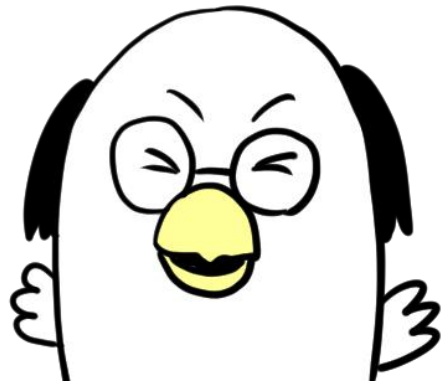
スレッド，例外処理

2018/5/22(火)　プログラミングIV 第七回

福井大学　工学研究科　情報・メディア工学専攻

長谷川達人

来週は月曜授業なので
プログラミングIVは
お休みです＼(^o^)／

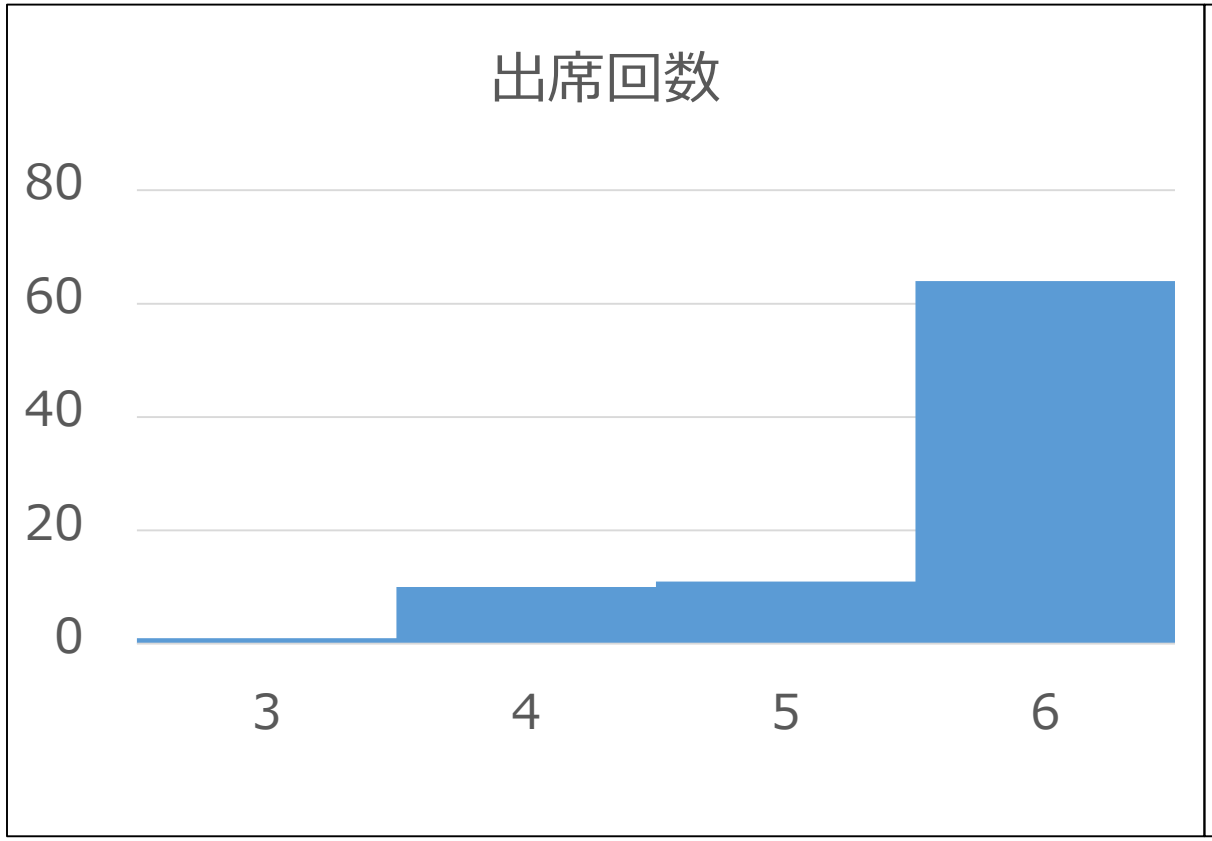


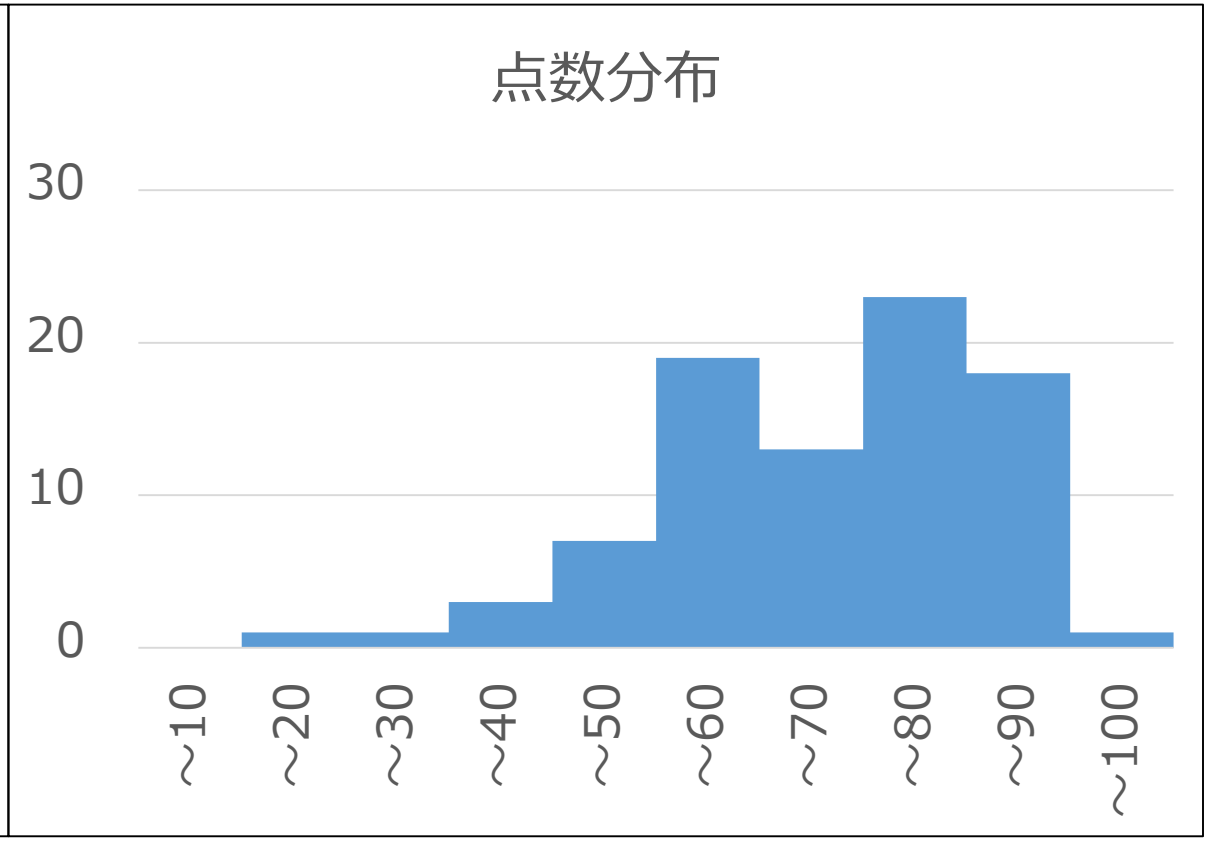
演習問題の解説

- HP上にアップロードした.
- 解説はコメントを参照して下さい.

- 学籍番号を入力すると，成績を表示する仕組みを開発してみましたので使ってみてください.

現在の状況





質問への回答

- マウスイベントの違いがわからなかった
  - mouseClicked(): マウスを**その場で**押して離した
  - mousePressed(): マウスを押した（その後移動OK）
  - mouseReleased(): マウスを離した（移動中でもOK）
  - mouseEntered(): マウスがPanelに入った
  - mouseExited(): マウスがPanelから出た
  - mouseDraged(): マウスを押している間，常に
  - mouseMoved(): マウスが動いている間，常に
- mouseEntered()とmouseExited()の使いみち
  - 前回の画像がついてくる課題で，フレームアウトしたときに画像が残ってしまうのを消すためにはmouseExited()が必要

質問への回答

- Dimensionとは何か
  - コンポーネントの幅（width）と高さ（height）をカプセル化するクラスである．（widthとheightをセットにしただけ）
- ２つの図形が重なった場合，どちらかを前に出すことはできるのか.
  - fillRect()で全体を白塗りして，マウスカーソルを描画したように，後から描画される方が前に出る.
- 画面の中心を取得するメソッドはないのか.
  - JPanelのサイズを半分にしたものが画面の中心である.
- Panelの絵を保存できないか
  - できる．BufferedImage型インスタンスを作成し，そのGraphicsに対して書き込めば良い.

質問への回答

- クラスやインタフェースを実装すると，メソッドがもとはどのクラスのものなのかよくわからなくなる.
  - あるある．ある程度のメソッドは慣れて覚えるしかない.
- ダブルクリック等は自分で実装しなきゃいけないのか
  - 基本的なものは標準APIである．もちろんキーボードイベントもある．ググってみると良い.
- MouseListenerを使う際には全メソッドをオーバーライドして必要なものだけを使うという認識で良いか.
  - それで良い．MouseListenerはインタフェースなので，必ずオーバーライドはしなければならない.
- ColorはRGBで生成できないのか
  - new Color(int r, int g, int b)でできる.

質問への回答
自由制作について

- 自由課題はどの程度著作権に配慮すればよいか.
  - ちゃんと配慮すること.（フリー素材を推奨する）
- 自由課題はソースコードも評価対象か.
