卒業ワークショップB

# 1 心得

- 自分の将来に向けての備え
- プログラムなどの課題を完成させるだけが目的ではない
- 学んだ知識などの活用力をつける
- チェックポイントはあるが，ゴールはない．安易な近道もない．
- 欠席や遅刻は厳禁 (どうしても欠席する必要がある場合は事前に連絡する)

**プログラミング・工作・加工** – 美しく書くこと (美的センス)

- 論理的に考える (コンピュータ [機械] は情緒的な動作をしない)
- 「動けば良い」ではない
- 他人が読んでもわかり易い．修正・拡張・保守が容易なプログラム
- 考えが一貫している
- 変数名 – その役割りが推測できる名前
- 段下げ
- A4 用紙 1 枚以内 (長くなる関数は機能毎に分割する)
  1 つの関数が、できるだけ 1 枚に収まるようプログラムを書く
- コメント分，コメント行を活用する
  無駄なコメントは不要

# 2 目標

**このプリントで学ぶことでの達成目標** 統合開発環境である NetBeans IDE を使用して次の項目を含む Java 言語の基本的なプログラムが作成できるようになる．

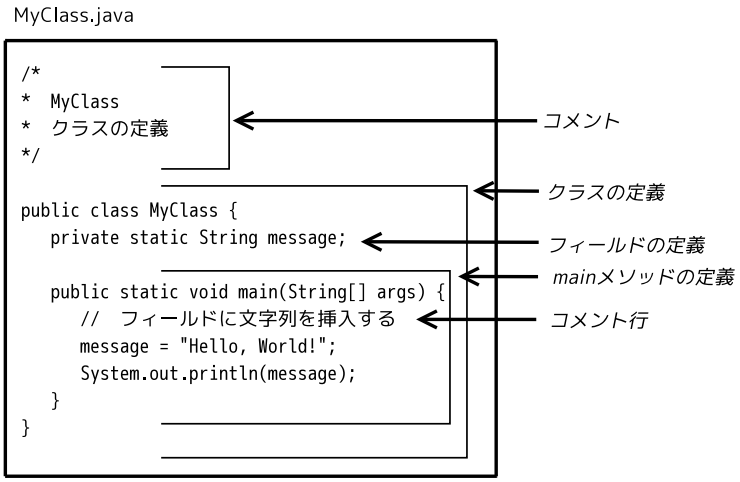
- Java Swing などの GUI の利用
- イベント処理
- 図形の描画
- 例外処理
- マルチスレッドによる並列処理
- アニメーション

# 3 Java 言語の基礎に関して

**文法** – C 言語に類似.

- わからない，知らなかったと思ったら自分で調べる
- オブジェクト指向言語
  - クラス，メソッド，変数 (フィールド)

# 4 プログラムの構造




- ファイル名とクラス名は一致させる必要がある，ファイル識別子は java とする.
- クラス内の main メソッドから処理は開始される.

# 5 データの型

- 基本型

| 型名 | 内容 | 値の範囲など |
|---|---|---|
| byte | 1 バイト整数 | $-128$ から $+127$ |
| short | 2 バイト整数 | $-32768$ から $+23767$ |
| int | 4 バイト整数 | $-2^{31}$ から $2^{31}-1$ |
| long | 8 バイト整数 | $-2^{63}$ から $2^{63}-1$ |
| float | 4 バイト浮動小数点数 | およそ 7 桁の精度 |
| double | 8 バイト浮動小数点数 | およそ 15 桁の精度 |
| char | 2 バイト文字 | ASCII および Unicode 文字 |
| boolean | 論理型 | true または false の 2 値 |

- 参照型

  配列型，クラス型，インターフェイス型

**プログラム例** – 演算プログラム

```
1  public class Enzan {
2     public static void main(String[] args) {
3        int a;
4        a = 4;
5        int b = 3;
6        int wa = a + b;
7        System.out.println("wa " + wa};
8        System.out.println("sa " + (a - b));
9     }
10 }
```

