# Study 課題 01

1250373 溝口洸熙 *

2022 年 5 月 12 日

**概要**

このレポートは，Study 課題の各問題の工夫点をまとめたものである．コードの転記には `listings`,`jlisting` を用いており，描画には TikZ を用いている．このレポートは，ソースコードの行番号を消している．

## プログラミングをする上で注意したこと

### 0.1 コメント

まず，ある処理に対して，昨日の自分と，今日の自分が同じ考えをするとは限らない．その上で，誰が見ても何の処理をしているかどうかをよりわかりやすくするために，可能な限りコメントを残している．

### 0.2 インデント

インデントは，プログラムを直感的に捉えるために必要である．多重分岐や多重ループなどの処理は階層構造になっており，階層を一目見ただけで理解できると，作業の効率化や，間違いを少なくできる．今回作成したプログラムは，演算子と値の間も適切にスペースを挿入することも含めてインデントを適切に使用している．また，**誰が見ても分かりやすいコードを目指して変数名も工夫している**．

### 0.3 メソッド化

ある処理 A を，処理 B・処理 C が利用することはしばしばある．その場合，処理 B・処理 C にそれぞれ処理 A を書くと，万一処理 A を変更したいときに両方の処理 A を変更する必要がある．この時に，処理 A をメソッドとしておくと，処理 A の変更が 1 度で済み，変更によるエラーが起きにくい．また，メソッド化することで，処理過程が非常に見やすいソースコードを実現できる．

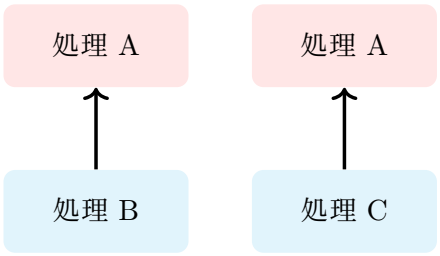



図 1　メソッド化されていない

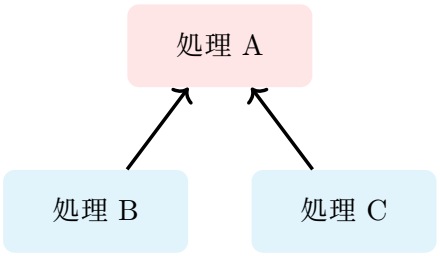



図 2　メソッド化されてる

* 高知工科大学 情報学群 学士 2 年

# 課題 1

課題ファイル名　Study01_1.java

## 1.1　大まかな処理過程

src. 1　多次元配列の作成

```
int input = Integer.parseInt(args[0]);
String[][] array = new String[input * 2 + 1][];// 配列の"行"を作成
for (int i = 0; i < array.length; i++) {// 配列の"列"を作成
    if (i < input) {
        array[i] = new String[input * 2 + 1 - i];// 必要な配列のみ作成.
    } else {
        array[i] = new String[i + 1];
    }
}
```

src. 2　三角形と逆三角形を処理

```
for (int i = 0; i < array.length; i++) {
    if (i <= input) {// 逆三角形の処理
        for (int j = 0; j + i < array[i].length; j++) {
            array[i][j + i] = "*";
        }
    } else {// 三角形の処理
        for (int j = 0; array.length - i - 1 + j < array[i].length; j++) {
            array[i][array.length - i - 1 + j] = "*";
        }
    }
}
```

大まかな処理過程

1) 多次元配列を作成する．**必要な配列のみを作成**．（src.1）
2) 逆三角形を出力し，三角形を出力する．（src.2）
3) 配列を初期化していない場合，その配列の要素は `null` であるので，配列の要素 `null` をスペースへ変換するメソッドに通す．
4) 完成した配列を，配列を出力するメソッドに通す．

## 1.2 工夫点

多次元配列の利用

実験の Lesson 課題でも似たような問題が出たが，その時に利用した多重ループを利用するのではなく，多次元配列を用いて実行してみようと試みた．

多次元配列を用いるにあたって，コマンドライン引数 `args[0]`[1]に対して（src.3）のように配列を作成する方が楽である．ただ，この配列の組み方には問題がある．

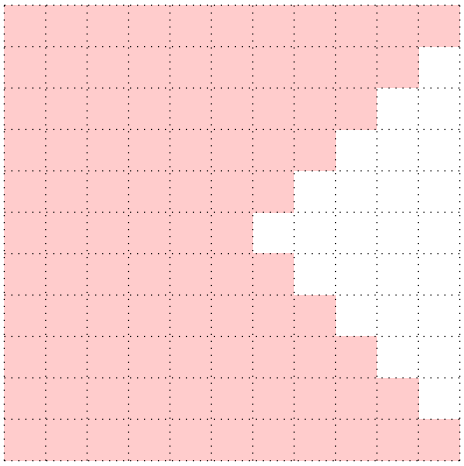
src. 3　無駄が多い配列の組み方

```
input = 5;// コマンドライン引数の仮
String[][] array = new String[input * 2 + 1][input * 2 + 1];
```