  - 成果物を評価対象とする．ただし評価軸にオリジナリティを入れる予定なので流用は避けたい.（コピペは論外）
- 自由課題を２つ作ったらどうなるのか.
  - 発表時間が足りないので１つ自信作を発表する.
- パズドラはもう作れそう？
  - 技術的にはできると思うが，容易ではないだろう．是非挑戦を.

質問への回答

評価等について

- 演習が０点の場合欠席になるのか.
  - ならない（講義出席で代用する）.
- 次回の資料の完成はいつか.
  - いつもギリギリで申し訳ないが，月曜日がデフォルトである.
- 試験時に資料等は参考にしてよいか.
  - WebClass（ブラウザで入力したり選択したりするやつ）で実施する予定で，資料持ち込みやWeb検索は禁止とする.
- クーラー直下で風が寒い.
  - 自由席なので，席を替えることを推奨する.

本講義の概要

前半		後半	
第1回	基本文法の復習	第9回	標準ライブラリ
第2回	クラス～カプセル化の復習	第10回	ファイル入出力
第3回	抽象クラス，インタフェース	第11回	デバッグ，インポート，高速化
第4回	ポリモーフィズム	第12回	オブジェクト指向
第5回	GUI 1	第13回	自由開発演習 1
第6回	GUI 2	第14回	自由開発演習 2
第7回	**スレッド，例外処理**	第15回	自由開発演習発表会
第8回	ジェネリクス，コレクション	第16回	期末試験

何を開発するか，少しずつ考えておくこと

本日の目標

概要

例外処理とマルチスレッドの基礎をかじる.

目標

例外処理とマルチスレッドを
とりあえず使う事ができる.

本日の提出課題

講義パート

課題を意識しながら
講義を聞くと良い.

課題 1

本日の授業を聞いて,
よくわかったと思う内容を2点簡潔に述べよ.

課題 2

本日の授業を聞いて,
質問事項または**気になった点**を1点以上簡潔に述べよ.

課題 3

感想（あれば）

例外処理

3種類のエラー

syntax
文法エラー
error

コードの形式的な誤りで，コンパイル時にエラーになる

↓

コンパイルエラーに従ってデバッグする

runtime
実行時エラー
error

実行中に，想定外の事態が発生し継続できず強制終了する

↓

想定外の事態への例外処理を実装する

本日はココの話

logic
論理エラー
error

実行はできるが想定していたものと異なる結果が返ってくる

↓

原因を究明し，ロジックを改善する

例外処理
例外対応の書き方

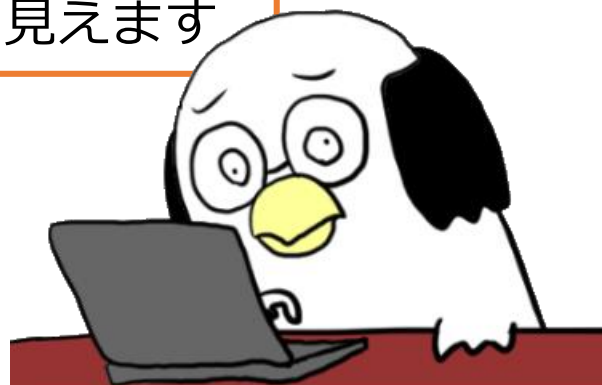
C言語の時の例外対応

```
// (1) ファイルを開く（失敗時戻り値は定数NULL）
FILE fp = fopen("c:¥¥test.txt");
if(fp == NULL){
        printf("エラーなので終了します. ");
        exit(1);
}

// (2) ファイルに文字を書き込む
fputs("hello!", fp);

// (3) ファイルを閉じる（失敗時戻り値は定数EOF）
int c = fclose(fp);
if(fp == EOF){
        printf("エラーなので終了します. ");
        exit(1);
}
```

