応用Javaプログラミング
第6回

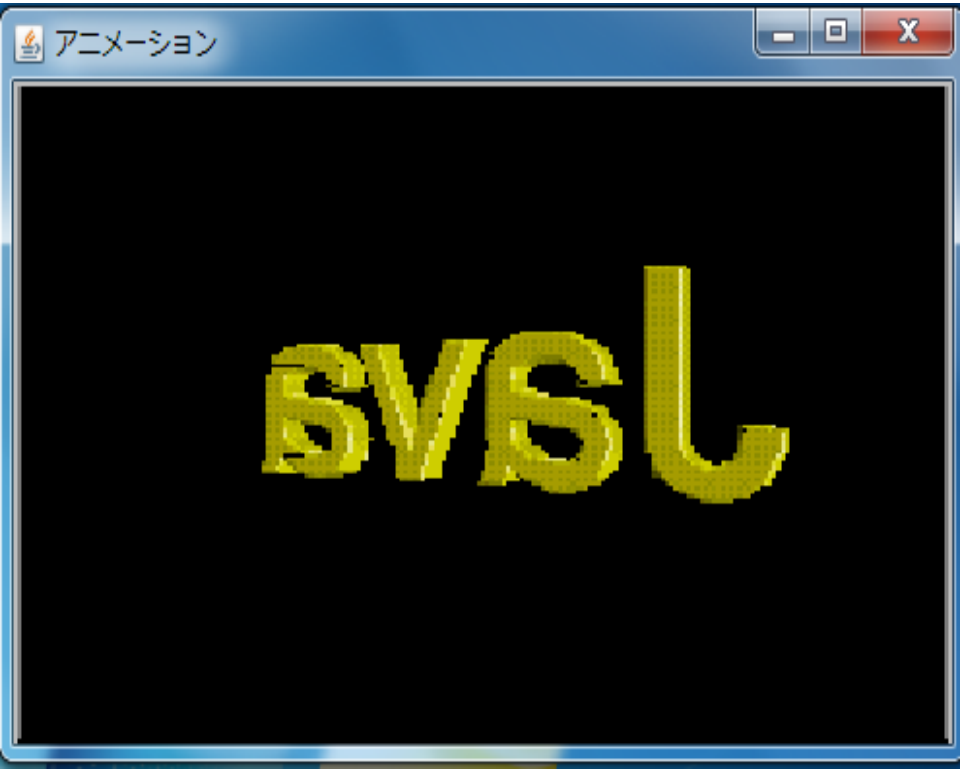
1. アニメーション
2. スレッド（入門編）
3. 画像の読み込み
4. メディア・トラッカー
5. Template Methodデザインパターン（再）