- クラスの中に main 関数 (メソッド) を定義する
- 実行は main メソッドから開始される

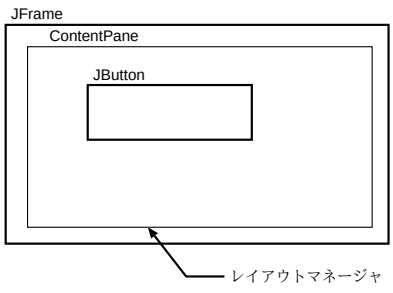
# 6 GUI プログラミング

Swing ツールキットを使用した GUI プログラミング
GUI を利用する場合は

1. ボタンなど部品の配置
2. 処理のアルゴリズム

の 2 つをプログラミングする.
部品を配置するだけで処理を含まないプログラムを以下に例示する.
[例 1] JFrame と JButton

```
1  import javax.swing.*;                                  // javax.swing 下のライブラリを利用する
2  import java.awt.*;                                     // java.awt 下のライブラリを利用する
3  public class JFrameTest
4  {
5     public static void main(String[] args)
6     {
7        JFrame  f = new JFrame("Swing -- JFrame");      // フレームを生成
8        JButton b = new JButton("MyButton");           // ボタンを生成
9        f.setSize(300, 200);
10       f.getContentPane().setLayout(new FlowLayout()); // レイアウトマネージャの設定
11       f.getContentPane().add(b);                     // フレームにボタンを追加
12       f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
13       f.setVisible(true);                            // フレームを表示
14    }
15 }
```



**処理内容など**

- 第 1,2 行： Java Swing クラスライブラリおよび，Java AWT ライブラリを利用する場合に指定する
- 第 5 行： 処理は main メソッドから開始される
- 第 7 行： フィールド名 (変数名) を f として，フレームを生成する．
- 第 8 行： ボタンを生成 (フィールド名は b)
- 第 9 行： フレーム f のサイズを指定
- 第 10 行： フレーム f の下にレイアウトマネージャを生成し，レイアウトを Flowlayout とする
- 第 11 行： フレーム f にボタン b を追加する
- 第 12 行： 終了時のフレームに対する処理
- 第 13 行： フレームを可視化 (表示) する

**実行までの操作**

1. エディタを使用して，ファイル「JFrameTest.java」にプログラムを書き込む
2. コンパイル　　`javac JFrameTest.java`
3. 実行　　`java JFrameTest`

[注意] ファイル名はクラス名と一致していなければならない．また，拡張子は.java とする必要がある．

**レイアウトマネージャ** – ボタンなど部品の配置を管理する

## 6.1 イベントとイベントリスナ

- イベント – マウス，キーボートからの入力，タイマーなど
- イベントリスナ – イベントを受け取って処理するオブジェクト

[例 2] ボタンがクリックされるとフレームの色を変える

```
import javax.swing.*;
import java.awt.*;
import java.awt.event.*;
public class SwingEventTest
{
    public static void main(String[] args)
    {
      JFrameListener f = new JFrameListener("Swing event test");
      JButton b = new JButton("Change Color");
      f.getContentPane().add(b);
      b.addActionListener(f);      // ボタンにイベントリスナを設定する
      f.setVisible(true);
    }
}
// JFrame を拡張し，イベントリスナを実装したクラスを定義する
class JFrameListener extends JFrame implements ActionListener
{
    public JFrameListener(String title)  // コンストラクタ
    {
      setTitle(title);
      setSize(300,200);
      getContentPane().setBackground(Color.BLUE);
      getContentPane().setLayout(new FlowLayout()); // レイアウトマネージャの設定
      setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }

    // ボタンクリックが発生したときに呼び出されるメソッドを定義する
    public void actionPerformed(ActionEvent e)
    {
      if (getContentPane().getBackground() != Color.RED)
          getContentPane().setBackground(Color.RED);
      else
          getContentPane().setBackground(Color.BLUE);
    }
}
```

# 7 NetBeans IDE の利用

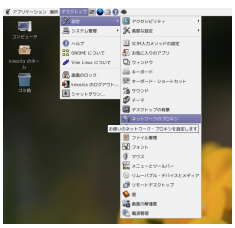
NetBeans IDE: 統合開発環境

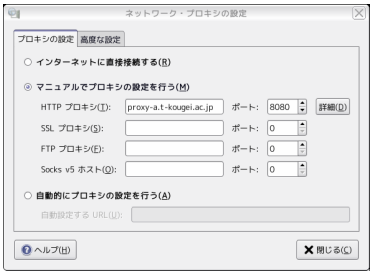
- 始めて使用する前に必要な設定

  プロキシ・サーバの登録

  1. 「デスクトップ」⇒「設定」⇒ 「ネットワークのプロキシ」
  2. HTTP プロキシ「proxy-a.t-kougei.ac.jp」，ポート「8080」と設定する．

