、センサーの種類を表すSensorクラスの定数を設定します。サンプルアプリでは、加速度センサーを表す**TYPE_ACCELEROMETER**を引数として設定しています。

また、SensorManagerの**registerListenerメソッド**に取得したセンサーのイベントリスナー、取得したセンサーオブジェクト、反応速度を設定します。反応速度には、SensorManagerの定数を使用しましょう。サンプルアプリでは、onResumeメソッドにこれらの処理を実装しています。

リスト2.2.2　ColorBallSnapSampleActivity.java (44行目～ 53行目)

```
// 加速度センサーオブジェクト取得
List<Sensor> accelerometerSensors =
    sensorManager.getSensorList(Sensor.TYPE_ACCELEROMETER);

// 加速度センサーオブジェクトが取得できた場合
if (accelerometerSensors.size() > 0) {
    // SensorManagerインスタンスにセンサーイベントリスナーを設定する
    sensorManager.registerListener(sensorEventListener,
        accelerometerSensors.get(0), SensorManager.SENSOR_DELAY_GAME);
}
```

⑥ センサー情報取得

センサー情報は、センサーの精度や値が変わったときに呼び出されるSensorEventListenerの**onAccuracyChangedメソッド**や**onSensorChangedメソッド**で取得できます(**リスト2.2.3**)。

サンプルアプリでは、onSensorChangedメソッドで検知した加速度センサーの値を取得しています。

リスト2.2.3　ColorBallSnapSampleActivity.java（63行目～ 82行目）

```
// センサーイベントリスナーの定義
private final SensorEventListener sensorEventListener =
    new SensorEventListener() {

(省略)                  :                  :                  :

    // onAccuracyChangedメソッド(精度変更時イベント)
    @Override
    public void onAccuracyChanged(Sensor sensor, int accuracy) {
        // 処理なし
```

```
    }

    // onSensorChangedメソッド(センサー情報変更時イベント)
    @Override
    public void onSensorChanged(SensorEvent event) {・・・
```

onSensorChangedメソッドの引数には、センサーが検知した情報を持つSensorEventのインスタンスが引き渡されます。SensorEventの**valuesメソッド**で引数にX軸、Y軸、Z軸を表すSensorManagerの定数**DATA_X**、**DATA_Y**、**DATA_Z**を指定して各方向のセンサーの値を取得します(**リスト2.2.4**)。

リスト2.2.4　ColorBallSnapSampleActivity.java (80行目～ 86行目)

```
// onSensorChangedメソッド(センサー情報変更時イベント)
@Override
public void onSensorChanged(SensorEvent event) {
    // センサーが検知した値を取得
    float x = event.values[SensorManager.DATA_X];
    float y = event.values[SensorManager.DATA_Y];
    float z = event.values[SensorManager.DATA_Z];
```

また、SensorEventの**getTypeメソッド**でセンサーの種類を取得して、対象のセンサーが検知された場合のみ必要な処理を行うようにします(**リスト2.2.5**)。

サンプルアプリでは、加速度センサーを表すSensorの定数TYPE_ACCELEROMETERの場合、カラーボールを描画するビュークラスに各方向のセンサーの値を引き渡して、再描画しています。加速度センサー以外は、何の処理も行っていません。

リスト2.2.5　ColorBallSnapSampleActivity.java (93行目～ 106行目)

```
switch (event.sensor.getType()) {

    // 検知されたセンサー情報が加速度センサーの場合
    case Sensor.TYPE_ACCELEROMETER:
        drawableView.effectAccelaration(lowX, lowY, lowZ);

        // 画面を再描画
        drawableView.invalidate();
        break;

    // 加速度センサー以外は無視
    default:
        break;
}
```


7 画面の表示方向を固定

サンプルアプリでは、端末を前後左右に傾けてカラーボールを転がしますが、このまま実行すると、端末を動かしている間に画面レイアウトの縦横が頻繁に切り替わってしまうことになります。