簡単なアニメーション

アニメーションとは，フレーム等を連続して表示することによって描かれているものが動いているかのように見せる技術である．

Javaを使ってアニメーションを作る際には，1枚ずつフレームを表示していきながらループさせればよいことになる．

次に，アニメーションを説明する．



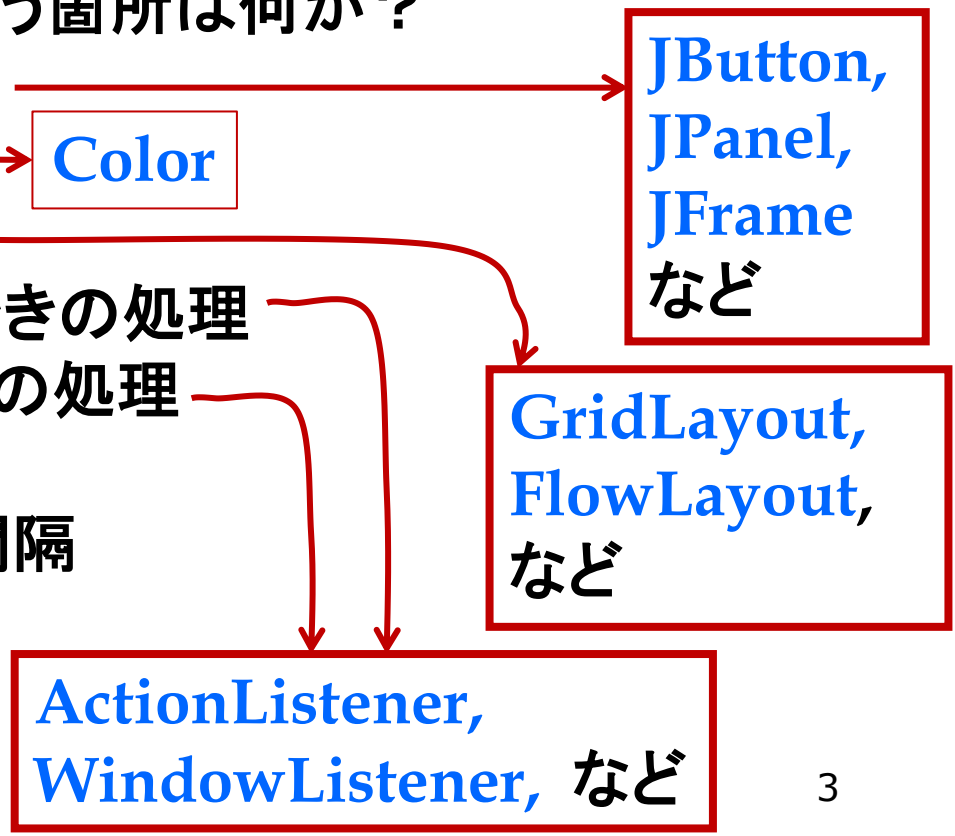
オブジェクト指向によるアプリケーション開発とは

変更されない箇所を軸に、頻繁に変更されるであろう箇所を
クラスやインターフェースに抽出するプログラミングスタイルである.

例: 簡単なアニメーション作成問題.

問: 頻繁に変更されるであろう箇所は何か？

答: ① 使われる部品や画像
② 部品の色
③ 部品の配置方法
④ ボタンをクリックしたときの処理
⑤ フレームを閉じたときの処理
⑥ アニメの起動
⑦ アニメの更新とその間隔
⑧ アニメの停止
…などなど



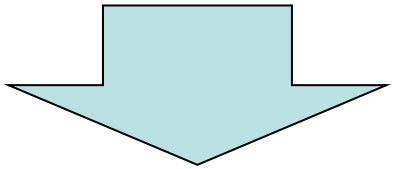
オブジェクト指向によるアプリケーション開発とは

変更されない箇所を軸に、頻繁に変更されるであろう箇所を
クラスやインターフェースに抽出するプログラミングスタイルである.

例：簡単なアニメーション作成問題.

問：変更されないであろう箇所は何か？
答：処理順序：初期化，部品の生成と配置，リスナーの生成と登録，アニメの起動.

この処理順序はほとんどの簡単なアニメアプリケーションにおいて共通と言えそうなので，固定しておきたい.



AbstractAnimeクラスを用意して，多くの簡単なアニメアプリケーションで使えるようにする.

javax.imageio.ImageIOクラス

このクラスは，ImageReader および ImageWriter を検索する静的な簡易メソッドを保持して，簡単な符号化を実行するクラスである．

主なメソッド
public static BufferedImage read(File input)
現在登録されているものの中から自動的に選択された ImageReader を使用して指定されたFileを復号化した結果として，BufferedImageを返す．登録されたImageReaderが，結果のストリームを読み込みできないような場合，nullを返す．
public static boolean write(RenderedImage img, String format, File output)
指定された形式をサポートする，任意のImageWriter を使用してイメージを指定されたFileに書き込む．書き込めなかった場合 falseを返す．（注：サポートされている形式： GIF, JPEG, PNG, BMP, WBMP. ）

アニメーション付きGIFファイルのように，一部の画像ファイルは複数のイメージからなる．ImageIOクラスのreadメソッドはその中の1つしか読まない．別の方法が必要．