  最初に一度設定すれば，以後，この操作は不要

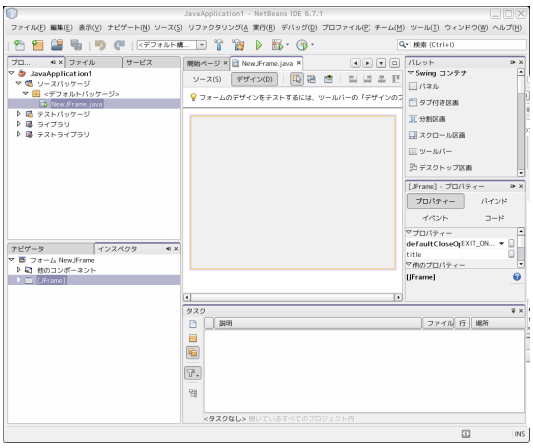





- 起動

  「アプリケーション」⇒「プログラミング」⇒「NetBeans IDE 6.7.1」

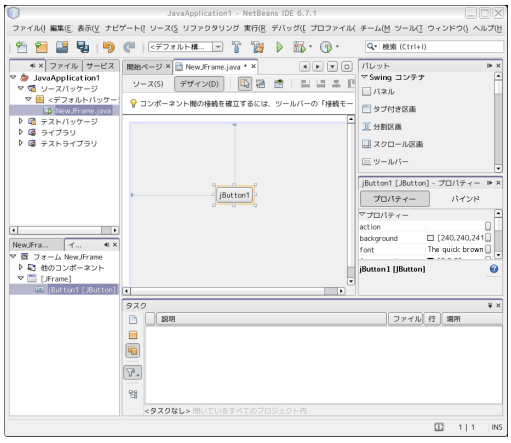

1. プロジェクトの設定
   (a) 「ファイル」⇒「新規プロジェクト」
   (b) カテゴリーに「Java」，プロジェクトに「Java アプリケーション」を選択して,「次へ >」をクリックする
   (c) プロジェクト名を入力し,「主クラスのクラスを作成」のチェックを外して,「完了」をクリックする
2. ファイルの設定
   (a) 「ファイル」⇒「新規ファイル」
   (b) カテゴリーとして「Swing GUI フォーム」を選択し，ファイルの種類として「JFrame フォーム」を選び,「次へ >」をクリックする
   (c) 名前と場所では，ファイル名を入力して,「完了」をクリックする



- 「デザイン」タブでデザイン・エディタを開き部品の配置などを行なう
- 「ソース」タブをクリックするとソース・エディタが開く

ボタンクリックでフレームの色が変わる (前の例題と同じ動作をする) プログラムを作る.

1. ボタンの配置
   パレット内の Swing コントロールからボタンを取り出し (マウスを左クリックする)，デザインエディタ画面のフレーム内に置く (ボタンを置く場所でマウスを左クリック)



ボタンの名前は「jButton1」となっている.

2. イベントとイベントハンドラの登録

   ボタン (jButton1) を右クリックして，表示されたメニューから

   「イベント」⇒「アクション」⇒ 「actionPermormed」

   を選択する.


3. テキスト・エディタで `jButton1ActionPerformed` メソッドを作成する：

   ```
   // TODO add your handling code here:
   ```

   の部分にキーボードから以下のプログラム部分を入力して，メソッド `jButton1ActionPerformed` を完成させる

   ```
   if (getContentPane().getBackground() != Color.RED)
       getContentPane().setBackground(Color.RED);
   else
       getContentPane().setBackground(Color.BLUE);
   ```

4. デバッグ

   「デバッグ」⇒「ファイルをデバッグ」

   でデバッガが起動し，実効が開始される.

5. ソースプログラムの印刷

   (a) 「ファイル」⇒「印刷」

   (b) 「印刷オプション」を選択し，背景色を消し，印刷スケールを紙に合わせる：

   - 「背景色」⇒「AWT パレット」⇒「white」⇒「了解」
   - 「幅を合わせる (1 ページ)」をチェック
   - 「了解」

   (c) 「印刷」

---

[課題] NetBeans IDE を使用して出来上がったプログラムのソースコードを読んで，どのようなプログラムが作られたか調べる.

---

(注意) ソーププログラム中の知らない機能やキーワードは，本やインターネットで調べる.

**web マニュアルの活用** Sun のホームページから Java の各クラスについて調べる.

`http://sdc.sun.co.jp/java/docs/j2se/1.4/ja/docs/ja/api/`

索引から Color を調べる：

java.awt
クラス Color

とあるので， java.awt パッケージ内に置かれたクラスであることがわかる．
また,「フィールドの詳細」を読むと，幾つかの色が定数として定義されている．

| キーワード | 意味 |
|---|---|
| import | |
| extends | |
| public | |
| private | |
| nwe | |
| static | |

# 8 NetBeans を利用した GUI プログラミング

[例 8-1] 例として，お絵書きプログラムを作成する．

**機能** – 作成するソフトの機能

- マウス・ドラッグにより絵を描く
- ペンの色を選べる
- ペンの太さを選べる
- 全画面の消去機能を持つ

### プログラム作成のための操作

1. プロジェクトを新規に作る (プロジェクト名： MyPaint)