プログラムの本質部分

例外対応

本来必要な処理は3行で
済むのに，間の例外処理
により煩雑に見えます



例外処理

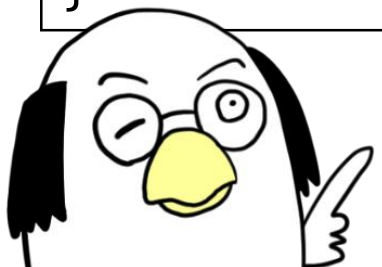
例外対応の書き方

Javaの例外対応

正常系の処理

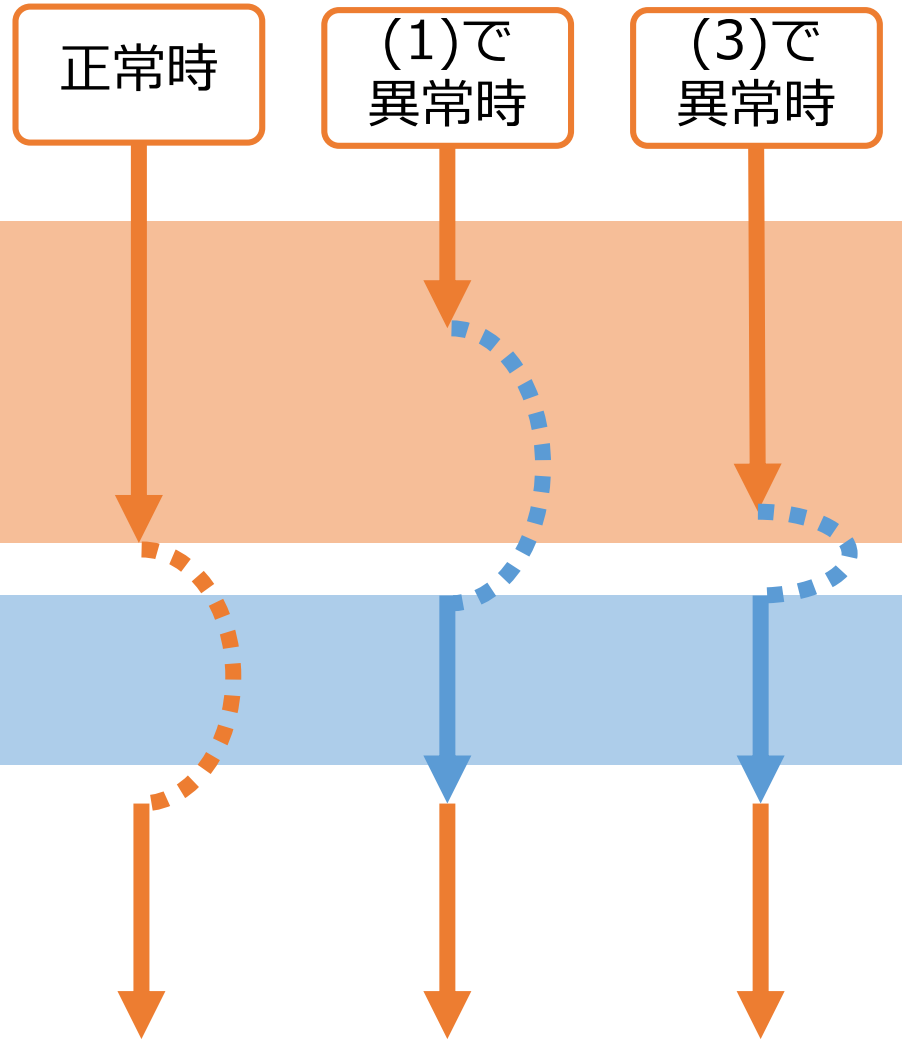
例外対応

```
try{
    // (1) ファイルを開く
    FileWriter fw = new FileWriter("c:¥¥data.txt");
    // (2) ファイルに文字を書き込む
    fw.write("hello!");
    // (3) ファイルを閉じる
    fw.close();
}catch(IOException e){
    // エラー発生時の対応方法を書く
    System.out.println("エラーなので終了します。");
    System.exit(1);
}
```



正常系の処理と異常系の処理が分かれていてわかりやすい

複数の例外処理を集約して記述できる



例外処理

例外対応の書き方

※C言語にはなかったが，C++にはtry-catchがある

```
try{
    // 正常処理
    // ・例外が発生した際にシステムが強制終了しない.
    // ・例外が発生した処理以降を中断しcatchへ.
    // ・例外1，2以外発生時はcatchできず強制終了する.
} catch([例外1の型] [変数名]){
    // 例外1発生時の処理
} catch([例外2の型] [変数名]){
    // 例外2発生時の処理
} finally{
    // 例外に関わらず実行される処理
}
```

複数種類の例外をcatchすることもできる
優先順位は上から順（else ifと同じ）

実行されずに終了すると困る処理はfinallyに
例：ファイルやDBのclose()処理等

めんどくさいので全体をtry-catchで囲んでおこうと思います．．．

正常系と異常系が離れすぎてしまうと逆に可読性が下がります．処理のグループごとに必要に応じてくらいが良いでしょう．

例外処理
例外対応の書き方

例外もクラスである.
以下の例だと，**ArithmeticException**型の**e**である.
インスタンス**e**は**Exception**の詳細内容を保持している.

```
catch(ArithmeticException e){
  System.out.println("1: " + e.toString());
  System.out.println("2: " + e.getMessage());
  e.printStackTrace();
}
```

0除算を**catch**したときの出力例

```
1: java.lang.ArithmeticException: / by zero
2: / by zero
java.lang.ArithmeticException: / by zero
at Shiryo.main(Shiryo.java:8)
```

