

オブジェクト指向プログラミング

黒瀬 浩

kurose@neptune.kanazawa-it.ac.jp

出席はEシラバスアンケートで確認します

前回の復習

パッケージ **package** パッケージ名;
クラス名の有効範囲を管理する単位
packageが違えば同じクラス名でも別物
パッケージ名.クラス名.メソッド名()または変数名 のように指定

例外処理 **try** { 試す部分 }
catch (例外の型 例外のインスタンス){
例外が起きた場合に実行する部分
}
// catchは例外の型を並べて複数定義できる
finally { いずれの場合にも実行する部分 }

例外生成
throw new Exception(例外の文字列);

外部のクラスで例外を処理する

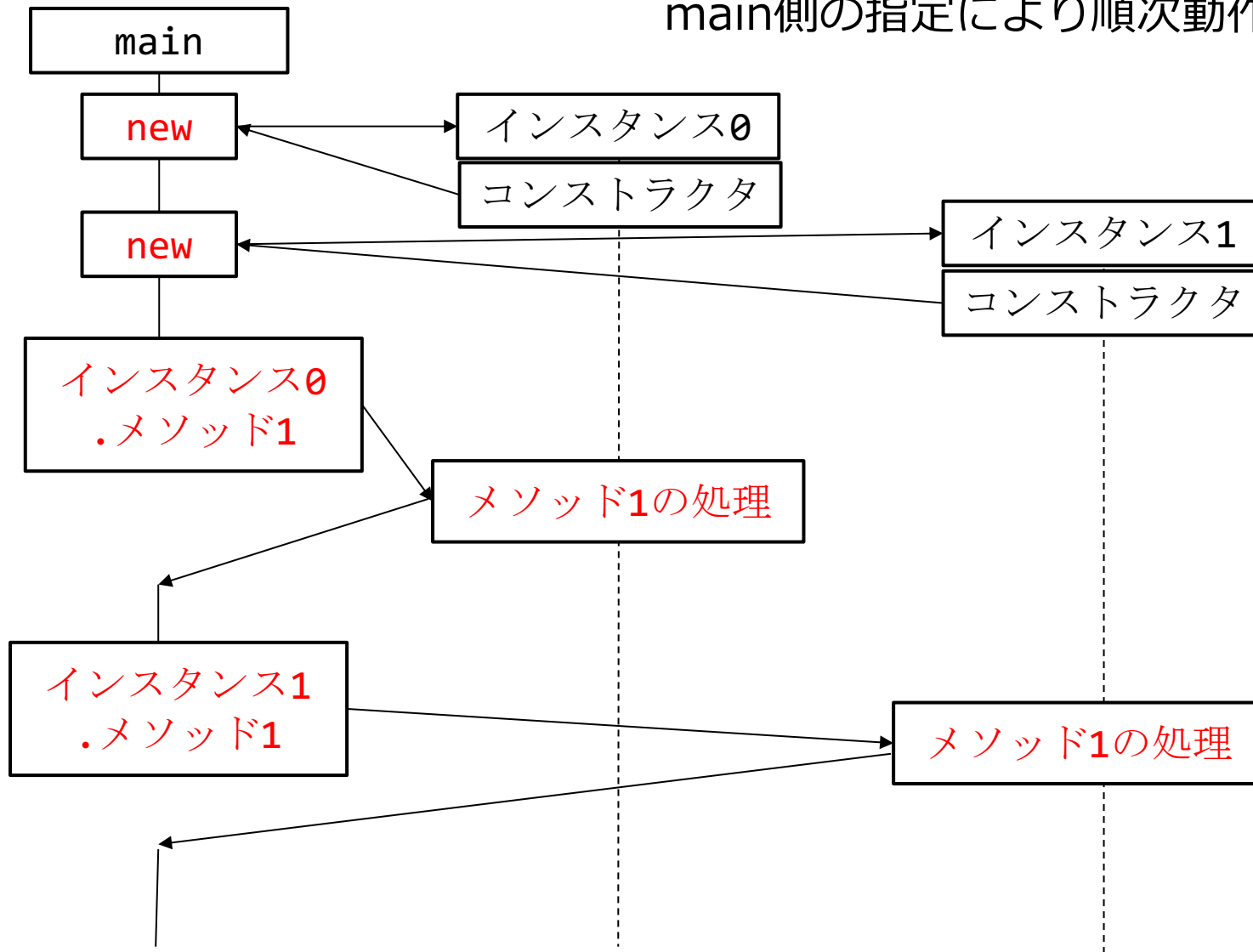
```
class 例外を処理するクラス extends Exception {  
    コンストラクタ(String message){ super(message); }  
}
```

