情報科学演習D
よく使うクラス（ライブラリ）について

2013/10/17

肥後芳樹

Javaのドキュメント

- 英語： http://docs.oracle.com/javase/6/docs/api/
- 日本語： http://docs.oracle.com/javase/jp/6/api/

- JDKに含まれるすべてのクラスについて，以下の情報を得ることができる
  - クラスの継承関係
  - クラスで定義しているフィールド，コンストラクタ，メソッドの一覧

文字列（String）処理関連でよく使うクラス

- ファイル読み込み，ファイル書き出し
  - java.io.BufferedReader, java.io.FileReader,
  - java.io.BufferedWriter, java.io.FileWriter,
- 文字列連結
  - java.lang.StringBuilder
- 文字列分割
  - java.lang.StringTokenizer
- データ構造
  - java.util.Collectionを実装したクラス群

java.util.Collection

- 多数のデータを扱うためのデータ構造
- リスト，集合等の頻繁に使うデータ構造を提供
- Iteratorを使って「順番にアクセスする」ことが容易にできる

リスト

- java.util.Listインターフェースを実装したクラス群
  - よく使うのはjava.util.ArrayListとjava.util.LinkedList
- 追加した順番を保ったまま格納される
- addで追加，getで取得，removeで削除

セット

- java.util.Setインターフェースを実装したクラス群
  - よく使うのはjava.util.HashSet
- 同じ物を2回以上入れることはできない
- 追加した順番は保たれない
  - HashSetの場合は順番という概念がない
  - java.util.SortedSetを使えば，決まった順序に並べて保持することができる
- addで追加，removeで削除

マップ（Collectionではない）

- java.util.Mapインターフェースを実装したクラス群
  - よく使うのはjava.util.HashMap
- keyとvalueを対応付けて格納する
- マップには順番の概念がないが，java.util.SortedMapを使えば，keyを決まった順序で並べて保持することができる
- 追加はput(key, value)，削除はput(key)

イテレータ

- リストやセットに格納された要素を順番にアクセスする仕組み
- うまく利用することでコードを短くできる

Listはjava.lang.Iterableを継承しているからこのような処理が記述可能

```
void method1(List<Integer> array){
   for(int i = 0 ; i < array.size() ; i++){
      Integer value = array.get(i);
      System.out.println(i);
   }
}

void method2(List<Integer> array){
   Iterator<Integer> iterator = array.iterator();
   while(iterator.hasNext()){
      Integer value = iterator.next();
      System.out.println(value);
   }
}

void method3(List<Integer> array){
   for(Integer value : array){
      System.out.println(value);
   }
}
```

オブジェクトのソート

- java.util.SortedSetやjava.util.sortedMapを使えば，登録したオブジェクトを自動的にソートすることが可能
- ソートはcompareToメソッドによって行う
  - compareToメソッドの詳細については，java.lang.Comparableインターフェースを調べると良い

```
public class A implements Comparable<A> {

    private int value;

    public A(int v) {
        this.value = v;
    }

    public int getValue() {
        return this.value;
    }

    @Override
    public int compareTo(A a) {
        if (this.value < a.value) {
            return -1;
        } else if (this.value > a.value) {
            return 1;
        } else {
            return 0;
        }
    }
}
```

列挙型

- 複数の定数のみを値として取る型を定義できる
- 列挙した定数には具体的な値を格納することも可能
- switch文と親和性が高い
- Javaの列挙型は機能が豊富，興味のある人は自分で調べてみて下さい

```
enum Type {

    ONE(1), TWO(2), THREE(3);

    int value;

    Type(int v) {
        this.value = v;
    }

    int getValue() {
        return this.value;
    }
}
```

```
class Test01{

    void method(Type t){
        switch(t){
            case ONE:
                …
                break;
            case TWO:
                …
                break;
            case THREE:
                …
                break;
        }
    }
}
```