コレクション

2018/6/5(火)　プログラミングIV 第八回

福井大学　工学研究科　情報・メディア工学専攻

長谷川達人

折返し地点です

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.

例外処理
復習

- 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
復習（printStackTrace()）

プログラムが例外で強制終了した際に赤く出力されるエラーメッセージが**StackTrace**である.

try-catch()するとStackTraceが本来表示されて強制終了したところをcatchするので，StackTraceが表示されない．これを意図的に表示するのが`e.printStackTrace()`である.

```
catch(ArithmeticException e){
  e.printStackTrace();
}
```

0除算をcatchしたときの出力例

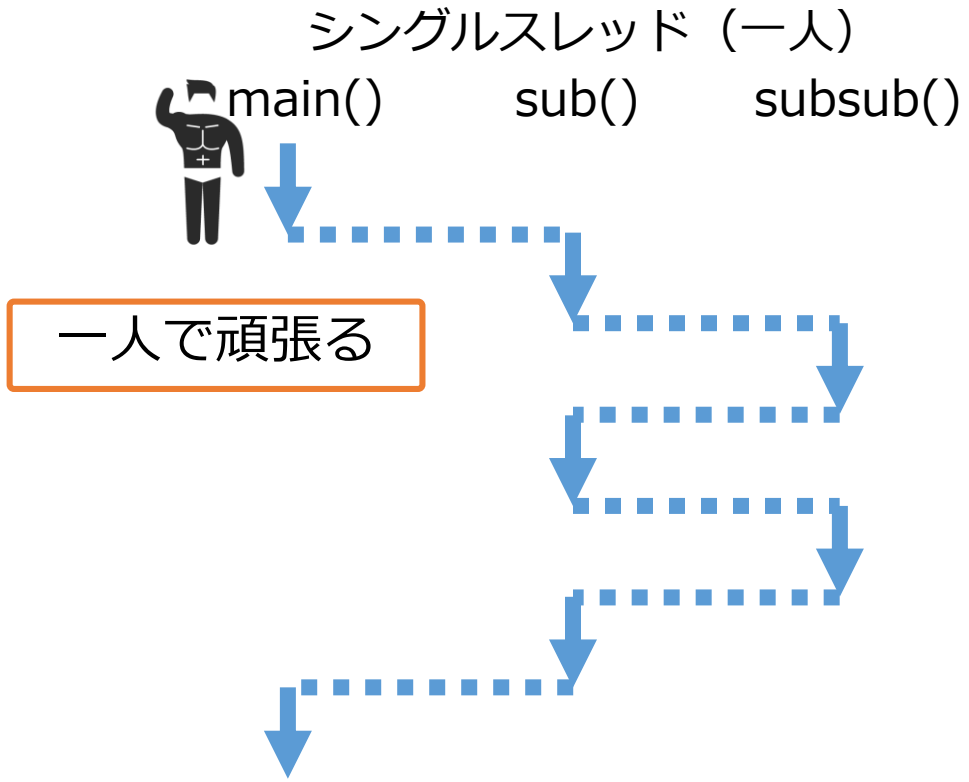
```
java.lang.ArithmeticException: / by zero
at Shiryo.main(Shiryo.java:8)
```

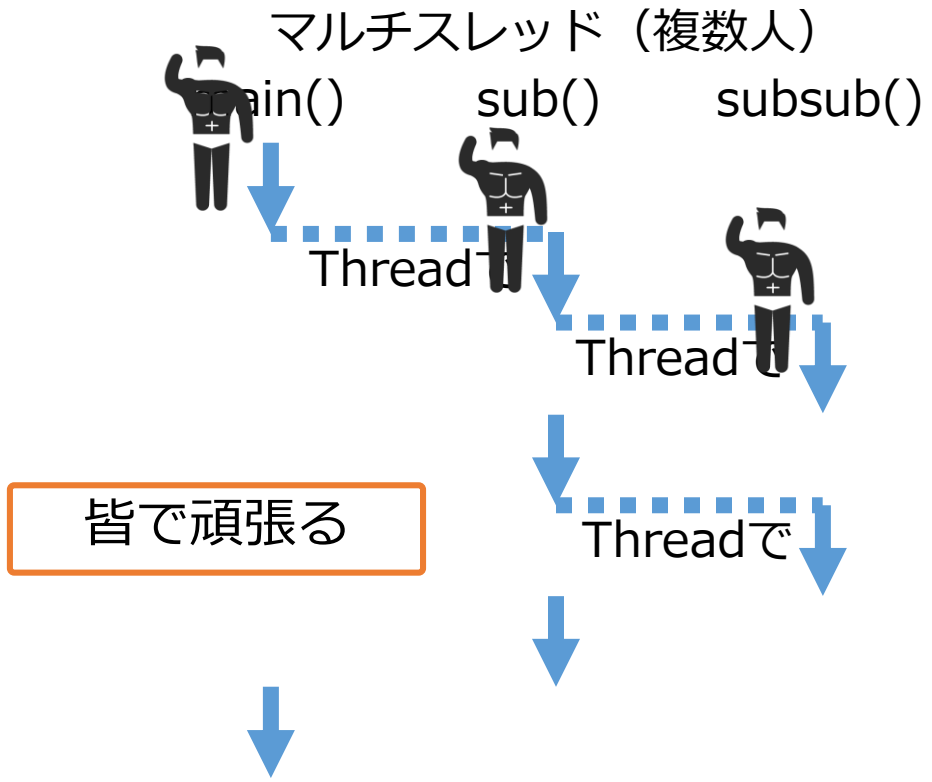
StackTraceはException発生源を追跡した情報でありデバッグに超便利

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

スレッド
復習

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
復習

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）

個人的によく使う２の方法を次のページで復習する.

なお，**ThreadもRunnableもrun()をオーバーライドしたいというだけ**なので，難しく考えなくて良い.