1つの画像をファイルから読み出すには？

答え:

```
Image image ; // 画像を覚えるバッファへの参照を用意する.
try {
    File f = new File( 画像を保存したファイルの名前 ) ;
    image = ImageIO.read( f ) ; // 画像をファイルから読み出してバッファに置く.
} catch (IOException e) {
    JOptionPane.showMessageDialog(null, e) ; // 例外をダイアログで表示.
}
```

複数の画像をファイルから読み出すには？

```
Image imgs[ Num_Images ] ; // 画像を覚えるバッファへの参照の配列を用意.
try {
    for ( int i = 0 ; i < Num_Images ; i++ ) {
        File f = new File( i番目の画像を保存したファイルの名前 ) ;
        imgs[i] = ImageIO.read( f ) ; // 画像をファイルから読み出してバッファに置く.
    }
} catch (IOException e) {
    JOptionPane.showMessageDialog(null, e) ; // 例外をダイアログで表示.
}
```

java.awt.MediaTrackerクラス

画面に複数の画像を描画しようとしたときに，画像がロードされるので画像は少しずつ描画される．それでは，画像のロード時にちらつきが発生する．そのちらつきを防ぐ方法 として，MediaTrackerクラスのインスタンスを使い，必要な画像がすべて ロードされるまでアニメーションを開始しないようにすればよい．

コンストラクタ
public MediaTracker(Component comp)
指定されたコンポーネントの画像を監視するメディアトラッカーを生成.

主なメソッド
public void addImage(Image image, id)
このメディアトラッカーによって監視されているリストに画像を登録.
public void waitForAll( )
このメディアトラッカーによって監視されているすべての画像のロード（読み込み）を開始し，ロードが完了するまで待機する.

複数の画像をファイルから読み出すより良い方法

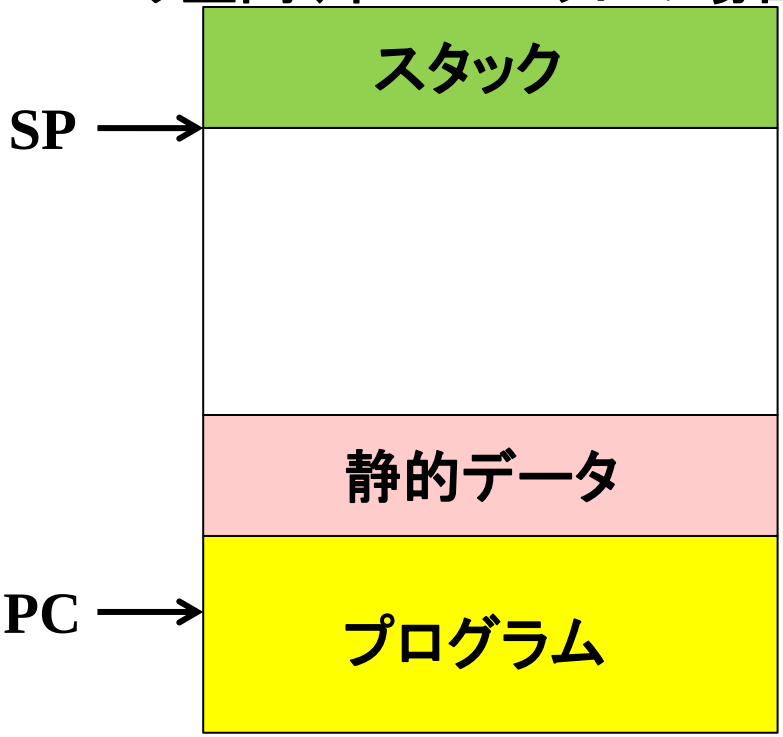
```
Image imgs[ Num_Images ] ; // 画像を覚えるバッファへの参照の配列を用意.
MediaTracker tracker = new MediaTracker(this);  // 画像を監視するための
                                                // メディアトラッカーを生成
try {
    for ( int i = 0 ; i < imgs.length ; i++) {   //画像をファイルから読み込む
        File file = new File( i番目の画像を保存したファイル名 ) ;
        imgs[i] = ImageIO.read( f ) ; // 画像をファイルから読み出してバッファに置く.
        tracker.addImage(imgs[i], i); // ID番号iで画像をメディアトラッカーに登録
    }
    tracker.waitForAll( ) ; // すべての画像のロードが完了するまで待つ
} catch ( Exception ex ) {
    JOptionPane.showMessageDialog(this, ex) ;
}
```

スレッド（thread）とは

A *thread* is a single sequential flow of control within a program.

