2016 年 12 月 1 日（木）実施

SurfaceView による高速描画

SurfaceView とは

SurfaceView は, View の階層の中に描画専用の面を埋め込むためのクラスである。SurfaceView を継承したクラスでは，View を継承したクラスと異なり，描画処理専用の**スレッド**（並列処理に対応した OS 上でのプログラムの最小の実行単位；Thread）を主要なスレッドとは別に割り当てることが可能で，高速な描画が可能となる。

必要となる Java の基礎知識

コンストラクタ

Java 言語では，クラスの定義にフィールド，メソッドの他に**コンストラクタ**と呼ばれる，クラスのインスタンス化の際に呼び出されて実行される処理を記述出来る。

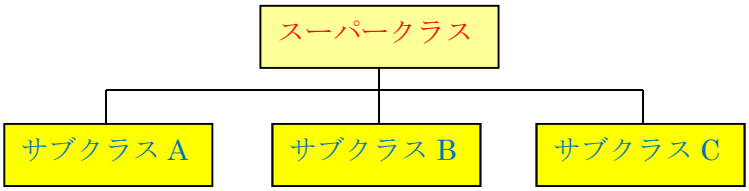
```
修飾子 class クラス名 {
  型 フィールド名;

  修飾子 コンストラクタ名 ( 引数 ) {
    コンストラクタの定義
  }

  修飾子 戻り値のデータ型 メソッド名( 引数 ) {
    メソッドの定義
  }
}
```

コンストラクタの特徴としては，次の様なものが挙げられる。

1) 名称はクラス名と同一である。
2) 戻り値はない。
3) インスタンスの初期設定に利用する。
4) コンストラクタからメソッドを呼び出すことも可能である。

スーパークラスとサブクラス －継承とオーバーライド－

あるクラスを拡張して作られたクラスを元のクラスの**サブクラス**と呼び，サブクラスから見て元のクラスを**スーパークラス**と呼ぶ。あるクラスから複数のサブクラスを作ることは可能であるが，あるサブクラスを複数のスーパークラスから作ることは出来ない。（図は次のページ）



サブクラスの作り方は次の様になる。

```
修飾子 class サブクラス名 extends スーパークラス名{
  型 フィールド名;

  修飾子 コンストラクタ名(=サブクラス名)( 引数 ) {
    コンストラクタの定義
  }

  修飾子 戻り値のデータ型 メソッド名( 引数 ) {
    メソッドの定義
  }
}
```

サブクラスはスーパークラスのフィールド及びメソッドを引き継ぎ，このことを**継承**と呼ぶ。サブクラスには，スーパークラスには無いフィールド及びメソッドを新規に定義出来ると共に，スーパークラスから継承したフィールド及びメソッドを再定義することも可能である。サブクラスでスーパークラスから継承したメソッドを再定義することを**オーバーライド**(override)と呼ぶ。なお，サブクラスにはスーパークラスからコンストラクタは継承されないので注意が必要である。

インタフェース

インタフェースとは実装の無いクラスで，**実装とは，枠組みだけが定義されているものに，中身を与えること**である。インタフェースはクラスの階層にとらわれずに実装が可能であるが，記述出来る内容は定数フィールドと**抽象メソッド**に限られる。ここで，**抽象メソッドとは，インタフェースを実装したクラスで必ずオーバーライドしなければならないメソッド**である。

インタフェースの定義の一般形は次のようになる。

```
修飾子 interface インタフェース名 {
  型 フィールド名;

  戻り値のデータ型 メソッド名( 引数 );
}
```

インタフェースでは明示しなくても，全てのフィールドは public static final の修飾子を付

けた扱いとなる。また，全てのメソッドは public abstract の修飾子を付けた扱いとなり，その処理の定義は記述しないので，ブロックにはならず，セミコロンを必要とする。

なお，インタフェースは直接インスタンス化することが出来ないので，コンストラクタを宣言することは出来ない。（インタフェースを実装するクラスには，コンストラクタを宣言出来る）

