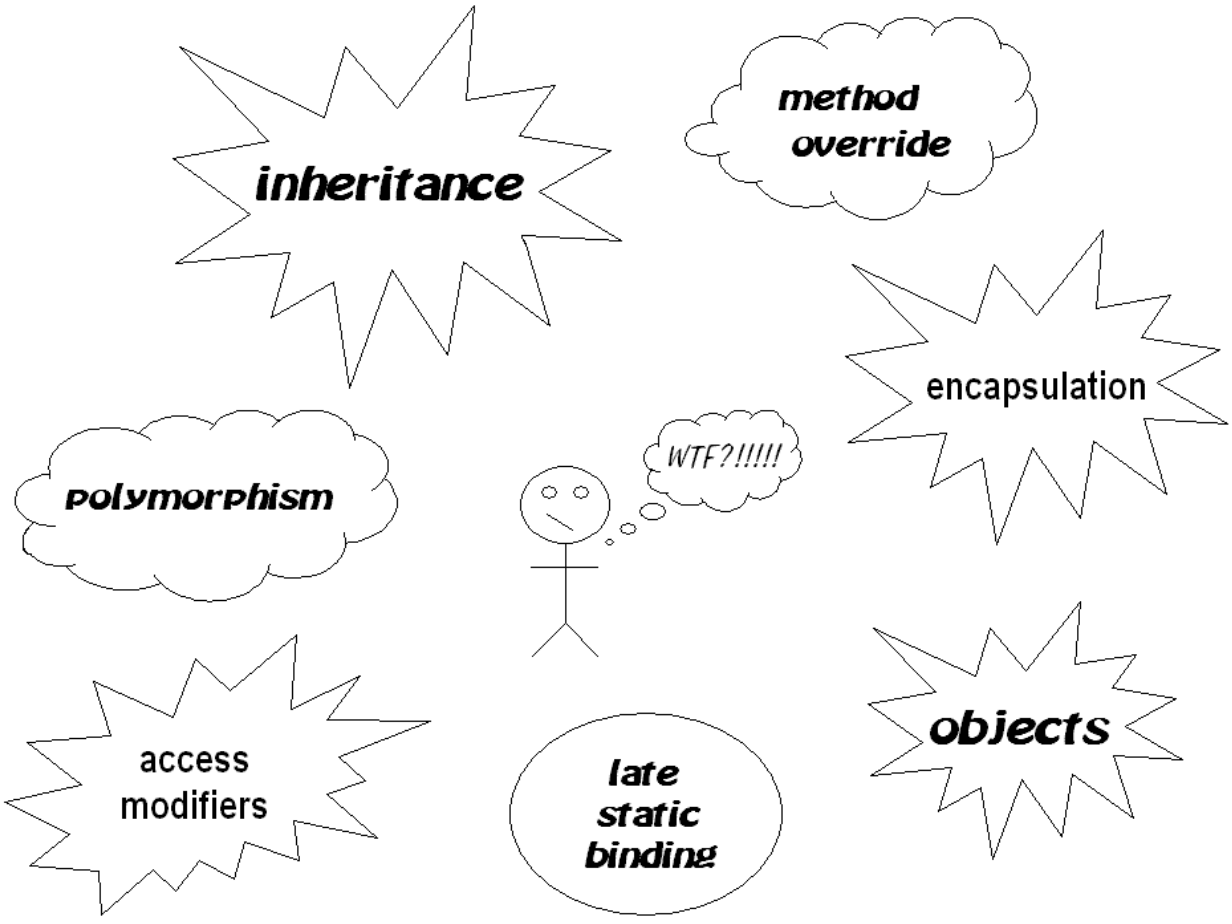
PHP はこれ！

オブジェクト指向ＰＨＰ
を初めて学ぶ人に



MICHELLE GOSNEY

ＰＨＰはこれ！．　オブジェクト指向ＰＨＰを初めて学ぶ人に、

ISBN-13 （電子媒体）：978-1-456-61529-1

目次

はじめに **7**

第１章：なぜこの本なのか？ **8**

初級 PHP 開発者　求人内容 8

なぜオブジェクト指向 PHP を学ぶのか 10

開発環境を設定する 11

Zend Server CE(Community Edition)のインストール 13

MySQL Workbench 5.2 CE 15

統合開発環境の設定 16

第２章　PHP オブジェクトとクラス **19**

最初の困惑 19

PHP オブジェクトとクラス 20

インスタンシエーション 22

$this 変数 22

アクセス・モディファイアー 23

インヘリタンス 25

メソッドのオーバーライド 25

親のメソッドの呼出し 26

水平インヘリタンス　―　trait の使用 27

カプセル化(encapsulation) 27

ポリモルフィズム(Polymorphism) 28

メソッドの多重定義(Method Overloading) 29

メソッドのオーバーライド 29

実行時結合(Late Binding)（ダイナミック結合） 30

第３章　マジック・メソッド **32**

__construct() 32

親のコンストラクタとデストラクタを使う 33

__destruct() 33

__get(), __set(), __isset() 33

“User” の例１ 34

“User” の例２ 35

__call() 36

__sleep() 37

__wakeup() 38
__clone() 38
第４章　抽象的クラスとメソッド 40
abstruct のキィーワード 40
抽象メソッド 42
final キィーワード 43
第５章　PHP インタフェース 44
キィーワード interface と implements 44
インタフェイスに向けたプログラミング 45
第６章　Static メソッドとプロパティ 52
Static プロパティ 53
static メソッド 56
シングルトン設計パターン 60
実行時 static 結合 61
第 7 章　PHP のエラー制御と例外の取り扱い 64
組み込み例外クラス 64
例外の発生 65
エラーへの感度のレベルを適度に設定する 67
第 8 章　モデルービュウーコントローラ 77
