# システムソフトウェア特論 課題#1

1931098 中島 涼

2020 年 2 月 17 日

## 1 演習 1-1 Java 言語入門

### 1.1 a. 入力された 2 つの数値の和を出力するアルゴリズムを変更し、減算, 乗算, 除算, 剰余を行え。

- 方針: println で出力する段階で演算子を + からそれぞれ対応するものに変更する。
- ソースコード及び実行結果: 減算, 乗算, 除算, 剰余の順にソースコードとその実行結果を以下に示す。

ソースコード 1　Sam11_a_sub.java

```
1  import java.util.*;
2  public class Sam11_a_sub {
3    public static void main(String[] args) {
4      Scanner sc = new Scanner(System.in);
5      System.out.print("s1> "); String s1 = sc.nextLine();
6      System.out.print("s2> "); String s2 = sc.nextLine();
7      double d1 = Double.parseDouble(s1);
8      double d2 = Double.parseDouble(s2);
9      System.out.println("sub: " + (d1-d2)); //+から-に変更
10   }
11 }
```

```
% java Sam11_a_sub
s1> 1.2
s2> 3.7
sub: -2.5
%
```

ソースコード 2　Sam11_a_mul.java

```
1  import java.util.*;
```

```
 2 public class Sam11_a_mul {
 3   public static void main(String[] args) {
 4     Scanner sc = new Scanner(System.in);
 5     System.out.print("s1> "); String s1 = sc.nextLine();
 6     System.out.print("s2> "); String s2 = sc.nextLine();
 7     double d1 = Double.parseDouble(s1);
 8     double d2 = Double.parseDouble(s2);
 9     System.out.println("mul: " + (d1*d2)); //+から*に変更
10   }
11 }
```

```
% java Sam11_a_mul
s1> 2.3
s2> 4.6
mul: 10.579999999999998
% java Sam11_a_mul
s1> 20.1
s2> 5.7
mul: 114.57000000000001
%
```

ソースコード 3　Sam11_a_div.java

```
 1 import java.util.*;
 2 public class Sam11_a_div {
 3   public static void main(String[] args) {
 4     Scanner sc = new Scanner(System.in);
 5     System.out.print("s1> "); String s1 = sc.nextLine();
 6     System.out.print("s2> "); String s2 = sc.nextLine();
 7     double d1 = Double.parseDouble(s1);
 8     double d2 = Double.parseDouble(s2);
 9     System.out.println("div: " + (d1/d2)); //+から/に変更
10   }
11 }
```

```
% java Sam11_a_div
s1> 1.2
s2> 0.4
sum: 2.9999999999999996
% java Sam11_a_div
s1> 4.5
s2> 9
sum: 0.5
%
```

ソースコード 4　Sam11_a_mod.java

```
 1 import java.util.*;
 2 public class Sam11_a_mod {
```

```
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("s1> "); String s1 = sc.nextLine();
    System.out.print("s2> "); String s2 = sc.nextLine();
    double d1 = Double.parseDouble(s1);
    double d2 = Double.parseDouble(s2);
    System.out.println("mod: " + (d1%d2)); //+から %に変更
  }
}
```

```
% java Sam11_a_mod
s1> 5
s2> 2
mod: 1.0
% java Sam11_a_mod
s1> 4.9
s2> 1.3
mod: 1.0000000000000002
```

## 1.2 b. 前述したアルゴリズムを実数のかわりに整数で計算するよう変更せよ。

- 方針: 上の例では標準入力から文字列を読み込み $s1, s2$ に格納したあとクラス Double のメソッド parseDouble() で double 型に変換している。この動作をかわりにクラス Integer の parseInt() で行う。
- ソースコード及び実行結果:

ソースコード 5　Sam11_b.java

```
import java.util.*;
public class Sam11_b {
  public static void main(String[] args) {
    Scanner sc = new Scanner(System.in);
    System.out.print("s1> "); String s1 = sc.nextLine();
    System.out.print("s2> "); String s2 = sc.nextLine();
    int d1 = Integer.parseInt(s1);
      //Double ではなく Integer のメソッドを呼び出し int 型変数に格納
    int d2 = Integer.parseInt(s2);
    System.out.println("sum: " + (d1+d2));
  }
}
```

```
% java Sam11_b
s1> 19
s2> 31
```

```
sum: 50
```

## 1.3 c. 同様に整数のかわりに長精度変数で計算するよう変更せよ。

- 方針: b と同様にクラス Long のメソッドで Long.parseLong() で変換を行う。
- ソースコード及び実行結果:

ソースコード 6 Sam11_c.java

```
1  import java.util.*;
2  public class Sam11_c {
3    public static void main(String[] args) {
4      Scanner sc = new Scanner(System.in);
5      System.out.print("s1> "); String s1 = sc.nextLine();
6      System.out.print("s2> "); String s2 = sc.nextLine();
7      long d1 = Long.parseLong(s1);
8      long d2 = Long.parseLong(s2);
9      System.out.println("sum: " + (d1+d2));
10   }
11 }
```

```
% java Sam11_b
s1> 1342674898
s2> 5784938237
Exception in thread "main" java.lang.NumberFormatException: For input string: "5784938237"
at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
at java.lang.Integer.parseInt(Integer.java:583)
at java.lang.Integer.parseInt(Integer.java:615)
at Sam11_b.main(Sam11_b.java:8)
% java Sam11_c
s1> 1342674898
s2> 5784938237
sum: 7127613135
%
```

## 1.4 考察

b の実行結果から double 型の計算は小数点以下込み 17 桁で打ち切られると予想できる。また、double 型の計算 (確認した限りでは乗算と除算) では少数点以下に誤差が出る場合があった。この対策として小数点以下を誤差なく計算するための BigDecimal というクラスが存在することを知った。

int 型の最大値は 2147483647 であり、入力の際これを超える値を入力すると計算が実行されないが計算の結果が 2147483647 が超えるような入力を行った場合、計算は行われ負の数が出力された。これは 2 進数の計算において桁溢れした最上位のビットが − の符号として処理されたためだ

と考えられる。

## 2　アンケート

Q1: プログラミングは得意ではないけれど好きです。持っている知識を増やし、その知識を組み合わせ少しづつプログラムを組んでいるときが特に楽しいと感じます。
Q2: 学習する側の立場としてはプログラミング言語が多いのは大変だと感じますが、それぞれに特徴や長所があり現在でも多くの種類の言語が使われていることには理由があると感じます。
Q3: 基礎的な課題でしたがいざやってみると知らない仕様等が意外と多く、基礎は蔑ろにできないと改めて感じました。