CollectionとLambda式

オブジェクト指向プログラミング特論

2020年度

只木進一：工学系研究科

今日のサンプルプログラム

- https://github.com/oop-mc-saga/Lambda

同じクラスのインスタンスの集まり

- 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

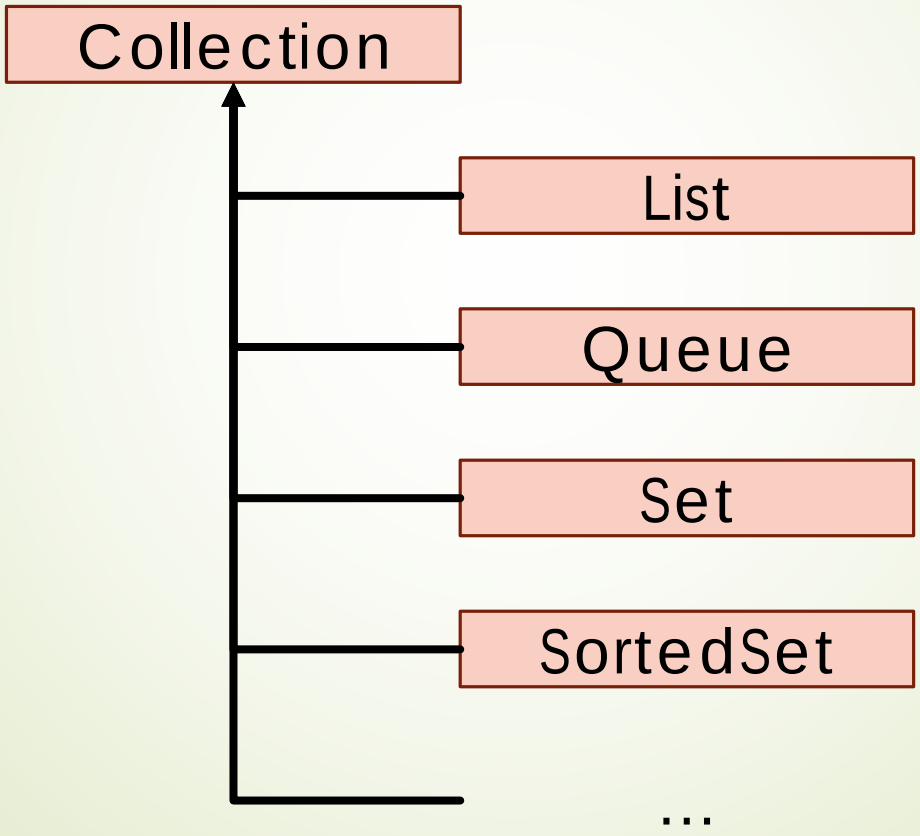
- 型パラメタを明示することで、コンパイラが型の整合性を確認できるようにすること。
- Collectionなどでは、保存する型を明示することで、出し入れを安全に行う。

