第26章　コレクションフレームワーク

目次

コレクションフレームワークとは

Javaでは提供されている、複数のデータを扱うためのクラスのこと。
配列よりも柔軟に複数のデータを扱える。

配列は作成された時点で要素数が固定され、
状況に応じて柔軟に要素数を変更したい処理には適さない。
コレクションフレームワークを利用すれば、
要素数の変更にも柔軟に対応できる。

代表的なコレクションフレームワーク

インターフェイス	説明
java.util.List	順序通りに並んだ要素の集まりを管理するデータ構造
java.util.Map	キーと値を関連付けたものを1つの要素として管理するデータ構造

実装元のインターフェイス	実装クラス	説明
java.util.List	java.util.ArrayList	配列と同様に0開始の連番で要素を管理する。
java.util.Map	java.util.HashMap	キーと値のペアを1つの要素として管理する。

ArrayListとは

Listインターフェイスの実装クラスであり、配列と同じように
0開始の連番で各要素を管理できるコレクションフレームワーク。

要素数を状況に応じて柔軟に増減できる。

リストの生成

ArrayListを利用するには、オブジェクトを生成する。

```
List<型名> 変数名 = new ArrayList<>();
```

new演算子を用い、インターフェイスであるList型の変数に代入する。
ArrayListを使用する場合は、一般的にList型の変数に代入する。

（List型の変数に代入する理由）

1. Listインターフェイスに定義してあるメソッドだけで十分な場合が多い。
2. ArrayList以外のListインターフェイスの
   実装クラスを代入することもできる。

リストの生成

ArrayListも配列と同様に、扱うデータの型を指定する。その際に使用するのが、ジェネリクス(< >)。

ジェネリクスは扱う型を指定するための機能。
「<>」内に型を記述することで、記述された型のデータをArrayList内で扱える。

```
List<型名> 変数名 = new ArrayList<>();
```