   **カテゴリ** Java

   **プロジェクト** Java アプリケーション

   **プロジェクト名** MyPaint

   **注意事項** 「名前と場所」の設定に於て,「主クラスを作成」のチェックを外す (GUI アプリケーションの作成では，このチェックを外すことが必要)

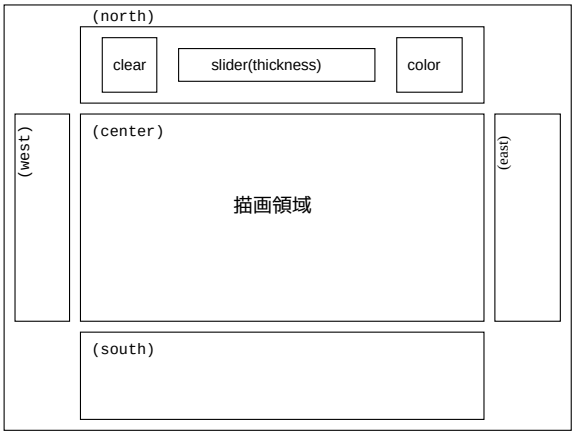
2. 新規にファイル (クラス) を作成する (クラス名： MyPaint)

   **カテゴリ** Swing GUI フォーム

   **ファイルの種類** JFrame フォーム

   **クラス名** MyPaint

3. デザインエディタによる部品の配置




   以下の手順で，図に示すような各部品を配置する．

   (a) レイアウトの設定 (BorderLayout)
   デザインエディタ内に表示されている JFrame 内でマウスを右クリックして表示されるメニューから「レイアウトを設定」⇒「ボーダレイアウト」を選択する

   (b) パレット内の Swing コンテナよりパネルを選び，JFrame の上部 (north) と中央部 (center) に各 1 つのパネルを配置する．
   以下の説明では，上部のパネルを jPanel1 ，中央部のパネルを jPanel2 と呼ぶ

(c) パレット内の Swing コントロールよりボタンとスライダを取り出し次のように配置する．
上部パネル jPanel1 内にボタン 2 つとスライダ 1 つを配置する．

(d) 色を選択する際に使用するウィンドウとして，カラーチューザを次の操作で「他のコンポーネント」に配置する．
パレット内の Swing ウィンドウ下の「色を選択」をクリックしてカラーチューザを取り出し，フレームの外，あるいは,「他のコンポーネント」にドラッグ&リリースする．
このとき，インスペクタ画面には次のように部品の配置が階層化されて表示される．

- フォーム MyPaint
  - 他のコンポーネント
    - jColorChooser1 [JColorChooser]
  - `[JFrame]`
    - BorderLayout
    - jPanel1 [JPanel]
      - jButton1 [JButton]
      - jButton2 [JButton]
      - jSlider1 [JSlider]
    - Jpanel2 [JPanel]

4. イベントの登録 I(ボタン操作)

(a) jButton1 に対するイベントハンドラを作成する

i. デザインエディタ上でボタンを右クリックして表示されるメニューから「イベント」⇒「Action」⇒「actionPerformed」を選択する．
表示されるソース・エディタ内の `jButton1ActionPerformed` メソッド (関数) は，後で，コードを書き入れる．

(b) デザインエディタに戻り，上と同様にして，jButton2 に対するイベントハンドラを作成する

5. イベントの登録 II(描画操作)

描画のための jPanel2 内でのマウス操作に対するイベントとして，mousePressed, mouseDragged を登録する

(a) jPanel2MousePressed メソッド：
jPanel2 内でマウスボタンを右クリックして,「イベント」⇒「Mouse」⇒ 「mousePressed」

(b) jPanel2MouseMouseDragged メソッド：
イベント ⇒MouseMotion→MouseDragged

6. ソース・コーディング

マウスをドラッグするとドラッグした座標に沿って短い直線を描くことで，マウスの軌跡を描画する．このために次の変数とメソッド (関数) をコーディングする．

- 変数
  - startPoint, endPoint
    線を描く際の始点と終点の座標
  - graphics
    グラフィックスコンテキスト (描画のための情報を格納したオブジェクト)

ソースコードの最後にボタンなどの変数宣言部分がある．この後ろに次の変数宣言を追加する：

```
// End of variables declaration           この後ろに追加
private Point startPoint, endPoint;   // 始点と終点
private Graphics2D graphics;          // 描画用グラフィックス
```

さらに，グラフィックスライブラリパッケージを利用することから，プログラムの先頭に次の import 文を追加する：

```
import java.awt.*;        // Java AWT ライブラリ
import java.awt.geom.*;
```

