プログラミングIII

黒瀬　浩

kurose@neptune.kanazawa-it.ac.jp

パッケージ

クラスライブラリ

必要な機能を持つクラスをあらかじめ用意したもの

パッケージ

用途に応じた管理単位

主なパッケージ

java.lang	java基本機能
java.util	よく使われる機能(utility)
java.io	入出力(input/output)
javafx.scene.control	GUIコントロール
javafx.scene.shape	GUI図形
javafx.event	GUIイベント
java.net	ネットワーク

パッケージの指定

完全限定名 トップレベルから ． でつないでパッケージ位置を指定

```
// java.utilパッケージのRandomクラスでrandインスタンスを作成
java.util.Random rand = new java.util.Random();
```

完全限定名が長いので短く書く場合

```
import java.uril.Random;
Random rand = new Random();
```

```
// 複数のimport
import java.util.ArrayList;
import java.util.Random;
```

```
//  まとめてimport
import java.util.*;
```

java.utilの直下のクラスを取り込む
java.util.zip.*は取り込まない

importを省略した場合は
`import java.lang.*;` と同じ

クラスパス

実行するプログラムは　環境変数PATHに　記載されたディレクトリのリスト

Javaのclassファイルやjarファイル(クラスファイルをまとめたライブラリ)は　環境変数CLASSPATH　に記載されたディレクトリから探す

CUIでは　echo %CLASSPATH% (Linuxではecho $CLASSPATH)

統合エディタには，クラスパスを設定するメニューがある

パッケージは．単位で区切られるが実際にはディレクトリ(フォルダ)に対応して格納されている

API: Application Program(ming) Interface

https://docs.oracle.com/javase/jp/10/docs/api/

10はJDKのバージョン

必要な機能は既にあるものを活用する

・Java言語が提供しているもの
・言語が提供しているが指定しないと使えないもの
・誰かが作って提供しているもの

どのようなパッケージにどのようなメソッドがあるか調べる
　　書籍，ネットの記事，パッケージの仕様書

行たいことが決まっていたらその分野のパッケージを探す
　　統計解析，画像処理，グラフ解析，ネットワーク，
　　Webアプリケーション，データベース，．．．

パッケージには，他のパッケージを前提にしているものがある
　　使いたいパッケージの他にも追加でインストールしなければ
　　ならない場合がある（パッケージの依存関係）

基本的なクラス String

```
// String型的な扱い
String message = “こんにちは”;

// インスタンス的な扱い
String message = new String(“こんにちは”);

// 文字列長さを得るメソッドとフィールド
message.length();
message.length;                    何が違うのか

String s = “Javaの学習”;
int s.indexOf(“学習”);   // 開始位置を返す　ない場合は?
boolean s.contains(“学習”);     // 含む場合はTrue
String s.replace(”Java”, “java言語”); // 置換　複数ある場合は?
String[] s.split(“/”); // 区切り記号で分割，正規表現使用可
```

注：正規表現は，文字列を汎用的に表す書き方 Regular Expression

Mathクラス　importしなくても利用可

以下は変数名を省略して書いている．戻り値の型も省略している場合がある
importしなければMath. をつける必要がある

絶対値 `double abs(double), int abs(int)`

三角関数 `double sin(double), cos(double), tan(double)`

算術関数 `double sqrt(double), pow(double, double),`
`log(double)`

最大　`double max(double, double), int max(int, int)`
最小　`double min(double, double), int min(int, int)`

乱数　`double random()`

円周率 `PI`　　定数は全て大文字で書く慣習

パッケージの作成

```
package パッケージ名;  // ソースの最初に書く
```

```
package mypacage;
```

定義

外部からは，　パッケージ名．　でアクセス

```
mypackage.MyClass a = mypackage.MyClass();
```

か

```
import mypackage;
MyClass a = MyClass();
```

同じパッケージに含まれるクラスはimport不要

Eclipseではファイル→新規→パッケージ　でパッケージが作れる
クラス作成はパッケージ作成の後に行う

リスト1-3改　の演習

```
package mypackage;

public class MyClass1 {
    public void printMessage(){
        system.out.println( "mypackage.Myclass.printMessage()" );
    }
}
```