// 別クラスのメソッドで指定

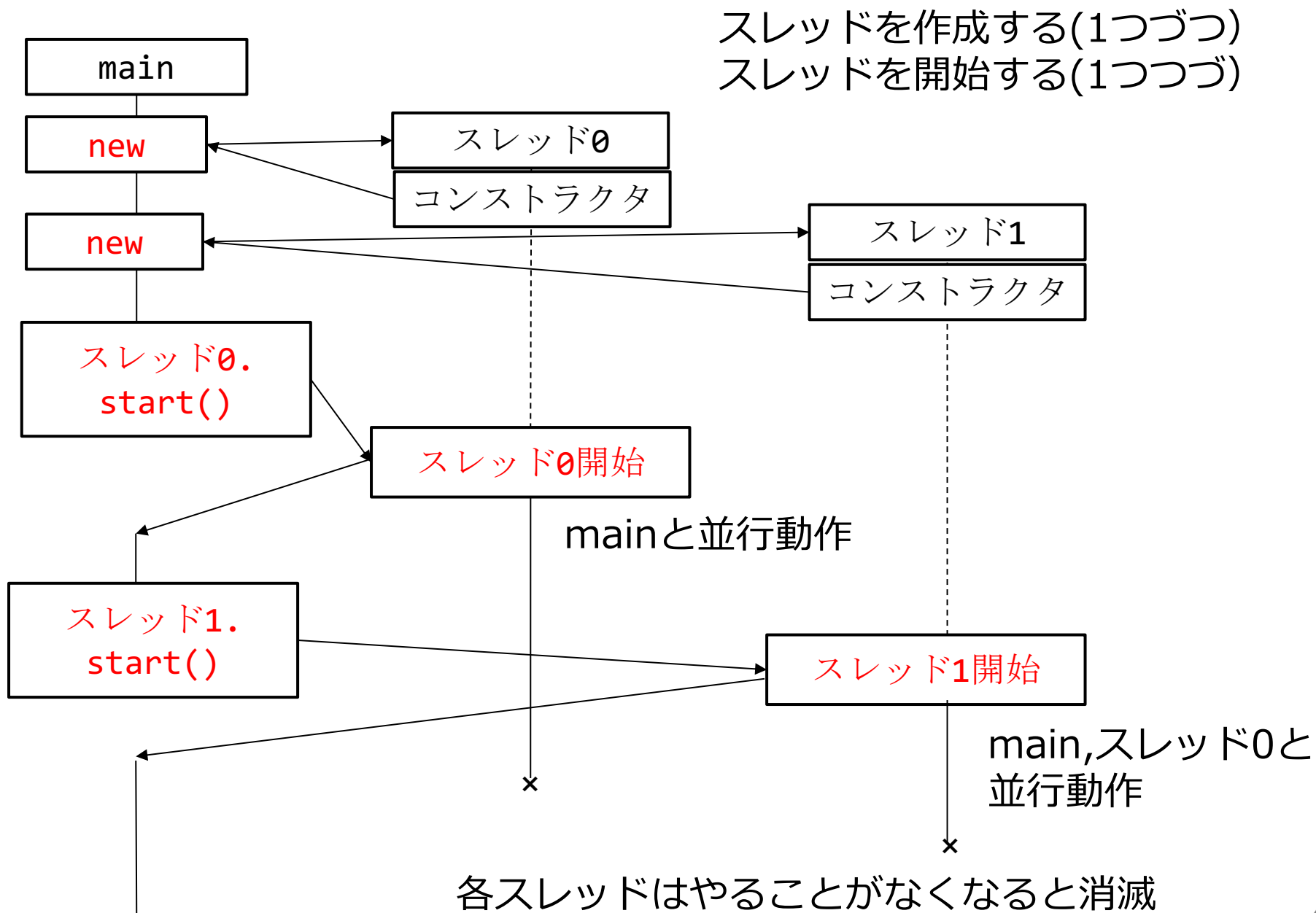
```
例外を発生させるメソッド(int age) throws 例外を処理するクラス { }
```