モデルービュウーコントローラによる設計方式 77
モデル 77
ビュウ 77
コントローラ 77
MVC URL 構造と URL マッピング 77
index.php ファイル 78
MVC フォルダーの構造 79
カスタム MVC 設計　－　レストラン・メニュー管理応用 83
メニューを見せる 83
新メニュー品目を追加する機能の開発 91
メニュー品目追加機能のためのコントローラ 92
メニュー品目をメニューに割り当てる機能のためのモデル 98
メニュー品目の編集あるいは削除 100
レストラン・メニュー管理応用ソース・コードのダウンロード 104
第 9 章　リフレクション API 106

リフレクション・クラス 106
第 10 章　PEAR ライブラリ 113
PEAR の概要 113
Windows Zend Framework Environment に PEAR をインストールする 113
Windows XAMPP Environment に PEAR をインストールする 115
Windows WAMP Environment に PEAR をインストールする 116
Mac OS に PEAR をインストールする 116
Linux に PEAR をインストールする 117
PEAR を使う- サンプル 117
第 11 章　PHPUnit 122
PHPUnit でするユニット・テスト 122
PHPUnit をインストールする 122
Windows Zend Framework Environment で PHPUnit をインストールする 123
Windows XAMPP Environment で PHPUnit をインストールする 123
Mac OS X で PHPUnit をインストールする 123
Linux で PHPUnit をインストールする 124
テスト固定具 124
テストのフェーズ 124
テストのケースを書く 124
判定メソッド 125
HTML のフォームから提出されたデータをテストする 130
テスト組 131
模擬と控え 131
模擬クラスを生成する 132
プラグインなしで Eclipse から PHPUnit を走らす 135
第 12 章　Subversion を用いたソース・コードの管理 137
Windows での Subversion 138
Windows でサーバー用、クライアント用の Subversion を立ち上げる 138
自分のソースコードを初めて入れる。 141
作業するためのコピーを生成する 141
更新とワーキング・コピーの送出し 141
TortoiseSVN の立ち上げ 141
提出したものの輸出 147