上記をMyClass1.java で作成

PackageEx1.javaを別に作成
その中で
```
mypacket.Myclass1 a = new pypackage.MyClass();
```
で呼び出すJavaソースを完成させよ

それができたら,
```
import mypackage.MyClass1;
Myclass1 a = new MyClass1;
```
で呼出せるように改造せよ

演習

Randomクラス(java.util.Random) を使い,
10回乱数を表示させ値のばらつきを調べよ

10要素の配列を作り，乱数範囲の区間により発生頻度を数えて表示する
プログラムを作れ

作成できたら
発生回数を100回，1000回，10000回で区間の頻度のばらつきを確認せよ

Random()が返す値の範囲を確認すること
範囲を10区間に分割しどの区間の値か判定すること

例外処理

例外　予測できない事態が実行時に起きる

0で割った

⇨ AlthmeticException発生

要素が存在しない配列にアクセスした

⇨ ArrayIndexOutOfBoundException発生

．．．

例外を考慮しないとプログラムは強制終了

try catch文で処理を継続することが可能

```
try {
        例外が発生するか試す処理
}
catch(例外の型　変数){
        例外発生時の処理
}
finally {
        いずれの場合も実行する処理
}
```

pythonの例外処理は

```
try:
    試行するブロック
except 例外:
    例外発生時実行するブロック
else:
        正常だった時実行するブロック
finally:
      いずれの場合でも最後に実行する
```

List2-3,4抜粋改

```
int a=3;
int b=0;
try {
    int c= a/b;     // 定数0で割る場合はコンパイル時に検出可能
    System.out.println("c=" + c );
}
catch(AlthmeticException e){
    System.out.println(e);
}
finally {
    System.out.println("end");
}
```

tryブロック実行で例外がでなければfinallyブロック実施

AlthemticException例外が発生してもfinallyブロック実施

catchは例外の型別に複数続けて書ける

例外を発生させる(試験や自分で例外を定義するとき)

```
// 例外の作成
Exception e = new Exception( 例外の文字列 );
throw e;     // 例外を意図的に発生させる

// newしないで例外を発生させる
throw new Exception( 例外の文字列 );

// メソッド外へ例外を渡す
戻り値 メソッド名(引数列) throws 例外の型 {
        メソッドの内容
}
```

throwsの後ろの例外は複数指定可能か？

例外をメソッド外へ渡す

```
//  具体的にはリスト2-8
class InvalidAgeException extends Exception {
    InvalidAgeException(String message){  // コンストラクタ
        super(message);                   // 上位へ丸投げ
    }
}
class Person {
   int age;
   void setAge(int age) throws InvalidAgeException {
       if(age<0){
          throw new InvalidAgeException("年齢異常");
       }
       this.age = age;  // このプログラムはage<0でも代入する
   }
}    // mainメソッドは教科書参照
```

setAge(-10)で呼ばれた場合，InvaridAgeExceprionクラスが生成され例外が渡される．そのクラスでは上位Exceprionクラスにメッセージを渡す

スレッド(thread)