インスタンス生成とメソッド呼び出し

main側の指定により順次動作する



スレッド生成と開始



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();
```

```
t.start();
```

並行で動く部分

// ①スレッド作成

// ②スレッド開始

// 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
```

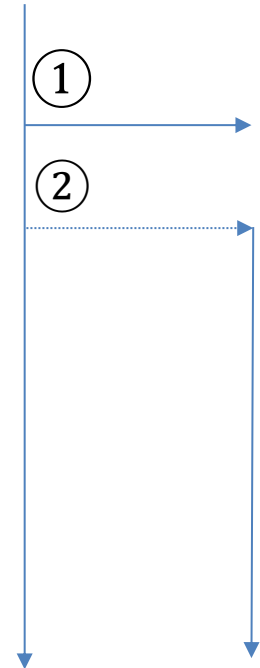
```
t.start();
```

// 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)

main スレッド



Javaでスレッドをつくる

// 実装方法2 Runnableインターフェースを使う

```
class MyThread implements 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にまだ処理があるのに作成側が終了するとどうなるか？

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

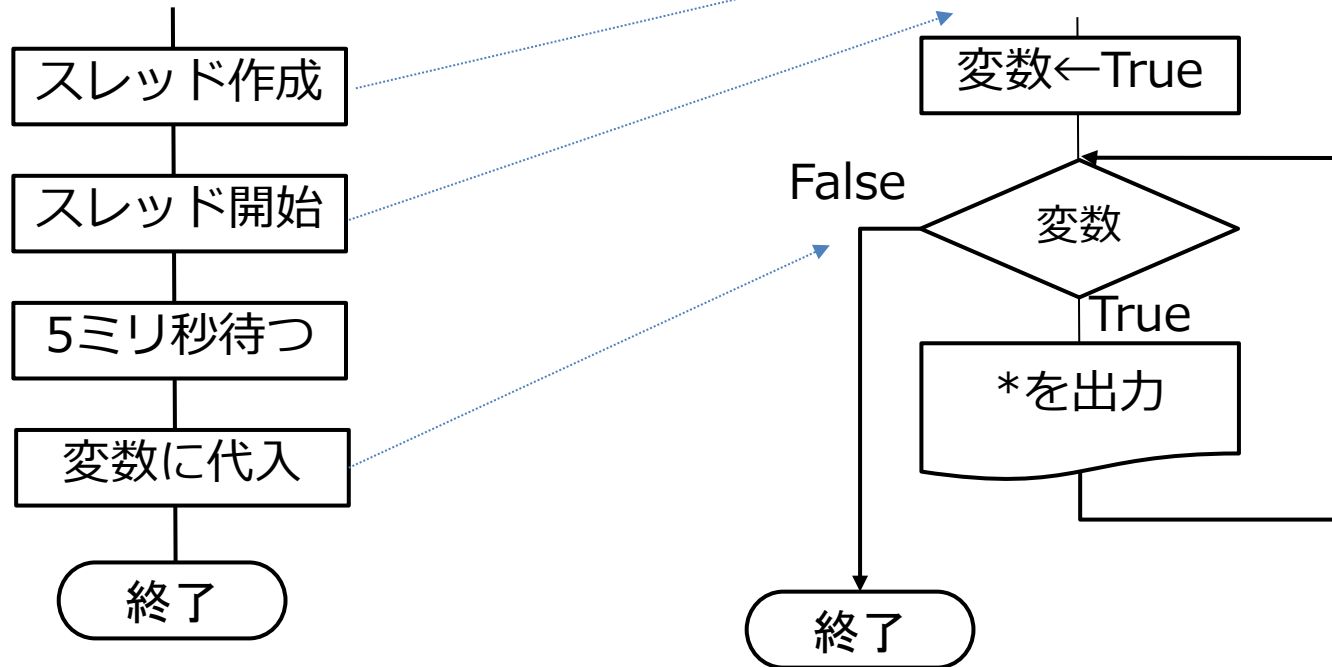
終了を待つ `t.join();` // tはMyThreadクラスのインスタンス

List3-4で `catch(InterruptedException 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)

クラスThreadStopExample

クラスMyClassのインスタンス



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

レポート3 (4点) Eシラバス提出窓口は第9回に用意

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.LocalDateTime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.printf(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)

