Javaプログラミング実習

27. Collection / Map

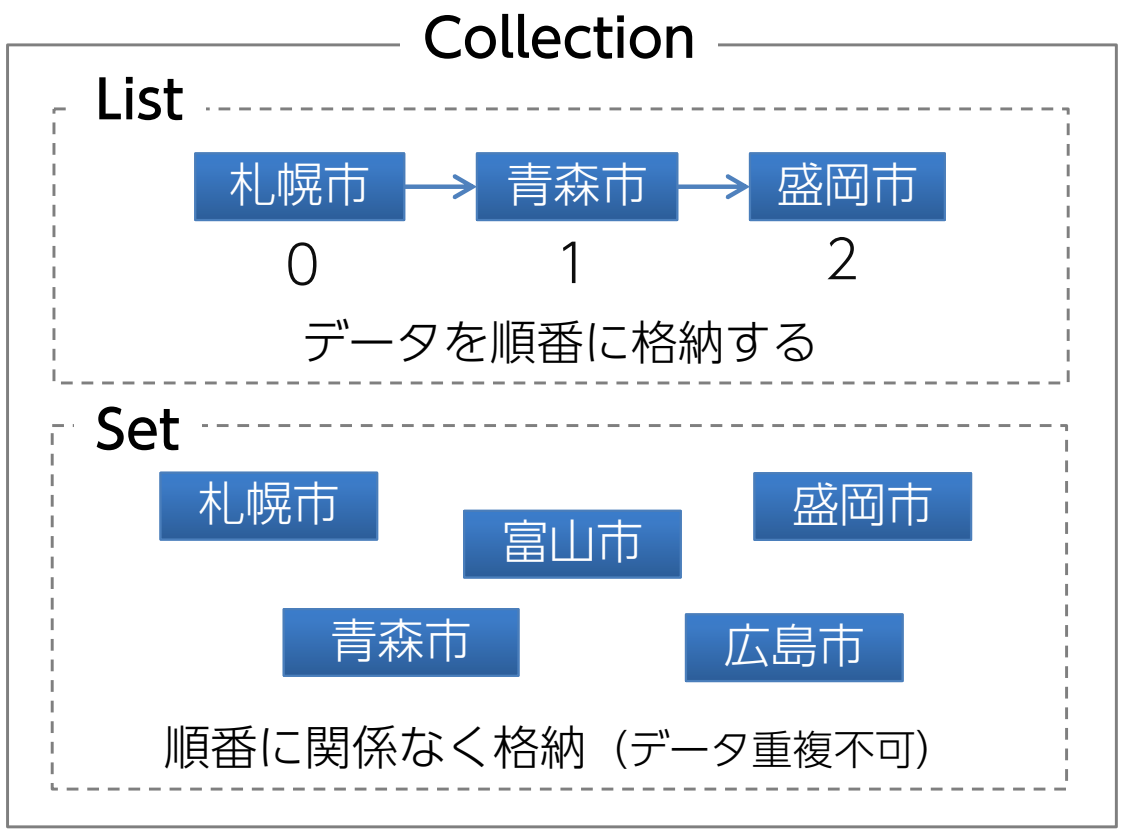
株式会社ジードライブ

今回学ぶこと

- コレクションの概要
- Listインターフェース
- Setインターフェース
- Mapインターフェース

Collection / Map

- java.utilパッケージに属し、高度な配列機能を提供するインターフェース
  - Collectionを継承するものとして、List, Setが存在する



Map

Key	Value
北海道	札幌市
青森県	青森市
岩手県	盛岡市

キーと値(バリュー)の組でデータを格納する

List

- 高機能な配列機能を実現するインターフェース
  - java.utilパッケージに含まれる
  - Listインターフェースを実装しているクラスとして、**ArrayList** や LinkedList などがある
- Listの特徴
  - 格納できる要素の数が可変

    ⇒ 通常の配列は格納できる要素の数が固定
  - for文や拡張for文を使って要素処理が可能

Listの書式

書式

```
List<保持するデータの型> 変数 = new ArrayList<>();
List<保持するデータの型> 変数 = new LinkedList<>();
```

- < >はダイヤモンド演算子と呼ばれ、保持するデータの型を指定する
  - プリミティブ型は指定できない

記述例

```
List<Integer> scoreList = new ArrayList<>();
```

ArrayListでも問題ないが、
Listの方が柔軟性がある

LinkedListも可

Listのメソッド

- List<E>インターフェースの主なメソッド
  - E は要素のデータ型を示す

メソッド	説明
`void add(E e)`	指定された要素をリストの最後に追加する。
`void add(int index, E e)`	指定された要素を指定された位置に挿入する。 index は 0 が先頭。
`E get(int index)`	指定された位置の要素を返す。
`E set(int index, E e)`	指定された位置にある要素を指定された要素に置き換える。
`E remove(int index)`	指定された位置の要素を削除する。 戻り値は指定された位置に以前あった要素。
`int size()`	リスト内の要素の数を返す。

