PHPで雑に通貨クラスを実装してみた結果 🤑

I implemented a **Money** class, and I'm sleepy.

pixiv

2018-02-23 ピクシブ社内勉強会

お前誰よ

- うさみけんた（@tadsan）/ Zonu.EXE
  - GitHub/Packagistでは id: zonuexe
- ピクシブ株式会社技術基盤チーム（pixiv.net）
- Emacs Lisper, PHPer
  - Emacs PHP Modeのメンテナ引き継ぎました
  - 好きなリスプはEmacs Lispです
- Qiitaに記事を書いたり変なコメントしてるよ

pixiv

伏線

void
@tadsan

PHPで、たかだか小数点以下2桁程度の精度があれば十分の金額計算をするときにどうしますか。

Translate from Japanese

58% int/floatで保持して組み込み演算子で計算

2% stringで保持してBC Math関数で計算

25% 通貨計算用のライブラリを利用する

15% 通貨計算用クラスを自作する

48 votes・5 days left

7:20 PM - 21 Feb 2018

日本で流通する最低
貨幣は一円硬貨

USで流通する最低
貨幣は一セント硬貨

セントは1/100ドル

$0.01と表記される

コンピュータで扱える数

PHPでは型が

3つある

整数型（int）

- PHPの整数は signed int
- ビット数はPHP_INT_SIZE
  最大値はPHP_INT_MAXで定義
- 概念上、この式で最大値が得られる
  `2 ** (PHP_INT_SIZE * 8 - 1) - 1`

浮動小数点数型（float）

- PHPの整数はdouble
（IEEE 754 倍精度フォーマット）
- 型名としては(real)の別名もあるが、別に実数ではない

とても
大事なこと

PHP_INT_MAX

9223372036854775807

PHP INT MAX + 1

どうなる？

```
var_dump(PHP_INT_MAX + 1);
//=> float(9.2233720368548E+18)
```

```php
$bytes = PHP_INT_SIZE * 8;
var_dump(2**($bytes-1)-1);
//=> float(9.2233720368548E+18)
```

突然のfloat

そうです

PHPの演算子は整数演算でPHP_INT_MAXを超えるとfloatになります

もひとつ
大事なこと

浮動小数点は
「ぴったり」の数
を表現できない

```
>>> sprintf('%.53f', 1.0)
=> "1.00000000000000000000000000000000000000000000000000000"
>>> sprintf('%.53f', 0.9)
=> "0.90000000000000022204460492503130808472633361816406250"
>>>
```

エラー（誤差） [ソースを編集]

「誤差#計算誤差の種類」および「数値解析#誤差の発生と伝播」も参照

オーバーフロー／アンダーフロー

演算結果が指数部で表現できる範囲を超える場合があるが、最大値を超えた場合は**オーバーフロー**、絶対値の最小より小さい場合は**アンダーフロー**という。IEEE 754の場合、アンダーフローは、まず結果が非正規化数となり精度が低下し、さらに進むと結果が0になる。

桁落ち

絶対値がほぼ等しい異符号の数値同士の加算後や、同符号でほぼ等しい数値同士の減算の後、正規化で有効数字が減少すること。詳細は桁落ちを参照。

情報落ち

浮動小数点数値を加減算するときはまず指数を揃えなければならない。絶対値の非常に小さな値と絶対値の非常に大きな値との浮動小数点数の加減算を行うときは、まず指数を絶対値の大きい方に揃える桁合わせを行ない、それから加減算を行なう。そのため絶対値の小さな値は仮数部が右側に多くシフトされて下位の桁の部分が反映されずに結果の値が丸められて情報が欠落する。情報欠落ともいう。詳細は情報落ちを参照。

積み残し

情報落ちが繰り返し起こる場合を言う。たとえば $\sum_{n=0}^{100} \frac{1}{1.5^n}$ を n=0 の初項からn=100に向かって順番に加えて計算しようとすると、ある項から先で情報落ちが起こり、それ以降の項は無視されてしまうことになる。これを積み残しと呼ぶ。値の小さい項から大きい項に向かって加算をする（この例では逆順に加算をする）ことで対処できる場合もある（必ずしもそうすれば全て完全にうまくいくとは限らない）。

