抽象クラスの抽出

オブジェクト指向プログラミング特論

2018年度

只木進一：工学系研究科

抽象クラスの抽出

- 既存のクラスから共通的部分を抽出
- Netbeansでは、「リファクタリング」機能で可能

MergeSortからの抽出

- そのまま抽出
  - less()、isSorted()、list
- 抽象化して抽出
  - sort()
- AbstractSortクラス

AbstractSortの継承クラス

- MergeSort
  - sort()を上書き
- BubbleSort
  - sort()を上書き
  - swap()をAbstractSortへ

SelectionSortを継承クラスとして作成

n //要素数

for ($i = 0$; $i < n - 1$; $i++$){

  m = ($i < j < n$の範囲の最小要素の位置)

  if ($i \neq m$)swap(i, m)

}

AbstractSort.java

```java
package example2;

import java.util.List;

/**
 *
 * @author tadaki
 * @param <T>
 */
public abstract class AbstractSort<T extends Comparable> {

    protected final List<T> list;

    public AbstractSort(List<T> list) {
        this.list = list;
    }

    /**
     * 整列の実行
     *
     * @return 整列済みのリスト
     */
    public abstract List<T> sort();

    protected boolean less(int i, int j) {
        return list.get(i).compareTo(list.get(j)) < 0;
    }

    protected void swap(int i, int j) {
        T t = list.get(i);
        list.set(i, list.get(j));
        list.set(j, t);
    }

    public boolean isSorted() {
        boolean b = true;
        for (int i = 0; i < list.size() - 1; i++) {
            if (!less(i, i + 1)) {
                return false;
            }
        }
        return b;
    }

}
```

MergeSort.java

```java
package example2;

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

/**
 * MergeSort
 *
 * @author tadaki
 * @param <T>
 */
public class MergeSort<T extends Comparable> extends AbstractSort<T> {


    public MergeSort(List<T> list) {
        super(list);
    }

    /**
     * 整列の実行
     *
     * @return 整列済みのリスト
     */
    @Override
    public List<T> sort() {
        sortSub(0, list.size());
        return list;
    }

    /**
     * 再帰的整列
     *
     * @param left リストの整列対象のうち左端のインデクス
     * @param right リストの整列対象のうち右端のインデクス＋１
     */
    protected void sortSub(int left, int right) {
        if (right <= left) {
            throw new IllegalArgumentException("illegal range");
        }
        if (right == left + 1) {
            return;
        }
        int middle = (right + left) / 2;
        //再帰呼び出し
        sortSub(left, middle);
        sortSub(middle, right);
```

```
        //リストの結合
        List<T> tmpList = mergeList(left, middle, right);
        for (int p = 0; p < tmpList.size(); p++) {
            list.set(left + p, tmpList.get(p));
        }
    }

    /**
     * リストの結合
     *
     * @param left 左端
     * @param middle 右側要素の先頭
     * @param right 右側要素の終端＋１
     * @return
     */
    protected List<T> mergeList(int left, int middle, int right) {
        List<T> tmp = new ArrayList<>();
        int leftIndex = left;
        int rightIndex = middle;
        while (leftIndex < middle || rightIndex < right) {
            if (leftIndex >= middle) {//左側終了
                for (int k = rightIndex; k < right; k++) {
                    tmp.add(list.get(k));
                }
                return tmp;
            }
            if (rightIndex >= right) {//右側終了
                for (int k = leftIndex; k < middle; k++) {
                    tmp.add(list.get(k));
                }
                return tmp;
            }
            if (less(leftIndex, rightIndex)) {
                tmp.add(list.get(leftIndex));
                leftIndex++;
            } else {
                tmp.add(list.get(rightIndex));
                rightIndex++;
            }
        }
        return tmp;
    }


}
```