この実装方法ではcoefficients[p] = k、exponents[p] = mとするとp番目の項はkx^mになり、前述の方法での問題点は解決できていますが、非常に煩雑でもあります。1つの多項式に対して2つの配列を追う必要があります。もし2つの配列が違う長さであれば、「未定義」の値を含んでいることにもなります。また、2つの配列を返す必要があるので多項式のデータを返す場合にも困ります。

```
1  ??? sum(double[] coeffs1, double[] expon1,
2          double[] coeffs2, double[] expon2) {
3     ...
4  }
```

良い実装

この問題の良い実装は、多項式を独自のデータ構造で設計することです。

```
1  class PolyTerm {
2     double coefficient;
3     double exponent;
4  }
5
6  PolyTerm[] sum(PolyTerm[] poly1, PolyTerm[] poly) {
7     ...
8  }
```

これは「過度な最適化」という議論があるかもしれません。そうかもしれませんし、違うかもしれませんが、過度な最適化かどうかにかかわらず、上記のコードはあなたがどのようにコードの設計をするかについて考え、できる限り最速な方法を考えるあまり見落としてしまった部分がないことをアピールすることになります。

適切なコードの再利用

2進数の値と16進数の値がいずれも文字列として与えられたとき、それらが等しいかどうかをチェックする関数を書くという問題を出された場合を考えてみましょう。

この問題のエレガントな実装は、コードを再利用することです。

```
1  public boolean compareBinToHex(String binary, String hex) {
2     int n1 = convertToBase(binary, 2);
3     int n2 = convertToBase(hex, 16);
4     if (n1 < 0 || n2 < 0) {
5        return false;
6     } else {
7        return n1 == n2;
8     }
9  }
10
11 public int digitToValue(char c) {
12    if (c >= '0' && c <= '9') return c - '0';
13    else if (c >= 'A' && c <= 'F') return 10 + c - 'A';
14    else if (c >= 'a' && c <= 'f') return 10 + c - 'a';
15    return -1;
16 }
17
18 public int convertToBase(String number, int base) {
19    if (base < 2 || (base > 10 && base != 16)) return -1;
20    int value = 0;
21    for (int i = number.length() - 1; i >= 0; i--) {
```