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(){
        // ココに書かれた処理が並列処理される
        // 長時間かかる処理や，ループで定期的に実行したい処理を書く．
    }

    public void test(){
        Thread thread = new Thread(this);

        thread.start();              // A. 並列処理でrun()を呼び出す場合
        thread.run();                // B. 普通にrun()を呼び出す場合
    }
}
```

1．Runnableを実装する

2．run()をオーバーライド

3．Threadインスタンスの作成

4．実行

この使い方は最低限できてほしい．
AとBの実行の違いが説明できてほしい．

スレッド
復習：無名クラス

今までインタフェース（例えばRunnable）を実装したクラスをインスタンス化するには次のようにするしかなかった.

```
public class MyRunnable implements Runnable{
    @Override
    public void run() {
        // ココに並列処理したい内容を書く
    }
}
```

1．Runnableをimplementsしたクラスを実装

```
public class Main {
    public static void main(String[] args){
        Runnable runnable = new MyRunnable();
    }
}
```

2．上記クラスをインスタンス化

スレッド
復習：無名クラス

以下のように，わざわざファイルを分けなくても，インスタンス化する際に処理を定義できる．この一度だけ使われるクラスを無名クラスと言う．

```
public class Main {
    public static void main(String[] args){
        Runnable runnable = new Runnable(){
            @Override
            public void run(){
                // ココに並列処理したい内容を書く
            }
        };
    }
}
```

1．上記クラスをインスタンス化しながら

2．Runnableの実装内容を同時に定義することができる．

スレッド
復習：無名クラス

```
public class Main {
    public static void main(String[] args){
        Runnable runnable = new Runnable(){
            @Override
            public void run(){
                // ココに並列処理したい内容を書く
            }
        };

        Thread thread = new Thread(runnable);
        thread.start();
    }
}
```

この部分が，
インスタンス化（ new ）と
クラスの定義（void run(){}等）
をまとめてやっている．

Runnableを実装したクラスの
インスタンスを生成している．

Threadに入れて，start()すれば，
並列処理が実行できる．

スレッド
復習：無名クラス

```
public class Main {
    public static void main(String[] args){
        Thread thread = new Thread(new Runnable(){
            @Override
            public void run(){
                // ココに並列処理したい内容を書く
            }
        });
        thread.start();
    }
}
```

更にまとめて記述するとこんな感じ

無名クラスが書けると，**わざわざThread用にクラスを作らなくても良くなる**．単純な処理は上記のようにその場でサッと並列処理ができて便利！

質問への回答

- 例外の必要性がわからなかった．全部コンパイルエラーにして．
  - 文法上問題あるものがコンパイルエラー
  - 実行してみて，中の値に応じて発生したりシなかったりするのが例外
- 例外は全てException型でcatchしたらだめなのか．
  - 別に良い．catch(Exception e){}と書いておくだけで全てcatchできる．
  - しかし，本当に安全なシステムというものを目指す場合は，きちんと状況に応じて対処を変えられる方が良い（企業が提供するシステム等）．
- try-catchの中でtry-catchすることはできるのか．
  - できる．前回のclose()が処理されない問題は，これで対応される事が多い．

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");
}
```

write()で例外発生時にfwがclose()されずメモリリークにつながる．

例外処理

裏課題：練習問題１の実は．．．

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

finallyにclose()で対応するのが一般的である．その際try-catchを忘れずに書く．

この冗長な書き方はJavaSE7からtry-with-resource文という新しい構文で，少し簡潔に書けるようになった．興味のある人は調べてみてほしい．

質問への回答

- sleep()の精度はなぜ保証されないのか.
  - 100%信用しないでほしいというだけで，基本的には問題ない.
  - sleep(1000)して1ずれるかどうかくらい.
- sleep()はなぜtry-catchしなければならないのか.
  - **InterruptedException e**をcatchしなければならないのだが，これは割り込みが発生したときに起こる例外である．すなわちsleep()中は割り込みが起こり得て，起こった場合どうすればよいかを書いてほしいので.
- thread.stop()はなぜ非推奨なのか.
  - 安全にスレッドを停止することが保証できないから.
  - https://docs.oracle.com/javase/jp/7/technotes/guides/concurrency/threadPrimitiveDeprecation.html

質問への回答

- Threadを休止するのがThread.sleep()ならmain()を休止するのは？
  - Thread.sleep()はmain()というスレッドも休止させられる.
- スレッドを使うメリットは？
  - 音を鳴らしながら，100万桁の足し算をしながら，GUI表示を変えながら，，，みたいなときに使うと便利である.
  - シングルでも実装できないことはないが，並列っぽく見えるためには，タスクA～Cを適宜切り替えながら処理する必要があり面倒である.
- スレッドを使うデメリットは？
  - 使い方を誤ると死なないThreadが誕生しメモリリークが発生しうる.
  - 共通のデータを複数Threadから参照する場合，データ整合性を確保するためには色々な工夫が必要である（授業ではあえて触れていない）.

質問への回答

- スレッドが使えればコールバックはわからなくてもよいか.
  - ThreadもActionListenerもまずは使えることが大事である.
  - しかし，動作の理屈が理解できている方が応用の幅が広くて良い.
- Threadの数に制限はないのか.
  - 基本的にはメモリの限界まで可能.
  - これはJava全般に言える（例えば配列要素数の上限など）.
- 今いくつのスレッドが同時に動いているのか知る方法はあるか.
  - 各Threadはインスタンスとして変数に代入して保持している必要がある. これを怠ると正しくThreadが終了させられなくなるリスクがある.
  - ということで，保持しているインスタンスを数えるしかない.
  - Javaは基本的にインスタンス変数として保持されなくなると，ガーベージコレクションが勝手にデータを破棄していく.

質問への回答

- プログラミングが苦痛になってきたがどうすればよいか.
  - 授業なので色々な機能を説明しているが，Javaはポリモーフィズムまでがわかっていれば後はググりながら対応できる.
  - わからない回があっても，まぁそれはそれとして，作りたいものを作るという点で気楽に学んでほしい.
- 演習の点が低いと単位を取るのは厳しいか.
  - 一応第四回に回答したとおりと考えている.
  - 各回演習：4割，自由開発：3割，期末試験：3割
- ３回くらい休んでいるが秀は狙えるのか.
  - 3回休み/15回*4割=8点を失っているので，まだギリ狙える.
- 自由開発でスレッドやコレクションを使うみたいな条件はあるか.
  - 条件はないが，色々使ったことをアピールしてくれると点数が上がる.