Listの使用例(1)

例：アイテムの追加と拡張for文を利用したListの処理

```
List<String> fruits = new ArrayList<String>();
fruits.add("りんご");
fruits.add("バナナ");
fruits.add("ぶどう");
fruits.add("いちご");
fruits.add("りんご");

for (String item : fruits) {
  System.out.println(item);
}
```

```
りんご
バナナ
ぶどう
いちご
りんご
```

Listの使用例(2)

例：add()メソッドとset()メソッド

```
List<String> fruits = new ArrayList<>();
fruits.add("りんご"); // 0番に「りんご」を追加
fruits.add("バナナ"); // 1番に「バナナ」を追加
fruits.add("ぶどう"); // 2番に「ぶどう」を追加
fruits.add(1, "みかん"); // 1番に「みかん」を追加⇒以下の番号ずれる
fruits.set(0, "アップル"); // りんごを「アップル」に変更
System.out.println(fruits.get(2)); // 「バナナ」と表示

// for文による出力
for (int i = 0; i < fruits.size(); i++) {
  System.out.println(i + ":" + fruits.get(i));
}
```

```
バナナ
0:アップル
1:みかん
2:バナナ
3:ぶどう
```

Listの使用例(3)

- ArraysクラスのasList()メソッドを使うと初期値を持つリストの作成が容易になる

例：固定サイズのリストの作成

```
List<String> fruits =
   Arrays.asList("りんご", "バナナ", "ぶどう");
```

例：可変サイズのリストの作成

```
List<String> fruits = new ArrayList<>(
    Arrays.asList("りんご", "バナナ", "ぶどう"));
```

Listの使用例(4)

例：氏名、年齢の情報を保持するMemberクラスのListを作成する

Memberクラス

```
public class Member {
    private String name;
    private int age;
    public Member(String name, int age) {
        this.name = name;
        this.age = age;
    }
    public void showInfo() {
        System.out.println(name + "(" + age + ")");
    }
}
```

Listの使用例(4)

前頁の続き

Memberを利用するクラス

```
// MemberのListを作成
List<Member> memberList = new ArrayList<>();
memberList.add(new Member("山田太郎", 28));
memberList.add(new Member("木村次郎", 33));
memberList.add(new Member("佐藤花子", 27));

// 拡張for文による利用
for(Member member : memberList) {
    member.showInfo();
}
```