この画面の切り替えを抑制するために、**リスト2.2.6**のように「AndroidManifest.xml」に縦長レイアウト固定の設定を追加しておきましょう。

リスト2.2.6　AndroidManifext.xml (12行目～ 16行目)

```
<!-- 画面の向きを縦画面表示に固定 -->
<activity
  android:name=".ColorBallSnapSampleActivity"
  android:label="@string/app_name"
  android:screenOrientation="portrait">
```

activityタグの**android:screenOrientation属性**に「portrait」を設定します。これで対象のアクティビティが縦長レイアウト固定になります。

android:screenOrientation属性には、**表2.2.7**に示す設定が可能です。

表2.2.7　android:screenOrientation属性の設定値

設定値	縦持ち	横持ち	説明
unspecified (デフォルト)	縦長	横長	システムが最適な画面レイアウトを選択し表示 ユーザの操作によっても変更される
landscape	横長	横長	常に横長のレイアウトで表示
portrait	縦長	縦長	常に縦長のレイアウトで表示
user	縦長	横長	ユーザの現在の設定に従って表示
behind	親アクティビティに依存	親アクティビティに依存	親アクティビティの画面レイアウトに従って表示
sensor	縦長	横長	傾きセンサーに従って表示 ユーザの操作によっても変更される
nonsensor	変化なし	変化なし	傾きセンサーは無視 ユーザの操作によっても変更されない

サンプルアプリでは、加速度センサーを使用していますが、他のセンサーについても同様に処理することでそのセンサー情報が取得できます。

2-3 グラフィックス描画

次に、サンプルアプリのようにカラーボールを描画するためのグラフィックスの描画処理について説明します。

① Androidのグラフィックス描画

少し凝った画面のアプリケーションを作成したい場合、標準で提供されているテキストボックスやボタンだけでは表現に限界があります。Androidでは、画面の柔軟なデザインを可能にするためのグラフィカルな描画方法として、次に挙げる方法が提供されています。

1. Drawableクラスを利用した描画

抽象クラスである**Drawableクラス**のサブクラスを使用して、図形の描画や画像データの表示を行います。

2. Canvasクラスを利用した描画

Canvasクラスで、直接ペンや筆で自由にキャンバスに描くように描画処理を行います。

3. アニメーションXMLを利用した描画

画像を回転したり、拡大縮小したり、移動したりといった画像を動作させる処理を行います。

4. Open GL ESライブラリを利用した描画

OpenGL ESライブラリを使用して2D、3Dを高速に描画する方法です。描画の自由度も高く、ゲームアプリなど描画が多く必要なアプリケーションでは多く利用されています。

本節では、サンプルアプリをもとに「1. Drawableクラスを使用した描画」を中心に説明します。「2. Canvasクラスを利用した描画」および「3. アニメーションXMLを利用した描画」は、サンプルアプリでは扱っていませんが、知っておくと便利な方法なので、併せて本章で解説します。なお、「4. Open GL ESライブラリを利用した描画」については、難易度が高いこと、ボリュームのあるトピックであることから、本書では割愛いたします。

② Drawableクラスを使用した描画

android.graphics.drawableパッケージの**Drawableクラス**を継承したクラスは複数ありますが、サンプルアプリでは、円や四角形などの基本図形を描画する**ShapeDrawable**を使用しています。

また、カラーボールや各色の穴を描画するのに、描画用ビュークラスをアクティビティとは別に定義しています。

アクティビティのonCreateメソッドで描画用クラスのインスタンス化を行い、setContentViewメソッドに描画用ビュークラスのインスタンスを設定すると、画面表示のときに描画処理が実行されます(**リスト2.3.1**)。なおその際に、バックライトが自動消灯しないように**setKeepScreenOnメソッド**にtrueを設定しておきましょう。

リスト2.3.1　ColorBallSnapSampleActivity.java (18行目～32行目)

```
// DrawableViewインスタンス
private DrawableView drawableView;

// onCreateメソッド(画面初期表示イベント)
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // DrawableViewインスタンス生成
    drawableView = new DrawableView(this);

    // バックライトが自動消灯しないように設定
    drawableView.setKeepScreenOn(true);

    // 画面に表示するDrawableViewインスタンスを指定
    setContentView(drawableView);
```

描画用ビュークラスは、Viewクラスを継承して定義し、ShapeDrawableの配列を宣言しています(**リスト2.3.2**)。

リスト2.3.2　DrawableView.java (15行目～72行目)

```
// 画面の描画クラス
public class DrawableView extends View {

(省略)                  :                  :                  :

    // 穴と描画オブジェクト用配列
    private ShapeDrawable[] drawables;

(省略)                  :                  :                  :

// 落ちたボールの描画オブジェクト配列
private ShapeDrawable[] fallBallDrawables;
```