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）
    }
}
```

actionPerformed()が実装された場所を渡すと考えても良い.　（今回は自分が実装しているので**this**）

run()が実装された場所を渡すと考えても良い.　（今回は自分が実装しているので**this**）

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	**コレクション，ジェネリクス**	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

コレクションの使い方と
ジェネリクスという考え方について学ぶ

目標

コレクションを**とりあえず使う**事ができる.
ジェネリクスを説明できる.

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）

コレクション

ArrayList

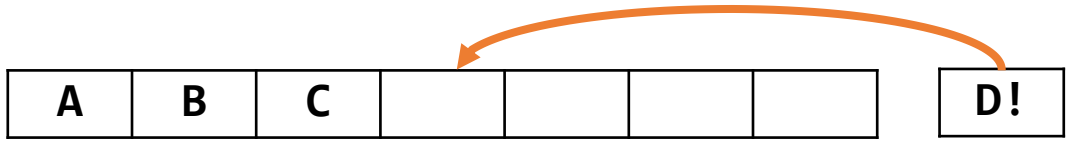
サイズの上限がない（長さが自由な）配列を扱う方法として
ArrayListがある.

```
// ArrayListインスタンスの生成
ArrayList<String> list = new ArrayList<String>();

// 要素の追加：コンポーネントのようにadd
list.add("1つ目");
list.add(0, "2つ目");

// 要素の取得：配列のようにindexで取得
System.out.println(list.get(0));
System.out.println(list.get(1));
```

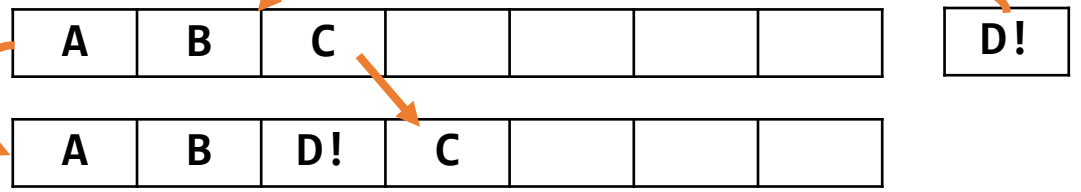
出力	2つ目 1つ目

配列と異なりサイズを指定する必要がない

add(xxx)は一番末尾に要素を追加する



add(num,xxx)はnumの位置に要素を挿入する



コレクション

ArrayList

要素の追加や削除が配列より楽なので**使えると非常に便利**である.

```
// 前のページの続き...
// 要素の削除: indexを指定して削除
list.remove(0);
// 再度中身を表示
System.out.println(list.get(0));
// 要素数の取得
System.out.println(list.size());
```

出力	1つ目 1

コレクション
ArrayList

ArrayListをインスタンス化するとき<String>と記述した．これはArrayListに**代入する要素の型**を指定するものである．

自作のMonster型を指定し，大量のインスタンスを格納する，といった処理も可能である．

コレクション
ジェネリクス

APIリファレンスでArrayListを調べると，次のように書かれている.

```
public class ArrayList<E>
extends AbstractList<E>
implements List<E>, RandomAccess, Cloneable, Serializable
```

この ”E” は**型パラメータ**と呼ばれ「後から型を決めて」という意味になる．同様にArrayListのget()メソッドを調べると，戻り値がE型になっていることが確認できる.

```
public E get(int index)