ファイルのタグと枝分け 148
Mac OS X 148
Mac OS X でサーバー用、クライアント用の Subversion を立ち上げる 148
Subversion リポジトリの立ち上げ 149
Subversion でプロジェクトを生成する 149
Subversion からファイルを取り出す 150
更新と変化の送り出し 150
ファイルとディレクトリの追加と削除 151
リリースのタグ付けと輸出 152
プロジェクトの枝分けと統合 153
グラフィカル・ユーザ・インタフェイス 154
Linux 155
Subversion のサーバーをライナックスのサーバーに立ち上げる 155
更新と変更のコミット 158
ファイルとディレクトリの追加と削除 159
リリースのタグ付けと輸出 160
プロジェクトの枝分けと統合 161
まとめ 163
付録Ａ 164
参考文献 164
ウエブ上のリソース 164
書籍 164
インデックス 165

はじめに

二、三年前 PHP5.3 の新しい機能を調べようとしたとき、長く勉強していなかったことのつけを感じた。わたしが PHP を使っていたのは 9 年前のことだった。その当時は使いやすくて、型指定の緩いスクリプト言語だった。PHP5.3 の新機能を調べていてまず感じたのは、「ほう、あのこじんまりした PHP がずいぶん成長したな」ということだった。

急いで追いつこうとして、何冊かの本を読み、オンライン・フォーラムを覗いて直ぐに、それらの新機能の説明がいずれも同じように退屈で単調でさえない例が使われているのにいらだちを覚えた。私はオブジェクト指向プログラミングに人並み以上の経歴を持ち合わせているので、「もし誰かがオブジェクト指向にあまり経験がないのにこれらの概念を学ぼうとしているならどうなんだろうか？」と自問し続けていた。

この本の目的は、オブジェクト指向 PHP の最新の機能と特徴を、単純に理解しやすく説明し、後で君が学んだことを思い出し使えるようにしたいということだった。

もし、君が就職の面接の準備をしていて、 PHP 5+ の最新機能についての質問が出そうだというのであれば、この本はとても役立つだろう。実際、君は就職の面接でこの本で取り上げている肝心な概念のほとんどを思い出せると確信が持てるだろう。さらに面接で問われそうなその他のいくつかの機能も含めるようにした。

第１章：なぜこの本なのか？

この本はオブジェクト指向 PHP あるいは多分オブジェクト指向プログラミングそのものを知らないので新しい概念の理解に困難を感じているプログラマーのために書かれている。この本を読むべき理由はいくつかあるが、まず次の仕事内容の記述から始めよう。この本は君が PHP に使われているオブジェクト指向の概念を理解し思い出すのに助けとなり、さらに次のステップの学習へと進むのにしっかりした基礎を与えてくれるはずだ。次のような仕事内容の就職面接の場に臨んだとき、何のことを話しているのか理解し、そして面接の相手が出すだろうと一番に予想される PHP についての基本的な質問にどう答えるべきか分かるだろう。

初級 PHP 開発者　求人内容

求人内容

テキサス州アービングで２名の正社員の求人。１名は上級 PHP 開発者、もう１名は中級/初級 PHP 開発者。給料は８万 5000 ドル - 12 万 5000 ドル。

職場は自由に拡張可能な e-コマース・ウェブサイトの開発を行う快適な環境。

待遇

＊Solr,PHP5.3, Gearman, Varrish, PHPUnit, Phing, Hudson, Doctrine 2, JQuery, MySQL など多くのオープン・ソース・ソフトで開発。

