簡単なJavaプログラム
その２

オブジェクト指向プログラミング特論

2016年度

只木進一：工学系研究科

前回のプログラムで発生しそうな不都合

- 通常、ソートは、数字を並べ替えるのが目的ではない。
  - データを何かの順に並べ替える
    - 順序が定められれば、何でもよい
- データの種類に対応して、コードを作り替える?
  - ソートは、標準的手法なのに

クラスを使って改善してみよう

- 名前と点数を保持するクラスEntry

```
public class Entry {
    final private String name;//名前
    final private int score;//点数
    public Entry(String name, int score) {
        this.name = name;        this.score = score;    }

    public Entry(final Entry entry) {
        this.name = entry.getName();   this.score = entry.getScore();    }

    public String getName() {return name;}
    public int getScore() {return score;}
    public String toString() {return getName() + ":" + getScore();    }
}
```

- 新しいインスタンスの生成new
  - 配列を一度に作ることもできる

```
new Entry[]{
        new Entry("Bob", 90),
        new Entry("Mary", 70),
        new Entry("Tom", 95),
        new Entry("Mark", 85),
        new Entry("Betty", 80)
    }
```

- Entry.getScore()で値を調べ、その大小でソートする

```
private List<Entry> bubble(Entry d[]) {//泡立ち法
    for (int j = d.length - 1; j >= 1; j--) {//後ろからループを回す
        for (int i = 0; i < j; i++) {
            //順序が逆の場合
            if (d[i].getScore()>d[i + 1].getScore()) {
                Entry c = d[i];
                d[i] = d[i + 1];
                d[i + 1] = c;
            }
        }
    }
    return Arrays.asList(d);
}
```

何が問題か・何を学ぶか

- メインのクラスがEntryクラスの中身を知らねばならない
  - 別のデータには別のプログラムが必要になる
  - ソートの手法は同じなのに
  - もっと普遍的な方法が欲しい

- データの実装とデータを処理する過程を分離
  - クラスの抽象化
  - 抽象的データ構造
  - 抽象的インターフェイス
  - デザインパターン

Javaにおける抽象クラス

- Abstract Class
  - クラスの原型・共通的構造
  - フィールドを持つ
  - 一部のメソッドが実装されていない
    - Abstractメソッド
  - extends を使って継承

Javaにおける抽象クラス

- Interface
  - 他のクラスからの呼ばれ方を定義
  - 定数と実装されていないメソッドだけを持つ：例外あり
    - Defaultメソッド
  - implements を使って継承

インターフェイスの利用例

- java.lang.Comparable
  - 要素の比較を定義する抽象インターフェイス
  - 「比較できる」と実際の比較方法を分離
- 新しいEntryクラス：EntryNew.java
- 新しいWithClassNewクラス：WithClassNew.java

EntryNewクラス

- 大小関係を比較できるクラスとして宣言

```
public class EntryNew
        implements Comparable<EntryNew> {
          //EntryNewクラスのインスタンスと比較できる
```

比較のためのメソッド

```
/**
   * Comparableインターフェイスに必要な比較のメソッド
   * @param e 比較対象
   * @return 自分が大きければ1、小さければ-1、同じならば0
   */
  public int compareTo(EntryNew e) {
     if (e.getScore() > score) {
        return -1;
     }
     if (e.getScore() < score) {
        return 1;
     }
     return 0;
  }
```

クラステンプレート

- 特定のクラスを特定しない
- クラスの中に現れるテンプレート

```
public class クラス名<T>
```

- メソッドの中に現れるテンプレート

```
public <T> メソッド名(T t, ...)
```

WithClassNew

- Comparableインターフェイスを持ったクラスならば、どんなクラスのインスタンスでもソートできるクラス
  - 対象クラス名をテンプレートで表示
  - クラス名EntryNewは表れない！
  - Comparableの拡張であることだけ

```
public class
    WithClassNew<T extends Comparable<T>>
```

課題

- ソートプログラムの再利用性を高める、もう一つの方法は、要素の比較をおこなう方法を指定する方法である。
- java.util.Comparatorを用いる
  - メソッド public int compare(T t1,T t2)
    - 正の整数：t1>t2
    - 負の整数：t1<t2
    - ゼロ：t1=t2
- サンプルプログラムの動作を理解すること。