このリスト内の指定された位置にある要素を返します。
```

このように，後から決まる型のことを**ジェネリクス型（総称型）**と呼び，このような仕組み自体を**ジェネリクス**と呼ぶ.

コレクション
様々なコレクション

複数オブジェクトを管理するためのクラスやインタフェースを総称して**コレクション**と呼び，主に3つに分類できる.

- **リスト**（implements List）
  - 要素の並び順に意味のある場合に使用する（配列に近い）.
  - ArrayList，LinkedList等がある.
- **マップ**（implements Map）
  - キーと値（オブジェクト）をペアで管理する場合に使用する.
  - HashMapやLinkedHashMap，TreeMap等がある.
- **セット**（implements Set）
  - 要素の並び順に意味がなく，重複を許さない場合に使用する.
  - HashSetやTreeSet等がある.

ListやMapインタフェースが実装されている＝内部手続きはさておき，皆同じようなメソッドを持っているよという意味

コレクション
リストコレクション

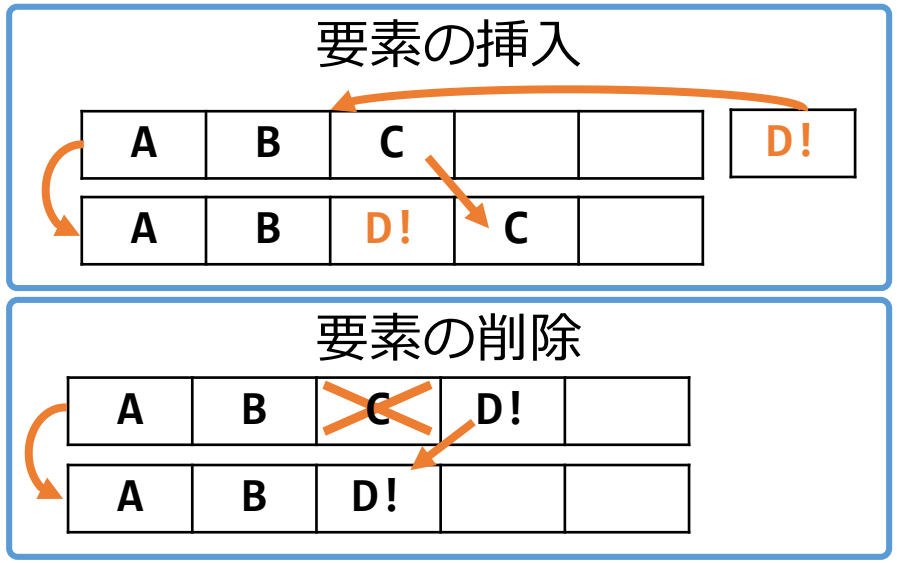
ArrayListやLinkedListがあるが，基本的に使い方は同じである.

よく使うメソッド	説明
`boolean add(E e)`	要素eをリストの末尾に追加する
`boolean add(int i, E e)`	要素eをリストのiの位置に挿入する
`void clear()`	すべての要素をリストから削除する
`boolean contains(Object o)`	要素oがリストに含まれているかを返す
`E get(int i)`	iの位置の要素を返す（indexは0スタート）
`int indexOf(Object o)`	要素oがリスト中最初に発見されたindexを返す
`boolean isEmpty()`	リストが空かどうかを返す
`E remove(int i)`	iの位置の要素を削除する（削除する要素を返す）
`boolean remove(Object o)`	要素oがあれば削除する（削除したかを返す）
`E set(int i, E e)`	iの位置の要素を要素eと置き換える（戻り値は同上）
`int size()`	リスト内の要素数を返す

コレクション
リストコレクションの違い（ArrayListとLinkedList）

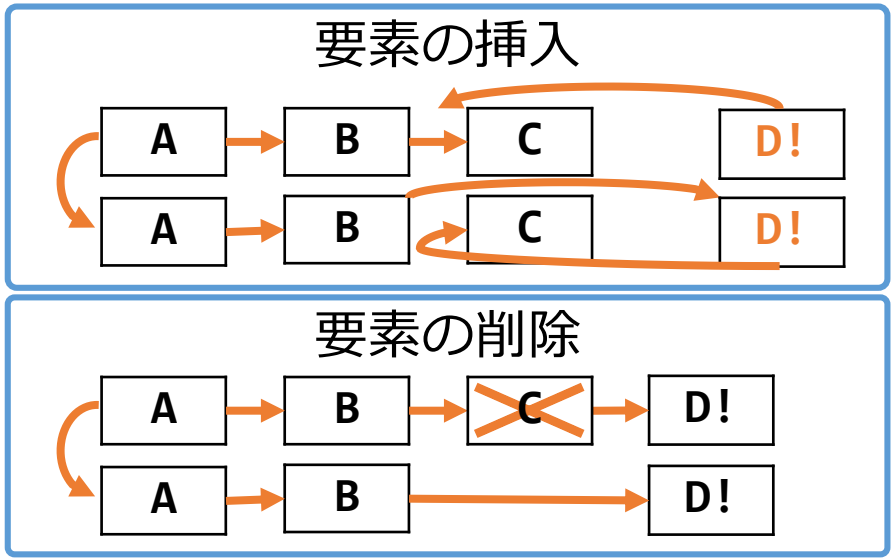
ArrayListは配列の拡張版であり，LinkedListは「データ構造とアルゴリズム」の授業で学習したリストをイメージすると良い.

ArrayList



長所：要素へのアクセスが高速
短所：挿入削除が低速

LinkedList



長所：挿入削除が高速
短所：要素へのアクセスが低速

コレクション

練習問題１：ArrayListを使ってみる

1. String型を要素として持つArrayListインスタンスlistを生成し，次の要素を順次代入せよ.
   要素[“あ”, “い”, “う”, “え”, “お”, “あ”, “え”]
2. for文を用いてすべての要素をprint()でコンソール出力せよ.
3. 改行を出力せよ（System.out.println();と書け）.
4. 3番目の要素（index=2）を削除せよ.
5. “え”の要素をすべて削除せよ.
6. 拡張for文を用いてすべての要素をprint()でコンソール出力せよ.

質問への回答

どうでもいいコメントたち

- 福井のお薦めの遊び場所は？一乗谷には行ったことある？
  - 待って，長谷川はまだ福井歴1年ちょい.
  - 一乗谷は面白いところ？
- 夏は暇な時何をしている？
  - 大学教員の夏休みと春休みは実は学会シーズンである（暇がない）.
  - 趣味的にはインラインスケート，ボードゲーム，漫画，ゴルフもしたい.
- ハトヤマスタンプもっと売れてほしい.
  - 確かに売れてほしい.
  - 売れると利益が出るがたくさん使っても別に利益は出ない.
- javaを始めたら大学生活が楽しくなりました！
- javaを初めて勉強もスポーツもうまくいきました！
  - 進研ゼミのヤツ！！

質問への回答

どうでもいいコメントたち

- ポケモンやモンスト的なものはもう作れそう？
  - 頑張れば作れる．モンスト作るの楽しそう．ポケモンはひたすらゴリゴリコーディング系な予感.
- パズドラまだやってる？
  - 先日日曜に「スマホ依存改善支援」に関する研究で東京で発表.
  - その時，幕張でちょうどパズドラのイベントがあったので，朝5時半に起きて幕張寄ってきたレベルで，パズドラやってる.
- ダンジョンメーカーってアプリすごく面白いのでおすすめ.
  - 実はバズってるの見て東京往復の電車でプレイしたところドハマリした.
  - こいつのおかげで，少しパズドラ熱が冷めた.
  - 繰り返しやりたくなるゲーム要素がたくさん組み込まれていて，研究的にも非常に参考になる（と言い訳しつつプレイ）.

コレクション
マップコレクション

マップコレクションはキーと値（オブジェクト）をペアで管理する方法である．例えばHashMapだと次のように使用する．

```
// HashMapインスタンスの生成
HashMap<String, String> map = new HashMap<String, String>();

// 要素の追加
map.put("Name", "はせがわ");
map.put("Tel", "0776-27-8929");
map.put("eMail", "t-hase@u-fukui.ac.jp");