そして、**リスト2.3.3**に示すように、描画用のクラスのコンストラクタでShapeDrawableのインスタンスを生成し、描画する図形や色などの設定を行っています。

コンストラクタの最初にShapeDrawableの配列を生成し、各配列のShapeDrawableをインスタンス化して描画オブジェクトを生成しています。ShapeDrawableの描画オブジェクトを生成するときに、引数に描画する基本図形の種類を指定するクラスのインスタンスを指定します。

リスト2.3.3　DrawableView.java（80行目～92行目）

```
// コンストラクタ(描画オブジェクト生成)
public DrawableView(Context context) {
    super(context);

    // ShapeDrawableのインスタンスを生成
    drawables = new ShapeDrawable[HOLES_SIZE];
    fallBallDrawables = new ShapeDrawable[FALL_BALL_STOCKS_SIZE];

    // 穴の描画
    drawables[RED_OVAL] = new ShapeDrawable(new OvalShape());
    drawables[GREEN_OVAL] = new ShapeDrawable(new OvalShape());
    drawables[BLUE_OVAL] = new ShapeDrawable(new OvalShape());
    drawables[WHITE_OVAL] = new ShapeDrawable(new OvalShape());
```

サンプルアプリでは、**OvalShapeクラス**のインスタンスを設定し、円として描かれるオブジェクトを生成しています。

主要な基本図形のクラスは、**表2.3.1**のとおりです。

表2.3.1　基本図形のクラス

クラス名	説明
OvalShape	円形を描画するクラス
RectShape	四角形を描画するクラス
RoundRectShape	角の丸い四角形を描画するクラス
PathShape	連続する複数の点を結んで多角形を描画するクラス
ArcShape	円弧を描画するクラス

ShapeDrawableのインスタンスが生成できたら、続いて**リスト2.3.4**のように、描画する基本図形の色やグラデーションを設定します。

サンプルアプリでは、各色をベースにグラデーションをつけています。

リスト2.3.4　DrawableView.java（94行目～106行目）

```
// 穴の色の設定
drawables[RED_OVAL].getPaint().setShader(
    new RadialGradient(25, 25, 20, Color.BLACK, Color.RED,
        Shader.TileMode.MIRROR));
drawables[GREEN_OVAL].getPaint().setShader(
    new RadialGradient(25, 25, 20, Color.BLACK, Color.GREEN,
        Shader.TileMode.MIRROR));
drawables[BLUE_OVAL].getPaint().setShader(
    new RadialGradient(25, 25, 20, Color.BLACK, Color.BLUE,
        Shader.TileMode.MIRROR));
drawables[WHITE_OVAL].getPaint().setShader(
    new RadialGradient(25, 25, 20, Color.BLACK, Color.WHITE,
        Shader.TileMode.MIRROR));
```

このように、ShapeDrawableの**getPaintメソッド**で**Paintクラス**のインスタンスを取得します。Paintクラスは、図形や線を描画するときの筆のような役割で、色やグラデーションのような特殊効果の設定をするメソッドを持っています。

サンプルアプリでは、Paintの**setShaderメソッド**でグラデーションをつけています。引数には、中心から外側に向かってグラデーションをつける**RadialGradient**を指定しています。RadialGradientのインスタンス生成時には、次の6つの引数を設定します。

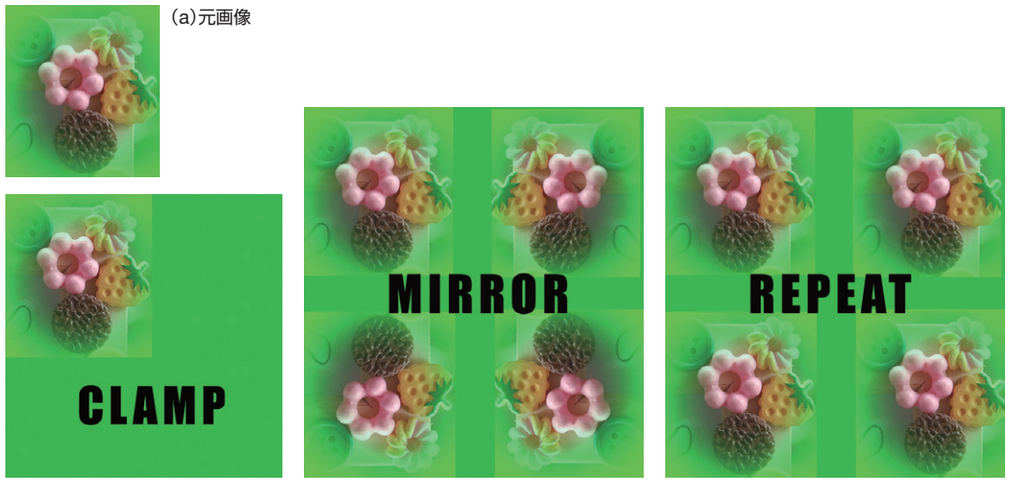
・第1引数：中心点のx座標
・第2引数：中心点のy座標
・第3引数：半径
・第4引数：中心点の色
・第5引数：円のエッジの色
・第6引数：グラデーションのモード