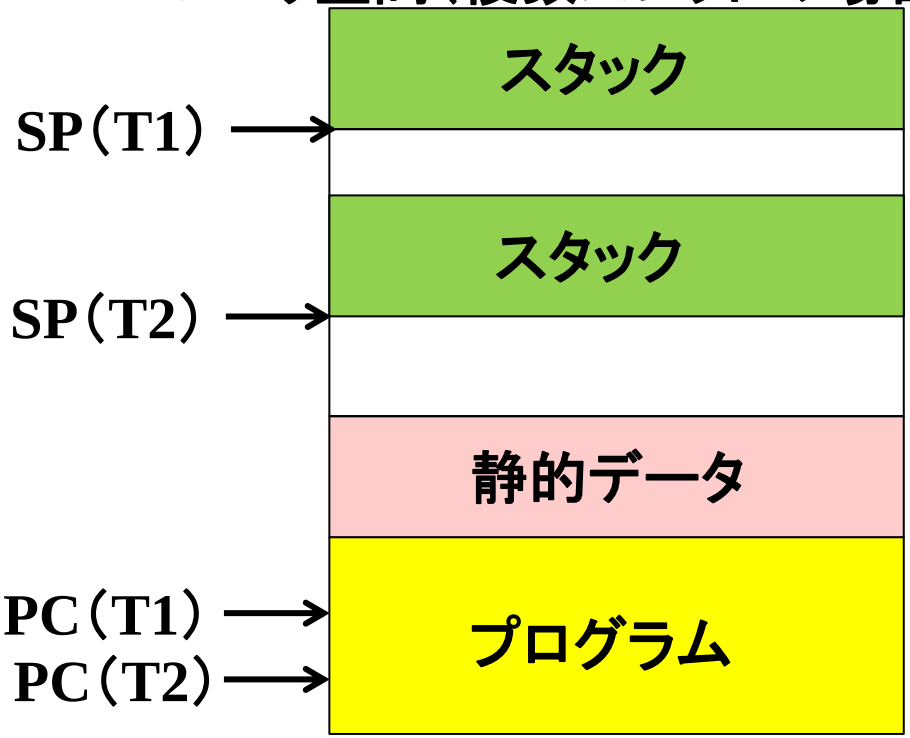
（Javaにおける）スレッドとは，プログラム中の独立した制御の流れのことである．