// 要素の取得
System.out.println(map.get("eMail"));
System.out.println(map.get("Tel"));
```

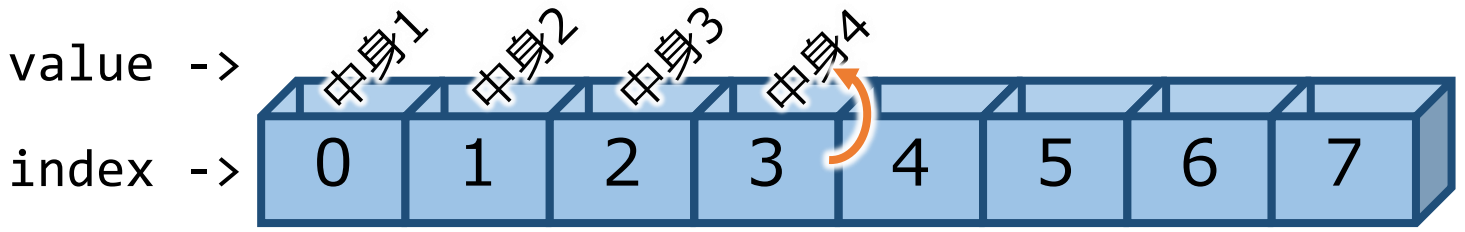
“Name”といえば”はせがわ”
“Tel”といえば” 0776-27-8929”
のようにキーと値を紐づけて
管理する

出力	t-hase@u-fukui.ac.jp 0776-27-8929

コレクション
マップコレクション

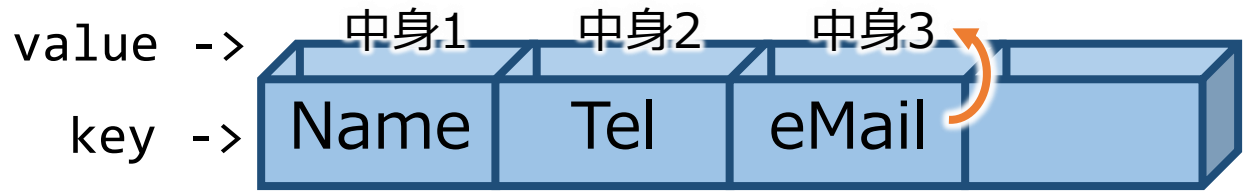
“Name” といえば ”はせがわ”のように，配列のindexの代わりにオブジェクトが使用できる．**連想配列**とも呼ばれる．

普通の配列
String[] array



array[0]で“中身1”にアクセスできる

連想配列
HashMap array



array.get(“Name”)で”中身1”にアクセスできる

コレクション
マップコレクション

キーの重複はできないため，第二回:課題３のTreeMapでやったように，賞味期限をキーにして在庫数を管理するといった処理に応用できる.

```
// 前のページの続き...

// 要素の削除
map.remove("Tel");

// 要素数の取得
System.out.println(map.size());
System.out.println(map.get("Tel"));
```

存在しないキーにアクセスするとnullが返ってくる.

出力

```
2
null
```

コレクション
マップコレクション

表中のVやKは，前述のEと
同義のジェネリクス型である

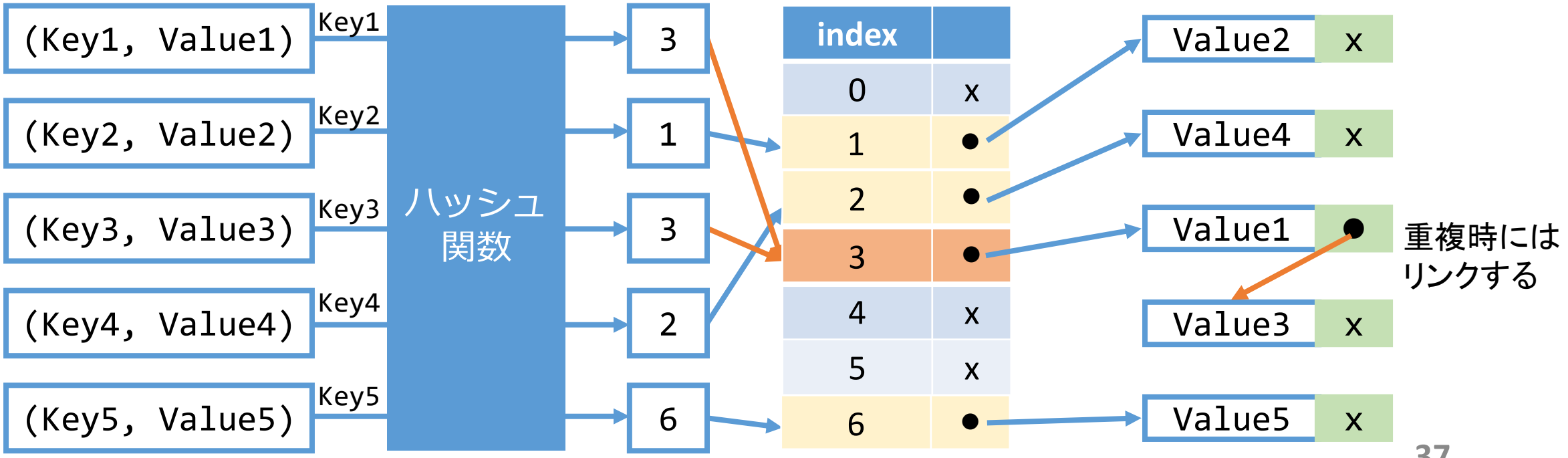
HashMapやTreeMapがあるが，基本的に使い方は同じである.

よく使うメソッド	説明
`boolean containsKey(Object key)`	指定されたkeyがマップに含まれているかを返す
`boolean containsValue(Object value)`	指定されたvalueがマッピングされているかを返す
`V get(Object key)`	keyに対応する値を返す（ない場合はnull）
`V getOrDefault(Object key, V default)`	keyに対応する値を返す（ない場合はdefault）
`boolean isEmpty()`	リストが空かどうかを返す
`Set<K> keySet()`	このマップに含まれるキーのSetビューを返す
`V put(K key, V value)`	keyとvalueのペアをマップに追加する
`V remove(Object key)`	マップからkeyを削除する(削除したValueを返す)
`V replace(K key, V value)`	指定されたkeyがマッピングされている場合に値を置換する（削除したValueを返す）
`int size(), void clear()`	リストと同じ

コレクション

HashMap

HashMapは内部のハッシュテーブルに値を格納する．その際，キーからハッシュ値を計算しハッシュ値をindexとすることで素早く要素にアクセスできるデータ構造である．



コレクション
マップコレクションの違い（HashMapとTreeMap）

HashMapはキーの順序を保証しないのに対して，TreeMapではキーを二分探索木のアルゴリズムでソートし，順序を保証する.

- キーをString型やMonster型にしたい場合は，順序関係をComparatorで定義する必要がある（ソートの時に説明する）.

したがってTreeMapは，「XXX以上，xxx以下のキーを持つ要素の取得」といった操作が可能となる.

- 例：賞味期限が20180605以前の要素を取得

当然ながら，要素の追加・削除・検索は，内部の二分木をたどる必要があるため，HashMapより処理時間がかかる.

コレクション
セットコレクション

セットコレクションは値（オブジェクト）の重複がないことを保証する．例えばHashSetだと次のように使用する．