Exceptionの詳細を取得可能
→Exceptionに応じて対応を変更可能

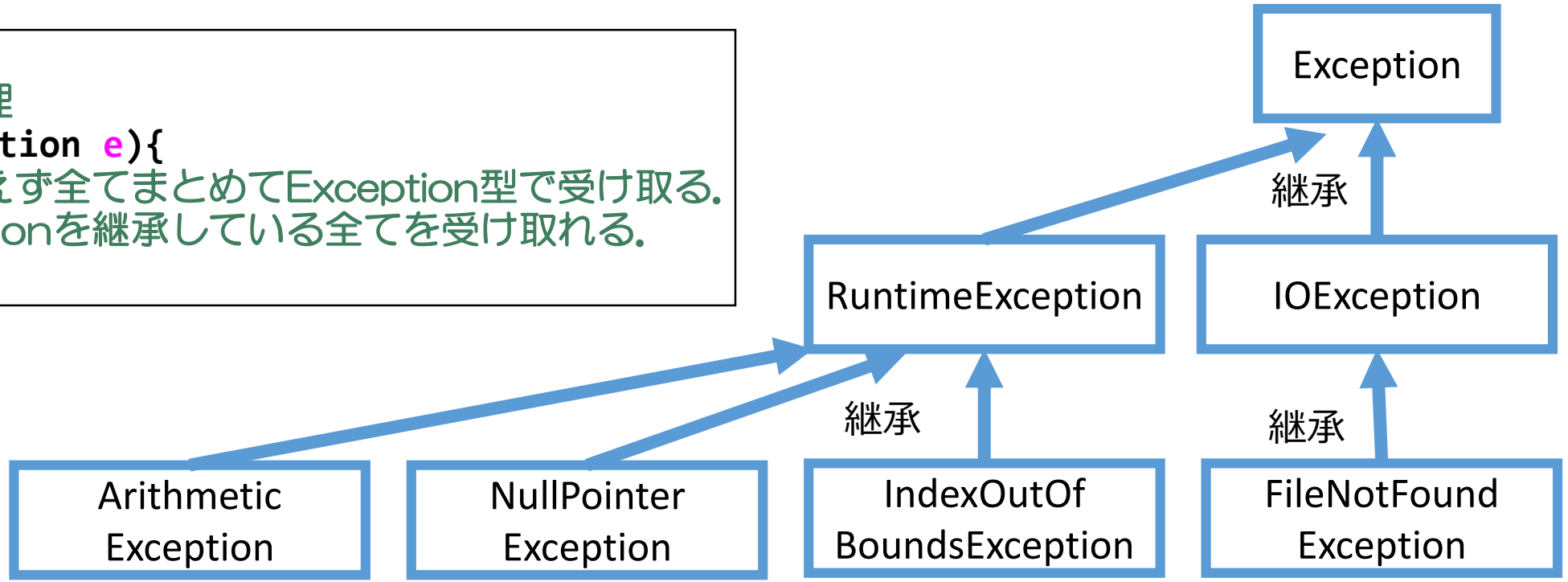
StackTraceはException発生源を追跡した情報でありデバッグに超便利

特にすることがなければとりあえずprintStackTrack()しておけばよい

例外処理
例外対応の書き方

例外もクラスなので継承関係がある.
すなわちポリモーフィズムを応用して
次のようにざっくりと例外を**catch**できる.

```
try{
    // 正常処理
} catch(Exception e){
    // とりあえず全てまとめてException型で受け取る.
    // Exceptionを継承している全てを受け取れる.
}
```



例外処理

よくあるException

Arithmetic Exception	**算術エラー**　ArithmeticException 不正な算術処理が発生している（大体0で除算している）
NullPointer Exception	**ぬるぽ**　NullPointerException Nullにアクセスした（大体インスタンスの初期化忘れ）
IndexOutOf BoundsException	**アウトオブバウンズ**　IndexOutOfBoundsException インデックス範囲外にアクセスした（大体配列外参照）
FileNotFound Exception	**ファイルノットファウンド**　FileNotFoundException ファイルが見つからない

例外処理
チェック例外

右図は本日はじめに出てきたファイル入出力の例

try-catchしないと図のようにコンパイルエラーとなり実行できない.

```
import java.io.FileWriter;

public class Shiryou {
    public static void main(String[] args){
        // ファイルを開く
        FileWriter fw = new FileWriter("c:¥¥data.txt");
        // ファイルに文字を書き込む
        fw.write("hello!");
        // ファイルを閉じる
        fw.close();
    }
}
```

処理されない例外の型 IOException

2 個のクイック・フィックスが使用可能です:

スロー宣言の追加

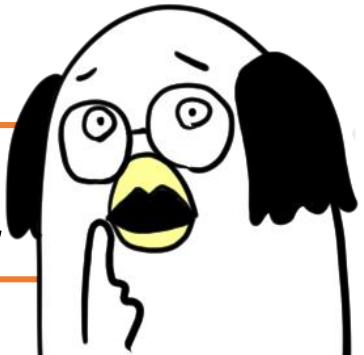
try/catch で囲む

フォーカスするには 'F2' を押下

このようにメソッドによっては例外対応を強制してくる.
このような例外を**チェック例外**と呼ぶ.

上記例の場合fwに関するメソッド
3つともチェック例外である

これなら忘れっぽい私でも
例外対応をせざるを得ません



例外処理
まとめ

- Javaは例外をExceptionとしてcatchする仕組みがある.
- 必ずtry-catchしなければならないメソッドがある.

```
try{
      // 正常処理
      //  ・例外が発生した際にシステムが強制終了しない.
      //  ・例外が発生した処理以降を中断しcatchへ.
      //  ・例外1，2以外発生時はcatchできず強制終了する.
} catch([例外1の型] [変数名]){
      // 例外1発生時の処理
} catch([例外2の型] [変数名]){
      // 例外2発生時の処理
} finally{
      // 例外に関わらず実行される処理
}
```

Exceptionを予測して回避できるプログラムが書けると良いですね.

例外処理

練習問題１：出力を予測せよ

`new FileWriter("data.txt");`
でIOExceptionが発生するとする.

