4.4 偶数の分解

それではゴールドバッハの予想を確認してみよう。これまでにプログラムの蓄積があるので、比較的作りやすいはずだ。[TwinPnum.java] を少し手直ししておこう。[TwinPnum.java] では $6m \pm 1$ の二つの数の素数判定をしたので、その部分を替えるだけで済むだろう。つまり、偶数を二つの奇数に分けたとき、それらの素数判定をするように組めばよいだけだ。

programming list [Goldbach.java]

```
 1: import java.util.Scanner;
 2:
 3: public class Goldbach {
 4:
 5:     public static void main(String[] args) {
 6:         Scanner s = new Scanner(System.in);
 7:
 8:         System.out.print("input an even number: ");
 9:         int m = s.nextInt();
10:
11:         System.out.print(m);
12:         for(int r = 3; r <= m / 2; r += 2) {
13:             boolean sgn = pchk(r) && pchk(m - r);
14:
15:             if(sgn == true)
16:                 System.out.print(" = " + r + " + " + (m - r));
17:         }
18:         System.out.println();
19:     }
20:
21:     public static boolean pchk(int n) {
22:         int i = 1;
23:         boolean flag = true;
24:
25:         while((i += 2) <= Math.sqrt(n)) {
26:             if(n % i > 0) {
27:                 ;
28:             } else {
29:                 flag = false;
30:                 break;
31:             }
32:         }
33:         return flag;
34:     }
35: }
```

プログラムは長ったらしいが新しいことはない。1:–9:行目はいつもの入力要求で、21:–34:行目までは [TwinPnum.java] と同じである。SUP をなくしたのは、ある偶数の入力に対して、素数の和に直すようなプログラムにしたからだ。上限 SUP を決めて、そこまでの偶数の分解の一覧を表示させるには、もう一つ for 構文を必要とする。それは、あとで君たちの課題になるだろう。結局のところ、[Goldbach.java] の本質は 11:–18:行目である。

そのようなわけで 9:行目で、候補となる偶数がテキストフィールドから取り込まれてる。11:行目でその数を表示させているが、これはもとの数が何であったか分かるようにしただけだ。不要と思えばこの行は要らない。

12:行目の for 構文で使われている変数 r は、m を二つの素数に分解するとき、一方の素数を代入するための変数になる。では、もう一方の素数を代入する変数は用意しなくていいの？　そう、その必要はない。なぜなら、もう一方の素数は m - r だからだ。プログラムでは不必要な変数は使わない方がよろしい。

13:行目は [TwinPnum.java] と同様の処理をしている。r と m - r が共に素数かどうか調べているのだ。共に素数のときに限り sgn が真となる。調べる範囲が r <= m / 2; であることに注意してほしい。

EX.　調べる範囲が r <= m / 2 で十分な理由は分かるね？

そして 15:行目で素数の和に分解されたとき、すなわち sgn に真が代入されたとき 16:行目で画面表示がされる。どのように表示されるか分かるだろうか。

プログラム [Goldbach.java] は、入力された偶数に対して、確実に素数の和に分解してくれるだろう。それも、すべてもれなく表示してくれる。ちょっとだけ困るのは、$4 = 2 + 2$ を示すことができない点だ。$4 = 2 + 2$ の表示は自明のことだから、かまわないと言えばかまわないのだけれど。

TRY!　6 以上のすべての偶数に対して、素数の和を表示するプログラムにせよ。とりあえず上限として SUP の値を 100 で試してみよ。

これで 6 以上の偶数の分解が **Java** 任せにできた。ところが 6 以上の偶数に対しては問題ないのだが、[Goldbach.java] には致命的な欠陥があるのだ。それは、うっかり奇数を入力してしまった場合に表面化する。奇数 m が入力されると、プログラムはこれを 3 以上の奇数 r と m - r に分けて

素数判定に回す。しかし m - r は偶数なのだ。

このときは大変困ったことになってしまう。というのは、pchk() メソッドは奇数を受け取ることしか想定してないからだ。pchk() は受け取った数を、3 から先の奇数で順次割り算を試して、どうにも割れないときに素数と判定する。ところが偶数を受け取った場合、3 から先の奇数で順次割り算を試しても、どうにも割れない数がある。たとえば 8 がそうだ。すなわち、偶数が素数と判定されてしまうのだ。

これを回避するには、奇数が入力されたとき偶数を入力するメッセージを出せばよい。これは if 構文を使えば容易にできるはずだ。

TRY! if 構文を追加して、奇数の入力にも耐えられるプログラムにせよ。

TRY! これで奇数入力の危機は去った。しかし、プログラムは 6 以上の偶数を入力しないと正しく機能しないのだ。2 や 4 が入力されては困る。それを回避するにはどのようなコードを書くべきか考えてみよ。