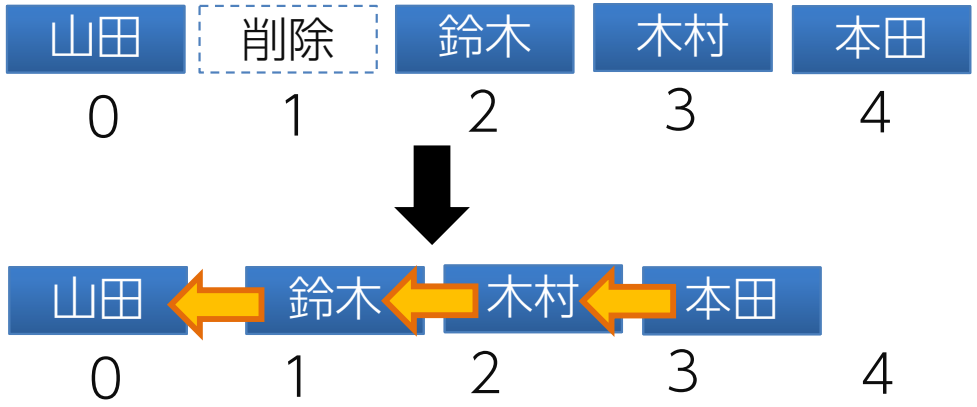
```
山田太郎(28)
木村次郎(33)
佐藤花子(27)
```

ArrayListのイメージ

- ArrayListは要素が隙間なく並ぶイメージで捉えることができる

番号の上に要素が整列しているイメージ

データ削除時の挙動イメージ



隙間を詰めるために、全ての要素が一つずつ動く

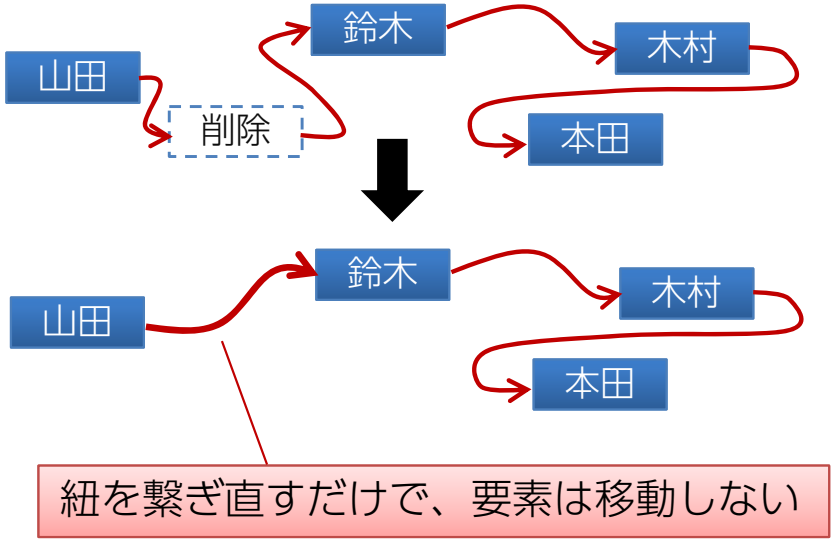
ArrayListは、番号がついているため要素の取得が得意だが、削除や追加は処理が遅くなる可能性がある

LinkedListのイメージ

- LinkedListはデータ同士が順番に紐づいているイメージで捉えることができる

順番に紐で繋がっている
イメージ

データ削除時の挙動イメージ



LinkedListは、要素の追加や削除への対応が得意だが、番号がついていないため要素の取得に時間を要する可能性がある

練習

- 練習27-1
- 練習27-2
- 練習27-3

Set

- 要素の集合を表現するインターフェース
  - java.utilパッケージに含まれる
  - Setインターフェースを実装しているクラスとして、**HashSet** や **LinkedHashSet**、**TreeSet** などがある
- Setの特徴
  - for文や拡張for文を使って要素処理が可能
  - 同じ値の要素は複数追加できない
  - 要素を追加した順番は保持されない(HashSetの場合)

Listとの違い

Setのメソッド

- Set<E>インターフェースの主なメソッド

メソッド	説明
`boolean add(E e)`	指定された要素を追加する。
`boolean remove(Object o)`	指定された要素を削除する。
`boolean contains(Object o)`	指定された要素が含まれている場合`true`を返す。
`int size()`	Set内の要素の数を返す。

HashSetの使用例

- 順番を持たない要素の集合

```
Set<String> fruits = new HashSet<>();
fruits.add("りんご");
fruits.add("バナナ");
fruits.add("ぶどう");
fruits.add("いちご");
fruits.add("りんご"); // 2回目のりんご

for (String item : fruits) {
    System.out.println(item);
}
```

- 追加の順序と取り出しの順序が異なっている
- 2回目のりんごは表示されない

```
りんご
ぶどう
いちご
バナナ
```

LinkedHashSetの使用例

- 追加順を保持する要素の集合

```
Set<String> fruits = new LinkedHashSet<>();
fruits.add("りんご");
fruits.add("バナナ");
fruits.add("ぶどう");
fruits.add("いちご");
fruits.add("りんご"); // 2回目のりんご

for (String item : fruits) {
    System.out.println(item);
}
```

• 追加の順序が保持されている

```
りんご
バナナ
ぶどう
いちご
```

TreeSetの使用例

- 自然順序に並ぶ要素の集合
  - 数値の小さい順、アルファベット順、五十音順…

```
Set<String> fruits = new TreeSet<>();
fruits.add("りんご");
fruits.add("バナナ");
fruits.add("ぶどう");
fruits.add("いちご");
fruits.add("りんご"); // 2回目のりんご

for (String item : fruits) {
    System.out.println(item);
}
```

- 平仮名、カタカナの順番
- 五十音順に並んでいる

```
いちご
ぶどう
りんご
バナナ
```

練習

- 練習27-4

Map

- キー(Key)と値(Value)の組(マッピング)を表すインターフェース
  - java.utilパッケージに含まれる
  - Mapインターフェースを実装しているクラスとして、**HashMap**や**LinkedHashMap**などがある
- Mapの特徴
  - 同一のキーは複数登録できない
  - 各キーには1つの値しか割り当てられない
  - HashMapの場合、登録されるキーの順序は保障されない