```
// HashSetインスタンスの生成
HashSet<String> set = new HashSet<String>();

// 要素の追加
set.add("はせがわ");
set.add("ふくま");
set.add("はせがわ");    // 既にあるので追加されない

// 要素の取得（というより存在確認）
System.out.println(set.contains("はせがわ"));
```

出力	true

コレクション
セットコレクション

セットコレクションは値（オブジェクト）の重複がないことを保証する．例えばHashSetだと次のように使用する．

```
// 前のページの続き...

// 要素の削除
set.remove("はせがわ");

// 要素数の取得
System.out.println(set.size());
System.out.println(set.contains("はせがわ"));
```

出力	1 false

コレクション
ラッパークラス

（復習）基本型：intやdouble等，値を直接格納する型
参照型：インスタンスや配列等，参照値を格納する型

ラッパークラス（Wrapper Class）は，基本型を参照型で包み込んだ（Wrapした）クラスである．色々と便利なメソッドを持つ．

基本型
int型やdouble型
値を直接持つ

↓ ラッパークラス

参照型
Integer型やDouble型
参照値を持つ

こんなイメージ

```
public class IntValue {
    private int value = 0;
    public IntValue(int v){
        this.value = v;
    }
    public int getValue(){return this.value;}
    // ･･･色々と便利なメソッドがある．．．
}
```

int型の1つの値を管理するだけのクラス

コレクション
基本型とジェネリクス

実は，ArrayList<int>はエラーが出る．これは，ジェネリクスが実際は**インスタンスへの参照**を格納するものだからである．

したがって，基本型をArrayListで使用したい場合，参照型である**ラッパークラス**を用いる必要がある．

int型のラッパークラス

```
ArrayList<Integer> list = new ArrayList<Integer>();
list.add(new Integer(10));          // 本来はこのようにInteger型を代入
Integer i = list.get(0);            // インスタンスの取得
System.out.println(i.intValue()); // 値の取得としなければならないが

list.add(10);                       // このように省略できるので気にしなくてよい．
System.out.println(list.get(0));
```

コレクション
イテレータ

イテレータとは何か？ということだけ知っておいてほしい.

練習問題１で拡張for文が使えることをこっそり示したが，拡張for文が使える条件としてIterable<E>インタフェースを実装しているということがある.

Iterableを実装するとIterator（イテレータ）インスタンスを取得することができるようになる．**イテレータはコレクションの中の要素を1つずつ順に参照する**能力（hasNext()とnext()メソッド）を持っている.

```
Iterator<Integer> it = list.iterator();
while(it.hasNext()){                    // 次の要素があるか確認
    System.out.println(it.next());  // 次の要素を取得
}
```

これを簡潔に使えるようにしたものが拡張for文である.

コレクション
拡張for文

ArrayListやHashSetはIterableなので拡張for文が使える.

```
ArrayList<String> list = new ArrayList<String>();
for(String str : list){
    System.out.println(str);
}
HashSet<String> set = new HashSet();
for(String str : set){
    System.out.println(str);
}
```

HashMapはIterableではないのでそのままでは使えない.
一組のペアをMap.Entryが定義しているので以下のように使う.