- 初期設定

  処理の初期の段階でグラフィックスコンテキストを graphics に記憶させる．
  このためには，MyPaint のコンストラクタに次の 1 行を追加する：

```
public MyPaint() {
    imitComponents();
    graphics=(Graphics2D)jPanel2.getGrachics();  //  この 1 行を追加する
}
```

イベントを登録した際に省略したイベントハンドラを完成させる：

- ボタン処理 1

  clearRect により画面を消去する

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    graphics.clearRect(0,0,jPanel2.getWidth(),jPanel2.getHeight());
}
```

- ボタン処理 2

  カラーチューザで選択した色を color に記憶する

```
private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    Color c = jColorChooser1.showDialog(this,"色を選択",Color.BLACK);
    graphics.setColor(c);
}
```

- マウスボタン処理
  - マウスボタンが押された場所の座標を startPoint に記憶
  - スライダの値から描画する線の太さを決める

```
private void jPanel2MousePressed(java.awt.event.ActionEvent evt) {
    startPoint = evt.getPoint();
    graphics.setStroke(new BasicStroke((float)jSlider1.getValue()));
}
```

- マウスドラッグ処理

  startPoint と endPoint の 2 点の間の直線を描く

```
private void jPanel2MouseDragged(java.awt.event.ActionEvent evt) {
    endPoint = startPoint;
    startPiont = evt.getPoint();
    graphics.draw(
        new Line2D.Double(startPoint, endPoint));
}
```

### 8.1 調査項目

- AWT(Abstruct Window Toolkit), Swing

  Java 上で GUI(Graphics User Interface) 機能を提供するクラス・ライブラリ

上記以外，以下の項目についても調べよ．

- この章の例題を実際にプログラミングして実行することで，動作を理解すること
- レイアウトとレイアウト・マネージャ
- Graphics2D クラス

  2 次元の画像や図形を取り扱うメソッド (関数) を含むクラス．

  このクラスが含むメソッドの機能についてオンラインマニュアルを一読しておくこと．
- java.awt.geom パッケージ

  2 次元図形を描くための図形クラスを含むパッケージであり．次のクラスを含む：

  - Line2D
  - Rectangle2D
  - RoundRectangle2D
  - Ellipse2D
  - Arc2D
  - QuadCurve2D (2 次元曲線)
  - CubicCurve2D
  - GeneratePath

  このクラスの基本的な使いかた (例えば直線の描き方) について理解すること．

## 9 図形の描画

**paint** – JFrame や JPanel に図形を描くには，図の描画手続きをメソッド paint に記述する．

**repaint** – 処理の途中で図形を再描画がする必要が生じた場合は，paint ではなく repaint メソッドを呼び出す．

[例 9-1] ボタン 1 をクリックすると図形を描き，ボタン 2 をクリックすると図形を消去する．

以下の説明では，jButton1ActionPerformed が描画のためのイベントハンドラ，JButton2ActionPerformed が画面消去のためのイベントハンドラとする．また，jPanel1 が描画用のパネルとする．

```
private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
    repaint();
}

private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
    Graphics g = jPanel1.getGraphics();
    g.clearRect(0, 0, jPanel1.getWidth(), jPanel1.getHeight());
}

public void paint(Graphics gr) {
    super.paint(gr);
    Graphics g = jPanel1.getGraphics();
    g.setColor(Color.RED);
    g.drawRect(40, 10, 100, 25);
    g.drawRoundRect(40, 50, 100, 25, 15, 15);
    g.drawOval(40, 100, 100, 25);
    g.fillRect(160, 10, 100, 25);
    g.fillRoundRect(160, 50, 100, 25, 15, 15);
    g.fillOval(160, 100, 100, 25);
}
```

上のプログラム部分では,

- jButton1ActionPerformed 中で repaint メソッド (関数) が呼び出され，再描画が行なわれる
- JButton2ActionPerformed 内では，jPanel1 に描かれた図が消去される
- paint は描画のための手続きであり，6 個の基本的な図形を描く命令を含んでいる

[課題] NetBeans IDE を使用して，このプログラムを作成し，実行せよ.
以下の手順でプログラムを作成することができる.

- NetBeans IDE を利用してプログラムを作成する
- 図形描画用のパネルとボタンを置くためのパネルを配置する
- 一方のパネル内に描画と消去のための合計 2 つのボタンを配置する.
  配置する部品の階層は次のようになる
  - JFrame
    - jPanel1
    - jPanel2
      - jButton1
      - jButton2
- 描画と消去のためのイベントを登録する
  `jButton1ActionPerformed`
  `jButton2ActionPerformed`
- 上に示したプログラムコードにより図形の描画，および，消去を行なう

### 9.1 調査項目

- paint と repaint による描画の仕組みを確認する

# 10 Image

Image オブジェクトは写真などのイメージや描画内容の保存，バッファリングに使用される．
ここでは，前出のお絵書きソフトに Image オブジェクトを使用してみる．

- マウスドラッグで Image に絵を描く
- paint メソッドで Image をグラフィクスに描画する

このようにすると，ウィンドウが後ろに隠れた場合でも，前面に表れた場合に再描画が可能となる．
このための変更点を以下に示す．

- 変数にイメージ保存のための buffer を追加する