第6引数のグラデーションのモードには、**表2.3.2**に挙げる設定が可能です。これらのモードが画像にどのような効果をもたらすのか、こちらについては**図2.3.1**を参照下さい。

表2.3.2　グラデーションのモード

モード名	説明
Shader.TileMode.CLAMP	画像の外側にグラデーションが描画される場合、エッジの色をコピーする。
Shader.TileMode.MIRROR	グラデーション画像を水平垂直方向に鏡像で繰り返す。
Shader.TileMode.REPEAT	グラデーション画像を水平垂直方向に繰り返す。

図2.3.1　グラデーションのタイルモード



(b) CLAMP　(c) MIRROR　(d) REPEAT

グラデーションや陰影効果をつける場合、RadialGradientクラス以外に**表2.3.3**に挙げるandroid.graphicsパッケージの**Shaderクラス**のサブクラスを使用することができます。

表2.3.3　グラデーション・陰影効果のクラス

クラス名	説明
BitmapShader	ビットマップに対して陰影をつけるクラス
ComposeShader	2つのShaderを合成して効果をつけるクラス
LinearGradient	線状のグラデーションをつけるクラス
SweepGradient	中心と半径から円状のグラデーションをつけるクラス

なお、サンプルアプリケーションにはありませんが、グラデーションなしで色を設定したい場合には、**リスト2.3.5**に示すようにPaintの**setColorメソッド**で色を指定します。色は**Colorクラス**の定数やカラーコードを指定すれば反映されます。*1

リスト2.3.5　図形に色を設定

```
drawables[RED_OVAL].getPaint().setColor (Color.RED);                    // 赤
drawables[BLUE_OVAL].getPaint().setColor (Color.BLUE);                  // 青
drawables[WHITE_OVAL].getPaint().setColor (Color.parseColor("#FFFFFF")); // 白
```

ここまでで、表示する色のついた穴の描画準備は完了です。そのあとにカラーボールについても同様に描画準備を行っています(**リスト2.3.6**)。

リスト2.3.6　DrawableView.java (108行目～ 299行目)

```
        // ボールの描画
        ballDrawables = new ShapeDrawable(new OvalShape());

        // ボールの色を設定(初期値)
        ballDrawables.getPaint().setShader(getRandomRadialGradient());

(省略)                  :                  :                  :

}

(省略)                  :                  :                  :

// ランダムグラデーション処理
private Shader getRandomRadialGradient() {

(省略)                  :                  :                  :

}
```

また、Viewを継承したクラスで**onDrawメソッド**をオーバライドします(**リスト2.3.7**)。このonDrawは、描画処理のときに実行されるメソッドで、引数にCanvasオブジェクトが引き渡されます。描画領域をCanvasの**getWidthメソッド**と**getHeightメソッド**を使用して取得し、drawColorHoleメソッドで4つの穴を描画する処理を行っています。

描画処理は、ShapeDrawableの**setBoundsメソッド**で描画する場所を指定し、ShapeDrawableの**drawメソッド**で描画します。

リスト2.3.7　DrawableView.java (119行目～ 275行目)

```
// onDrawメソッド(画面描画処理)
@Override
protected void onDraw(Canvas canvas) {

    // 画面描画領域
    int width = canvas.getWidth() - OFFSET_X;
    int height = canvas.getHeight() - OFFSET_Y;

    // 画面中央に円を表示する座標
    int cX = width / 2 - RADIUS;
    int cY = height / 2 - RADIUS;

    // 4色の穴を画面に表示
    drawColorHole(canvas, width, height, cX, cY);

(省略)              :              :              :
```

*1　リスト 2.3.5 の場合、「Color.RED」が定数で「"#FFFFFF"」がカラーコードになる。