```
HashMap<String, String> map = new HashMap();
for(Map.Entry<String, String> e : map.entrySet()){
    System.out.println(e.getKey() + ", " + e.getValue());
}
```

Entry型のSetを取得して拡張forしている.

コレクション

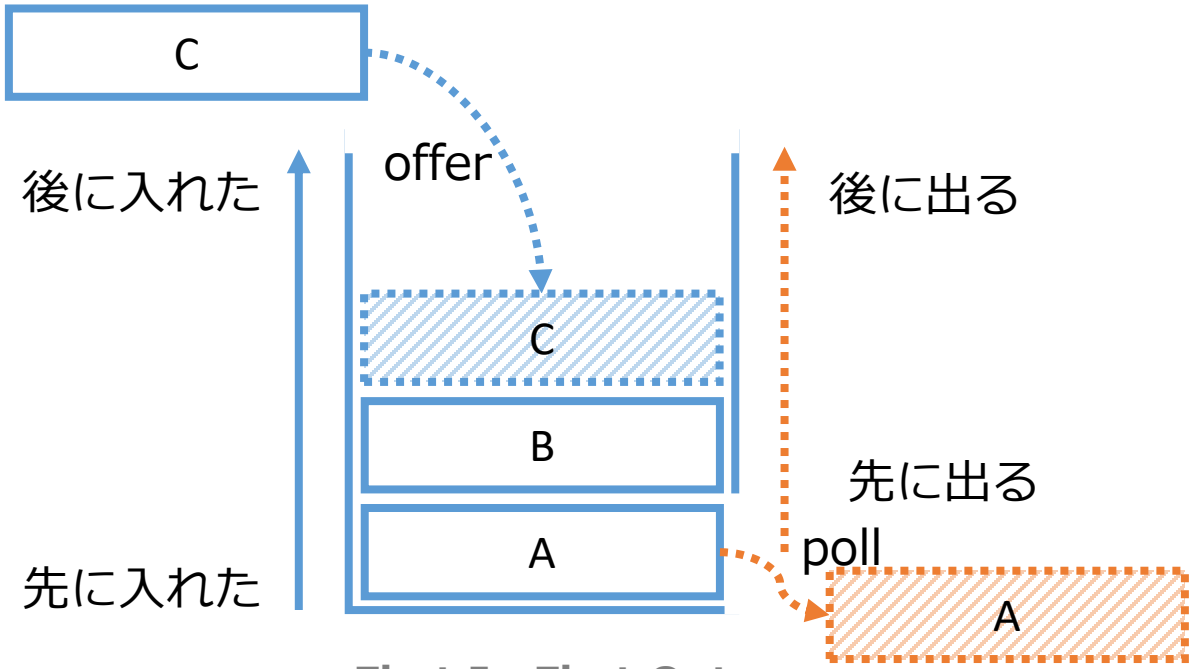
練習問題２：HashMapを使ってみる

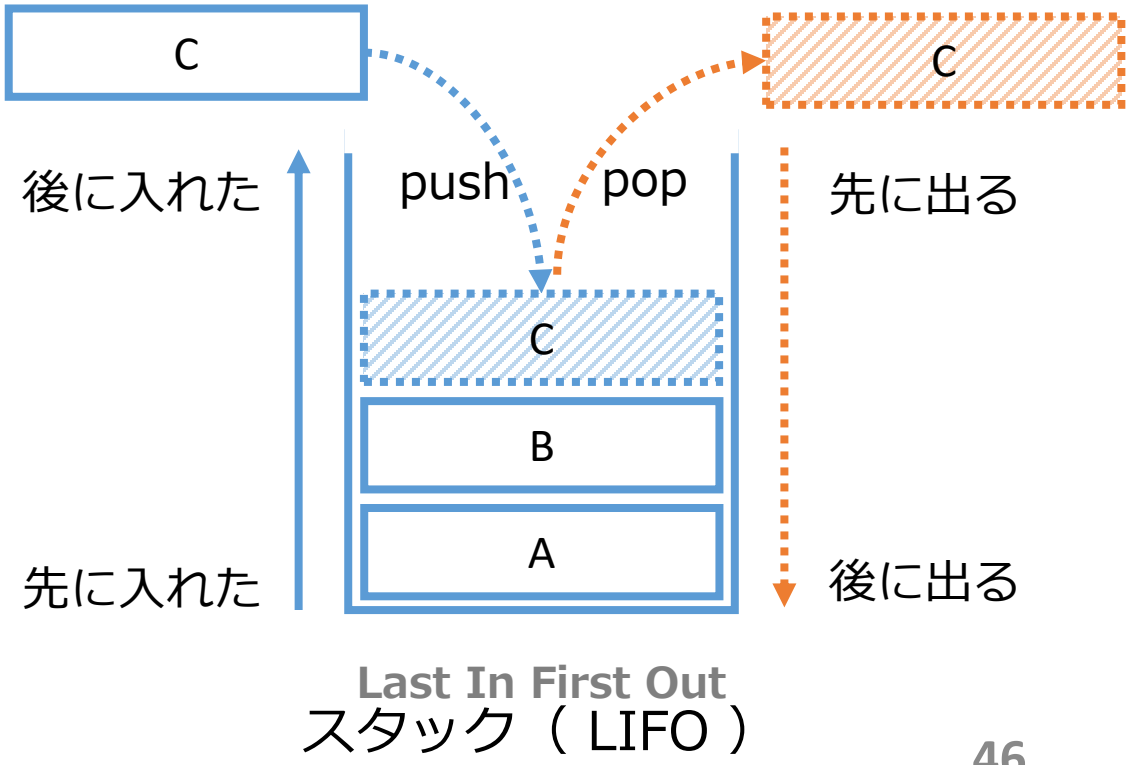
Key	Value
MacBookAir	89000
MacBookPro	129000
iPad	37000
iPhone8	85000
iPhoneX	132000

1. String型の品名をキーに，int型の標準価格を値として持つHashMapインスタンスmapを生成し，左の表の要素を順次代入せよ.
2. キーが“iPad”, “iPhone8”の標準価格をコンソール出力せよ.
3. “iPad”が販売停止になったので削除せよ.
4. “iPhone8”の標準価格が10%オフになったので更新せよ.
5. 拡張for文を用いてすべてのキーと値のペアを次の書式でコンソール出力せよ.
   品名：MacBookAir, 標準価格：89000円

コレクション
キューとスタック

データ処理の基本手法としてキューとスタックがある.
学習済みとは思うが念のため復習する.





コレクション
キューとスタック

LinkedListはキューとスタックを既に実装している.
キューの機能はQueueインタフェースで定義され，Queueを拡張しスタックの機能も使えるようにしたものがDequeである.