＊医療、歯科、眼科保険

＊有給休暇, ２週間　＊休日、休憩時間

＊社員割引制度

＊在宅勤務制度

必須能力

＊PHP ５DOP 十分に使いこなすこと

＊PHP 5.3, Javascript, CSS, JQuery

＊LAMP サーバー管理

＊ソース・コード管理　GUT, SVN

＊E_NOTICE error を ON にしたコード

有利な条件

＊Solr サーチ

＊Magento eCommerce Platform

＊Unit テスト

＊Zend Framework, Magento, Doctrine 2

＊高トラフィックでのアーキテクチャー

図 1-1. *初級 PHP 開発者　求人内容*

この本は Extreme Programming, Design-by-Contract, Test Driven Development, Coding-to-the-Interface そして Agile Development といった高度な開発方法を可能にするオブジェクト指向 PHP の高度な機能を中心に述べている。この短い本を読むにつれ多くの簡単な例が核となる概念とそれがどのように機能するかを示してくれる。小さなデータベースを基にした応用と PHPUnit を用いたテスト用ツールを構築し、ソース・コードの管理のため Subversion(SVN) でバージョン管理も行う。

先の求人内容に有った JQuery やその他の Javascript ライブラリについては述べない。JQuery の良い本を手に入れてほしい。Javascript は１０年前にはブラウザーと連携したクライアント・サイドの開発用ツールとして実用的ではなかったが、今では重要な地位を獲得し、今後何年間も妥当なツールとして残るだろう。JQuery の本を探すときには、JQuery の機能は基本的に８つのカテゴリーに分けられることを覚えておくとよい。それらは Core 機能、DOM 選択と横行(traversal)、DOM 操作と CSS, Events, 効果とアニメーション、AJAX, ユーザ・インターフェイスと拡張性である。これらのカテゴリーの各々を包括的にカバーしている本が一番良い。

なぜオブジェクト指向 PHP を学ぶのか

オブジェクト指向プログラミングを学ぶべき理由がいくつかある。その６つの主要な利点を紹介しよう。

何よりも第１に、PHP であろうと他のオブジェクト指向ベースの言語であろうと、オブジェクト指向プログラミングを学ぶことで君のプログラマーとしての価値を上げ、より高い給料を得、より自由に勤務先を選択できることになる。

第２に、オブジェクト指向の原理はすべてのオブジェクト指向言語とプラットフォームに通用する。一旦オブジェクト指向 PHP を学べば、他の言語を理解するのは容易だ。それが C#, Java, Visual Basic, Ruby on Rails であれ C++ であれ知らなければならないことのほとんどをすでに知っていることになる。シンタックスとセマンティクスは言語によって少しづつ異なるかも知れないが、それらが動作する方法は本質的に同じなのだ。

オブジェクト指向プログラミングを学ぶ第３の利点は拡張性だ。小さなプロジェクトで簡単なコードでも同じ目的を達成できるときに、オブジェクト指向の PHP を使うのは冗長でやりすぎと思えるかもしれない。しかし、その小さな応用がどんどん成長して、複雑になってくるとすると、オブジェクト指向 PHP で基本のコードを書いておくのに時間を費やしてもそれが良い投資であることが証明されるだろう。

第４の利点は、オブジェクト指向の PHP は再利用可能なコードのブロックを作ることになり、今のプロジェクト、あるいは将来のプロジェクトで再利用できるということだ。君は再利用可能で頑健なコードのライブラリを蓄積できるはずだ。

オブジェクト指向の PHP を使う第５の利点は、複数のプログラマーで同じプロジェクトあるいはコード・ベ―スのために働くのがずっと簡単だということだ。というのは全てがオブジェクトに分割されているからだ。さまざまなプログラマーが同じプロジェクトの他の部分のコードを変更したり壊したりする心配なく別々のオブジェクトを担当できる。それぞれのオブジェクトは最初から正しく設計されているという考え方なのだ。オブジェクト指向 PHP にこの特徴があるため、SCRUM/Agile 環境での PHP 開発が可能であり、また SVN や PHPUnit　のような頑健なツールが使えることになる。チーム環境でこれらのツールの利用方法を学ぶことは、君のプログラマーとしての価値を高めてくれることは確かだ。

オブジェクト指向の PHP を学ぶ第６の利点は、Zend のような PHP のフレームワークや PEAR のようなライブラリのオブジェクト指向のリソースを活用できるようになるということだ。これらのリソースを効果的に活用

