CollectionとLambda式

オブジェクト指向プログラミング特論

2018年度

只木進一：工学系研究科

同じクラスのインスタンスの集合

- 順序のあるもの：Listなど
- 待ち行列：Queue
- 要素の重複を許さないもの：Set
- 鍵と値の組：Map

Generic

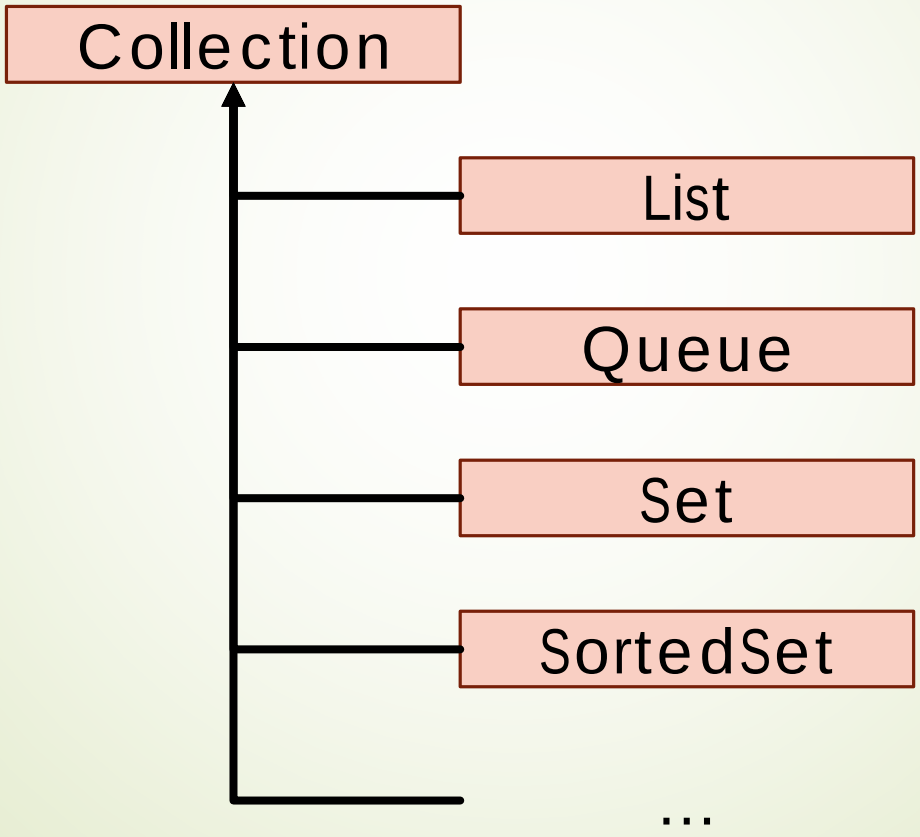
- 型パラメタを明示することで、コンパイラが型の整合性を確認できるようにすること。
- Collectionなどでは、保存する型を明示することで、出し入れを安全に行う。

java.util.Collection

- オブジェクトの集まりを保存するための最上位のインターフェース
- 保存する型を指定
- 基本的なメソッドが定義されている
  - boolean add() : 要素を追加
  - boolean contains() : 要素を含むか否か
  - boolean isEmpty() : 空か否か
  - boolean remove() : 指定された要素を削除
  - int size() : 要素数
  - Stream stream() : 逐次的streamを返す