ジェネリクス

リストの生成

ジェネリクスには参照型しか記述できない。
整数を扱うArrayListを作成する場合、
「<int>」と記述してしまうとコンパイルエラーになる。

ArrayListを生成する際はList型の変数に代入する。
ジェネリクス(< >)は型を指定する。
ジェネリクスには参照型しか指定できない。

要素の追加

要素の追加と値の代入の処理は、
ArrayListが提供しているadd()メソッドを使用して実行する。

メソッドを1回実行すると、リストの末尾に
新しい要素が1個追加される。
追加された要素には先頭から順に
0開始の連番が振られて、各要素が識別される。

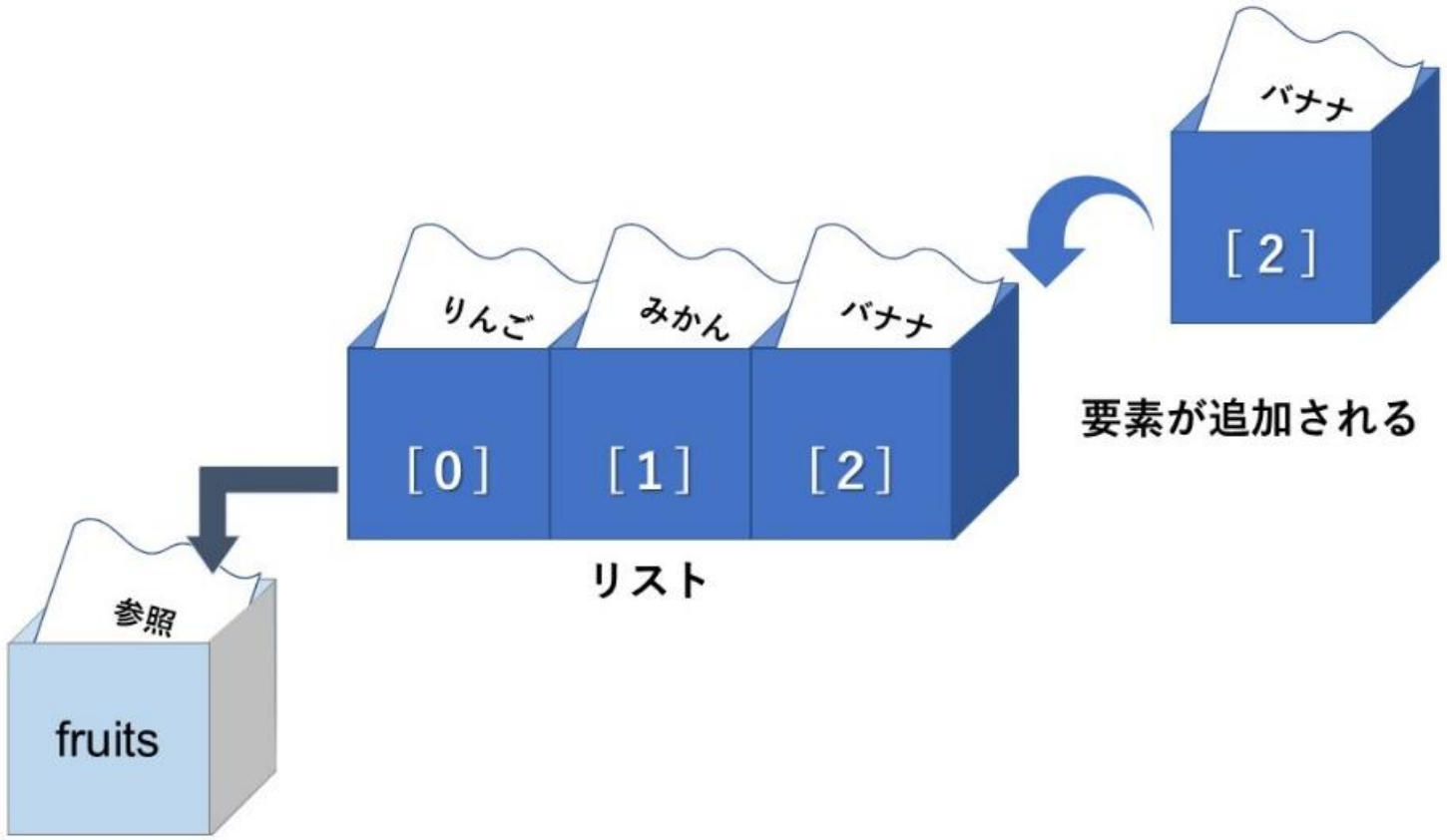
要素の追加

（例）String型の文字列を格納するリストを生成して、
　　要素の追加と値の代入を行う場合

```
List<String> fruits = new ArrayList<>();
fruits.add(″りんご″);
fruits.add(″みかん″);
fruits.add(″バナナ″);
```

add( )メソッドは、変数名の後ろに「.（ピリオド）」で続けて記述する。
add( )メソッドの「( )」内にはジェネリクスで指定された型に
該当する値を記述する。