複数のプログラムで連動して動かしたい場合

c/Unix(Linux)のfolk

自分の分身を作る．pid(プロセスid)により親か子か判断する。

言語仕様を拡張した並行動作

かつて言語仕様には並行動作が含まれなかったため,
コンパイラメーカが独自に仕様拡張した。
関数単位で並行動作

Ada 言語仕様として並行動作を採用した

タスクという単位で並行動作

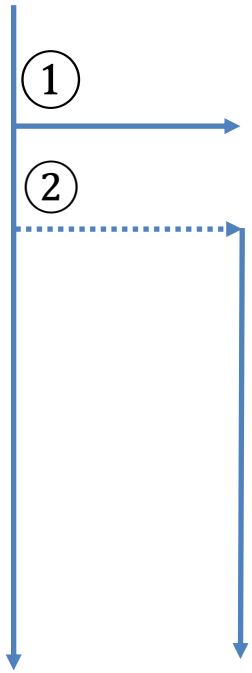
Java クラス単位で並行動作可能

Pythonのyieldは擬似的な平行動作(co-routine), 平行動作パッケージは他にもある

Javaでスレッドをつくる

```
// 何もしないスレッドの作成
Thread t = new Thread();          // ①スレッド作成
t.start();                        // ②スレッド開始
並行で動く部分

// 実装方法1 機能をオーバーライドするメソッドを定義
public class MyThread extends Thread {
    public void run(){   // runメソッドを再定義
        動かしたい処理処理
    }
}
mainのクラス，メソッドの中で
    Thread t = new MyThread(); // ①スレッド作成
    t.start();                 // ②スレッド開始
// start()内でrun()が呼ばれる
スレッドを作る例(リスト3-1参照)
```



```
// 実装方法2 Runnableインターフェースを使う
class MyThread implement Runnable {
    public void run(){  // runメソッドを定義
        動かしたい処理処理
    }
}
mainのクラス，メソッドの中で
   MyThread t = new MyThread();
   Thread thread = new Thread(t);
            // MyThreadのインスタンスでスレッドを作成
   t.start();
```

ソースはList3-2参照

スレッドの実行順

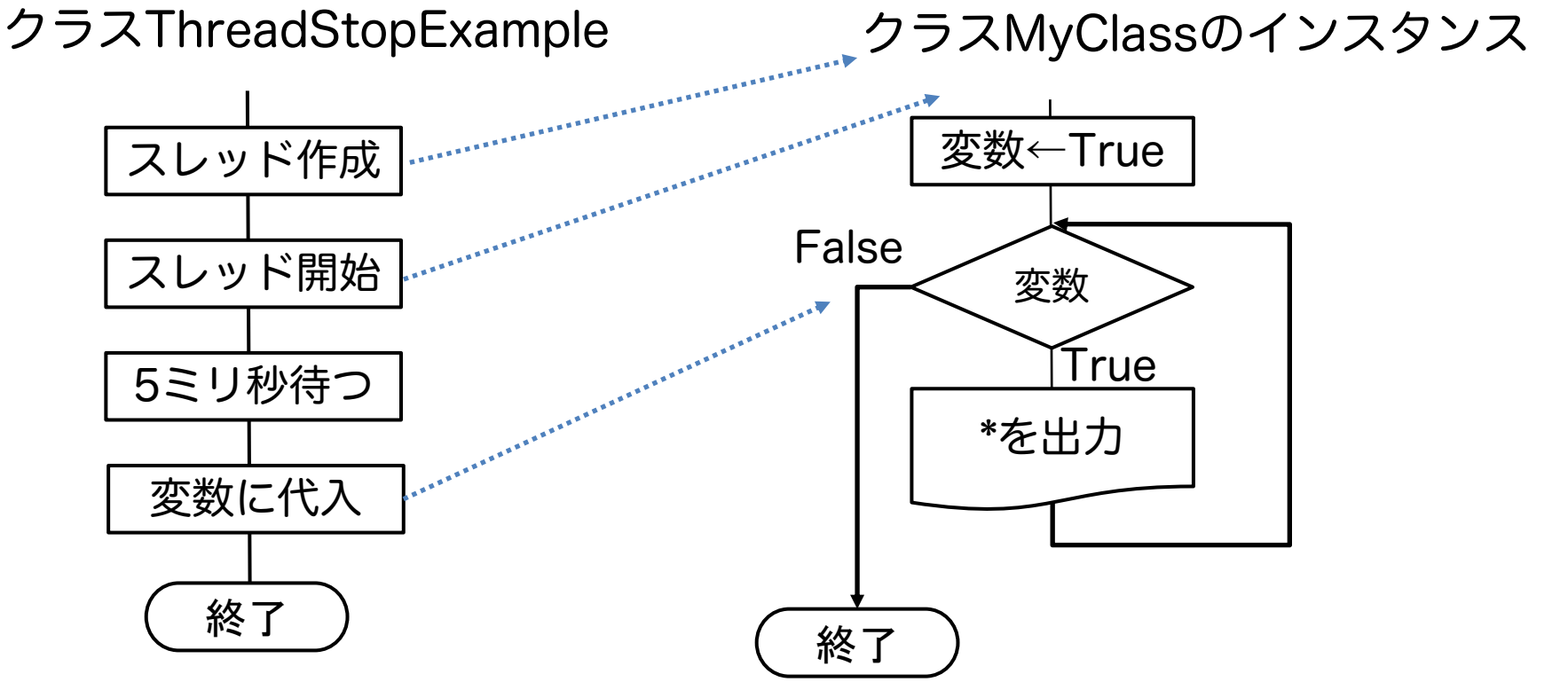
実行順は保証されない
通常は起動順
でもCPU資源を割り当てるのはJava VMやOSによる
同じ出力先（ファイルや標準出力）に複数のスレッドが出力した場合
順番が前後する場合がある
作成されたスレッドが処理があるのに作成側が終了してしまうこともある