ある実行中のプログラムのメモリ空間（単一スレッドの場合）



PC：プログラムカウンター
SP：スタックポインタ

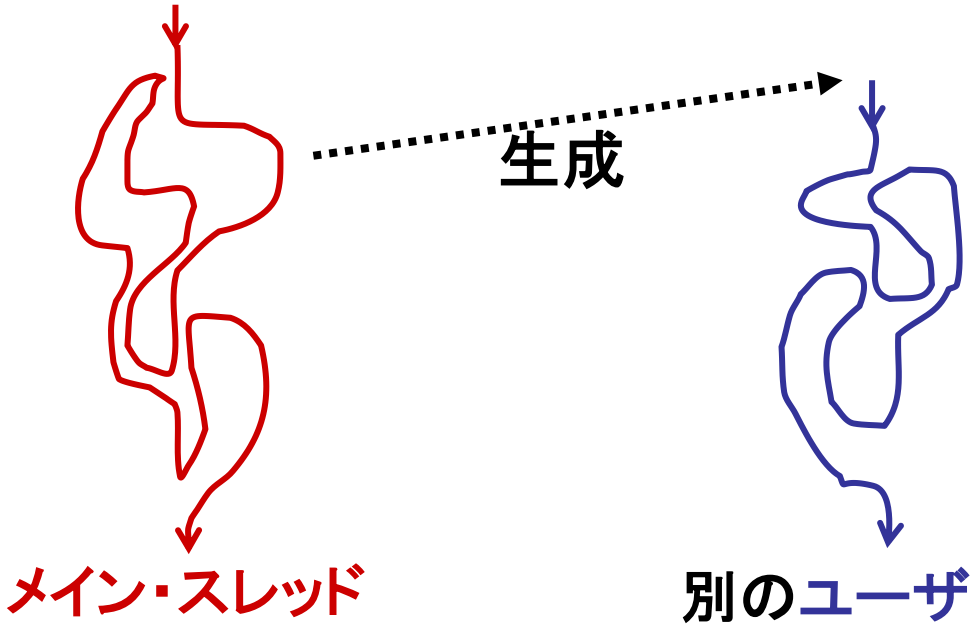
ある実行中のプログラムのメモリ空間（複数スレッドの場合）



PC（T1）：スレッドT1のプログラムカウンター
SP（T1）：スレッドT1のスタックポインタ

マルチ・スレッド（multi-thread）のプログラムとは

その実行時に，（そのプログラムを実行するユーザスレッド以外に）別のユーザスレッドを（1つ以上）生み出すプログラムのこと.



ユーザプログラムの実行はそのプログラムから生み出されたすべてのユーザスレッドが終了したときである.

ポイント：ユーザ・スレッド間でコミュニケーションが可能.

ユーザ・スレッド以外のスレッド

ユーザ・スレッド以外に，JVMが自動的に生み出すスレッドが沢山ある．

まず，ユーザーのプログラムで不要になったメモリを自動的に解放してくれる「garbage collection thread」がある．

また，JPanelクラスのインスタンスの paintComponentメソッドなどを実行する「event dispatch thread」 がある．

GUI部品の表示，追加，削除，更新などの操作は「event dispatch thread」に実行してもらったほうがよいが，このスレッドに重たい仕事をさせてはだめ．

```
EventQueue.invokeLater ( // 次の仕事をEDTにやってもらう
    new Runnable( ) {
        public void run( ) {
            ... // event dispatch thread に実行してもらう文をここに書く.
        }
    }
) ;
```

この部分で1つの仕事（無名仕事クラスのインスタンス）を生成する．

メソッドを実行するスレッドを知るには，そのメソッドの先頭に下記の2文

```
String name = Thread.currentThread( ).getName( );
System.out.println("The thread is: " + name);
```

を書けばよい．

ユーザ・スレッドの定義と生成と起動（方法1）

まず，自分のスレッドクラス（Threadクラスの子クラス）を
下記のように定義する：

```
private  class  MyThread  extends Thread {
    @Override
    public void run( ) {
       // ここに，このスレッドで行いたい処理を書く
    }
}
```

注意：runメソッドが元々Threadクラスにあるが，
自作スレッドクラスでそれを
オーバーライドすることが必須．

定義済みの自作スレッドクラスのインスタンス（スレッド）を生成して起動：

```
class  MyApplication {
...
   // 自作スレッドを生成する
   Thread t = new MyThread( );
...
   // 生成したスレッドを起動する
   t.start( );
...
}
```

Threadクラスのインスタンス・メソッドの1つで，生成されたスレッドを起動．

ユーザ・スレッドの定義と生成と起動（方法2）

まず，スレッドに実行してもらいたい仕事（タスク）クラス（Runnableインターフェースを実装したクラス）を下記のように定義する：

```
private class MyTask implements Runnable {
    public void run( ) {
        // ここに，仕事の中身（スレッドで行いたい処理）を書く
    }
}
```