インタフェースを実装するクラスは，implements（実装）により，次のように定義する。

```
修飾子 class クラス名 implements インタフェース名 1 [,インタフェース名 2, ・・・]{
  型 フィールド名;

  修飾子 戻り値のデータ型 メソッド名( 引数 ) {
    メソッドの中身
  }
}
```

＊ クラスの継承とインタフェースの実装とを同時に行うクラスの作成が可能である。

＊＊ インタフェースを extends で継承したサブインタフェースの作成も可能である。

授業の準備

1）Android Studio の初期設定

Android Studio を起動し，『Configure』→『設定のインポート』を選択し，第 3 回の教材の p.5 に従って設定をインポートする。

2）プロジェクトの新規作成

『Application name』(アプリ名)を「Prog_8th」(先頭は大文字,「_」は下線),『Company Domain』を「b6a0xxx.cuc.ac.jp」に書き換え,『Project Location』の先頭の「C:¥Users¥ b6a0xxx」を『H:』に書き換えて,『次へ』ボタンを押す。

第 1 回と同様に『Minimum SDK』では『API 22』を選択する（第 1 回教材 p.7)。

『Activity name』は「MainActivity8」とする。

3）AVD の設定

第 1 回の授業で作成した AVD の設定は H ドライブにあって残るが，SDK のシステムイメージは C ドライブにあるので，消失している。そこで,『Download』をクリックして，インストールし直す（第 2 回教材 p.4)。

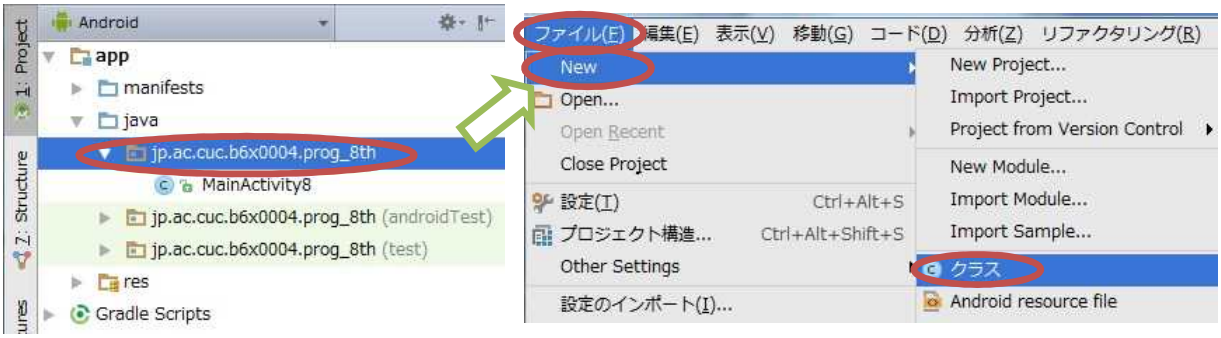
課 題

今回は，SurfaceView クラスを継承した新規のクラスを作成し，高速描画の基本を学ぶ。

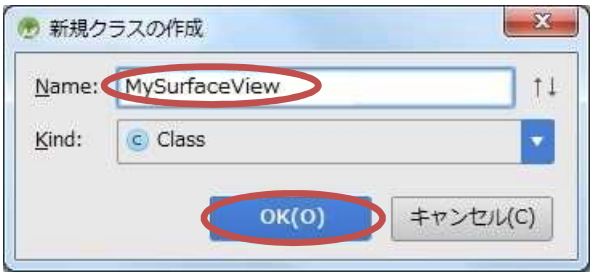
Android アプリの作成

『Project』タブを開いた状態で,『java』→『jp.ac.cuc.b6x0004.prog_8th』と選択した上で,『ファイル』→『New』→『クラス』と選択する。

