コレクションフレームワーク（List、Map、Set）

コレクション・・・複数の要素の集まり。
コレクションフレームワーク・・・コレクション（要素の集まり）を扱う機能。

※コレクションフレームワークでは、複数の要素（データ）を扱うための便利機能を提供しています。

具体的には以下 3 つの種類のインターフェースから成ります。

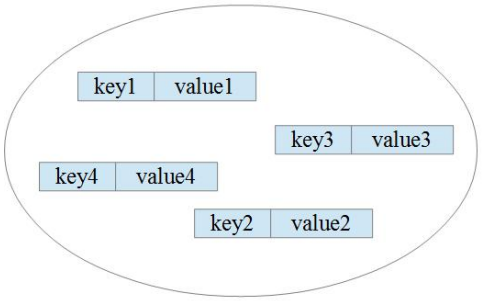
- List
- Map
- Set

それぞれの特徴

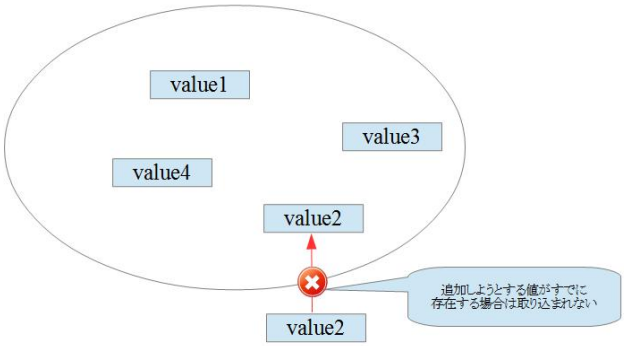
List：複数の要素の順番を保持する。配列の代わりとして利用することができます。

インデックス	0	1	2	3	4	5
要素	V1	V2	V3	V4	V5	V6

Map：キーと要素とのマッピングを表すデータ構造。
　　List のようなインデックスの代わりに、キーで要素を識別します。



Set：重複要素を持たない要素の集合。



コレクション・フレームワークの List、Map、Set は、それぞれ意味や機能が異なります。

List

List： 複数の要素の順番を保持する。
インデックスを利用して要素にアクセスするため、配列の代わりとして利用することができます。

List のイメージ図

配列の場合、宣言時に必要な個数を指定する必要がありましたが、List では後から要素数を変更することが出来るのが特徴です。

List を使う場合には、以下のいずれかを使ってインスタンス化します。

クラス	任意の要素へのアクセス	要素の途中追加、削除
ArrayList	高速	低速
LinkedList	低速	高速

List を使ったインスタンス化方法：

```
List<[要素の型]>  変数名  = new ArrayList<[要素の型]>();
List<[要素の型]>  変数名  = new LinkedList<[要素の型]>();
```

<[要素の型]>の書き方

この書き方を「ジェネリクス」といいます。
JDK1.5 からサポートされるようになった機能です。
Java は変数の型を厳格にチェックする言語ですが、JDK1.4 までは、ジェネリクスと呼ばれる仕組みが無く、いろいろな型のデータを詰め込むことが可能でした。

型をあらかじめ決めることができるのでジェネリクスで記述した場合、拡張 For 文を利用することもできるようになり値の取出しが楽になりました。（拡張 For 文は、以降のサンプルソース内で記述しています）

ジェネリクスは、現在、List、Map、Set いずれの場合にも記述します。

List サンプルソース

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

public class ListSample {

  public static void main(String[] args) {
    List<String> list = new ArrayList<String>();

    //値の記憶は add メソッドを利用
    list.add("1");
    list.add("2");
    list.add("3");
    list.add("4");
    list.add("5");

    //値の取得
    for (int i = 0; i < list.size() -1; i++) {
      //値の取得は get メソッドを利用
      System.out.println(list.get(i));
    }

    //拡張 for 文を利用するともっと簡単
    for (String s : list) {
      System.out.println(s);
    }
  }
}
```

List を使った場合には、
add()で要素を記憶できます。

List を使った場合には、
get()で要素を取得できます。

List を使った場合には、拡張 For 文でも要素を取得できます。
list 内の要素を都度 s に記憶して、これを画面表示しています。

List サンプルソース実行結果

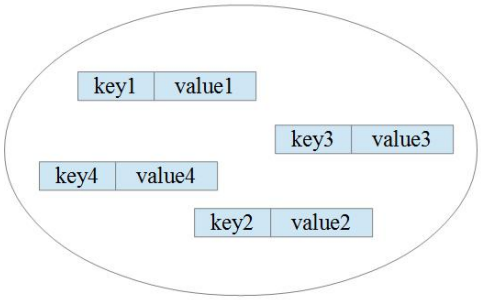
```
1
2
3
4
1
2
3
4
5
```

Map

Map：キーと要素とのマッピングを表すデータ構造。

List のようなインデックスの代わりに、キーで要素を識別します。

Map のイメージ図



Map を使う場合には、以下のいずれかを使ってインスタンス化します。

クラス	null の可否	順番
HashMap	可能	登録順は意識されずに記憶される
TreeMap	可能	登録したキーが昇順で記憶される
LinkedHashMap	可能	登録した順番で記憶される

Map を使ったインスタンス化方法：