要素の追加

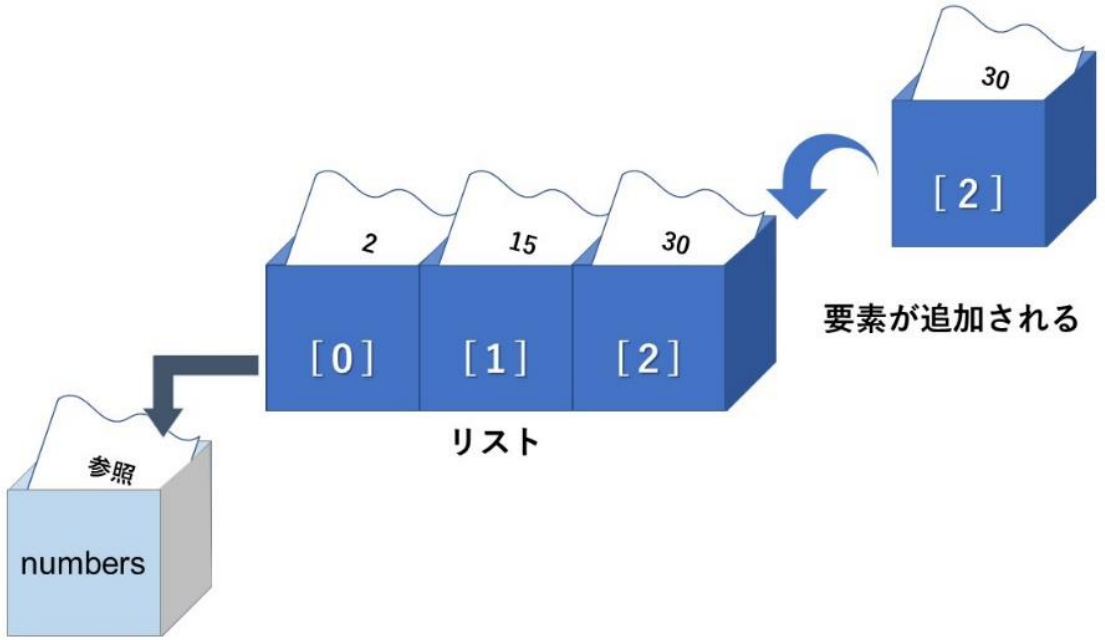


要素の追加

（例）整数を扱うリストを生成する場合

```
List<Integer> numbers = new ArrayList<>();
numbers.add(2);
numbers.add(15);
numbers.add(30);
```

要素の追加



リストに要素を追加し、値を代入するには、
「add()メソッド」を使います。

要素の取得

リストから要素とその値を取得するには、get( )メソッドを使う。
このメソッドは、指定された添字の要素に代入された値を
取得する機能を持つ。
（例）変数listが参照するリストから添字1番の要素の値を
　　取得して、その値を標準出力したい場合

```
System.out.println(list.get(1));
```

ArrayListでの値の取り出しは「get()メソッド」を使います。

【Sample2601 ArrayListを使う】を作成しましょう

Sample2601のポイント

ListやArrayListを使用する際には、
クラスブロックの手前にimport文を記述する。

```
import java.util.ArrayList;
import java.util.List;
```

Sample2601のポイント

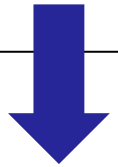
main()メソッド内では、リストの生成処理で
ジェネリクスにInteger型を指定している。
そのため、リストの各要素には整数値を代入する。
「list.get(0)」により、添字0番の要素の値を取得し、
標準出力している。

```
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(3);
list.add(5);
System.out.println("数値は" + list.get(0) + "です。");
```

Sample2601のポイント

存在しない要素の添字を指定した場合、
プログラム実行時にエラーが発生する。
配列と同様に、いくつの要素が存在するかを
意識して実装する必要がある。

```
System.out.println("数値は" + list.get(3) + "です。");
```

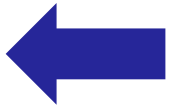


存在しないためエラー

要素の削除

要素を削除する際はremove( )メソッドを使う。
メソッドの「( )」内に削除したい要素の添字を指定することで、
リストから該当する要素を削除する。

```
list.remove(1);
```



添字の1番目の要素を削除する

要素の削除

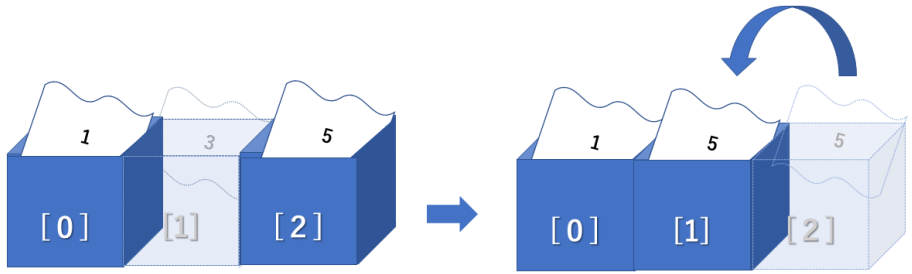
要素を削除した場合、後続する要素は
1つ手前に詰めて配置し直される。

```
List<Integer> list = new ArrayList<>();
list.add(1);
list.add(3);
list.add(5);
list.remove(1);
System.out.println("番号は" + list.get(1) + "です。");
```