（図は次のページ）



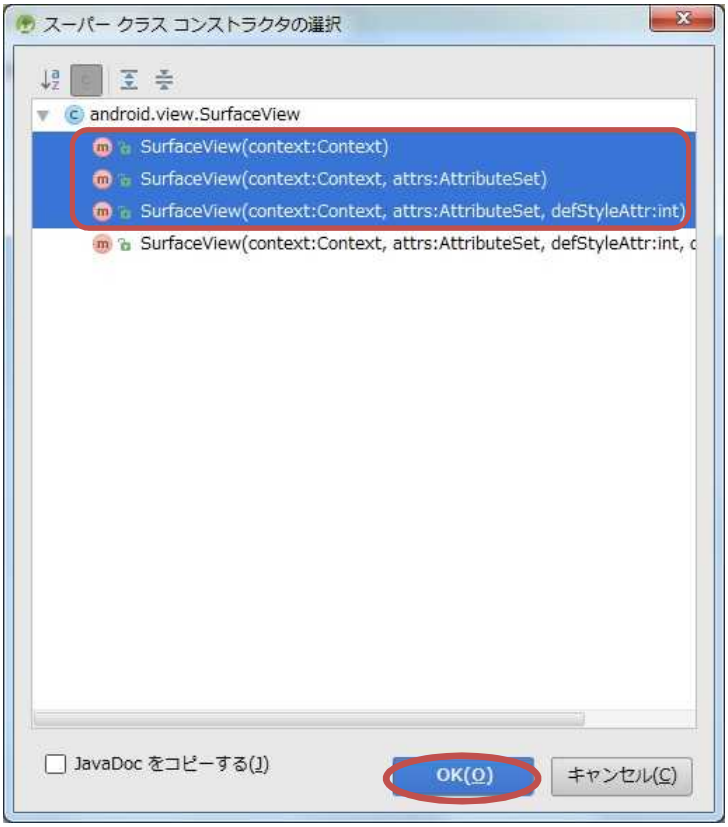
クラスの『Name:』を「MySurfaceView」とする。



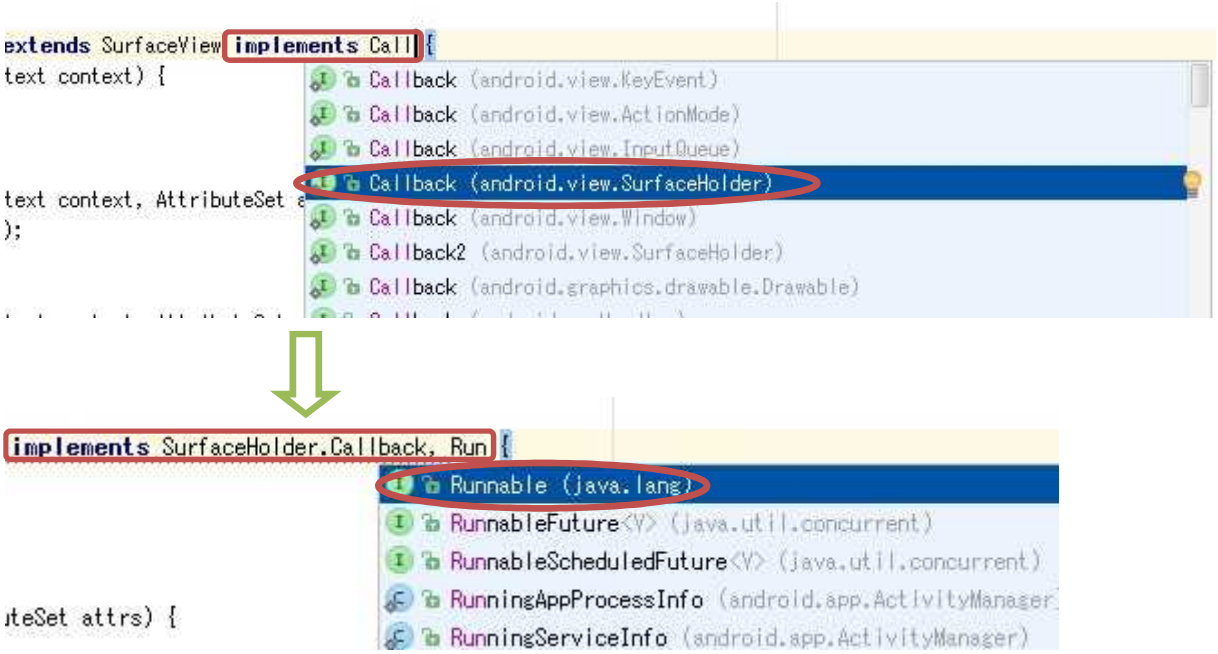
『MySurfaceView.java』のタブを開き，SurfaceView クラスを継承するために，「extends SurfaceView」をクラス名の『MySurfaceView』の後ろに書き加える。