問1

```
FileWriter fw = new FileWriter("data.txt");
fw.write("hello!");
fw.close();
```

(1)普通に実行し終了する
(2)IOExceptionが発生し強制終了する
(3)コンパイルエラーになる

問2

```
try{
    FileWriter fw = new FileWriter("data.txt");
    fw.write("hello!");
    fw.close();
} catch(IOException e){
    System.out.println(“あ");
}
```

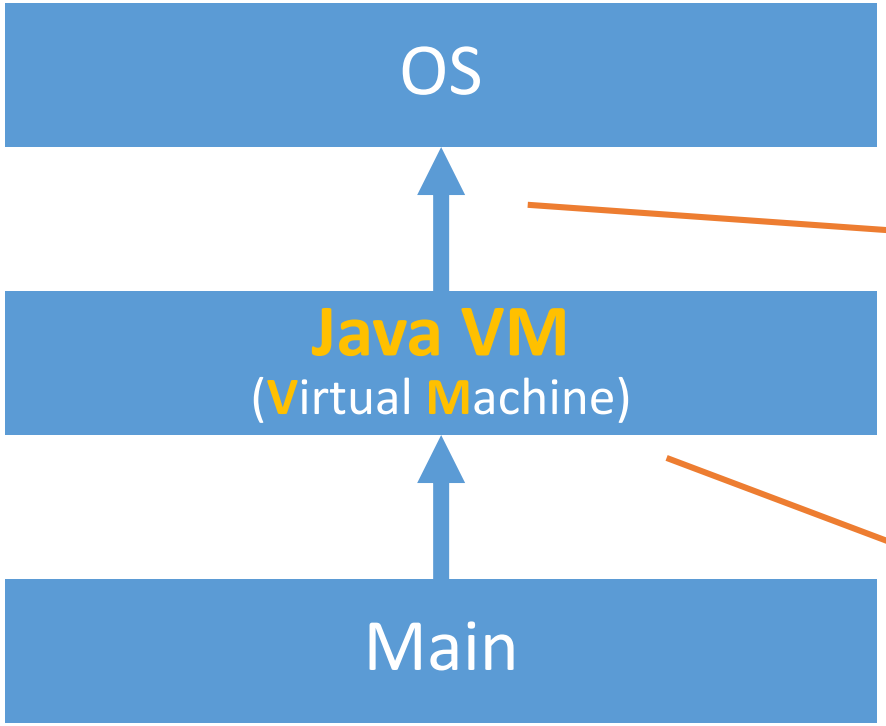
(1)「あ」と表示され普通に終了する
(2)IOExceptionが発生し強制終了する
(3)コンパイルエラーになる

ファイル入出力処理には実はもう少し
正しい書き方があるが，またいずれ.

補足説明

イベントや例外はどこからくるのか

Q:　例外はどこからくるのか？
A:　JVMからくる

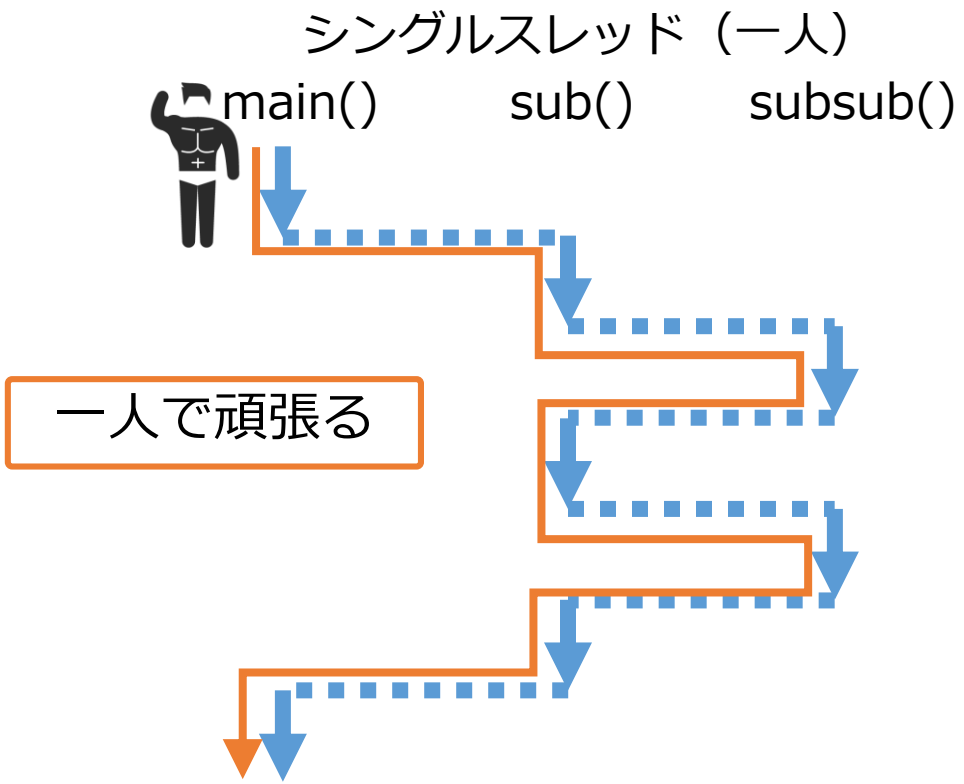


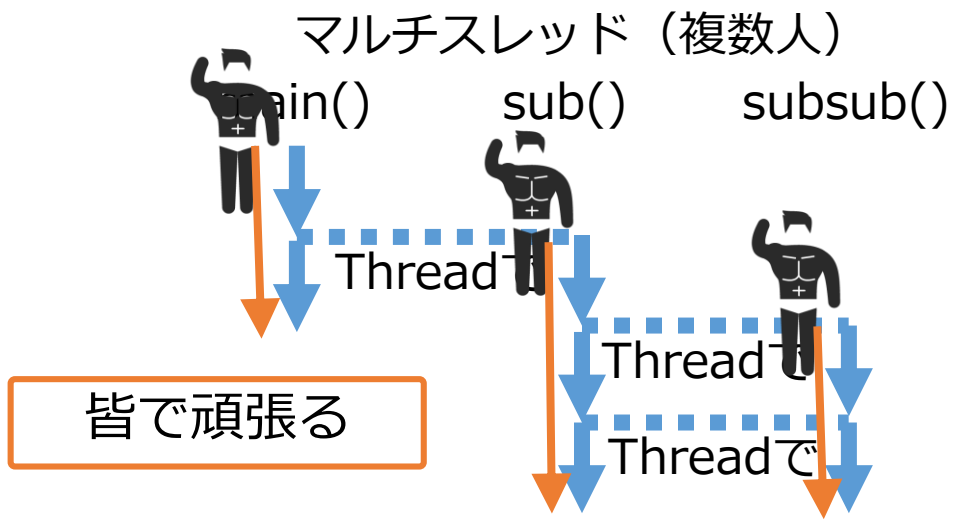
JavaはJava仮想マシン（JVM）が仲介役となる.
コンパイルするとJVMが理解できる形に変換される.
プログラムがJVM上で動作するため，次の特徴がある.
・OSに依存せずに動作する（JVMが動くならば）.
・エラーが発生してもOSの動作に影響が及ばない.
・仲介役がいる分，処理は多少遅くなる.