この配列の組み方は，無駄が多い．配列を宣言すると宣言した分のメモリが確保されるので，使わない変数を無駄に宣言するとメモリの無駄な消費につながる．

具体的には，図 3 の部分の着色部分が実際に使われる配列の要素．非着色部は使用されていないのに宣言されている無駄な配列である．

図 3　何かしらの要素が入る配列



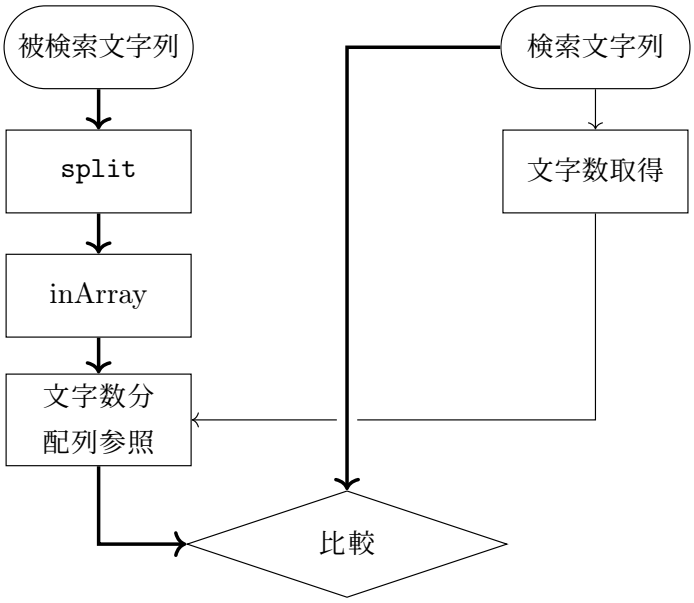
ここで，（src.1）のように，着色部のみの配列を宣言し，そこに文字（列）を代入する工夫をおこなった．

[1] 本来は `String` 型だが，ここでは `int` 型に変換しているものとする．

# 課題 2

課題ファイル名　`Study01_2.java`

## 2.1　大まかな処理過程

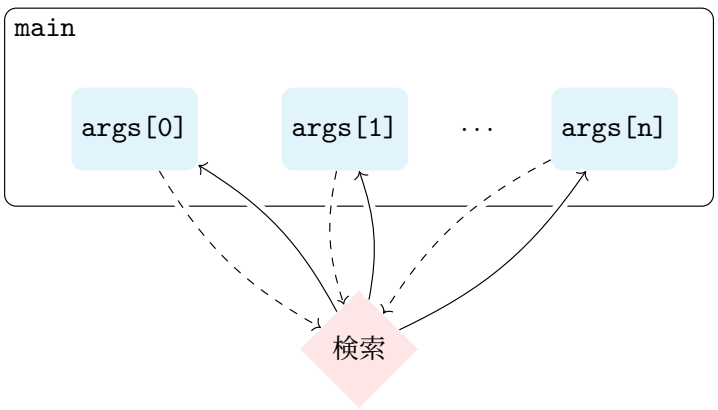



## 2.2　工夫点

検索メソッドの分離

p.1 第 0.3 節でも述べたが，同一の処理をメソッドとしてブロックのように扱うと一連の処理を簡単かつ明瞭にできる．

今回は，被検索文字列が，複数あることが前提であり，その被検索文字列を `split` メソッドを用いて分割し，それを配列に代入する．配列に対して行う処理が 1 度ではないので，検索するメソッドを分離することで，処理過程をより見やすくした．




- - - - → 配列参照引き渡し　　——→ 検索結果（`boolean`）値

# 課題 3

課題ファイル名　Study01_3.java

## 3.1　大まかな処理過程

src. 4　ある数に対して素数か否か判定する

```
int factorialCounter = 1;// 階乗のカウンタ
for (int i = 2; i <= number; i++) {
    if (number % i == 0) {//
      numberをiで割った剰余が0ならば，numberはその数を素因数にもつ
        number = number / i;// numberにnumberをiで割った商を代入
        if (number % i == 0) {// もう一度同じ数で割った剰余が0ならば，カウン
          タをインクリメント
            factorialCounter++;
        } else {
            printPrimeFactor(number, i, factorialCounter);// 素因数を出力する
              メソッドへ．
            factorialCounter = 1;// カウンタをリセット
        }
        i = 1;
    }
}
```

src. 5　その素数が最大であるか判断する

```
public static void printPrimeFactor(int number, int i, int factorialCounter)
  {
    if (number == 1) {// その素因数が最大の時
        if (factorialCounter == 1) {// その素因数が1乗の時
            System.out.print(i);
        } else {
            System.out.print(i + "^" + factorialCounter);
        }
    } else {// その素因数が最大でない時
        if (factorialCounter == 1) {// その素因数が1乗の時
            System.out.print(i + "␣*␣");
        } else {
            System.out.print(i + "^" + factorialCounter + "␣*␣");
        }
    }
}
```