java.util.Collectionから派生したインターフェース

注意：Mapは別種類

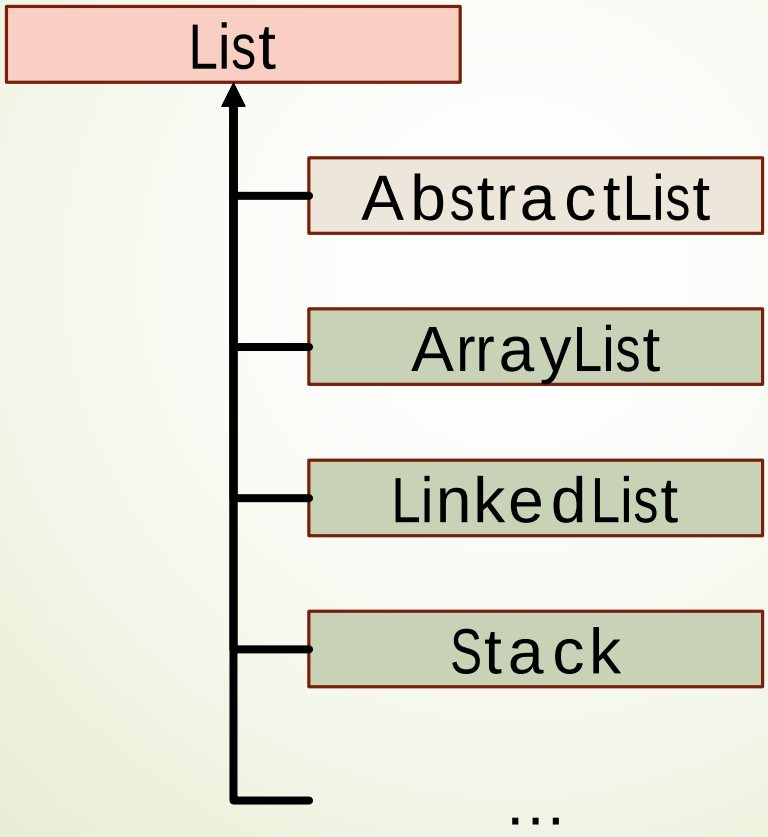


全てInterface

java.util.List

- 要素を順序つけて保存する
- 基本的メソッド
  - boolean add() : 要素を終端に追加
    - 失敗すると例外を発生
  - E get() : 指定された位置の要素
  - int indexOf() : 指定された要素の位置
  - E set() : 指定された位置に指定された要素を置く。戻り値は元の要素

java.util.Listの実装



要素へのアクセスが速い

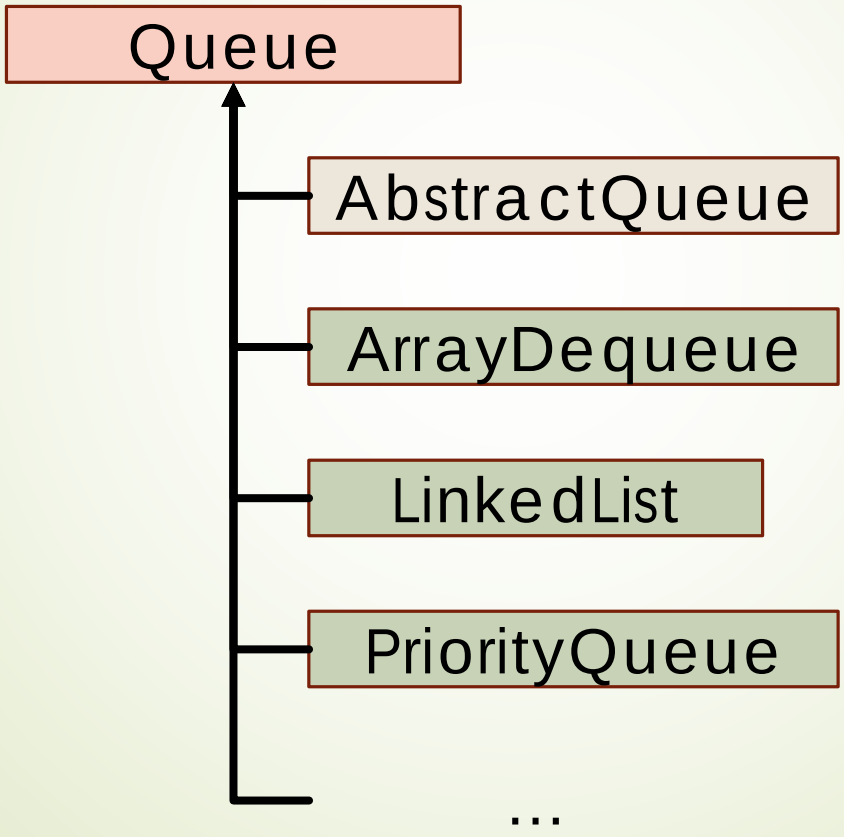
要素の追加・削除が速い

java.util.Queue

- 待ち行列
  - FIFOを想定したメソッド
  - 失敗した際に例外を返すメソッドと特殊な値を返すメソッド

動作	失敗時に例外を返す	失敗時に特殊な値を返す	失敗時の値
終端へ追加	add	offer	false
先頭の取出	remove	poll	null
先頭の値	element	peek	null