ということで，まずい処理が発生した場合，JVMが
それを検出し，Exceptionをプログラムに投げ返す.

スレッド
スレッドとは

スレッドはプログラムを実行している**１人の人間**とイメージすると良い.





スレッド

使い方α：Threadクラスを継承する

1. Threadクラスを継承したクラスを作成する.
2. 上記クラスでrun()メソッドをオーバーライドする.
3. 上記クラスをインスタンス化する.
4. インスタンスからstart()メソッドを呼び出す.

スレッド

使い方α：Threadクラスを継承する

```
public class MyThreadA extends Thread{
    @Override
    public void run(){
        for(int i=0;i<10;i++){
            System.out.println(this.getClass().getName() + "¥t" + i);
            try{
                Thread.sleep(10);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

Threadを継承しrun()をオーバーライド

run()の中が別スレッドで処理される
この場合，10ms毎にクラス名＋iを表示する

```
public static void main(String[] args){
    MyThreadA thread = new MyThreadA();        // MyThreadAをインスタンス化
    thread.start();          // 並列処理でrun()を呼び出す場合
    thread.run();            // 普通にrun()を呼び出す場合
}
```

スレッド

使い方β：Runnableインタフェースを実装する

1. Runnableインタフェースを実装したクラスを作成する.
2. 上記クラスでrun()メソッドをオーバーライドする.
3. Threadクラスをインスタンス化する（ただしnewの際の引数に上記クラスのインスタンスを代入する）.
4. 上記インスタンスからstart()メソッドを呼び出す.

スレッド

使い方β：Runnableインタフェースを実装する

```
public class MyRunnable implements Runnable{
    @Override
    public void run(){
        for(int i=0;i<10;i++){
            System.out.println(this.getClass().getName() + "¥t" + i);
            try{
                Thread.sleep(10);
            }catch(InterruptedException e){
                e.printStackTrace();
            }
        }
    }
}
```

Runnableを実装しrun()をオーバーライド

run()の中が別スレッドで処理される
この場合，10ms毎にクラス名＋iを表示する

```
public static void main(String[] args){
    Thread thread = new Thread(new MyRunnable());
    thread.start();          // 並列処理でrun()を呼び出す場合
    thread.run();            // 普通にrun()を呼び出す場合
}
```

ここに注意

スレッド

使い方β：Runnableインタフェースを実装する

Runnableのいいところ

以下のように，JFrameを継承しながら，ActionListenerを実装しながら，自身の中で並列処理を実装できる.

```
public class ThreadFrame extends JFrame
                              implements ActionListener, Runnable{}
```

スレッド

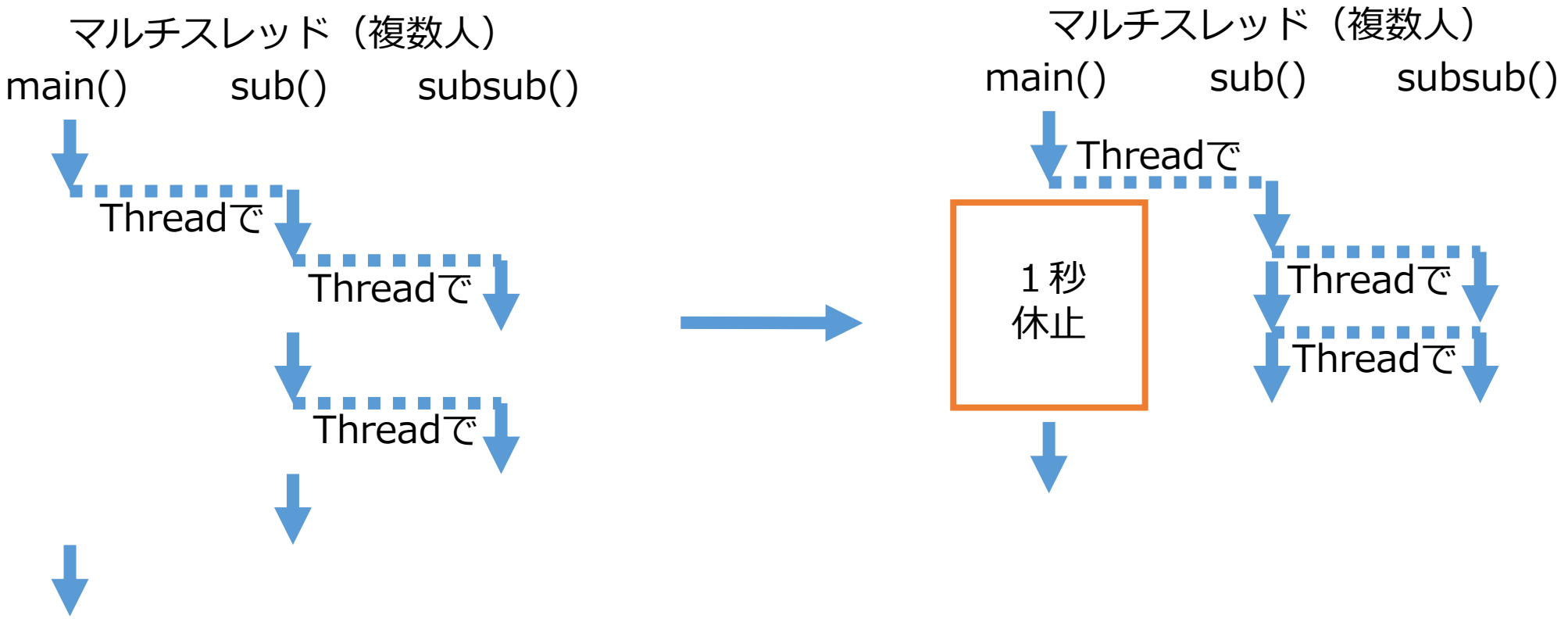
練習問題２：Runnableの便利な使い方

次の空欄に当てはまるコードを予測せよ.