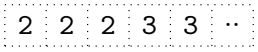
1) 入力数値 number に対して，ある数 i で割った剰余が 0 ならば，number は i を素因数にもつ．
2) 同一の複素数が 2 個以上存在する時，factoricalCounter をインクリメントする．(src.4)
3) 全ての素因数の中で，一番大きいもの以外は，　* 記号を後ろにつけて出力する．(src.5)
4) factoricalCounter が 2 以上の時，　^ 記号の後ろに factoricalCounter を添える．(src.5)

## 3.2 工夫点

指数カウンタ

はじめ，この問題に取り掛かるとき，配列を用いて解決しようとした．素因数個数分の配列に素因数を昇順に並べる方法である．昇順に並べた素因数が同じもの同士の個数を数えて指数を作る方法をとった．（図4）しかし，p.3 第 1.2 節にあるように，無駄な変数を置くことは避けた方が良い．

図 4　配列を用いた素因数の管理




ここでカウンタを用いると，配列を用いることなく指数の出力が可能になった．（src.4）

# 課題 4

課題ファイル名　Study01_4.java

## 4.1　大まかな処理過程

src. 6　コマンドライン引数から曜日シフトを決定

```
String[] weekName = { "sun", "mon", "tue", "wed", "thu", "fri", "sat" };
String[] week = { "土", "日", "月", "火", "水", "木", "金" };
int weekShift = -1;
for (int i = 0; i < weekName.length; i++) {// コマンドライン引数で曜日シフト
  数を決定.
    if (args[0].equals(weekName[i])) {
        weekShift = i;
    }
}
```

src. 7　経過日数を算出

```
int[] daysOfMoth = { 31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31 };
public static int daysElapsed(int inputMonth, int inputDay, int[] daysOfMoth)
   {
    int daysElapsed = 0;
    for (int i = 0; i < inputMonth - 1; i++) {
        if (inputMonth == 1) {
            return inputDay;
        }
        daysElapsed += daysOfMoth[i];
    }
    daysElapsed += inputDay;
    return daysElapsed;
}
```

src. 8　出力

```
int mod7 = (dayElapsed + weekShift) % 7;
System.out.println(inputMonth + "月" + inputDay + "日は," + week[mod7] +"曜日で
  す.");
```

## 4.2 工夫点

ユーザビリティを考慮した設計

コマンドライン入力や日付の入力など，ユーザの動作によって処理する事項は，思わぬエラーを引き起こす．このプログラムは，あらゆるエラーに対して，エラーメッセージを出力し，プログラムの中止や再入力を促す．

エラーメッセージは，基本的に使い回しが多いので，エラーをコード化し，それに対するメッセージを出力するメソッドを作成した．(src.9)

src. 9　エラーメッセージ・メソッド

```
public static void printError(int i) {
    if (i == 0)
        System.out.println("調べたい日付が不正な組です");
    else if (i == 1)
        System.out.println("コマンドライン引数の値が不正です");
    else if (i == 2)
        System.out.println("調べたい日付が不正な入力です");
    else if (i == 3)
        System.out.println("演算できませんでした");
}
```

コマンドライン引数が不正ではプログラムが動かないので，コマンドライン引数の不正を検出し，`System.exit(0);`で，プログラムを強制終了する．(src.10)

src. 10　コマンドライン引数の不正検出

```
int weekShift = -1;
if (args.length == 0 || args.length > 1) {// コマンドライン引数が不正の場合（
  入力なし，または，2つ以上の入力）にエラーを出し，終了
    printError(1);
    System.exit(0);
}
```

日付組み合わせの不正検出

2 月は 28 日までしかないので，2 月 30 日と入力があると，エラーである．しかし，全ての月の最後の日付に対して条件分岐を行うと，非常に見難いプログラムになる．そこで，論理演算子と配列を用いて 2 つの条件分岐で日付組み合わせの不正を検出するメソッドを作成した．（src.11）

src. 11　日付組み合わせ不正検出

```
public static boolean judgeMothAndDate(int inputMonth, int inputDay, int[]
  daysOfMoth) {// 月日の組み合わせの判定（不正であれば False）
    if (inputMonth > 12 || inputMonth < 1) {// 不正の条件
        printError(0);
        return false;
    }
    if (1 > inputDay || inputDay > daysOfMoth[inputMonth - 1]) {// 不正の条件
        printError(0);
        return false;
    }
    return true;
}
```