```
Map<key, [要素の型]>  変数名  = new HashMap<key, [要素の型]>();
Map<key, [要素の型]>  変数名  = new TreeMap<key, [要素の型]>();
Map<key, [要素の型]>  変数名  = new LinkedHashMap<key, [要素の型]>();
```

Map サンプルソース

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

public class MapSample {

  public static void main(String[] args) {
    Map<String, String> map = new HashMap<String, String>();

    //値のセットは put メソッドを利用
    map.put("key1", "value1");
    map.put("key2", "value2");
    map.put("key3", "value3");
    map.put("key4", "value4");
    map.put("key5", "value5");

    //値の取得
    String value = map.get("key1"); //value1 が取得できる
    System.out.println(value);

    String valueNull = map.get("key6"); //存在しない key の場合は null
    System.out.println(valueNull);

    //map に該当する key が存在するか否かチェックすることも可能
    if (map.containsKey("key1")) {
      System.out.println("key1 は存在します");
    } else {
      System.out.println("key1 は存在しません");
    }

    //拡張 for 文を利用してすべて情報を取得する
    for(Map.Entry<String, String> e : map.entrySet()) {
      System.out.println(e.getKey() + " : " + e.getValue());
    }
  }
}
```

Map を使った場合には、
put()で要素を記憶できます。

Map を使った場合には、
get()で要素を取得できます。

拡張 For 文：
map.entrySet()に要素が記憶されています。
これを Map.Entry で String, String で取得できるようにしています。
拡張 For 文を使って、取得したものは都度（ループ毎に）e に記憶されます。
e.getKey()とするとキーが、e.getValue()とすると、要素が取得できます。

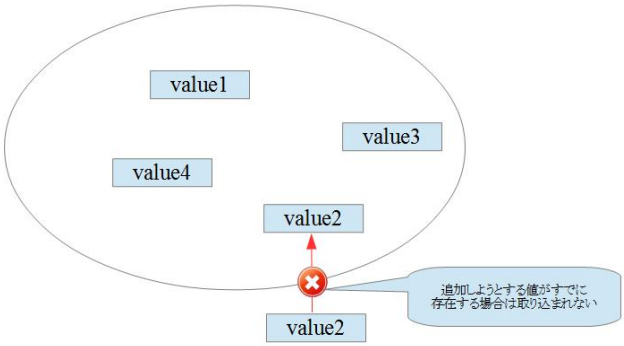
Map サンプルソース実行結果

```
value1
null
key1 は存在します
key1：value1
key2：value2
key5：value5
key3：value3
key4：value4
```

Set

Set：重複要素を持たない要素の集合。

Set のイメージ図



Set を使う場合には、以下のいずれかを使ってインスタンス化します。

クラス	重複の有無	順番
HashSet	無し	登録順は意識されずに記憶される
TreeSet	無し	登録したキーが昇順で記憶される
LinkedHashSet	無し	登録した順番で記憶される

Set を使ったインスタンス化方法：

```
Set<[要素の型]> 変数名 = new HashSet<[要素の型]>();
Set<[要素の型]> 変数名 = new TreeSet<[要素の型]>();
Set<[要素の型]> 変数名 = new LinkedHashSet<[要素の型]>();
```

Set サンプルソース

```
import java.util.HashSet;
import java.util.Set;

public class SetSample {

  public static void main(String[] args) {
    Set<String> set = new HashSet<String>();

    //値のセットは add メソッドを利用
    set.add("value1");
    set.add("value2");
    set.add("value3");
    set.add("value4");
    set.add("value5");

    set.add("value2"); //重複要素を add

    //拡張 For 文を利用
    for (String s : set) {
      System.out.println(s);
    }
  }
}
```

Set を使った場合には、
add()で要素を記憶できます。

Set を使った場合には、拡張 For 文で要素を取得できます。
set 内の要素を都度 s に記憶して、これを画面表示しています。

Set サンプルソース実行結果

```
value5
value2
value1
value4
value3
```

重複を排除したい場合に一旦 Set にデータを入れて、すべて取り出すといった使い方をすることが多いです。