することを学べば、優れたアプリケーションをより速く、より効果的に構築できるようになる。このことは君がフリーランスの開発者になろうというときに、頑健なアプリケーションをより速く制作し、複数のプロジェクトを簡単にこなせるわけで、より多くの収入を得られるということだ。

開発環境を設定する

何を差し置いてもまず最初に信頼できる開発環境を設定しよう。それぞれのマシンと環境が同じではないため、開発環境の構築はしばしば面倒なことになりかねない。だからここで君がこの本のサンプル・コードを試してみるのにきちんと動作してくれる場所をできるだけ簡単にしかも問題が起こらないように作れるよう教えよう。悩みの種は少なければ少ないほどよい。この指示に従えばこの本で教えることのすべてを全く問題なく学べる。

まだ開発環境がないなら、この節は AMP スタックを君のコンピュータに設定する手順を導いてくれる。AMP というのは、まだ知らないかもしれないが、AMP は PHP t と MySQL のアプリケーションを実行するためのソフトウエア・バンドルである。

AMP は次の略称である：

A: Apache 2 + HTTP ウェブ・サーバー

M: MySQL 5.x データベース・エンジン

P: PHP 5.x

もしこのセットをライナックスに載せるのならそれは **LAMP** として知られ、ウインドウズに載せるなら、**WAMP**、マックに載せるなら **MAMP**、クロス・プラットフォーム用なら **XAMP** として知られている。

XAMP は次の略称である：

X: クロス・プラットフォーム

A: Apache 2 + HTTP ウェブ・サーバー

M: MySQL 5.x データベース・エンジン

P: PHP 5.x

これから XAMP セットを Zend Server Community Edition でウインドウズ上にロードする手順を紹介する。**もし君がライナックス(LAMP) かマック(MAMP) のユーザなら、Zend Server CE (PHP 5.3) ダウンロードを探すとよい。**このダウンロードには Apache ウェブ・サーバー 7 、MySQL データベース・エンジンそして PHP 5.3 ライブラリが含まれている。ダウンロードは次のアドレスで可能だ：