注意：
Runnableインターフェースは runメソッドのみを持つ．

定義済みの自作仕事クラスを使って仕事インスタンスを生成して，
さらにその仕事インスタンスを実行するスレッドを生成して起動：

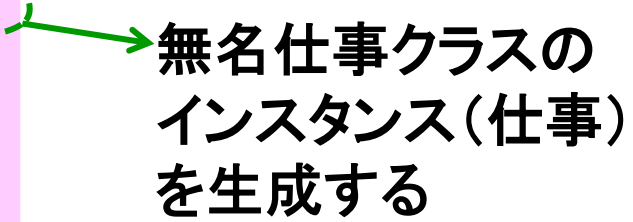
```
class MyApplication {
...
    Runnable task = new MyTask( ); // 自作仕事のインスタンスを生成
    Thread t = new Thread( task );// 生成した仕事を実行するスレッドを生成
...
    t.start( ); // 生成したスレッドを起動する
...
}
```

```
// この2行は次の1行にまとめてもよい
// Thread t = new Thread( new MyTask( ) );
```

ユーザ・スレッドの定義と生成と起動（方法2の続き）

ある仕事をプログラムの1カ所でしかスレッドにやってもらわない場合は，無名仕事クラスを使う：

```
class  MyApplication {
…
   // スレッドを生成する
   Thread t = new Thread( new Runnable( ){
      public void run( ) {
         // ここに，このスレッドで行いたい処理を書く
      }
   } );
…
   // 生成したスレッドを起動する
   t.start( );
…
}
```



java.lang.Threadクラス

主なコンストラクタ
public Thread( ) 
新しいスレッドを生成する．これだけでは，スレッドがまだ起動しない．
public Thread(Runnable task) 
指定された仕事を実行するスレッドを生成する．これだけでは，スレッドがまだ起動しない．

java.lang.Threadクラス（続き）

主なメソッド
public static Thread currentThread( ) 
現在実行中のスレッド（の参照）を返す.
public static void yield( ) 
現在実行中のスレッドを一時的に休止させ，他のスレッドが実行できるようにする.
public static void sleep(long millis) 
現在実行中のスレッドを，指定されたミリ秒数の間，一時的に実行を停止させる.
public void start( ) 
このスレッドの実行を開始．JVMはこのスレッドのrunメソッドを呼び出す.
public void run( )
このメソッドが何も行わない．∴子クラスでオーバーライドする必要がある.
public void interrupt( ) 
このスレッドに割り込む．大抵の場合，このスレッドの割り込みステータスが設定される．次頁のisInterruptedと合わせてスレッドを止めるのに使われる. 

java.lang.Threadクラス（続き）

主なメソッド（続き）
public boolean isInterrupted( ) 
このスレッドが割り込まれている場合 true を，そうでない場合 falseを返す.
public final boolean isAlive( )
このスレッドが生存している場合 true を，そうでない場合 falseを返す.
public final void setPriority(int newPriority)
このスレッドの優先順位を変更する.
public final int getPriority( )
このスレッドの優先順位を返す.
public final void setName(String name)
このスレッドの名前を引数 name に等しくなるように変更する.
public final String getName( ) 
このスレッドの名前を返す.

java.lang.Threadクラス（続き）

主なメソッド（続き）
public final ThreadGroup getThreadGroup( )
このスレッドが所属するスレッドグループを返す.
public final void join(long millis)
このスレッドが終了するのを，最高で millis ミリ秒待機する．0のタイムアウトは永遠に待機することを意味する.
public final void join( )
このスレッドが終了するのを待機する.
public final void setDaemon(boolean on)
このスレッドを，デーモンスレッドまたはユーザスレッドとしてマークする.
public final boolean isDaemon( )
このスレッドがデーモンスレッドであるかどうかを判定する.

