



# 泡立ち法とその実装

計算機アルゴリズム特論：2017年度  
只木進一



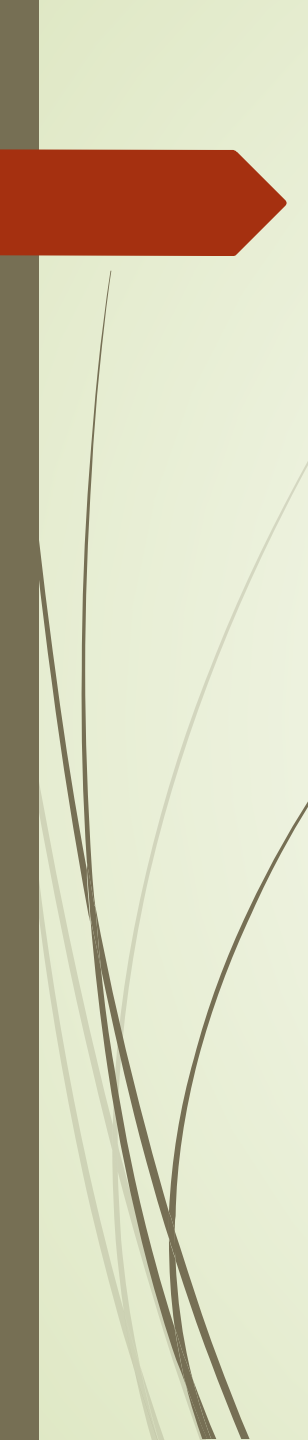
# 目的

- 泡立ち法を例に、
  - Comparableインターフェイスの実装
  - 抽象クラスの利用
  - 型パラメタの利用
  - 「比較」、「入替」の回数を計測



# Comparableインターフェイス

- クラスインスタンスが比較可能であることを示す
- `int compareTo()` メソッドを実装
- `Integer`、`Double`、`String`などには実装済み



```
public class Data implements Comparable<Data> {
    private final String label;
    private final int value;
    public Data(String label, int value) {
        this.label = label; this.value = value;
    }

    @Override
    public int compareTo(Data o) { return this.value - o.value; }
    @Override
    public String toString() { return label + ":" + value; }

    /**
     * テストデータの生成
     */
    static public Data[] createData(int numData) {
        Data[] data = new Data[numData];
        for (int i = 0; i < numData; i++) {
            int k = (int) (10 * numData * Math.random());
            data[i] = new Data(String.valueOf(i), k);
        }
        return data;
    }
}
```

# 型パラメタ

- クラスはメソッドが扱うクラスを表現
  - クラスやメソッドの記述では特定しない  
(ある程度制限することも可)
  - 例：リストへの保存
  - 例：整列の対象

# 整列を実行する抽象クラス

- 対象を表す型パラメタTは、Comparableの拡張クラス
- 実際の整列方法doSort()は未実装

```
public abstract class AbstractSort<T extends Comparable<T>> {  
    protected T[] data;//対象  
    private int numExch;//要素入替回数  
    private int numComp;//要素比較回数  
  
    abstract public T[] doSort();  
  
    ///他のメソッド  
}
```

# その他のメソッド

## ■ 配列設定

```
public void setArray(T[] data)
```

## ■ 要素比較

```
protected boolean less(T v, T w)
```

```
protected boolean lesseq(T v, T w)
```

```
protected boolean lessByIndex(int i, int j)
```

```
protected boolean lesseqByIndex(int i, int j)
```

## ■ 要素入れかえ

```
protected void exch(int i, int j)
```

# 課題

- データの数 $N$ を変化させ、比較が概ね  $N^2$  で大きくなる様子を図示しなさい。



AbstractSort.java

```
package sort;

import java.io. BufferedWriter;
import java.io. IOException;
import myLib.utils.FileIO;

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

    protected T[] data;//対象
    private int numExch;//要素入替回数
    private int numComp;//要素比較回数

    public AbstractSort(T[] data) {
        this.data = data;
        numExch = 0;
        numComp = 0;
    }

    public AbstractSort() {
    }

    public void setArray(T[] data) {
        this.data = data;
        numExch = 0;
        numComp = 0;
    }

    abstract public T[] doSort();

    /**
     * 大小関係 (vはwより小さい)
     *
     * @param v
     * @param w
     * @return
     */
    protected boolean less(T v, T w) {
        numComp++;
        return (v.compareTo(w) < 0);
    }
}
```