```
// End of variables declaration          この後ろに追加
private Point startPoint, endPoint;   // 始点と終点
private Graphics2D graphics;          // 描画用グラフィックス
private Image buffer;                 // バッファ(追加)
```

- コンストラクタ MyPaint

```
    public MyPaint() {
        initComponents();
        buffer=createImage(jPanel2.getWidth(),jPanel2.getHeight()); //  追加
        graphics = (Graphics2D)buffer.getGraphics();   // 変更
    }
```

- イベントハンドラ

```
  private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        graphics.clearRect(0, 0, jPanel2.getWidth(), jPanel2.getHeight());
        repaint();   // バッファ消去後に再描画 (追加)
    }
    private void jPanel2MouseDragged(java.awt.event.MouseEvent evt) {
        endPoint = startPoint;
        startPoint = evt.getPoint();
        graphics.draw(
                new Line2D.Double(startPoint, endPoint));
        repaint();   // 線を描いた後で再描画 (追加)
    }
```

- paint メソッド

```
    public void paint(Graphics gr) {
        super.paint(gr);
        jPanel2.getGraphics().drawImage(buffer, 0, 0, this);
    }
```

# 11 例外処理

**例外処理** : プログラムの実行途中でエラーが発生した場合に，その処理をする機能
try, catch による例外処理：

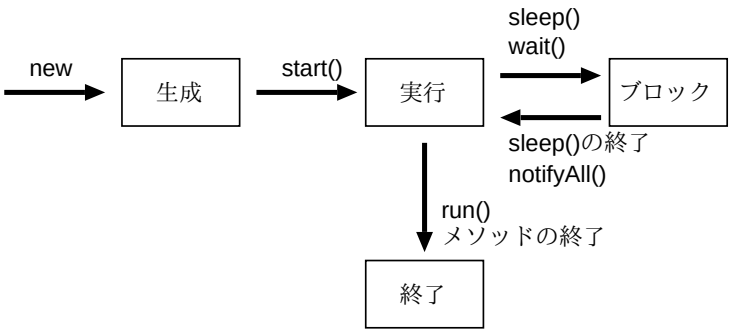
```
try {
        ...
    (処理)
        ...
} catch (例外) {
    例外処理 (例外が発生した場合の処理)
}
```

try と catch の間の処理で例外が発生した場合に例外処理を実行する.

# 12 スレッド

**マルチスレッド処理** : 短い時間で処理を切替えて実行することで複数のプログラムが同時に動いているように見せることができる．Java のプログラムの中でも複数の処理を同時に動かすことができる．このときの処理の単位をスレッドと呼ぶ.

**スレッドの実行** スレッドの状態変化を下図に示す.




- new でスレッドを生成する
- start() でスレッドを開始すると，スレッド内の run() メソッドが呼び出され，実行が開始される
- run() が終了するとスレッドの実行が終了する

スレッド上で処理を実行するには，スレッドを拡張したクラスを作成し，その中の run メソッドをオーバーライドする (書き換える).

```
public class MyClass extends Thread { ... }
```

**スレッドの主なメソッド**

| | |
|---|---|
| start() | スレッドの実行を開始する |
| run() | スレッド上での実行内容をこのメソッドに書く |
| sleep(*time*) | 実行を指定した時間 ( *time*) だけ休止する |

## 12.1 例

[例 12-1] マルチスレッド処理の例 (ThreadTest.java)
スレッド名，休止時間，および，くり返し回数を引数として TheradTest を初期化する.
ThreadTest の処理内容 (run メソッド)

1. for 文を用いて以下の内容をくり返し実行する
   (a) 何回目の処理であるかメッセージを書き出す
   (b) stime で指定された時間だけ処理を休止する
   (例外 InterruptedException が発生した場合，割り込み処理は何も行わない)
2. 終了と書き出し (，処理を終える)

以上の処理をする Threadtest オブジェクトを main メソッド内で 3 つ生成する．

**main メソッドでの処理**

1. thread1(休止時間 250[msec], くり返し回数 6) を生成する
2. thread2(休止時間 500[msec], くり返し回数 5) を生成し，実行を開始する
3. thread3(休止時間 1000[msec], くり返し回数 3) を生成し，実行を開始する
4. thread1 の実行を開始する

以下にプログラムを示す：