remove()メソッドの「()」内に「1」を指定して、
添字1番の要素と値「3」を削除している。

要素の削除

「5」という値はもともと添字2番の要素に代入されていた。
添字1番の要素が削除されたことで、
値「5」が代入された要素の添字が1番となった。



list.remove(1);
[1]番目の要素が削除されました

[2]番目だった要素が
[1]番目に繰り下がります

ArrayListでの要素の削除は「remove()メソッド」を使います。

ArrayListのメソッド

ArrayListにはadd()メソッド、get()メソッド、remove()メソッド以外のメソッドも用意されている。

メソッド名	働き	記述例（aは添字、bは値）
add()	リストの最後尾に要素の追加する	list.add(b)
	リスト中の指定した位置に要素を挿入する	list.add(a,b)
set()	指定した位置の要素の値を別の値に置き換える	list.set(a,b)
get()	指定した要素を取得する	list.get(a)
remove()	指定した要素を削除する	list.remove(a)
clear()	すべての要素を削除する	list.clear()
size()	リスト内の要素数を取得する	list.size()
isEmpty()	リスト内の要素が空（0個）であるか判定する	list.isEmpty()

【Sample2602 拡張for文でArrayListを扱う】を作成しましょう

Sample2602のポイント

拡張for文により、ArrayListの各要素の値を出力している。

```
for (int value : list) {
        System.out.println(value);
}
```

配列とリストの違い

● 配列
要素の数が固定的に決まっている場合
アクセス速度や処理の軽さが極端に求められる場合
● ArrayList
要素の数が処理によって変わる場合

→基本的にはArrayListを使い、
　要素数が将来にわたって完全に固定である場合のみ
　配列を使うとよい。

HashMapとは

「キー」と「値」のペアを1つの要素として管理しており、
キーを使用して特定の要素に参照できる仕組み。
キーの内容に連想する値を特定することから
「連想配列」とも呼ばれる。

配列やArrayListのような整数値ではなく、
他の形式の値で管理できた方が便利な場合に、
HashMapが利用される。

マップの生成

リストを生成する処理と同様、ジェネリクスによる型指定を行う。
1つの「<>」内に2つの型を指定する。
1つ目の型:「キーの型」
2つ目の型:「値の型」

```
Map<キーの型名, 値の型名> 変数名 = new HashMap<>();
```

変数にはマップの参照が代入され、
その変数を経由して間接的にマップを操作できるようになる。

マップの生成

（例）文字列型のキーで整数値型の値を管理したい場合

```
Map<String, Integer> map = new HashMap<>();
```

HashMapを生成する際は、Map型の変数に代入します。
ジェネリクスには「キーの型」と「値の型」の2つを指定します。

要素の追加

要素の追加はHashMapが提供しているput()メソッドで行う。
put()メソッドを1回実行することで、1つの要素が追加される。
第一引数にはキーを、第二引数には値を記述する。

```
Map<String, Integer> students = new HashMap<>();
students.put("山田", 21);
students.put("田中", 24);
students.put("高橋", 27);
```