レポート3 Eシラバスでレポート提出

クラス名列番号, 氏名を明記

- ・ソースコードが確認できるものの画面コピー
- ・実行結果が確認できるものの画面コピー

注意

作成するスレッド数が簡単に変更できるように, 作成したスレッドは配列に格納する

スレッド作成, スレッド開始start(), スレッド終了待ちjoin()は1つずつしかできない

スレッド作成, スレッド開始, 終了待ちは別のforブロックにする

開始にはJava, OSの処理が多いので, 作成し終わってからstartさせる

例外処理を入れないとエラーになるはず

出力は見やすいように桁合わせをして出力する(sprintf)

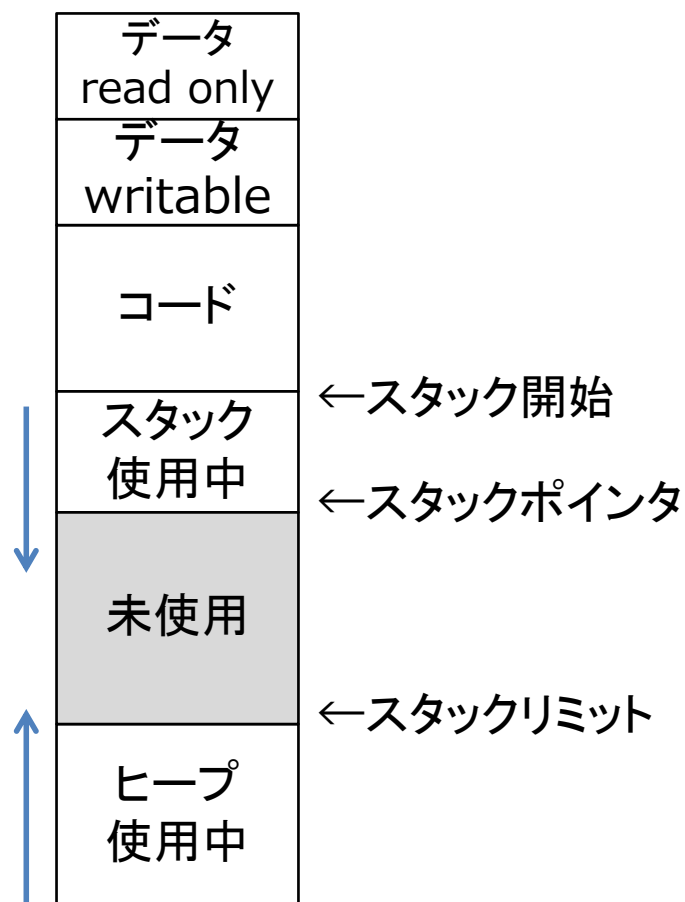
スレッド側の出力は1回で行う(複数回に分けると, 他のスレッドと出力が混ざってしまうことがある)

出力は待ち時間が短い順になるはずである, 短い順になっていなければプログラムの処理が間違っている

参考：メモリ配置

スタックはpush()で成長、pop()で縮小なので、伸び縮みするだけ
ヒープはアプリケーションの都合で使用、解放が行われる
=> スタックとは別の領域が必要

プログラムのアドレス空間



スタック、ヒープはどれだけ使うかわからない
多くの機種では、

スタックはアドレスの小さい方から
ヒープはアドレスの大きい方から

割りあてる

重なってしまったら、これ以上実行できない

stack overflow, heap overflow

のエラーを出して強制終了させられる

左図でスタックポインタ, スタックリミットは
状況により動的に変動する

ガーベッジコレクションの必要性

C言語ではプログラマが確保した領域はfree()してメモリを解放する

Javaではインスタンス終了時に不要な部分を解放するが
その都度、解放していると動作が遅くなる

そこで、

メモリが不足してきたら、Java VMが未使用部分をまとめて解放する

いつどのように解放するかはJava VMやOSにおまかせとなるが
途中でガーベッジコレクションを起こしたく無いこともある
(リアルタイム処理で遅延を許容できない場合)

以下で強制的にガーベッジコレクションを起こさせることができる