```
public class ThreadTest extends Thread {
    int stime;
    int count;
    public static void main(String[] args) {
        ThreadTest thread1 = new ThreadTest("スレッド 1", 250, 6);
        new ThreadTest(" スレッド 2", 500, 5).start();
        new ThreadTest("  スレッド 3", 1000, 3).start();
        thread1.start();
    }
    public ThreadTest(String st, int stime, int times) {
        setName(st);
        this.stime = stime;
        count = times;
    }
    public void run() {
        for(int i=0; i<count; i++) {
            System.out.print(getName()+": "+(i+1)+"回目\t");
            for(int k=i; k<count; k++) System.out.print("*");
            System.out.println("");
            try {
                sleep((long)stime);
            } catch (InterruptedException e) {}
        }
        System.out.println(getName()+": 終了");
    }
}
```

**実行結果**　以下に実行結果を示す.

```
 スレッド 2: 1 回目        *****
  スレッド 3: 1 回目       ***
スレッド 1: 1 回目         ******
スレッド 1: 2 回目         *****
 スレッド 2: 2 回目        ****
スレッド 1: 3 回目         ****
スレッド 1: 4 回目         ***
 スレッド 2: 3 回目        ***
  スレッド 3: 2 回目       **
スレッド 1: 5 回目         **
スレッド 1: 6 回目         *
 スレッド 2: 4 回目        **
スレッド 1: 終了
  スレッド 3: 3 回目       *
 スレッド 2: 5 回目        *
 スレッド 2: 終了
  スレッド 3: 終了
```

## 12.2　アニメーション

もう 1 つのスレッドの使用方法を例示するために，スレッドを利用したアニメーションを以下に示す.(ThreadTest2.java)

処理を含むクラスに Runnable インターフェイスを実装することでもスレッドを利用することができる.

```
public class MyClass  implements Runnable { ... }
```

この場合も run メソッド

```
    public void run()  { ... }
```

に実行内容を記述し，自分自身のオブジェクトをスレッドに渡す：

```
    Thread t = new Thread(this); //  自分自身を引数としてスレッドを生成し
    t.start();                   //  start() メソッドで実行を開始する
```

以下，作成した Circle クラスと ThreadTest2.class について順に説明する.

### 12.2.1　Circle.java

```
import java.awt.*;
import javax.swing.*;
public class Circle implements Runnable {
    JFrame frame;
    Graphics graphics;
    Thread thread;
    boolean running = true;
    int x, y;
    int xCenter, yCenter, radius;
```

```
    int t, delay;
    public Circle(JFrame f, Graphics g, int xCenter, int yCenter) {
        frame = f;
        graphics = g;
        radius = 100;
        this.xCenter = xCenter;
        this.yCenter = yCenter;
        t = 0;
        delay = 200;
        thread = new Thread(this);
        thread.start();
    }
    public void sw() {
        running = !running;
    }
    public void run() {
        while(!thread.isInterrupted()) {
            try {
                if(running) {
                    Point p = new Point();
                    t = (t+5) % 360;
                    x = xCenter + (int)(radius*Math.cos(Math.toRadians(t)));
                    y = yCenter + (int)(radius*Math.sin(Math.toRadians(t)));
                    frame.repaint();
                }
                thread.sleep(delay);
            } catch(InterruptedException e) {}
        }
    }
    public void show (){
        int r = 16;
        graphics.drawOval(x-r, y-r, 2*r, 2*r);
    }
}
```

---

**Circle クラスの動作概要**　パネル上で円運動をするボールのアニメーション

ボールの座標を移動させ，このオブジェクトを生成したフレームを再描画する．フレームでは，このオブジェクトの show メソッドを呼び出すことで，ボールを描画する．

**Circle クラスでの処理の概要**

- コンストラクタ Circle (Circle クラスの初期化)
  - フィールド (変数) を初期化する
    - frame: 呼出し元であるフレームを代入
    - graphics: 呼出し元での図を描くコンテキスト情報
    - xCenter: 描く図の運動の中心 (x 座標)
    - yCenter: 運動の中心 (y 座標)

    * delay: 動作の速度 (sleep 時間)

    など

  - このオブジェクトを引数としてスレッドを初期化する
  - スレッドでの処理を開始する

- sw メソッド

  論理変数 running の値 (true/false) を切替える

- run メソッド (実行が開始されるとこのメソッドの実行が開始される)

  割り込みが発生するまで処理をくり返し実行する

  1. running が真の場合に以下の処理を実行する
     (a) 回転角 t を変更する
     (b) 新たなボールの位置を (x,y) に計算する[1]
     (c) frame(このオブジェクトを呼び出したフレーム) を再描画する
  2. delay で指定された時間だけ休止する

- show メソッド

  座標 (x,y) を中心とする半径 r の円を描く

sw メソッドの呼び出しでアニメーションの停止と継続実行を切替えることができる.

### 12.2.2 ThreadTest2.java

NetBeans IDE で作成した main メソッドを含む ThreadTest2 クラスの一部を以下に示す.