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

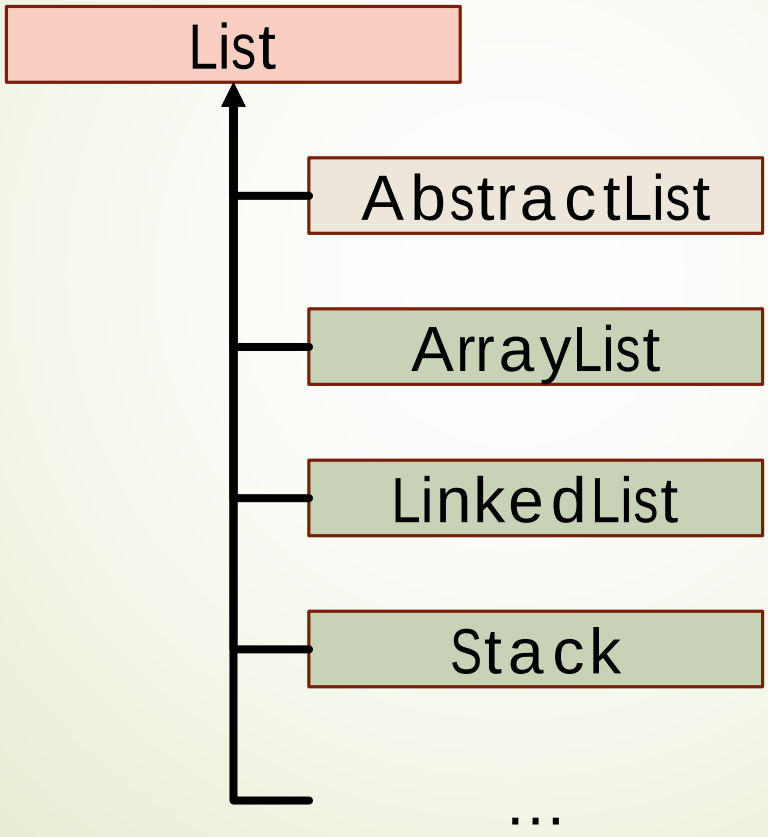


全てInterface

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

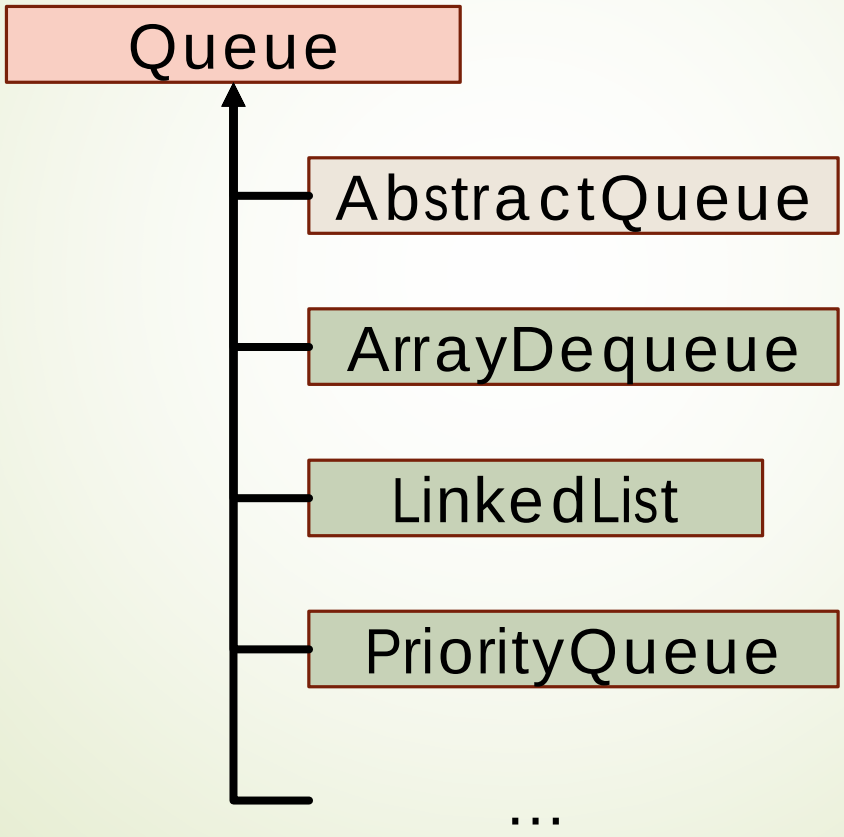
要素の追加・削除が速い

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

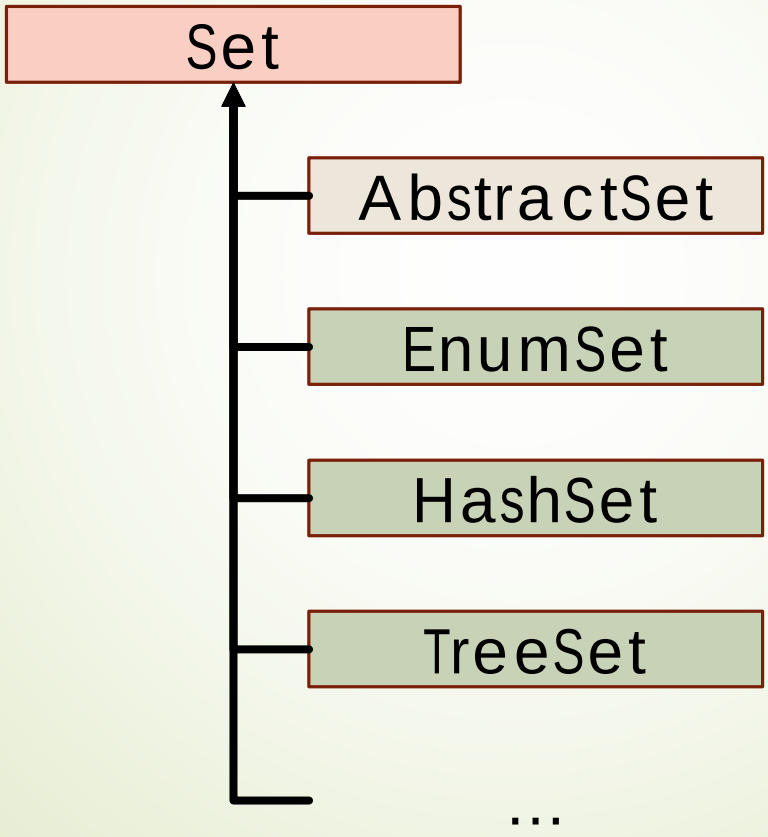
java.util.Queueの実装



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
- 変更不可のビュー作成

Lambda/collectionsSample/CollectionsSample.java

```
//要素を探す
int k = Collections.binarySearch(studentList, input[3]);
System.out.println(input[3] + " is found at " + k);

//最大の要素を見つける
Student best = Collections.max(studentList);
System.out.println(best + " marks the best");

//整列
System.out.println("sorted list");
Collections.sort(studentList);
studentList.forEach(
        s -> System.out.println(s)
);
System.out.println("----------------");
//配列へ出力
Student[] studentArray = new Student[studentList.size()];
studentArray = studentList.toArray(studentArray);
for(Student s:studentArray){
    System.out.println(s);
}
System.out.println("----------------");

//変更不可のビューを作る
List<Student> view = Collections.unmodifiableList(studentList);
try {
    Collections.reverse(view);
} catch (UnsupportedOperationException e) {
    System.err.println("このリストは変更できない");
}
```

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

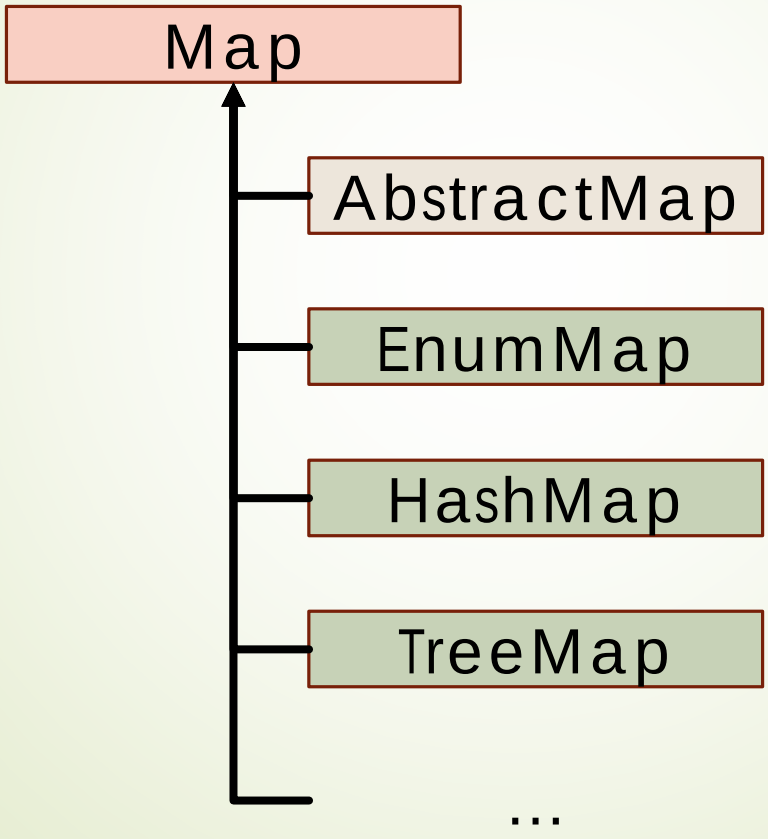
Lambda/arraysSample/ArraysSample.java

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
public class MapSample {

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        String codes[] = {"CTS", "FUK", "HSG", "HND", "KIX"};
        String names[] = {"新千歳空港", "福岡空港", "有明佐賀空港",
            "羽田空港", "関西空港"};

        Map<String, String> airports = new HashMap<>();