順番を保証するにはスレッド間で同期（待ち合わせ）を行う

単純に待つ　　`Thread.sleep(n); // nはミリ秒で指定`
終了を待つ　　`t.join();　　// tはMyThreadクラスのインスタンス`

List3-4で catch(InterruptException e)をしているのは
ctrl-CやUnixのkillコマンドで強制終了させようとした場合に受ける処理を定義するため

クラス外部から動作を制御する（List3-5)



MyClass内のフィールドを変更させるためにはpublicで宣言

このプログラムは環境やタイミングによって結果が異なる

5ミリ秒後にどれだけ動いているか（動いていないかも知れない）

それぞれが相手を気にせず終了する

test & set 問題　スレッドセーフ

```
if( a==0){
        a=1;
}
```

変数aが複数のスレッド（タスク，プロセス）から更新されるとき安全ではない
この処理は　値を取得，判定，値を設定　の命令に変換される
もし，取得後判定前に別のスレッドが値を変更したら処理が矛盾する
この問題の解決のために，値の取得から設定まで，他が動かないようにする
排他的処理
　　test&set命令　　取得から設定までを1命令で行う（マルチプロセッサでx)
　　セマフォ

同期処理(List3-6の問題解決)

```
class Bank {
    static int money = 0;
    static synchronized void addOneYen(){ // 同時に動かせない
       money++;
    }
}
```

レポート3予告 （4点） ソースと結果をA4 1枚(両面可) 10回開始時提出

親からスレッドを30個作る（スレッドを管理する配列を用意する）
各スレッドを開始させ，全ての終了を待つ
終了メッセージProgram endを出力する
各スレッドでは
スレッドのIDを取得，開始時刻を取得，乱数で2から10秒待ち，
終了時刻を取得し，スレッドID，開始時刻，待ち秒数，終了時刻を表示
参考
スレッドID取得　Thread.currentThread().getId()
現在時間取得 java.time.LocalTimeのLocalTime.now()
時刻書式指定 java.time.format.DateTimeFormatterのofPattern()
整数値nまでの乱数 randomインスタンスのnextInt(n)
整形して表示 System.out.prinf(書式指定，表示項目)
出力例

```
    23 start 15:48:11    wait    2 end 15:48:13
    13 start 15:48:11    wait    3 end 15:48:14
    (中略)
Program end
```

注：各スレッドは平行動作させること（全てのスレッドを作成し終えてから開始させる）