http://www.zend.com/products/server-ce/downloads

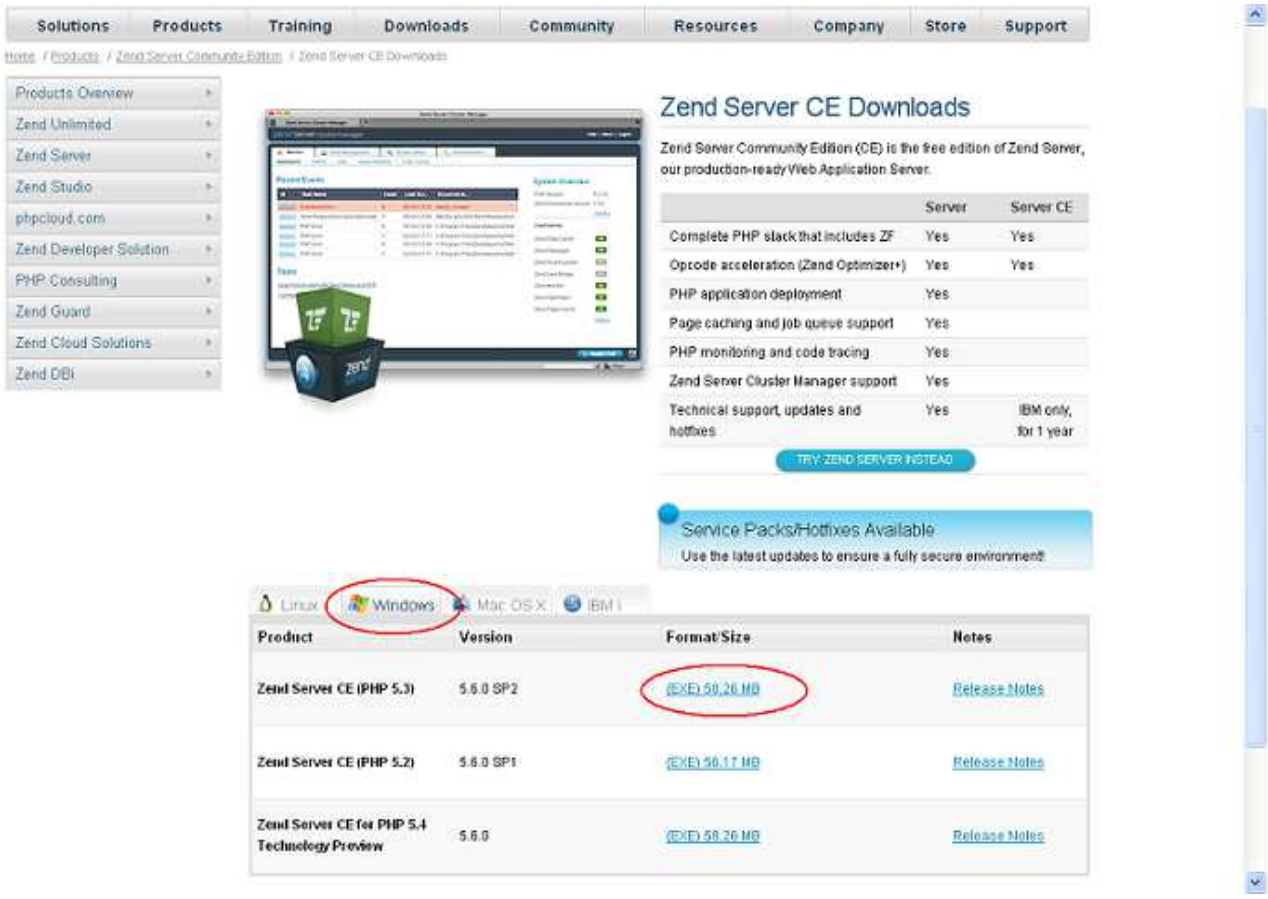


図　1-2.　*Zend Server CE ダウンロード・ページ*

同じく詳細な利用手引きは次にある：

http://www.zend.com/products/server/resources この中の section 3 には次のシステム要件が載っている。

3. システム要件

* サポートしているＯＳ、プラットフォームとＯＳのバージョン：

-Linux x86 と x86-64:

-RHEL 6.x, 5.x, CentOS5.x, 6.x, OEL5.x と Fedora 13 及びそれ以降―RPM パッケージからのもの

-SLES 10.x 11.x と OpenSUSE 11.x, 12.x ―RPM パッケージからのもの

-Debian GNU/Linux 6.0/5,x, Ubuntu Linux 10.04 とそれ以降―DEB パッケージからのもの

−Windows x86 と x86-64：

−Windows XP SP 2 とそれ以降

-Windows Vista (スターター・エディションを除く)