丸め誤差

仮数部の桁数が有限であるため、収まらない部分の最上位桁で四捨五入（2進法では0捨1入）して仮数部の桁数に丸めることによる誤差。

https://ja.wikipedia.org/wiki/浮動小数点数#エラー（誤差）

浮動小数点数は
概算で十分な
計算に向く

誤差を許容
できない用途には
使ってはならない

われわれが欲するのは
誤差が生じずに小数点
以下を表現できる数

颯爽と現れる

BC Math 関数

BC Math 関数

目次

http://php.net/manual/ja/ref.bc.php

bcadd

(PHP 4, PHP 5, PHP 7)
bcadd — 2つの任意精度の数値を加算する

説明

```
string bcadd ( string $left_operand , string $right_operand [, int $scale = 0 ] )
```

left_operand を **right_operand** に加算します。

http://php.net/manual/ja/function.bcadd.php

突然のstring

冷静になって
ほしい

文字列で入出力を表現すれば桁は落ちない

$scaleオプションで任意の精度を指定できる

裸の文字列のまま

取り回したくない…

クラスを作ろう

zonuexe / kane.php

Unwatch 1

Code Issues 0 Pull requests 0 Projects 0 Wiki Insights Settings

Idol time is Money.

README.md

🤑 kane.php 🤑

PHPで金額計算をするためのクラスです。

> Idol time is money!!

メイクマネー

```php
$十円 = Kane\JPY(10);
$二十円 = Kane\JPY(20);

echo $十円->add($二十円), PHP_EOL;
// "30.0000000000"
```

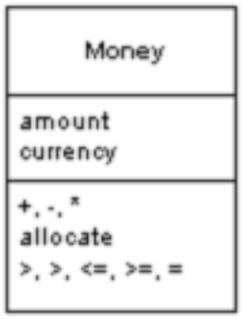
https://github.com/zonuexe/kane.php

定石がある

Money

Represents a monetary value.

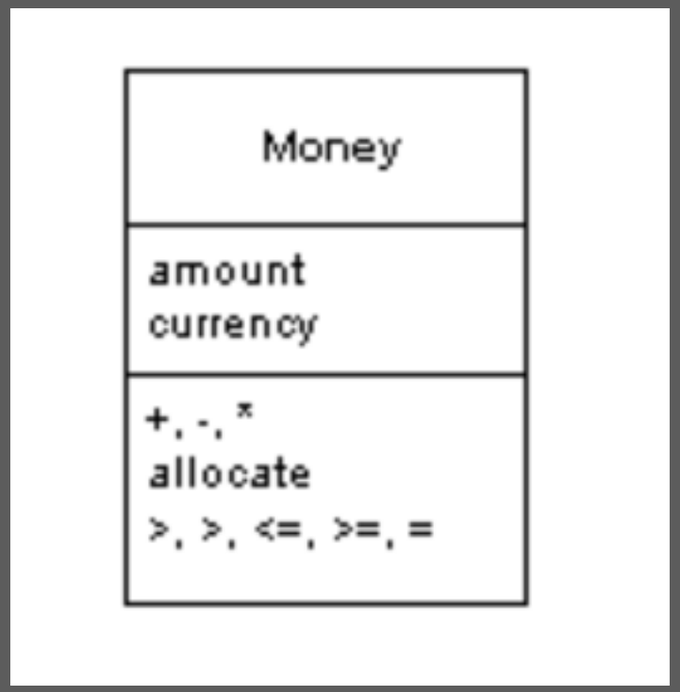
For a full description see P of EAA page **488**



A large proportion of the computers in this world manipulate money, so it's always puzzled me that money isn't actually a first class data type in any mainstream programming language. The lack of a type causes problems, the most obvious surrounding currencies. If all your calculations are done in a single currency, this isn't a huge problem, but once you involve multiple currencies you want to avoid adding your dollars to your yen without taking the currency differences into account. The more subtle problem is with rounding. Monetary calculations are often rounded to the smallest currency unit. When you do this it's easy to lose pennies (or your local equivalent) because of rounding errors.