java.lang.Threadクラス（続き）

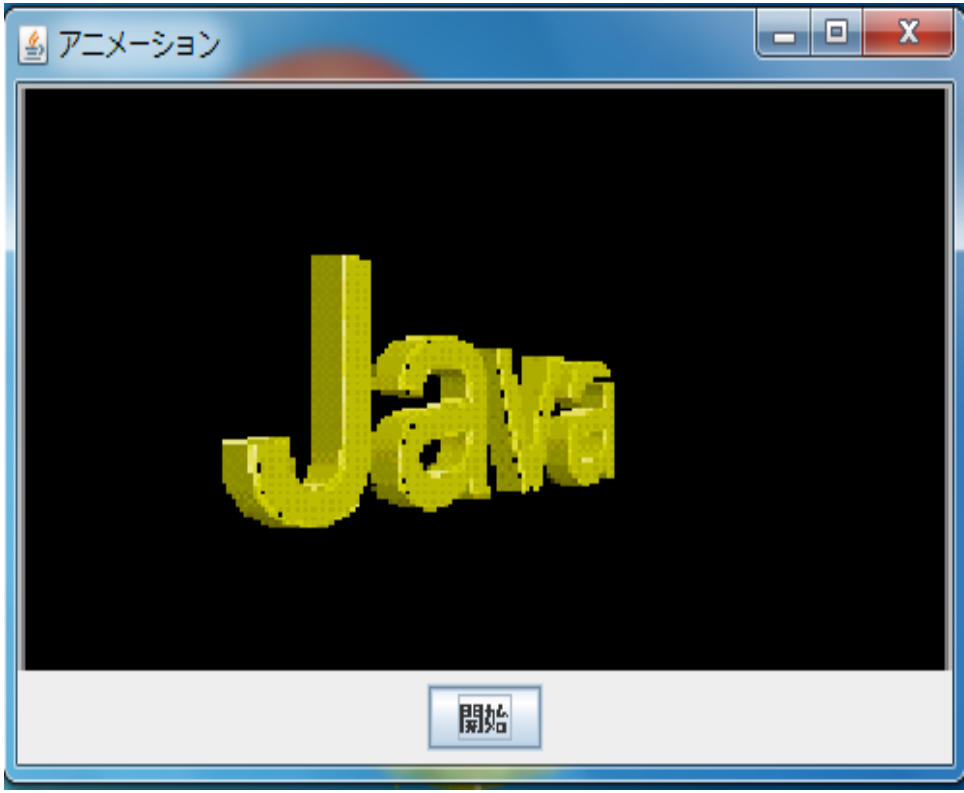
主なメソッド（続き）
public String toString( )
スレッドの名前，優先順位，スレッドグループを含むこのスレッドの文字列表現を返す．
public long getId( )
このスレッドの識別子を返す．スレッドIDは一意である．
public Thread.State getState( )
このスレッドの状態を返す．
public final void setDaemon(boolean on)
このスレッドを，デーモンスレッドまたはユーザスレッドとしてマークする．
public final boolean isDaemon( )
このスレッドがデーモンスレッドであるかどうかを判定する．

注意：使ってはいけないメソッド：stop, suspend, resume, ...

アニメーションの例

アニメーションが動いてる

アニメーションが止まっている



1つのボタンで「停止」と「開始」の2つの機能を果たさせる.

Template Method デザインパターン（再）

他のアニメにも使えそうな部分を抽象化した
（Template Method デザインパターン を利用）.

結果のプログラムは AbstractAnime.java と
SimpleAnime2.java にある.

上記の抽象化に基づいて，デジタル時計を簡単に作れる.
結果のプログラムは Clock.java にある.

3つのスレッドの関係（アプリケーションの場合）

メイン・スレッド

```
…
    // コンストラクタの定義
    public SimpleAnime2( ) {
        …
        animeThread.start( ) ; // アニメスレッドを起動
    } // コンストラクタの終わり
…
```