-Windows Server 2003

-Windows Server 2008

-Windows 7

-Mac

-Apple Mac OS X 10.6, 10.7

Zend Server CE(Community Edition)のインストール

Zend Server をダウンロードした後、インストーラーを実行する。

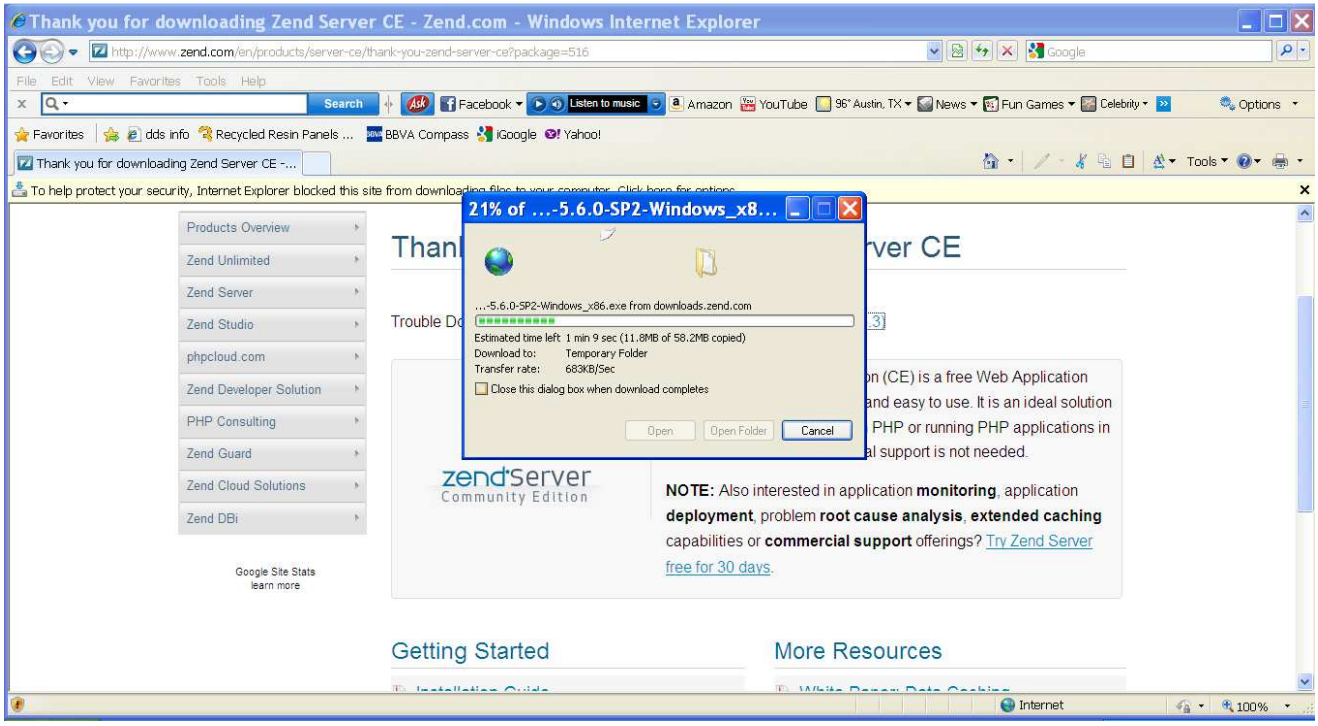


図 1-3.　*Zend Server CE インストーラー*

Apache のポート番号を問われたら次のように入力する。

