今週の一コマ画像

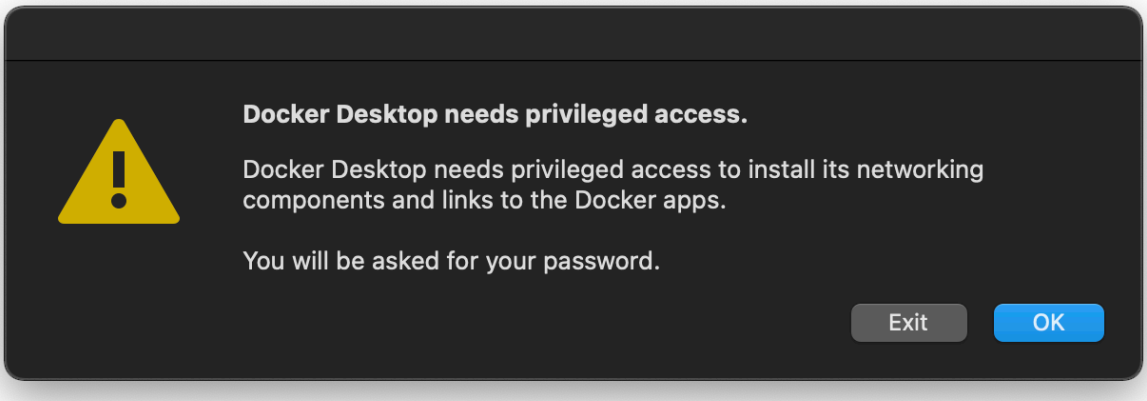

図0…いくらパスワードを打ってもこの画面から変わらないBetaOS

APJ05

学籍番号: 744366

氏名: 岸 典樹

2020年10月24日

第1章 プログラム設計とプログラム実装について

本章ではプログラミングの前段階で執り行う基本設計(外部設計)や詳細設計(内部設計)について、当該のプログラム設計とプログラム実装がどのようにつながるのかについて、ソフトウェアプロセスに基づいてウォーターフォールモデルで示す。

1.1 分析・要件定義

要件分析や要件定義はシステムの開発を行う上で、最も初めに行う作業であり、利用者がそのシステムに何を求めているのかを考えていく。

1.2 設計

設計には大きく分けて基本設計と詳細設計があり、基本設計では要件定義で決定された事項を具体化し、詳細設計では基本設計で具体化したそれを画面単位やプログラム単位で詳細に設計していく。状態遷移図、データフロー図については基本設計で、クラス図やフローチャートなどの設計は主にプログラム設計で行う。

1.3 実装

設計された使用をプログラムに落とし込むフェーズがこの実装である。基本設計・詳細設計により使用がかなり整理されたため、設計図をもとに徐に書いてゆく。通常、設計図は必ず用意するが、例えば用意されていなかった場合、このフェーズでシステムの仕様についてボトムアップで考えなければならないため、多くの時間を必要することと、エンジニア間、エンジニアとクライアント間で多くの齟齬が発生してしまうため、これは絶対避けなければならない。Javaなど一部の言語は専用のエディタを使うことでUMLから直接雛形を作ることが可能であり、作業量を減らしながら作業もれを防いでゆく。

1.4 テスト

プログラムがひと段落すると、テストに入る。テスト仕様書を作成し、列挙された項目に対して単体テスト、結合テスト、総合テスト、運用テストを行う。単体テストと結合テストではJestやxUnitなどを使用してテストコードを作成し、テストを行う。クライアントの利用環境と同じ環境で総合テストを行った上で、運用テストを行い、納品の流れになる。

1.5 保守・運用

この工程では報告されたバグの解消やメンテナンス、追加実装を行う。追加実装を行う場合は、また1.1節に戻り、設計から行ってゆく。

1.6 設計書の意味

建築においても洋裁においても、設計図は欠かせない。それを元に物事を忠実に形にしていくことが技能者に求められる仕事である。同時に、設計図を見て作り上げる人々はチームとしての作業内容が整理されているため、開発においても大幅な効率UPが期待できる。実装フェーズでは、backlogやgitlabなどを使用してタスク管理・個々のアサインを行うことで、担当者の整理や急な欠勤への対応が整理されるため、必ず使用されるだろう。

第2章 安直に「+」演算子の使用が許される場合について

本章では安直に「+」演算子の使用が許される場合について、プログラム断片を例示しながら記述していく。

2.1 オブジェクト指向としてJavaを使うこと

Javaはオブジェクト指向プログラミングというパラダイムを持つため、ラッパークラスとメッセージングの利用が強く推奨される。プリミティブ型はデータ単体の変数であるため、メソッドを持っていない。即ち、プリミティブ型を使用して文字列の連結や加算を行う際は「+」演算子の使用が必要とされる。しかし、ラッパークラスにおいて、Integer, Double, Floatにはsum, Stringにはjoinといった値の加算や結合を行うメソッドが存在しているため、これを使用することで安直な「+」演算子の使用が回避できる。