```java
// キューとして使う
Queue<String> queue = new LinkedList<String>();
queue.offer("A");
queue.offer("B");
queue.offer("C");
System.out.println(queue.poll());
System.out.println(queue.poll());
System.out.println(queue.poll());
System.out.println(queue.poll());
```

出力

```
A
B
C
null
```

```java
// スタックとして使う
Deque<String> stack = new LinkedList<String>();
stack.push("A");
stack.push("B");
stack.push("C");
System.out.println(stack.pop());
System.out.println(stack.pop());
System.out.println(stack.pop());
System.out.println(stack.pop());
```

出力

```
C
B
A
Exception in thread "main"
java.util.NoSuchElementException
```

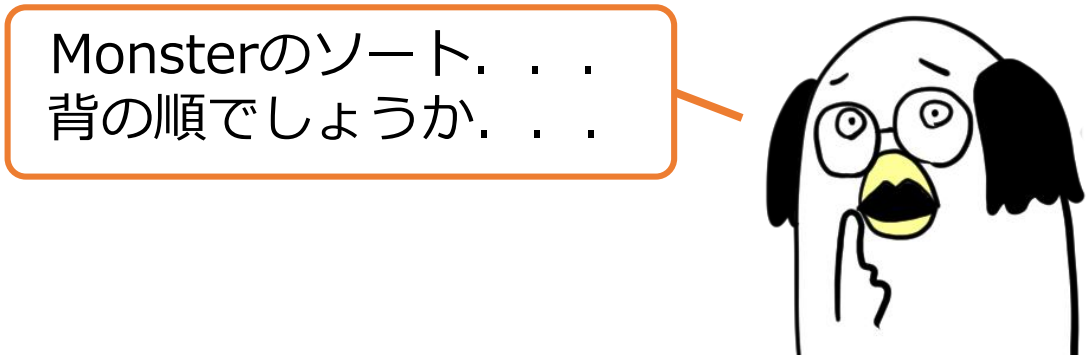
コレクション
ソート

リストの中身をソートするにはCollections.sort()を使う.

```
ArrayList<String> list = new ArrayList<String>();
Collections.sort(list);
```

数値や文字列の場合は標準で定義されたCompalatorが昇順を定義しているので昇順に並べてくれる.

では，Monster型をソートしたい場合どうすればよいだろうか？



コレクション
ソート：Comparableインタフェース

Collections.sort()でソートするには，引数のリストが持つ要素の型が**Comparableインタフェース**を実装している必要がある．

＞私は**比較できます**という機能を持ったインタフェース

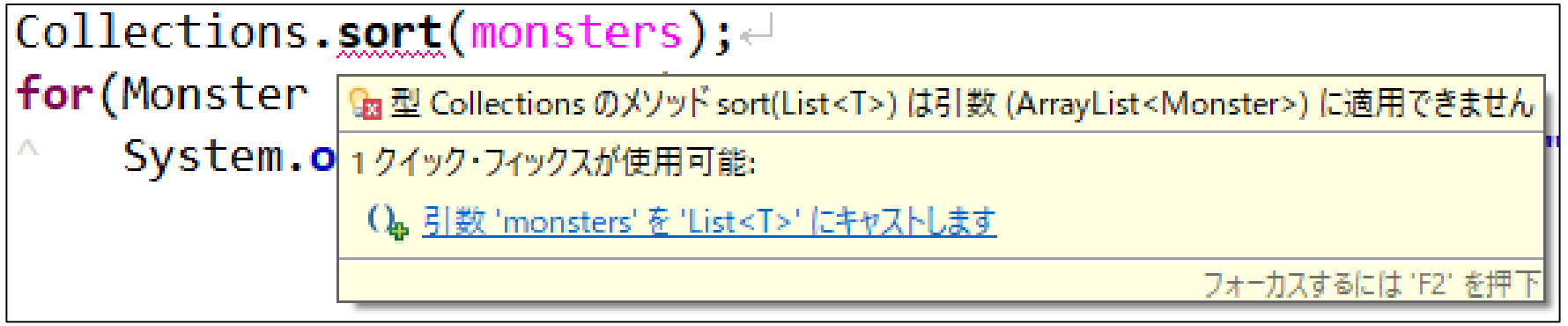
```
public class Monster implements Comparable<Monster>{
  public String name = "";
  public int hp = 0;
  public Monster(String name,int hp){
    this.name = name; this.hp = hp;
  }

  @Override
  // 比較対象(引数m)と自分を比較してソート基準を定義する．
  // この時，自分が大きいときに正の値を返すように定義する．
  public int compareTo(Monster m) {
    return this.hp - m.hp;
  }
}
```

Comparableを実装

並べ替える時には，何かしらの「大小」を比較する．その比較の基準を定義する必要がある．

コレクション
ソート：Comparableインタフェース

Monsterクラスが Comparableを実装していない場合実行できない.



実装している場合，Monsterのリストを HPでソートできる.

```
ArrayList<Monster> monsters = new ArrayList<Monster>();
monsters.add(new Monster("はせがわ", 70));
monsters.add(new Monster("ひょうどう", 120));
monsters.add(new Monster("とねがわ", 50));
for(Monster m : monsters)
  System.out.println("ソート前:" + m.name + "(" + m.hp + ")");
Collections.sort(monsters);
for(Monster m : monsters)
  System.out.println("ソート後:" + m.name + "(" + m.hp + ")");
```

出力

```
ソート前:はせがわ(70)
ソート前:ひょうどう(120)
ソート前:とねがわ(50)
ソート後:とねがわ(50)
ソート後:はせがわ(70)
ソート後:ひょうどう(120)
```

まとめ
コレクション

複数オブジェクトを管理するコレクションは主に3つに分類できる.

- **リスト**（implements List）
  - 要素の並び順に意味のある場合に使用する（配列に近い）.
  - ArrayList，LinkedList等がある.
- **マップ**（implements Map）
  - キーと値（オブジェクト）をペアで管理する場合に使用する.
  - HashMapやLinkedHashMap，TreeMap等がある.
- **セット**（implements Set）
  - 要素の並び順に意味がなく，重複を許さない場合に使用する.
  - HashSetやTreeSet等がある.

まとめ
コレクション

コレクションを扱う際などに，後から決まる型のことを**ジェネリクス型（総称型）**と呼び，このような仕組み自体を**ジェネリクス**と呼ぶ．以下のように<>で型を記述する．

```
ArrayList<Monster> list = new ArrayList<Monster>();
```

コレクションの中には，イテレータ（順番に取り出す）やソートが可能なものもあり，それぞれIteratableやComparableインタフェースを実装しているかどうかで判断できる．

＞APIリファレンスを読んで判断できるようになるとなお良い．

次週予告

※次週以降も計算機室

前半

Java開発時によく使う標準APIについて学ぶ

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

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．．．）