赤い波線が出たら，その行にカーソルを合わせるとヒントがあるサインが出るので，それをクリックしてヒントを出す。

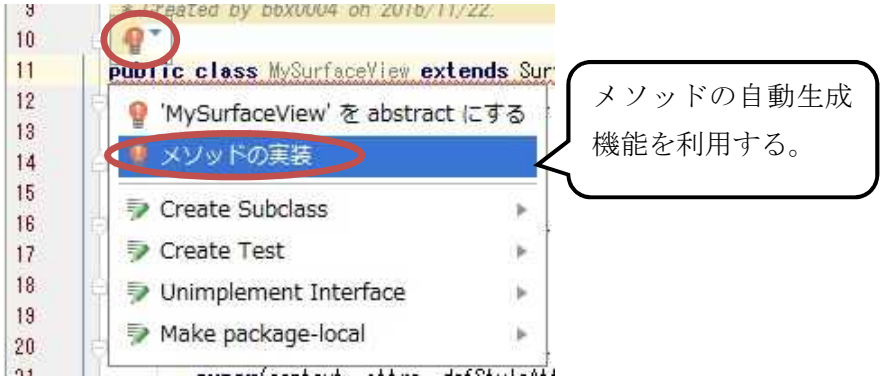
引数が 1 個，2 個及び 3 個の 3 つのコンストラクタを選択して「OK」を押す。



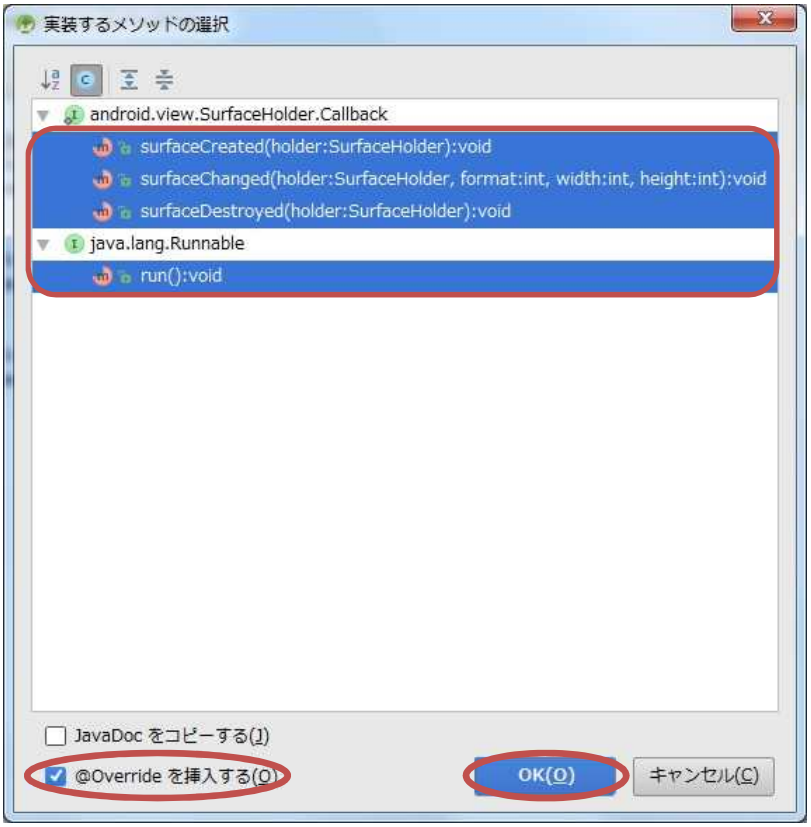
更に，android.view.SurfaceHolder.Callback インタフェース及び java.lang.Runnable インタフェースを実装するために，extend SurfaceView の後ろに「implements Callback, Runnable」を書き加える。