```
Runtime.getRuntime().gc();
```

ArrayList

配列は追加や削除が動的に行えない

作成 `ArrayList<型> 変数名 = new ArrayList<型>();`

メソッド

追加 `add(型 値)`

取得 `get(int 要素番号)`

削除 `remove(int 要素番号)`

要素数 `size()`

リスト5-1抜粋

```
ArrayList<String> months = new ArrayList<String>(); // []
months.add("Jan"); // ["Jan"]
months.add("Feb"); // ["Jan", "Feb"]
months.add("Mar"); // ["Jan", "Feb", "Mar"]
months.remove(1); // ["Jan", "Mar"]
months.size() // 2
```

基本型	ラッパークラス
boolean	Boolean
byte	Byte
char	Character
short	Short
int	Integer
long	Long
float	Float
double	Double

既に出てきた `Integer.parseInt()` はintラッパークラスのメソッド

List5-2抜粋

```
ArrayList<Integer> integerList = new ArrayList<Integer>();  
integerList.add(new Integer(50)); // Integerクラスに50を渡す  
Integer integer0 = integerList.get(0); // 型はInteger  
int i0 = integer0.intValue(); // 整数化
```

intは値を1つしか持てないがIntegerはメソッドも持てる

コレクション

データをまとめて扱う

配列 同じ型の値を連続して管理する
要素の追加削除ができない

ArrayListクラス 要素の追加削除可能(pythonのlistに類似)

HashMapクラス キーと値で管理する(pythonのdictに類似)

HashSetクラス 集合. 重複は1つになる(pythonのsetに類似)

型が異なるものをまとめる場合


C言語 構造体

Python list,dict,setの要素の型は同じでなくても可, クラスも利用可

Java データのみのクラスを定義

Javaのコレクションフレームワーク

リスト

要素の順番を管理する
要素の取り出し, 追加, 削除
ArrayList, LinkedList 

マップ

キーと値(Key Value)
他言語のハッシュ, 辞書と同じ
HashMap, LinkedHashMap

セット

要素に重複が起きない
他言語のセット, 集合と同じ
HashSet, TreeSet

LinkedListは, 要素から次の要素の位置と前の位置を管理している
前からたどる場合, 後ろからたどる場合, 要素の追加, 要素の削除で高速

HashMapの例

```
HashMap<String, String> map = new HashMap<String, String>();  
map.put("key1", "val1");          // 要素の追加 上の< >内で文字列2つと定義  
map.put("key2", "val2");  
map.entrySet(); // キーと値のペアの一覧取得  
// printすると [key1=val1, key2=val2] 順番は保証されない  
  
map.values();          // 値一覧取得  
// printすると [val1, val2] 順番は保証されない  
  
map.keySet();          // キー一覧取得  
// printすると [key1, key2] 順番は保証されない  
  
map.get("key1");        // キーがkey1の値を取得, キーがなければnull  
// printすると val1
```

値を指定してキー一覧を得たい場合は？
既に存在するキーに対してput()するとどうなるか？

イテレータ(Iterator) 反復子

一つづつとり出す

```
HashSet<String> set = new HashSet<String>();  
set.add("A");  
set.add("B");  
Iretator<String> it = set.iterator();  
while( it.hasNext() ){  
    String str = it.next();  
    System.out.println(str);  
}
```

キーボードから読む