The good thing about object-oriented programming is that you can fix these problems by creating a Money class that handles them. Of course, it's still surprising that none of the mainstream base class libraries actually do this.

https://martinfowler.com/eaaCatalog/money.html



https://martinfowler.com/eaaCatalog/money.html

よつしや

やってるで

Moneyは
Currency(通貨)と
量(数)の組である

```php
class Money
{
    use MoneyCalculator;

    /** @var string */
    private $amount;

    /** @var Currency */
    private $currency;

    /** @var int */
    private $scale;

    public function __construct(string $amount, Currency $currency, int $scale)
    {
        $this->amount = $amount;
        $this->currency = $currency;
        $this->scale  = $scale;
    }
```

数字から
金を作る

```php
function JPY(string $amount, int $scale = 10): Money
{
    return new Money($amount, Currency\JPY::getInstance(), $scale);
}
```

+, −, ×

どうやって実装

まじめに書くと
意外にめんどい

キラツと

ひらめいた

計算する処理は
MoneyCalculator
トレイトに分割

```php
/**
 * @param \Kane\Money $object
 * @return static
 * @throws \Kane\Currency\DifferenceException
 * @throws \Kane\Currency\ScaleMismatchException
 */
public function add(Money ...$object)
{
    return static::eval('+', $this, ...$object);
}
```

```php
public static function eval(...$expr)
{
    $operator = array_shift($expr);
    $operands = [];

    foreach ($expr as $obj) {
        $operands[] = is_array($obj) ? self::eval(...$obj) : $obj;
    }

    return self::eval1($operator, ...$operands);
}
```

```php
private static function eval1(string $operator, Money ...$operands)
{
    self::assertOperands(...$operands);

    $amount = null;

    switch ($operator) {
        case '+':
            $amount = self::bcadd($operands[0]->scale, ...$operands);
            break;
        case '-':
            $amount = self::bcsub($operands[0]->scale, ...$operands);
            break;
        case '*':
            $amount = self::bcmul($operands[0]->scale, ...$operands);
            break;
    }

    return new static($amount, $operands[0]->currency, $operands[0]->scale);
}
```

```php
private static function bcadd(int $scale, string ...$operands)
{
    switch (count($operands)) {
        case 0:
            throw new \BadMethodCallException();
        case 1:
            return $operands[0];
    }

    $v1 = array_shift($operands);
    $v2 = array_shift($operands);

    array_unshift($operands, bcadd($v1, $v2, $scale));

    return self::bcadd($scale, ...$operands);
}
```

ある程度の式は
書けるように

二項演算を採用しなかったので優先順位の評価は一切不要

副作用として
複数引数の処理
が簡単になった

void
@tadsan

通貨型を実装してたと思ったらLispを実装してた...

Translate from Japanese

12:20 AM - 23 Feb 2018

1 Retweet **2** Likes

1 1 2

void @tadsan · 18h
これをLispだと主張しておく

Translate from Japanese

```
 (string)通貨\JPY(10)->add(通貨\JPY(20))
"30"
 (string)通貨\Money::eval(['+', 通貨\JPY(10), 通貨\JPY(20)]);
"30"
 (string)通貨\Money::eval(['+', 通貨\JPY(10), ['+', 通貨\JPY(10), 通貨\JPY(20)]]);
"40"
```

通貨計算では List をProcessing する！

まとめ

雑に作った

実用性はない

MoneyPHPを
使ったほうがいい

Money for PHP

This library intends to provide tools for storing and using monetary values in an easy, yet powerful way.

Why a Money library for PHP?

Also see http://blog.verraes.net/2011/04/fowler-money-pattern-in-php/

This is a PHP implementation of the Money pattern, as described in [Fowler2002] :

> A large proportion of the computers in this world manipulate money, so it's always puzzled me that money isn't actually a first class data type in any mainstream programming language. The lack of a type causes problems, the most obvious surrounding currencies. If all your calculations are done in a single currency, this isn't a huge problem, but once you involve multiple currencies you want to avoid adding your dollars to your yen without taking the currency differences into account. The more subtle problem is with rounding. Monetary calculations are often rounded to the smallest currency unit. When you do this it's easy to lose pennies (or your local equivalent) because of rounding errors.

http://moneyphp.org/en/latest/

PHPで雑に通貨クラスを実装してみた結果 🤑

I implemented a **Money** class, and I'm sleepy.

pixiv

2018-02-23 ピクシブ社内勉強会

LISt と Processing
してた