赤い波線が出たら，その行にカーソルを合わせるとヒントがあるサインが出るので，それをクリックしてヒントを出す。



一覧に出る 4 個のメソッドを全て選択して「OK」を押す。



これで 3 個のコンストラクタ及び 4 個のメソッドが次のページの図の様に自動生成されるので，更に自作のメソッド surfaceInit を付け加えると共に，コンストラクタには surfaceInit メソッドの呼び出しを記述し，自動生成された 4 個のメソッドのうち surfaceCreated，surfaceDestroyed 及び run の 3 個のメソッドのそれぞれを実装する。

activity_main8.xml × | MainActivity8.java × | MySurfaceView.java ×

```
package jp.ac.cuc.b6x0004.prog_8th;

import android.content.Context;
import android.util.AttributeSet;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MySurfaceView extends SurfaceView implements SurfaceHolder.Callback, Runnable {
    public MySurfaceView(Context context) {
        super(context);
    }

    public MySurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);
    }

    public MySurfaceView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
    }

    @Override
    public void surfaceCreated(SurfaceHolder holder) {

    }

    @Override
    public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {

    }

    @Override
    public void surfaceDestroyed(SurfaceHolder holder) {

    }

    @Override
    public void run() {

    }
}
```

activity_main8.xml × | MainActivity8.java × | MySurfaceView.java ×

```
package jp.ac.cuc.b6x0004.prog_8th;

import android.content.Context;
import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.util.AttributeSet;
import android.view.SurfaceHolder;
import android.view.SurfaceView;

public class MySurfaceView extends SurfaceView implements SurfaceHolder.Callback, Runnable {
    private SurfaceHolder holder;
    private Thread thread;
    private int x, y, xadd = 5, yadd = 10;
    private final int r = 10;

    public MySurfaceView(Context context) {
        super(context);
        surfaceInit();
    }

    public MySurfaceView(Context context, AttributeSet attrs) {
        super(context, attrs);
        surfaceInit();
    }

    public MySurfaceView(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        surfaceInit();
    }
```

```
31
32      @Override
33      public void surfaceCreated(SurfaceHolder holder) {
34          thread = new Thread(this);
35          thread.start();
36      }
37
38      @Override
39      public void surfaceChanged(SurfaceHolder holder, int format, int width, int height) {
40      }
41
42      @Override
43      public void surfaceDestroyed(SurfaceHolder holder) {
44          thread = null;
45      }
46
47      private void surfaceInit() {
48          x = 15;
49          y = 20;
50          holder = getHolder();
51          holder.addCallback(this);
52      }
53
54      @Override
55      public void run() {
56          Canvas canvas = null;
57          Paint p = new Paint();
58          p.setColor(Color.YELLOW);
59
60          while (thread != null) {
61              try {
62                  canvas = holder.lockCanvas();
63                  if (canvas != null) {
64                      canvas.drawColor(Color.CYAN);
65                      canvas.drawCircle(x, y, r, p);
66                  }
67              } finally {
68                  if (canvas != null) {
69                      holder.unlockCanvasAndPost(canvas);
70                  }
71              }
72              x += xadd;
73              y += yadd;
74              if (x <= r || x > 500 - r)
75                  xadd = -xadd;
76              if (y <= r || y > 700 - r)
77                  yadd = -yadd;
78          }
79      }
80  }
```

MySurfaceView. java に上の図の色付きの枠で囲った箇所を付け加える。

1) 橙色の枠内　MySurfaceView クラスのフィールド ・・・ クラス内の複数のメソッドで使用。
SurfaceHolder のインスタンス holder，Thread のインスタンス thread，円の中心座標 x，y，円の中心座標の増分 xadd，yadd，円の半径 r