2.2 プラス演算子は使わなくてもなんとかなる

Javaにはさまざまなメソッドが存在しており、AtomicInteger, .forEachや先述のsum, joinなどのメソッド使用することでプラス演算子を使用しないプログラムは十分に書くことができる。for文やwhile文を用いて手続き的な処理をしない限りプラス演算子を使用することはないため、事実上、Javaという言語でオブジェクト指向プログラミングを守る場合、使用しなくても良いものが「+演算子」である。

```
import java.util.concurrent.atomic.AtomicInteger;
public class Example {
    public static void main (String… args) {
        try {
            Integer num = Integer.valueOf(args[0]);
            AtomicInteger i = new AtomicInteger();
            while(i.get() < num) {
                System.out.println(i.getAndIncrement());
            }
        } catch (NumberFormatException e) {
            System.out.println(“数値以外の値が指定されました。”);
        } catch (ArrayIndexOutOfBoundsException e) {
            System.out.println(“引数が不足しています。”);
        }
    }
```

```
  }
  return;
```

図2.2.1 AtomicIntegerを使用したインクリメント

2.3 他の演算子の場合…

算術演算子においては「+」以外にもさまざまな演算子が存在するが、どれも、ラッパークラスを使用することで手続き的にそれらの演算子の使用を回避することができる。さらに、ラッパークラスはそれぞれのクラスオブジェクトのキャストやバイナリ変換、進数変換、比較なども行えるため、これらの昨日をふんだんに使用することはオトクである。

2.4 では、許される場合は何か。

例えば、解の公式や微分積分など、なるべく数式に似せた記述をした方が可読性が向上する場合、それらの式に関しては使用しても良いと個人的には考えている。しかし、素の状態のJavaでは分数の使用ができないため、数式の計算に特化したライブラリを探すか、自らで作り出すことで、手続き型プログラミングの要素を排除できるため、時と場合によってはこういった選択肢もあるのではないだろうか。

第3章 自由記述

3.1 別の課題で、揚げ足を取ってみたんですよ〜。なぁ〜にぃ〜？やっちまったなぁ！

実践webアプリケーションという科目がありますが、神山祭期間を挟む形で、2つの課題が出題されていました。そのうち1つの課題が、以下になります。

> 1 から n までの整数の和を返す関数 sum を作れ.
> ーfor 文を使う.
> ー和の公式を使わずに，ここでは1 から n までの値を足し合わせるように書く.

図3.1.1 実践webアプリケーションの課題

PHPを使用した課題でありながら色々ツッコミどころはありますが、特に注目した点は和の公式を使わずにという点です。真意については不明ですが、少し面白そうだったので、わざと捻くれて、「+」演算子を一切使用せずプログラムを作成してみました。それが、以下になります。

```php
<?php
/**
 * このファイルでは1からnまでの合計値を計算する関数sumを提示します。
 *
 * @copyright 744366 Noriki Kishi All Right Reserved
 * @license https://opensource.org/licenses/mit-license.html MIT License
 * @author Noriki Kishi <g1744366@cc.kyoto-su.ac.jp>
 */

/**
 * このクラスでは加算に関する計算を行います。
 */
class Example {

    /**
     * コンストラクタ
```

```
     */
    public function __construct() {}

    /**
     * このメソッドでは1から与えられた数値までの合計値を計算します。
     * @param int $n
     * @return int $result
    */
    public function sum(int $n) {
        $result = 0;
        for($i = 1; $i <= $n; $i = $this->add($i, 1)) {
            $result = $this->add($result, $i);
        }
        return $result;
    }

    /**
     * このメソッドでは与えられた2つの数値の合計値を計算します。
     *
     * 和の公式(プラス演算子の使用)が制限されているためビット演算で加算を行います。
     * @param int $a
     * @param int $b
     * @return int $a
    */
    private function add(int $a, int $b) {
        while($b != 0) {
            $c = ($a & $b) << 1;
            $a ^= $b;
            $b = $c;
        }
        return $a;
    }
}
```

```php
// debug
$example = new Example();
echo $example->sum(10); // 55

?>
```

図3.1.2 課題プログラム

　ここまできたらおふざけの域に入ってきますが、for文の使用が条件(おそらく再帰やforeachはNG)であるため、ビット演算で対応してみました。この課題を行ったことで、会得したことは以下です。

①プラス演算子を使わなくても手続き的な足し算ができる
②PHPはオブジェクト指向プログラミングでありながらラッパークラスやsumメソッドが存在しない(動的型付けであるため)

ただふざけて書いたつもりが、思いのほか勉強になったため、棚からぼたもちだなと思いました。

(追記)
これをSNSに投稿したらとあるAOの面接官をされてる先生からこんなプログラムを提案されました。その断片を例示します。

```php
public function sum(int $n) {
    for(;1 != 1;); // 指示なのでforを使っておく
    $s = $n;
    if($n > 0) $s += $s += sum($n - 1);
    return $s;
}
```

図3.1.3 某先生がボケてくださったプログラムの断片

　昨日はこれでゲラゲラ笑っていました。