起動

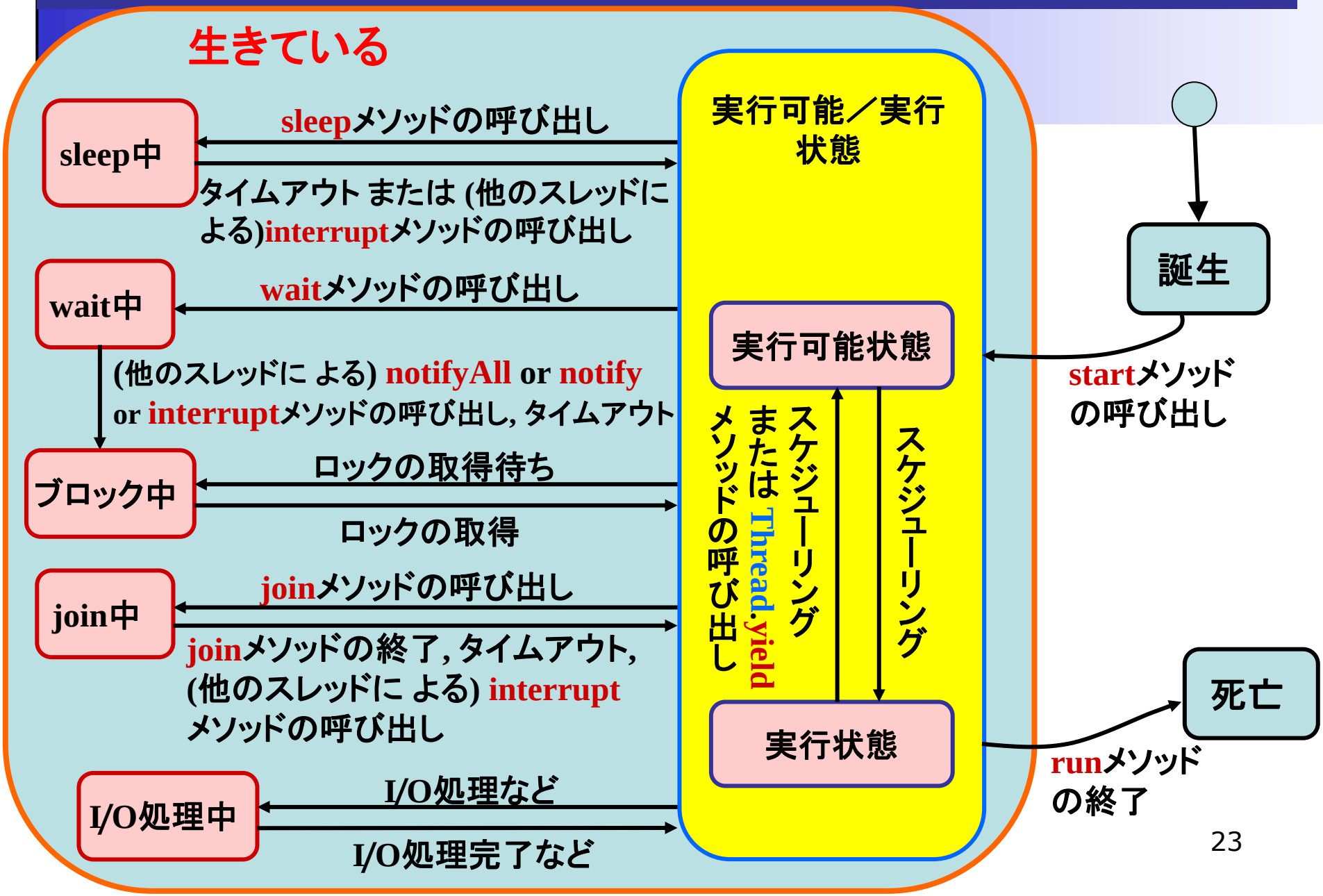
アニメーション・スレッド

```
…
    public void animeLoop( ) {
        try {
            while ( running ) {
                updateAnime() ;
                repaint( ) ; //画面の再描画を要求
                Thread.sleep( delay ) ;  // ポーズをかける
            }
        } catch (InterruptedException e) { … }
    }
…
```

描画要求
（0.5秒毎に
アニメ・
スレッドから
出される）

event
dispatch
thread

スレッドのライフサイクル



演習課題1

下記のように，SimpleAnime2.java を継承して，アニメーションの速度を変える ためのスライダーを追加せよ.

演習課題2

演習課題1でスライダーを使ってアニメの速度を変えたが，この課題ではアニメの速度を別のスレッド（速度変更担当スレッド）からランダムに変えるようにしたい．ただし，停止ボタンをクリックしたときにアニメスレッドだけでなく，速度変更担当スレッドも止め，開始ボタンをクリックしたときにアニメスレッドだけでなく，速度変更担当スレッドも新規作成して起動する．
また，SimpleAnime2.java を継承してクラスを作成すること．