2) 紫色の枠内　コンストラクタ MySurfaceView から自作のメソッド surfaceInit を呼び出して初期化。

3) 明赤色の枠内　描画用スレッドを生成し，開始する。

4) 黄色の枠内　描画面が破棄された時点で，描画用スレッドに null を設定して，run メソッドの while の継続条件を満たさなくなる様にして，繰り返しを止める。
5) 水色の枠内　getHolder メソッドで SurfaceHolder（描画面を保持，制御するためのインタフェース）を取得し，addCallback メソッドでホルダーholder にコールバックインタフェースを付加する。
6) 濃赤色の枠内　Canvas を用いて，run メソッド内に描画処理を記述。

【描画に必要な 4 つの要素】

(1) ビットマップ　　ピクセル（画素）を保持
(2) キャンバス　　描画コール（ビットマップへの書き出し要請）に対応
(3) 描画プリミティブ　描画領域，パス，テキスト，ビットマップ等
(4) ペイント　　画像の色及びスタイル

① Canvas のインスタンス canvas, Paint のインスタンス p を作成し, p に黄色をセットする。
② thread が空でない，即ち描画面が破棄されない限り描画を繰り返す。
(1) lockCanvas メソッドは，描画面上でピクセルの編集を開始する。戻り値のキャンバスは描画面のビットマップへの描画に利用可能となる。
(2) drawColor メソッドは，キャンバスの描画面全体のビットマップに特定の色をセットする。ここではシアン色をセットしている。
(3) drawCircle メソッドは，中心座標，半径，ペイントを指定して，円を描く。ここでは，ペイントは色のみ指定して，スタイルを指定していないので，黄色く塗り潰される。線のみを描画するためには，「p.setStyle(Style.STROKE);」の記述を追加しておく必要がある。
(4) unlockCanvasAndPost メソッドは，ピクセルの編集を終了する。このメソッドが呼ばれると，画面上に描画面のピクセルが表示されるが，描画面の内容は失われる。
(5) x, y の座標を xadd, yadd だけ移動する。但し，x, y が半径以下になった場合及び x が 500−半径，y が 700−半径を超えた場合は xadd，yadd の符号を反転する。

それぞれの色の枠内を次に示す。

```
private SurfaceHolder holder;
private Thread thread;
private int x, y, xadd = 5, yadd = 10;
private final int r = 10;
```

```
surfaceInit();
```

```
thread = new Thread(this);
thread.start();
```

```
thread = null;
```

```
private void surfaceInit() {
    x = 15;
    y = 20;
    holder = getHolder();
    holder.addCallback(this);
}
```

```
Canvas canvas = null;
Paint p = new Paint();
p.setColor(Color.YELLOW);

while (thread != null) {
    try {
        canvas = holder.lockCanvas();
        if (canvas != null) {
            canvas.drawColor(Color.CYAN);
            canvas.drawCircle(x, y, r, p);
        }
    } finally {
        if (canvas != null) {
            holder.unlockCanvasAndPost(canvas);
        }
    }
    x += xadd;
    y += yadd;
    if (x <= r || x > 500 - r)
        xadd = -xadd;
    if (y <= r || y > 700 - r)
        yadd = -yadd;
}
```

MainActivity8.java のタブを開き，下の図の色付きの枠で囲った箇所を付け加える。(onCreate メソッドでは setContentView の引数は書き換える。)

activity_main8.xml × | MainActivity8.java × | MySurfaceView.java ×

```
1  package jp.ac.cuc.b8x0004.prog_8th;
2
3  import android.os.Bundle;
4  import android.support.v7.app.AppCompatActivity;
5
6  public class MainActivity8 extends AppCompatActivity {
7      MySurfaceView sv;
8
9      @Override
10     protected void onCreate(Bundle savedInstanceState) {
11         super.onCreate(savedInstanceState);
12         sv = new MySurfaceView (this);
13         setContentView(sv);
14     }
15 }
```

それぞれの色の枠内を次に示す。

```
MySurfaceView sv;
```

```
sv = new MySurfaceView (this);
setContentView(sv);
```

『保存』のアイコンをクリックして，全てのファイルを上書き保存し，実行ボタンをクリックする。

提出物：

1）MySurfaceView クラスのソースファイル MySurfaceView. java
2）アクティビティのソースファイル MainActivity8. java