```
public class ThreadFrame extends JFrame implements ActionListener, Runnable{
    public ThreadFrame(){
        // JFrameの初期設定（省略）
        JButton button = new JButton("Threadスタート");
        button.addActionListener(   (1)   );
        this.getContentPane().add(button);
    }
    @Override
    public void actionPerformed(ActionEvent e) {
        Thread thread = new Thread(   (2)   );
        thread.start();
    }
    @Override
    public void run() {
        // run()の中身は同じとする（省略）
    }
}
```

スレッド
スレッドの休止

Threadを休止するには**Thread.sleep(int ms);**を用いる.
実行されたThreadが**ms**ミリ秒休憩に入る.



スレッド
スレッドの休止

Threadを休止するには**Thread.sleep(int ms);**を用いる.
実行されたThreadが**ms**ミリ秒休憩に入る.

```
while(true){
        this.repaint();
        try{
                Thread.sleep(1000);
        }catch(InterruptedException e){
                e.printStackTrace();
        }
}
```

この場合，1秒間隔で再描画repaint()を行う.
・repaint()→1秒休止→repaint()…

引数は目安であり，精度保証はされていない.

InterruputedExceptionはチェック例外なので，必ず記述しなければならない.

InterruputedExceptionは外部から割り込みが入ったときに発生する例外である.

スレッド
Threadを停止する

- Threadはrun()内の処理が完了すればThreadは自動的に停止する.
- 外部イベントにより終了させたい場合，flag変数をもたせて終了判定させることが多い.
- Threadインスタンスが持つstop()メソッドは，現在非推奨なので使うべからず.

スレッド
Threadを停止する

```
// ActionPerformed()が呼び出されたらThreadを停止したい場合
private boolean flag = true;
@Override
public void actionPerformed(ActionEvent e) {
    this.flag = false;
}
@Override
public void run() {
    while(this.flag){
        System.out.println("待機中");
        try{
            Thread.sleep(100);
        }catch(InterruptedException e){
            e.printStackTrace();
        }
    }
}
```

フラグでThreadの継続を管理する例

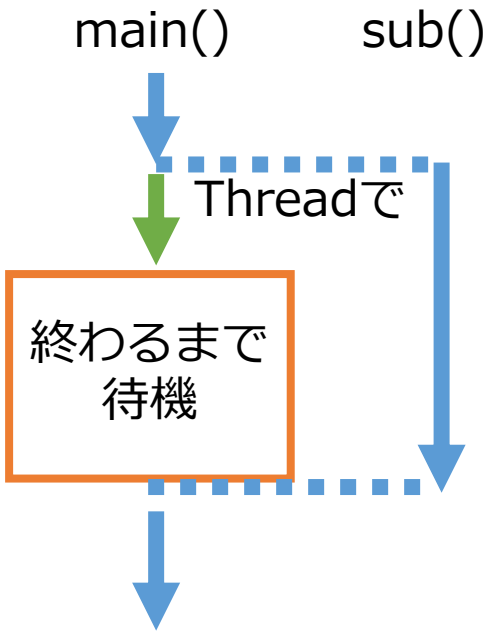
スレッド

Thread終了まで休止する

とあるスレッド終わるまで待機するメソッドjoin()がある.

```
thread.join();
[対象スレッドのインスタンス].join();
```

sleep()同様
try-catchが必要



- スレッドの終了後に何か実行したい時
- スレッド開始後，少し処理（右図の**緑矢印**）をして，完了まで待機したい時

等々に応用ができる.
コールバックで対応してくれてもよい.

スレッド

練習問題３：Threadを使ってみる

1. Threadを継承したMyThreadを定義する.
2. フィールドはString strとint numとしコンストラクタの引数でこれを初期化する.
3. run()をオーバーライドし，100ms毎にstrをnum回println()で出力する処理を記述する.

```
// 動作確認手順（実行前に出力を予測すること）
public static void main(String[] args){
    Thread thread1 = new MyThread("★", 5);
    Thread thread2 = new MyThread("  ▲", 3);
    Thread thread3 = new MyThread("    ●", 10);
    thread1.start(); thread2.start(); thread3.start();
}
```

スレッド

発展：無名クラスでマルチスレッドを実装する

以下のようにすることで，Thread専用のクラスを別のファイルで作成することなく，自身がRunnableを実装することなく，Threadを実装することができる.

```
Runnable runnable = new Runnable(){
    @Override
    public void run(){
        for(int i=0;i<10;i++){
            System.out.println("スレッド内です");
            try{
                Thread.sleep(100);
            }catch(Exception e){}
        }
    }
};
Thread thread = new Thread(runnable);
thread.start();
```

Runnableインスタンスを定義しつつそのままインスタンス化して使う.
このクラスを**無名クラス**という.

無名クラスの定義方法

```
new Runnable(){
    // オーバーライド等の
    // Runnableに必要な処理
};
```

ということで，実はインタフェースのインスタンス化には，このような方法もあったのだ.

GUIプログラミング

復習：様々なコンポーネント

GUIプログラミング1で，main()でJFrameを作成する時におまじないを書いてもらっていた.

```
public static void main(String[] args){
    SwingUtilities.invokeLater(new Runnable() {
        public void run() {
            // MyFrameのインスタンスを生成する
            MyFrame frame = new MyFrame();
        }
    });
}
```

実はおまじないとして書いていたこれの中身が**無名クラス**

スレッド

発展：無名クラスでマルチスレッドを実装する

おまじないの例のように，無名クラスのインスタンス化と，Threadインスタンスの生成をまとめて書くと次のようになる.