        for (int i = 0; i < codes.length; i++) {
            airports.put(codes[i], names[i]);
        }

        for (String code : airports.keySet()) {
            System.out.println(code + "->" + airports.get(code));
        }
    }

}
```

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
  - 例

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

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

```
10  public class SimplestSample {
11
12      /**
13       * @param args the command line arguments
14       */
15      public static void main(String[] args) {
16          int n=100;
17          List<Double> list = new ArrayList<>();
18          for(int i=0;i<n;i++){
19              list.add(Math.random());
20          }
21          //各要素を出力
22          list.stream().forEach(d ->  System.out.println(d));
23      }
24
25  }
```

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

Lambda式の基本

- (引数並び)->{処理}
- 引数の型は省略可能
- 引数が一つの場合には、括弧”()”も省略可能
- 処理が一行の場合には、括弧”{}”も省略可能

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の簡単な例

```
public static List<Integer> listOperation(List<Integer> inputList,
        IntFunction<Integer> func){
    List<Integer> outputList = new ArrayList<>();
    for(int i=0;i<inputList.size();i++){
        int input = inputList.get(i);
        int output = func.apply(input);
        outputList.add(output);
    }
    return outputList;
  }
```

Lambda/lambdaSample/SimpleSample.java

Lambdaの利用

- Collectionの要素に対する処理

```
studentList.forEach(s -> System.out.println(s));
```

- Comparator
  - 要素の比較方法

```
//名前順に整列
    Comparator<Student> comparator =
        (a,b)->a.name.compareTo(b.name);
    studentList.sort(comparator);
```

Lambda/lambdaSample/SortSample.java