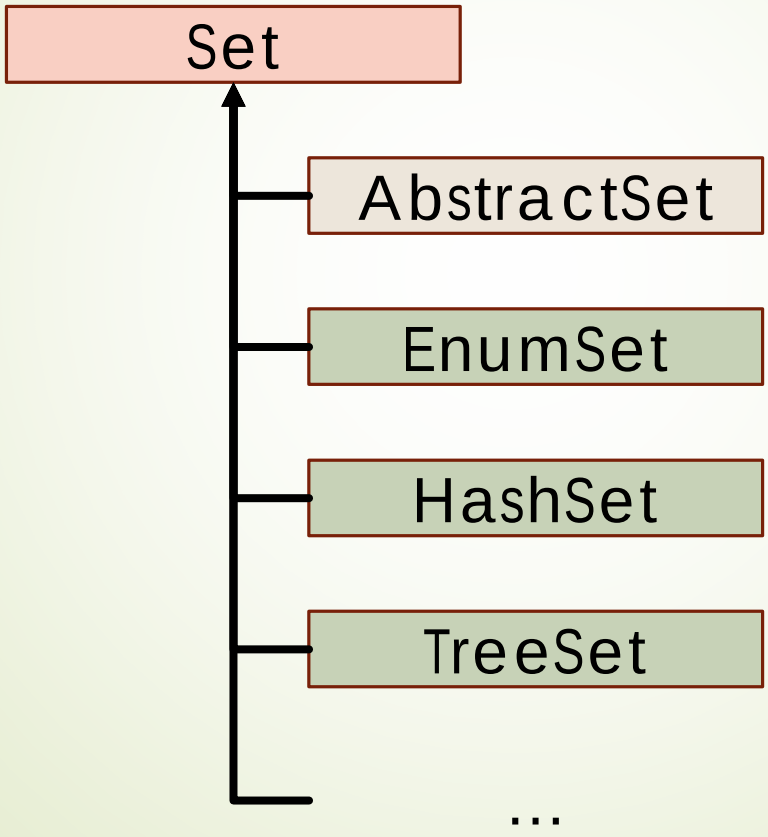
java.util.Queueの実装



java.util.Set

- 要素の重複を許さない
  - equal()がtrueとなることで判断
- 要素が入っているかの有無しかメソッドが無い

java.util.Setの実装



Collectionsクラス
コレクション操作メソッド群

- 探索
- 最大・最小
- 逆順
- スレッド保護：後述
- 整列
- スワップ

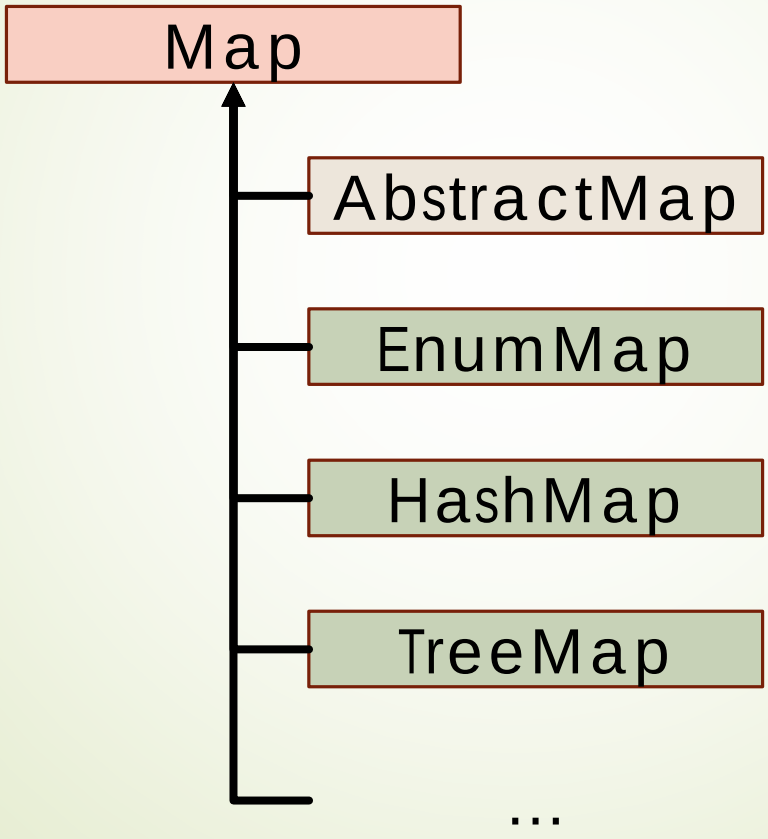
Arraysクラス
配列関連メソッド群

- 配列のリスト化（固定長）
- 探索
- 配列コピー
- 比較
- 整列
- 文字列化

java.util.Map

- 鍵と値の組を保存
- 基本的メソッド
  - V get() : 鍵に対応する値
  - Set<K> keySet() : 鍵の集合
  - V put() : 鍵と値を保存
  - Collection<V> values() : 値のコレクション

