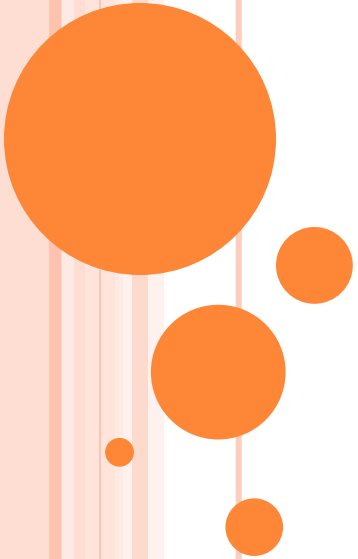

簡単なJavaプログラム その2

オブジェクト指向プログラミング特論

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ことになる
  - 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
  - Entry.java
- 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
- 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
- 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

インターフェイスの利用

- 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 java.lang.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;
   }
```

WithClassNew

- java.lang.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！

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

補足

- java.util.Listクラス
  - インターフェイス
  - リスト操作の標準的メソッド名が定義されている
  - stream()メソッド
    - 要素への順序付アクセス
  - Stream.forEach()メソッド
    - 各要素に対する処理

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。

WithClass.java

```java
package firstSample;

import java.util.Arrays;
import java.util.List;

/**
 * Entryクラスを使った例題
 *
 * @author tadaki
 */
public class WithClass {

    private final Entry entries[];//データを保存する整数配列

    /**
     * コンストラクタ
     *
     * @param entries Entryクラスの配列で、データを登録
     */
    public WithClass(Entry entries[]) {
        this.entries = entries;
    }

    /**
     * ソートの実行
     *
     * @return 結果を文字列で返す
     */
    public List<Entry> sort() {
        return bubble(entries);
    }

    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);
    }
```

WithClass.java

```
    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        WithClass withClass = new WithClass(new Entry[]{
            new Entry("Bob", 90),
            new Entry("Mary", 70),
            new Entry("Tom", 95),
            new Entry("Mark", 85),
            new Entry("Betty", 80)
        });

        List<Entry> list = withClass.sort();
        //streamとラムダ式による表記
        list.stream().forEachOrdered(
                p -> {
                    System.out.println(p.toString());
                }
        );
        /*
        for(Entry p:list){
            System.out.println(p.toString());
        }*/
    }
}
```

Entry.java

```java
package firstSample;

/**
 * Entry クラス 名前と点数を保持
 *
 * @author tadaki
 */
public class Entry {

    final private String name;//名前
    final private int score;//点数

    /**
     * コンストラクタ
     *
     * @param name 名前
     * @param score 点数
     */
    public Entry(String name, int score) {
        /**
         * this はこのインスタンスを表す
         */
        this.name = name;
        this.score = score;
    }

    /**
     * 他のインスタンスをコピーするコンストラクタ
     *
     * @param entry コピー元
     */
    public Entry(final Entry entry) {
        this.name = entry.getName();
        this.score = entry.getScore();
    }

    /**
     * 名前を返す
     *
     * @return このインスタンスの保持している名前
     */
    public String getName() {
        return name;
    }

    /**
```

Entry.java

```
     * 点数を返す
     *
     * @return このインスタンスが保持している名前
     */
    public int getScore() {
        return score;
    }

    @Override
    /**
     * 文字列への変換
     */
    public String toString() {
        return getName() + ":" + getScore();
    }
}
```

WithClassNew.java

```
package firstSample;

import java.util.Arrays;
import java.util.List;

/**
 * java.lang.Comparableを実装したクラステンプレートT を
 * 使うことを指示 明示的にクラスEntryNewが使われていない点に注意
 *
 * @author tadaki
 * @param <T>
 */
public class WithClassNew<T extends Comparable<T>> {

    private final T entries[];//データを保存する整数配列

    public WithClassNew(T entries[]) {
        this.entries = entries;
    }

    public List<T> sort() {
        return bubble(entries);
    }

    private List<T> bubble(T d[]) {//泡立ち法









        return Arrays.asList(d);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        EntryNew e[] = new EntryNew[]{
            new EntryNew("Bob", 90),
            new EntryNew("Mary", 70),
            new EntryNew("Tom", 95),
            new EntryNew("Mark", 85),
```

```
            new EntryNew("Betty", 80)
        };

        WithClassNew<EntryNew> withClass = new WithClassNew<>(e);
        List<EntryNew> list = withClass.sort();
        list.stream().forEachOrdered(
                (p) -> {
                    System.out.println(p.toString());
                }
        );
    }
}
```

EntryNew.java

```java
package firstSample;

/**
 * EntryNew クラス 名前と点数を保持
 * Comparableインターフェイスを使う
 * @author tadaki
 */
public class EntryNew
        implements java.lang.Comparable<EntryNew> {

    final private String name;//名前
    final private int score;//点数

    /**
     * コンストラクタ
     *
     * @param name 名前
     * @param score 点数
     */
    public EntryNew(String name, int score) {
        this.name = name;
        this.score = score;
    }

    /**
     * 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
     */
    @Override
    public int compareTo(EntryNew e) {
        if (e.getScore() > score) {
            return -1;
        }
        if (e.getScore() < score) {
            return 1;
        }
        return 0;
    }

    public String getName() {
        return name;
    }

    public int getScore() {
        return score;
```

```
    }

    @Override
    public String toString() {
        return getName() + ":" + getScore();
    }
}
```

WithClassNew.java

```java
package secondSample;

//必要なライブラリの取り込み
import java.util.Arrays;
import java.util.Comparator;
import java.util.List;

/**
 * Comparatorを使った例
 * @author tadaki
 * @param <T>
 */
public class WithClassNew<T> {

    private final Comparator<T> comparator;
    //クラスT のインスタンスを要素とするリスト
    private final T entries[];//データを保存する整数配列

    public WithClassNew(T entries[], Comparator<T> comparator) {
        this.entries = entries;
        this.comparator = comparator;
    }

    public List<T> sort() {
        return bubble(entries);
    }

    private List<T> bubble(T 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++) {
                if (comparator.compare(d[i], d[i + 1]) > 0) {
                    T c = d[i];
                    d[i] = d[i + 1];
                    d[i + 1] = c;
                }
            }
        }
        return Arrays.asList(d);
    }

    /**
     * @param args the command line arguments
     */
    public static void main(String[] args) {
        EntryNew entries[] = new EntryNew[]{
            new EntryNew("Bob", 90),
```

```
            new EntryNew("Mary", 70),
            new EntryNew("Tom", 95),
            new EntryNew("Mark", 85),
            new EntryNew("Betty", 80),
            new EntryNew("Ann", 85),
            new EntryNew("Kim", 95)
        };

        WithClassNew<EntryNew> withClass = new WithClassNew<>(entries,
                (x, y) -> x.getScore() - y.getScore());

        List<EntryNew> result = withClass.sort();
        result.stream().forEachOrdered((x) -> {
            System.out.println(x.toString());
        });
    }
}
```

EntryNew.java

```java
package secondSample;

/**
 *
 * @author tadaki
 */
public class EntryNew {

    final private String name;
    final private int score;

    public EntryNew(String name, int score) {
        this.name = name;
        this.score = score;
    }

    public String getName() {
        return name;
    }

    public int getScore() {
        return score;
    }

    @Override
    public String toString() {
        return getName() + ":" + getScore();
    }
}
```