```
import java.awt.*;
...
    private void jButton1ActionPerformed(java.awt.event.ActionEvent evt) {
        if(circle == null) {
            Graphics gr = jPanel1.getGraphics();
            gr.setColor(Color.RED);
            circle = new Circle(this, gr,
                    jPanel1.getWidth()/2,jPanel1.getHeight()/2);
        }
    }
    private void jButton2ActionPerformed(java.awt.event.ActionEvent evt) {
        circle.sw();
    }
      ...
    public void paint(Graphics g) {
        super.paint(g);
        if(circle != null) circle.show();
    }
    Circle circle;           // 追加するフィールド (変数)
      ...
}
```

[1] Math.cos, Math.sin, Math.toDegrees: 数学関数

TheradTest2 クラスの動作概要：

- ボタン 1 がクリックされると (jButton1ActionPerformed)，Circle オブジェクトが生成され，アニメーションが開始される
- ボタン 2 がクリックされると (jButton2ActonPerformed)，アニメーションが停止する．もう一度，このボタンが押されるとアニメーションが続行さえる．

ThreadTest2 クラスの処理概要：

- jButton1ActionPerformed

  circle オブジェクトが生成されていない場合には以下の処理を行う．

  1. jPanel1 で描く図の色を赤色に設定する
  2. circle に Circle クラスのオブジェクトを生成する．

- jButton2ActionPerformed

  circle オブジェクトの sw メソッドを呼び出し，アニメーションの休止/続行を切替える．

- paint

  1. このオブジェクトの親クラスの paint を呼び出す
  2. circle オブジェクトの show を呼び出してボールを描く．

# 13 Timer クラスと TimerTask クラスを利用したアニメーション

Timer と TimerTask クラスを利用すればスレッドによるくり返し処理をもっと簡単に記述することができる．

**Timer クラス** ：バックグラウンドで実行するスレッドをスケジュールすることができる．

**TimeTask クラス** ：このクラスを拡張して，Timer により実行されるタスクを定義する．

## 13.1 アニメーションの例

[例 13-1] jPanel1[JPanel] 内でのアニメーション
TimerTask を拡張してボールを描く MyTask クラスを定義し，Timer 上で定期的にくり返し実行する．

```
import java.util.TimerTask;     //  クラスパッケージのインポート
import java.util.Timer;
import java.awt.*;
import javax.swing.*;
...
public class TimerTest extends javax.swing.JFrame {
    /** Creates new form TimerTest */
    public TimerTest() {
        initComponents();
        f = this;                              //  ここから下を追加する
        g = jPanel1.getGraphics();
        task = new MyTask(120,120, 80);
        Timer t = new Timer();
        t.schedule(task, 0, 200);
    }
```

```
        ...
    public void paint(Graphics g) {
        super.paint(g);
        task.show();
    }                                          //  ここまでを追加
    // Variables declaration - do not modify
    private javax.swing.JPanel jPanel1;
    // End of variables declaration

    Graphics g;                                //  ここから下を追加
    JFrame f;
    MyTask task;
    class MyTask extends TimerTask {
        int t = 0;
        int r = 16;
        int x, y;
        int xCenter, yCenter, radius;
        public MyTask(int x, int y, int r) {
            xCenter = x;
            yCenter = y;
            radius = r;
        }
        public void run() {
            t = (t+5) % 360;
            x = xCenter + (int)(radius*Math.cos(Math.toRadians(t)));
            y = yCenter + (int)(radius*Math.sin(Math.toRadians(t)));
            f.repaint();
        }
        public void show() {
            g.drawOval(x-r, y-r, 2*r, 2*r);
        }
    }
}
```

上に示したものは NetBeans IDE で作成したプログラム (TimerTest.java) の一部である．run メソッドが Timer のスケジューリングにより，くり返し呼び出される．

run には 1 回の処理内容だけを記述することに注意すること．

**schedule**(*task*, *delay*, *period*) ：タスクのスケジューリング

*task* スケジュールするタスク

*delay* 実行開始までの遅延時間

*period* くり返し実行する際の待ち時間．この時間間隔でタスクをくり返し実行する

## 13.2 課題

MyTask クラスの各フィールド (変数) の役割と各メソッド (関数) の働きを調べ，動作を理解せよ．