java.util.Mapの実装



ThreadとCollection

- Collectionに対して複数のthreadから読み書きすることに対する保護
  - Collectionsクラスのstatic methods
    - Collections.synchronizedList()
    - Collections.synchornizedSet()
    - 他

Collectionの全ての要素に対する操作

- 拡張されたfor

```
List<T> list;
for(T t  : list ){
}
```

- StreamとLambda式の利用

java.util.stream.Stream

- Collectionの要素をstreamへ
  - 要素の一つ一つを取り出す（順序は別）
- 主要なメソッド
  - filter()：指定されたものを抽出
  - forEach()：各要素に
  - forEachOrdered()：順番に
  - reduce()：集約

Lambda式

- 関数を変数として扱う方法
- C/C++ならば関数ポインタを利用
- Javaではインターフェースを利用
- java.util.function.*に様々なインターフェース
  - 自分で定義しても良い

java.util.function.*の例

- BinaryOperator<T>
  - 同じ型Tの2つのオペランドに作用してオペランドと同じ型の結果を生成する演算
- DoubleBinaryOperator
  - 2つのdouble値オペランドに作用してdouble値の結果を生成する演算
- DoubleFunction<R>
  - 1つのdouble値引数を受け取って結果(型R)を生成する関数

Lambdaの利用

- Streamとともに
  - Collectionの要素に対する処理
- Comparator
  - 要素の比較方法

NewMain.java

```java
package example;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

/**
 *
 * @author tadaki
 */
public class NewMain {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int n = 20;
        Map<Integer, Double> map = new HashMap<>();
        for (int i = 0; i < n; i++) {
            map.put(i, Math.random());
        }

        List<Double> list = new ArrayList();
        for (Integer i : map.keySet()) {
            System.out.println(i + "->" + map.get(i));
            list.set(i, map.get(i));
        }

        for (Double d : list) {
            System.out.println(d);
        }
        /*
        double sum=0.;
        for(Double d:list){ sum += d; }
         */
        //和
        double sum = list.stream().reduce(0., (acc, _item) -> acc + _item);
        //最大値
        double max = list.stream().max(Comparator.naturalOrder()).get();
        //条件に合う要素の数
        int count = list.stream().filter(d -> (d > 0.5)).
                map(d -> 1).reduce(0, Integer::sum);
    }
```

NewMain.java

}

ArraysSample.java

```java
package example;

import java.util.Arrays;
import java.util.List;

/**
 *
 * @author tadaki
 */
public class ArraysSample {

    final private int n;
    private final Integer array[];

    public ArraysSample(int n) {
        this.n = n;
        array = new Integer[n];
        for (int i = 0; i < n; i++) {
            array[i] = i * i;
        }
    }

    /**
     * 配列からリストへ
     */
    public void convertList() {
        List<Integer> list = Arrays.asList(array);
        //要素を一つ変更
        int m = n / 2;
        list.set(m, 0);
        for (int i = 0; i < n; i++) {//配列側も変更になることの確認
            int x = array[i];
            int y = list.get(i);
            System.out.println(i + ": " + x + " " + y);
        }
        //エラーになる
        // list.add(1);
    }

    /**
     * 要素の探索
     * @param key
     */
    public void search(int key) {
        int k = Arrays.binarySearch(array, key,
                (a, b) -> {
```

```
                    return a - b;
                });
        if (k < 0) {
            System.out.println("Not found");
        } else {
            System.out.println(array[k] + " at " + k);
        }
    }

    /**
     * 配列コピー
     * @param m
     * @return
     */
    public Integer[] newArray(int m) {
        Integer array2[] = Arrays.copyOf(array, m * n);
        System.out.println("length of b is " + array2.length);
        for (int i = 0; i < m * n; i++) {
            array2[i] = (int) (100 * Math.random());
        }
        return array2;
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        int n = 10;
        int key = n / 2;
        ArraysSample sys = new ArraysSample(n);
        sys.convertList();
        sys.search(n / 2);
        sys.search((n - 2) * (n - 2));
        Integer array2[] = sys.newArray(2);
        //整列
        Arrays.sort(array2,
                (a, b) -> {
                    return a - b;
                });
    }

}
```