## AbstractSort.java

```
protected boolean lesseq(T v, T w) {
    numComp++;
    return (v.compareTo(w) <= 0);
}

protected boolean lessByIndex(int i, int j) {
    return less(data[i], data[j]);
}

protected boolean lesseqByIndex(int i, int j) {
    return lesseq(data[i], data[j]);
}

/**
 * 要素iとjを入れ替える
 *
 * @param i
 * @param j
 */
protected void exch(int i, int j) {
    int n = data.length;
    if (i < 0 || i >= n || j < 0 || j >= n) {
        throw new IllegalArgumentException("Indexes are out of bound");
    }
    T t = data[i];
    data[i] = data[j];
    data[j] = t;
    numExch++;
}

public boolean isSorted() {
    boolean b = true;
    int numCompSave=this.numComp;
    for (int i = 0; i < data.length - 1 && b; i++) {
        b = b & lesseqByIndex(i, i + 1);
    }
    this.numComp=numCompSave;
    return b;
}

public void printArray(BufferedWriter out) throws IOException {
    for (int i = 0; i < data.length; i++) {
        out.append(data[i].toString());
        out.newLine();
    }
}
```

AbstractSort.java

```
    }

    public T[] getArray() {
        return data;
    }

    public int getNumExchange() {
        return numExch;
    }

    public int getNumCompare() {
        return numComp;
    }

    /**
     * テストランを実施
     *
     * @param sort
     * @throws IOException
     */
    static public void testRun(AbstractSort sort)
        throws IOException {
        sort.doSort();
        if (sort.isSorted()) {
            System.out.println("Sorting Completes");
            System.out.println("Number of Compare " + sort.getNumCompare());
            System.out.println("Number of Exchange " +
sort.getNumExchange());
            try (BufferedWriter out = FileIO.openWriter("output.txt")) {
                sort.printArray(out);
            }
        } else {
            System.out.println("Sorting fails");
        }
    }
}
```

BubbleSort.java

```
package sort;

import java.io.IOException;

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

    public BubbleSort(T[] data) {
        super(data);
    }

    public BubbleSort() {
    }

    @Override
    public T[] doSort() {
        int n = data.length;
        for (int i = n; i > 0; i--) {
            for (int j = 0; j < i - 1; j++) {
                if (lessByIndex(j + 1, j)) {
                    exch(j + 1, j);
                }
            }
        }
        return data;
    }

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     */
    static public void main(String args[]) throws IOException {
        int numData = 100;
        Data[] data = Data.createData(numData);
        testRun(new BubbleSort<>(data));
    }
}
```

CountSteps.java

```
package observation;

import java.io. BufferedWriter;
import java.io. IOException;
import java.util. List;
import myLib.utils.FileIO;
import myLib.utils.Utills;
import sort.*;

/**
 * 整列アルゴリズムの評価を行う
 *
 * @author tadaki
 */
public class CountSteps {

    private AbstractSort<Data> sort;//対象となるアルゴリズム

    public CountSteps(AbstractSort<Data> sort) {
        this.sort = sort;
    }

    public void setRandomDataAndDo(int numData) {
        Data[] data = Data.createData(numData);
        sort.setArray(data);
        sort.doSort();
    }

    public List<PerformanceData> measure(int min, int max) {
        int n = min;
        List<PerformanceData> list = Utills.createList();
        while (n <= max) {
            setRandomDataAndDo(n);
            list.add(new PerformanceData(n,
                sort.getNumCompare(), sort.getNumExchange()));
            n *= 2;
        }
        return list;
    }

    /**
     * @param args the command line arguments
     * @throws java.io.IOException
     * @throws java.lang.ClassNotFoundException
     * @throws java.lang.InstantiationException
     * @throws java.lang.IllegalAccessException
     */
}
```

## CountSteps.java

```
*/
public static void main(String[] args)
    throws IOException, ClassNotFoundException,
        InstantiationException, IllegalAccessException {
    //対象となるクラスの名前を登録
    String[] classNames = {"sort.BubbleSort", "sort.MergeSort",
        "sort.QuickSort"};
    for (String s : classNames) {
        //クラス名からクラスインスタンスを生成
        @SuppressWarnings("unchecked")
        Class<AbstractSort<Data>> sortClass
            = (Class<AbstractSort<Data>>) Class.forName(s);
        AbstractSort<Data> sort = sortClass.newInstance();
        CountSteps cs = new CountSteps(sort); //評価プログラムへ
        String filename = sort.getClass().getSimpleName() + ".txt";
        List<PerformanceData> list = cs.measure(16, 4096);
        try (BufferedWriter out = FileIO.openWriter(filename)) {
            for (PerformanceData p : list) {
                FileIO.writeSSV(out, p.n, p.numComp, p.numExch);
            }
        }
    }
}
}
```

PerformanceData.java

```
package observation;
```

```
/**
```

```
 *
```

```
 * @author tadaki
```

```
 */
```

```
public class PerformanceData {
```

```
    final int n;
```

```
    final int numComp;
```

```
    final int numExch;
```

```
    public PerformanceData(int n, int numComp, int numExch) {
```

```
        this.n = n;
```

```
        this.numComp = numComp;
```

```
        this.numExch = numExch;
```

```
    }
```

```
}
```