```
Thread thread = new Thread(new Runnable(){
    @Override
    public void run(){
        for(int i=0;i<10;i++){
            System.out.println("スレッド内です");
            try{
                Thread.sleep(100);
            }catch(Exception e){}
        }
    }
});
thread.start();
```

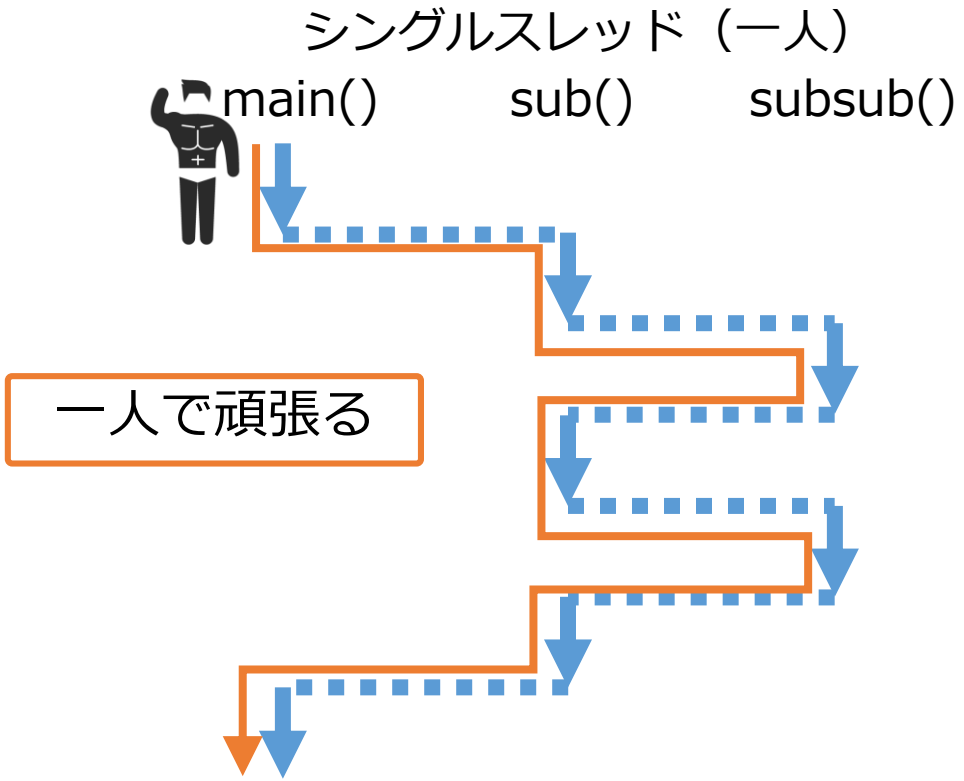
Threadインスタンスを作成する部分

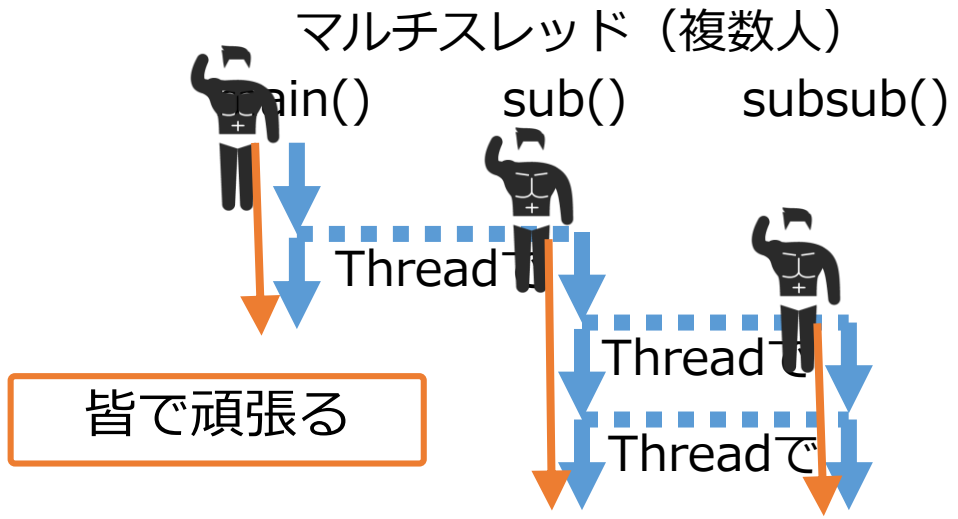
本来Runnableインスタンスが代入される部分

無名クラスでインスタンス化しつつ**new Thread()**の引数に代入しているのである.

スレッド
まとめ

マルチスレッド（並列処理）という新しい考え方を学んだ.
複数人で処理を分担し同時に実施するイメージである.





スレッド
まとめ

マルチスレッドの実装方法は3種類ある.

1. Threadクラスを継承したクラスを作成し，インスタンスに対してstart()を実行する.
2. Runnableインタフェースを実装したインスタンスをThreadインスタンス生成時に引数で渡し，start()を実行する.
3. Runnableインタフェースの無名クラスのインスタンスをThreadインスタンス生成時に．．．（略）

個人的には，複雑な実装時には１を，他のクラスの継承等と併用したい場合は２を，簡易な実装時には３を推奨する.

補足説明

イベントや例外はどこからくるのか

Q:　イベントはどこからくるのか？
A:　リスナの登録先からくる.

JButton

2．自分自身（ボタン）が押されることを,
今か今かと待ちわびています．（**別スレッド**で）
（この処理はJava側が実装してくれている）

1．私はactionPerformed()を完備しているので,
イベントが起こったら教えて下さい.

```
button.addActionListener(this);
```

3．イベントを検出したので
コールバックします！
（**２のスレッドで**）

MyFrame
implements ActionListener

次週予告

※次週以降も計算機室

前半

ジェネリクスとコレクションという新しい**便利な**概念を学ぶ.
なお，コレクションとはリストとかマップである.

後半

講義内容に関するプログラミング演習課題に取り組む.

本日の提出課題

講義パート

課題 1

本日の授業を聞いて，
よくわかったと思う内容を
2点簡潔に述べよ.

課題 2

本日の授業を聞いて，
質問事項または**気になった点**
を1点以上簡潔に述べよ.

課題 3

感想（あれば）

課題 4

例外処理

裏課題：練習問題１の実は．．．

練習問題1の問2（下記のコード）には実は欠陥がある．これについて，「FileWriterは使ったらclose()する必要がある．」をヒントに，どんな欠陥があるのか簡潔に述べよ．

```
try{
    FileWriter fw = new FileWriter("c:¥¥data.txt");
    fw.write("hello!");
    fw.close();
} catch(IOException e){
    System.out.println(“あ");
}
```

演習

- 昼休み，いつものWebページに演習問題をPDFで演習問題をアップロードする．各自実施してプロII同様のWebページから提出すること.
- 質問は３人体制で受け付けるので遠慮なく申し出る．質問の際は，どこまでわかっていて何がわからないのかを申し出ること.
- （ないとは思うが）コピペは発覚次第両成敗する.
  - ✓ {コピペ，カンニング} ∈不正行為
- つまらないミスも今回は問答無用で×とするので，最終チェックを怠らないこと．（昨年は目視で甘めに採点していたが，自動採点を開発している意味がないので．．．）