Main0.java

```java
package sortExample;

import java.util.Arrays;
import java.util.List;

/**
 *
 * @author tadaki
 */
public class Main0 {

    /**
     * @param args the command line arguments
     */
    static public void main(String args[]) {
        Student[] input = {
            new Student("Tom", 1, 88),
            new Student("Jane", 2, 80),
            new Student("Ray", 3, 70),
            new Student("Kim", 4, 75),
            new Student("Jeff", 5, 85),
            new Student("Ann", 6, 75),
            new Student("Beth", 7, 90)
        };
        MergeSort<Student> sort = new MergeSort<>(
                Arrays.asList(input),
                (a, b) -> {//comparator を定義
                    int ans = a.getRecord() - b.getRecord();
                    if (ans == 0) {
                        return a.id - b.id;
                    }
                    return ans;
                }
        );
        List<Student> output = sort.sort();
        if (sort.isSorted()) {
            output.stream().forEachOrdered(s -> System.out.println(s));
        }
    }

}
```

MergeSort.java

```java
package sortExample;

import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;

/**
 * MergeSort
 *
 * @author tadaki
 * @param <T>
 */
public class MergeSort<T> {

    final private List<T> list;
    final private Comparator<T> comparator;

    public MergeSort(List<T> list,Comparator<T> comparator) {
        this.list = list;
        this.comparator=comparator;
    }

    /**
     * 整列の実行
     *
     * @return 整列済みのリスト
     */
    public List<T> sort() {
        sortSub(0, list.size());
        return list;
    }

    /**
     * 再帰的整列
     *
     * @param left リストの整列対象のうち左端のインデクス
     * @param right リストの整列対象のうち右端のインデクス＋１
     */
    private void sortSub(int left, int right) {
        if (right <= left) {
            throw new IllegalArgumentException("illegal range");
        }
        if (right == left + 1) {
            return;
        }
        int middle = (right + left) / 2;
```

```
        //再帰呼び出し
        sortSub(left, middle);
        sortSub(middle, right);
        //リストの結合
        List<T> tmpList = mergeList(left, middle, right);
        for (int p = 0; p < tmpList.size(); p++) {
            list.set(left + p, tmpList.get(p));
        }
    }

    /**
     * リストの結合
     *
     * @param left 左端
     * @param middle 右側要素の先頭
     * @param right 右側要素の終端＋１
     * @return
     */
    private List<T> mergeList(int left, int middle, int right) {
        List<T> tmp = new ArrayList<>();
        int leftIndex = left;
        int rightIndex = middle;
        while (leftIndex < middle || rightIndex < right) {
            if (leftIndex >= middle) {//左側終了
                for (int k = rightIndex; k < right; k++) {
                    tmp.add(list.get(k));
                }
                return tmp;
            }
            if (rightIndex >= right) {//右側終了
                for (int k = leftIndex; k < middle; k++) {
                    tmp.add(list.get(k));
                }
                return tmp;
            }
            if (less(leftIndex, rightIndex)) {
                tmp.add(list.get(leftIndex));
                leftIndex++;
            } else {
                tmp.add(list.get(rightIndex));
                rightIndex++;
            }
        }
        return tmp;
    }
```

```
    /**
     * リストのi番の要素がj番の要素より小さい場合に真
     * @param i
     * @param j
     * @return
     */
    private boolean less(int i, int j) {
        return (comparator.compare(list.get(i),list.get(j))<0);
    }

    public boolean isSorted(){
        boolean b = true;
        for(int i=0;i<list.size()-1;i++){
            if(!less(i,i+1))return false;
        }
        return b;
    }
}
```

Student.java

```java
package sortExample;

/**
 *
 * @author tadaki
 */
public class Student {

    final String name;//名前
    final int id;//番号
    private int record;

    public Student(String name, int id, int record) {
        this.name = name;
        this.id = id;
        this.record = record;
    }

    public int getRecord() {
        return record;
    }

    public void setRecord(int record) {
        this.record = record;
    }

    public String toString() {
        return String.valueOf(id) + ":"+name +":"+ String.valueOf(record);
    }


}
```