```
Scanner sc = new Scanner(System.in);  
sc.next();    // イテレータと同じ
```

拡張for文 ひとつづつ取り出して処理する

```
for(型 変数 : コレクション){ }
```

以下と似ている

pythonのfor文

```
for 変数 in コレクション:  
    ブロック
```

phpのforeach文

```
foreach ( 変数 as 配列 ){ }
```

List5-7抜粋改

```
String [] months={"Jan", "Feb", "Mar"}  
for(String str : month){  
    System.out.println(str);  
}
```

キュー, スタック

キュー

末尾に追加, 最初から取り出し(先入先出し,FIFO)

スタック

末尾に追加, 末尾から取り出し(後入先出し,LIFO)

キュー, スタックの応用

キュー: 待ち行列, 構造探索

スタック: 演算の計算, 関数呼び出しの管理, 構造探索, 番号付け

LinkedListのメソッド

isEmpty()	空ならtrueを返す
offer(値)	キューに追加
poll()	キューから取り出し. 空なら null
push(値)	スタックに追加
pop()	スタックから取り出し

pythonではlistでキュー, スタックを実装している

appendで末尾に追加

pop(0)で先頭取り出し(キュー)

pop(-1)で末尾取り出し(スタック)

内部クラス リスト6-1

クラスの中にクラスを定義する

```
class Outer {
    private String message = "*Outer";
    void doSomething(){
        class inner {
            void print(){
                System.out.println("*Inner");
                System.out.println(message); // class外
            }
        } // doSomething()の中でインスタンスを作成している
        Inner inner = new Inner();
        inner.print();
    }
}

public class InnerClassExample {
    public static void main(String[] args){
        Outer outer = new Outer();
        outer.doSomething();
    }
}
```

匿名クラス 名前のないクラス

```
interface SayHello {  
    public void hello(); // 抽象メソッド  
}  
  
class Greeting {  
    // SayHelloインターフェースを実装するクラスを引数として受け取る  
    static void greet(SayHello s){ // (2) 渡されたpをsに  
        s.hello(); // (3) p.hello()と同じ  
    }  
}  
  
class Person implements SayHello {  
    public void hello(){ // (4) 最終的にここが呼ばれる  
        System.out.println("Hello");  
    }  
}  
  
public class SimpleExampe {  
    public static void main(String[] args){  
        Person p = new Person();  
        Greeting.greet(p); // (1) Person型のpインスタンスを渡す  
    }  
}  
  
// 型にインターフェース名を使うことにより  
// クラスを限定せずにそのクラスに応じたメソッドを呼び出す
```

ラムダ式 無名関数：簡単な処理をその場で定義して呼び出す

```
(int n) -> { return n+1; }
```

整数nを受け取りn+1を返す

```
(int a, int b) -> { return a+b; }
```

整数a,bを受け取りa+bを返す

引数の型は省略できる

```
(n) -> { return n+1; }
```

nを受け取りn+1を返す

引数が1つなら()を省略できる

```
n -> { return n+1; }
```

nを受け取りn+1を返す

実施する文が1つなら { return ;}を省略できる

```
n -> n+1;
```

nを受け取りn+1を返す

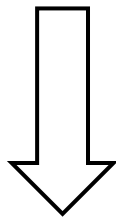
引数がない場合

```
() -> System.out.println("こんにちは");
```

コレクションフレームワークとラムダ式 list6-6抜粋

平面座標x,yを管理するPontクラスを定義

```
for(Point p : pointList){  
    p.x *= 2;  
    p.y *= 2;  
}  
  
for(Point p : pointList){  
    p.printInf();  
}
```



```
pointList.forEach( p -> { p.x *= 2; p.y *= 2; } );  
pointList.forEach( p -> p.printInfo() );
```


演習

4章, 5章の例題

4章, 5章の章末問題

出席はEシラバスアンケート

今後

11回 入出力, レポート4

12回 達成度確認試験(筆記)

13, 14回 アプリケーション作成(レポート5)

15回 解説・補足など