Key	Value
北海道	札幌市
青森県	青森市
岩手県	盛岡市

Mapのメソッド

- Map<K,V>インターフェースの主なメソッド
  - K：Key（キー）　V：Value（値）

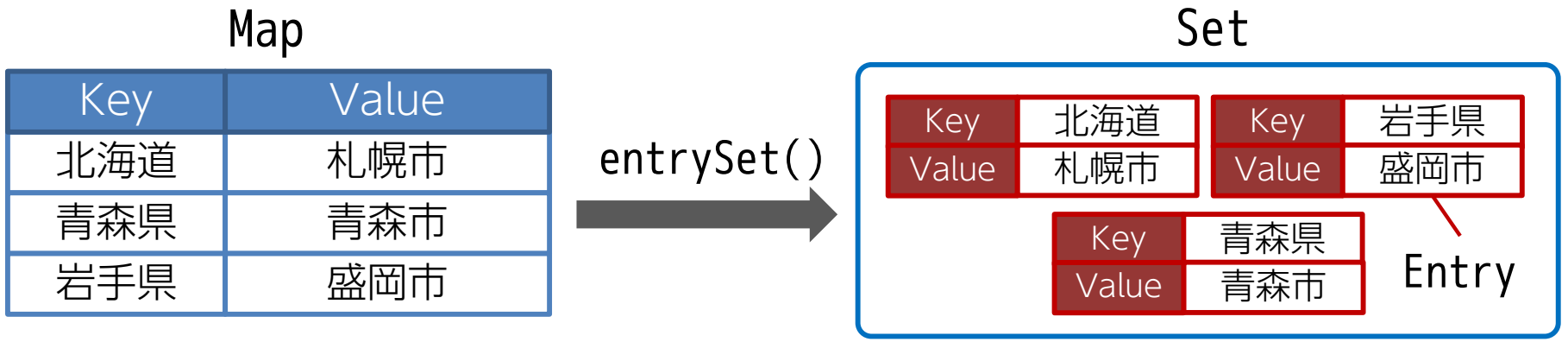
メソッド	説明
V put(K key, V value)	指定されたキーと値の組を追加する。
V get(Object key)	指定されたキーに対応する値を返す。
V remove(Object key)	指定されたキーの組を削除する。
boolean containsKey(Object key)	指定されたキーを含む組が存在する場合 true を返す。
boolean containsValue(Object value)	指定された値を含む組が存在する場合 true を返す。
Set<Entry<K,V>> entrySet()	このマップに含まれるデータを Entry の Set として返す。

Entry

- キーと値の一組を表すインターフェース

Entry<K,V>の主なメソッド

メソッド	説明
K getKey()	エントリーに対応するキーを取得する。
V getValue()	エントリーに対応する値を取得する。
V setValue(V value)	エントリーに対応する値を、指定した値に置き換える。戻り値は置き換える前の値。



Mapの使用例(1)

例：文字列(String)と数値(Integer)の組を格納するMap

- 同じキーで登録した場合、値が上書きされる

```
Map<String, Integer> fruits = new HashMap<>();
fruits.put("りんご", 100); // ("りんご", 100)を追加
fruits.put("バナナ", 200);
fruits.put("ぶどう", 300);
fruits.put("いちご", 400);
fruits.put("りんご", 150); // ("りんご", 100)を上書き

fruits.remove("いちご"); // ("いちご", 400)を削除

// "りんご"というキーがある場合は、対応する値を出力
if (fruits.containsKey("りんご")) {
  System.out.println(fruits.get("りんご"));
}
```

```
150
```

Mapの使用例(1)

例：前頁の続き

```
// EntryのSetを受け取る場合の例
Set<Entry<String, Integer>> items = fruits.entrySet();

// 拡張for文による出力の例
for (Entry<String, Integer> item : items) {
  System.out.print(item.getKey() + ":");
  System.out.println(item.getValue() + "円");
}
```

```
りんご:150円
ぶどう:300円
バナナ:200円
```

Mapの使用例(２)

例：スケジュールのMapを作成する

あらかじめ、1つの予定を保持するクラスを作成しておく

```
public class Plan {
    private String subject; // 予定の件名
    private String start;   // 開始時刻
    private String end;     // 終了時刻

    public Plan(String subject, String start, String end) {
        this.subject = subject;
        this.start = start;
        this.end = end;
    }

    public void show() {
        System.out.println(subject + " (" + start + "〜" + end + ")");
    }
}
```

日付や時間を扱うためのクラスについては次章で学習します

Mapの使用例(２)

例：スケジュールのMapを作成する

```
Map<String, Plan> planMap = new HashMap<>();

// スケジュールの追加
planMap.put("2024-09-03", new Plan("試験", "10:00", "15:00"));
planMap.put("2024-09-08", new Plan("歯科", "18:30", "19:30"));
planMap.put("2024-09-14", new Plan("面接", "14:00", "15:00"));

// スケジュールの出力
for(Entry<String, Plan> entry : planMap.entrySet()) {
    System.out.print(entry.getKey() + "の予定: ");
    entry.getValue().show();
}
```

```
2024-09-08の予定: 歯科（18:30〜19:30）
2024-09-03の予定: 試験（10:00〜15:00）
2024-09-14の予定: 面接（14:00〜15:00）
```

練習

- 練習27-5
- 練習27-6