これは，(index + 1) 月の最終日が格納してある配列 dayOfMonth（src.7）から不正条件を導き出している．この検出方法で，この配列と西暦の入力で，閏年の計算も可能になる．

曜日出力をシンプルに

曜日を出力するために，一般的に 経過日数 mod 7 を計算し，その解に応じて if 文，swith 文を使う．7 の剰余の条件分岐は多行に渡る．非常に見難いプログラムになることは明白である．（src.12）

ここで，曜日（漢字）を格納する配列を用意し，7 で割った剰余番目の配列を呼び出すシンプルな仕組みを作成した．（src.13）

src. 12　switch 文を用いたコード例

```
int mod7 = dayElapsed % 7;
switch(mod7){
    case 0:
        System.out.println("土曜日です．");
        break;
    case 1:
        System.out.println("日曜日です．");
        break;
        ・
        ・
        ・
}
```

src. 13　配列を用いたコード例

```
String[] week = { "土", "日", "月", "火", "水", "木", "金" };
int mod7 = dayElapsed % 7;
System.out.println( week[mod7] +"曜日です．");
```

# 課題 5

課題ファイル名　Study01_5.java

## 5.1　大まかな処理過程

src. 14　コマンドライン引数の不正を検出

```
if (args.length != 2) {
    System.out.println("コマンドライン引数は2つです");
    System.exit(0);
}
int firstNumber = Integer.parseInt(args[0]);
int secondNumber = Integer.parseInt(args[1]);
int linesCounter = 0;
if (firstNumber > 1000 || firstNumber < 1 || secondNumber > 1000 ||
  secondNumber < 1) {
    System.out.println("コマンドライン引数は1以上1000以下を入力してください");
    System.exit(0);
}
```

src. 15　コマンドライン引数の大小比較・条件分岐

```
if (firstNumber < secondNumber) {// 昇順
    for (int i = firstNumber; i <= secondNumber; i++) {
        if (isPrime(i) == true) {
            System.out.printf("%4d", i);
            if (linesCounter == 11) {// 改行判断
                System.out.println();
                linesCounter = 0;
            } else {
                linesCounter++;
            }
        }

    }
} else {...
```

src. 16　素数判断

```
public static boolean isPrime(int i) {// 素数であればTrue
    if (i == 1) return false;
    if (i % 2 == 0 && i != 2) return false;
    double sqrtI = Math.sqrt(i);
    for (int j = 3; j <= sqrtI; j += 2) {
        if (i % j == 0) return false;
    }
    return true;
}
```

## 5.2 工夫点

コマンドライン引数での不正を検出

p.8 第 4.2 章と同じく，コマンドライン引数でエラーを検出し，プログラムを終了する仕組みを作成した．(src.14)

出力終了ラインの調整

この問題は 12 列の素数出力で改行する必要がある．細かいところだが，ちょうど 12 列の出力の場合，最終出力行の次の行に = で造られた出力終了ラインを出力する必要がある．そのことを考慮した結果と，考慮しなかったっ結果を比べる．当然だが，考慮した方が題意に沿っている．

考慮しなかった場合

```
$ java Study01_5 3 41 Enter
=============   3 から  41 までの素数=============
   3   5   7  11  13  17  19  23  29  31  37  41


================================================
```

考慮した場合

```
$ java Study01_5 3 41 Enter
=============   3 から  41 までの素数=============
   3   5   7  11  13  17  19  23  29  31  37  41
================================================
```

列数カウンタ（linesCounter）に対して，条件分岐を設けた．(src.17)

src. 17　列数カウンタによる条件分岐

```
if (linesCounter == 0) {// 最後の数値が12列目か否か
    System.out.println("================================================");
} else {
    System.out.println();
    System.out.println("================================================");
}
```

素数判断の効率化

素数判定について，「素数でない数 $x$ は $\sqrt{x}$ 以下の約数をもつ」という性質を用いれば，劇的に処理が速くなる．(src.16)

本来（src.18），ある数 i に対して，その数が 2 以上，i 以下で割り切れるかどうかを判定するが，src.16 では，その数の 1/2 乗の数までのみ剰余演算をし，ループ回数が大幅に減って処理速度が大きくなる．

src. 18　素数判定の基本

```
    if (i == 1) return false;
    if (i == 2) return true;
    for (int j = 2; j < i; j++) {
        if (i % j == 0) return false;
    }
    return true;
```