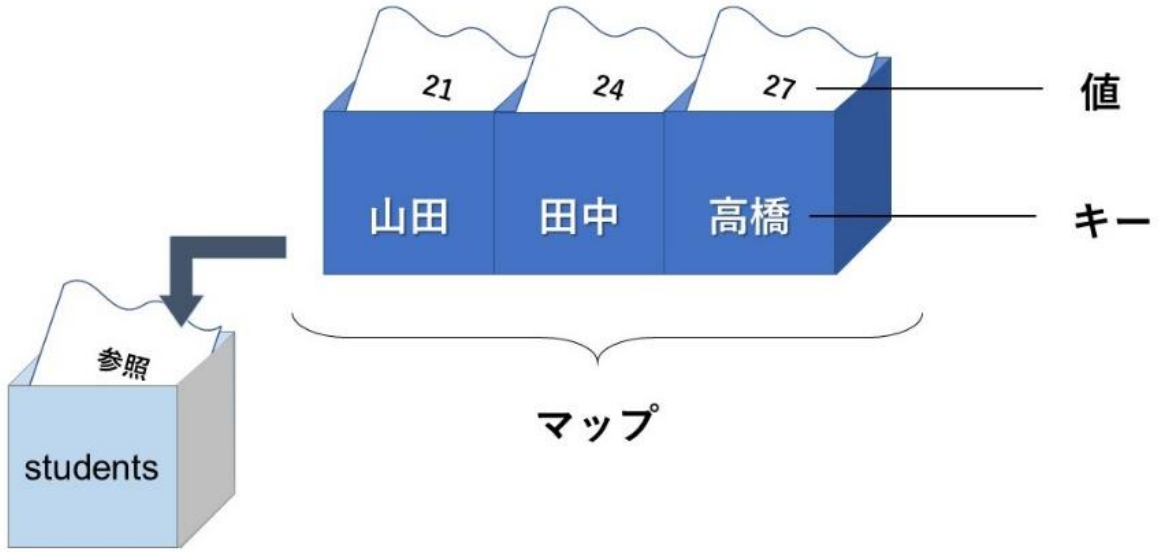
要素の追加

追加する要素のキーは、先に追加した要素のキーと
異なるキーを指定する。
キーが重複すると、後から追加したキーに紐づく値で
上書きされてしまう。各要素の値は重複していても問題なし。

```
students.put(″山田″, 21);
students.put(“山田″, 24);
```

山田の値が
24に上書きされる

要素の追加



マップに要素を追加するには「put()メソッド」を使います。

要素の取得

マップから要素とその値を取得するには、
get()メソッドを使用する。
get()メソッドには引数を指定する必要があり、
引数にはキーの内容を記述する。
（例）キー「山田」に該当する要素を取得して
　　その値を出力したい場合

```
System.out.println(map.get("山田"));
```

HashMapでの値の取り出しは「get()メソッド」を使います。

【Sample2603 HashMapを使う】を作成しましょう

Sample2603のポイント

MapやHashMapを使用する際には、
クラスブロックの手前にimport文を記述する。

```
import java.util.HashMap;
import java.util.Map;
```

Sample2603のポイント

キーと値がどちらも文字列型（String）のマップを生成している。
そのため、put()メソッドの「()」内には文字列を2つ記述する。

```
map.put(″犬″, ″dog″);
map.put(″猫″, ″cat″);
map.put(″猿″, ″monkey″);
```

Sample2603のポイント

マップに要素を追加した後、キー「猿」に該当する
要素の値を取得し、標準出力している。

```
System.out.println(″キー「猿」に該当する要素の値は″
    + map.get(″猿″) + ″です。″);
```

HashMapのメソッド

メソッド名	働き	記述例（aはキー、bは値）
put()	要素を追加する	map.put(a,b)
get()	指定したキーに該当する要素を取得する	map.get(a)
remove()	指定したキーに該当する要素を削除する	map.remove(a)
clear()	すべての要素を削除する	map.clear()
size()	マップ内の要素数を取得する	map.size()
containsKey()	指定したキーに該当する要素が存在するか判定する	map.containsKey(a)
containsValue()	指定した値に該当する要素が存在するか判定する	map.containsValue(b)
isEmpty()	マップ内の要素が空（0個）であるか判定する	map.isEmpty()

章のまとめ

- 代表的なコレクションフレームワークとして
  ArrayListとHashMapがあります。
- ArrayListは、要素数を柔軟に変更できます。
- ArrayListに要素を追加するには、
  「add()メソッド」を使います。
- ArrayListでの要素の取得は「get()メソッド」を使います。
- HashMapは、キーと値をペアとした要素を管理します。
- HashMapに要素を追加するには、
  「put()メソッド」を使います。
- HashMapでの要素の取得は「get()メソッド」を使います。