Web Server Port: 80

Zend Server Interface Port: 10081

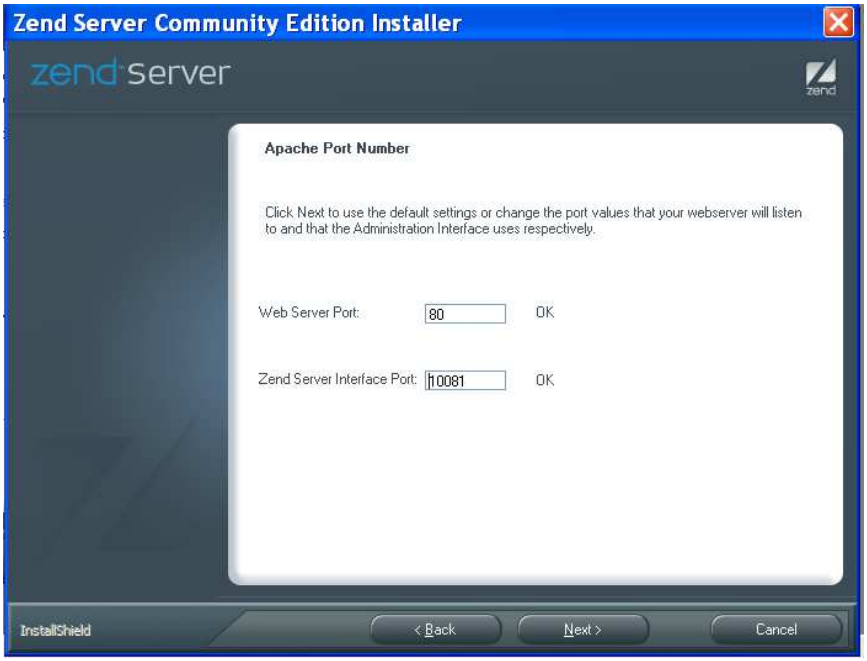


図 1-4. *Apache ポート番号*

“Next”をクリックする。”Custom Install”を選択し、次のスクリーン・ショットのようにチェックを入れるか外すかする。

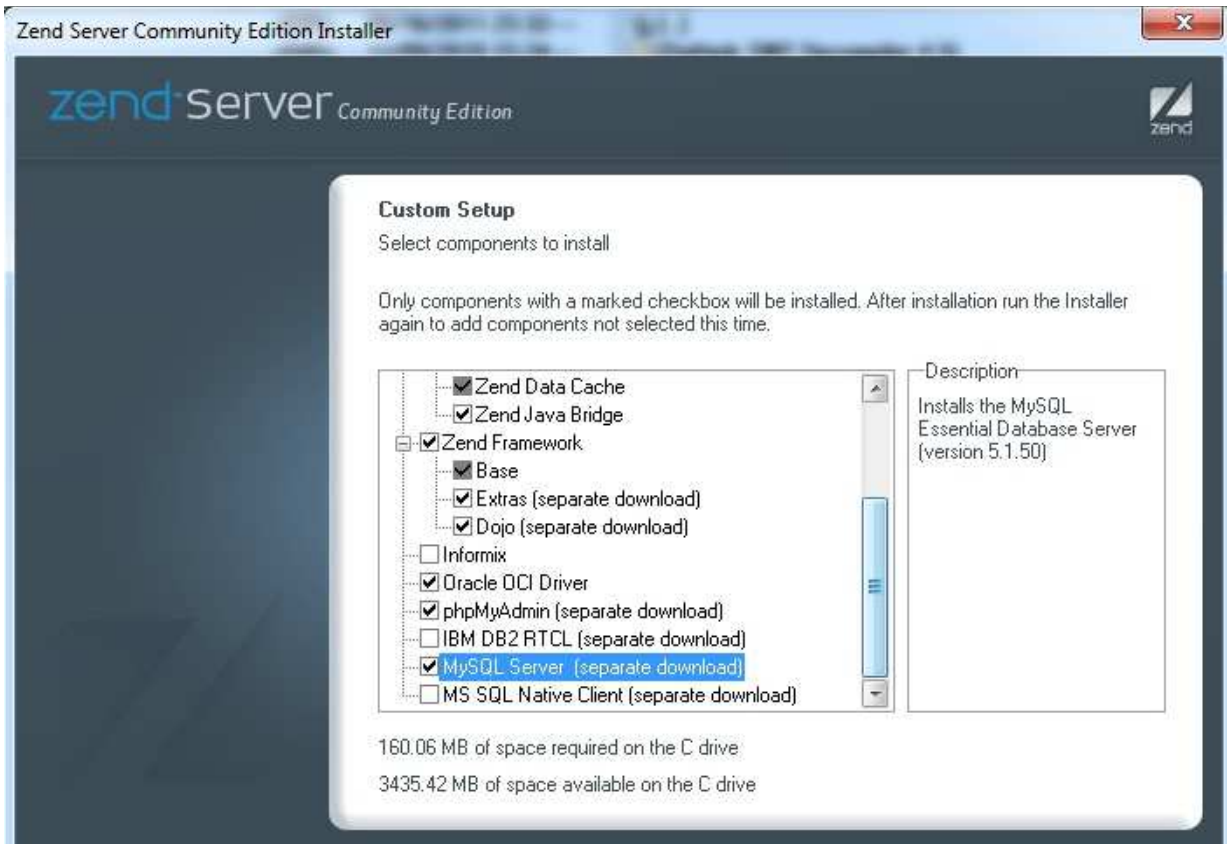


図 1-5. *Zend Server CE カスタム・セットアップ*

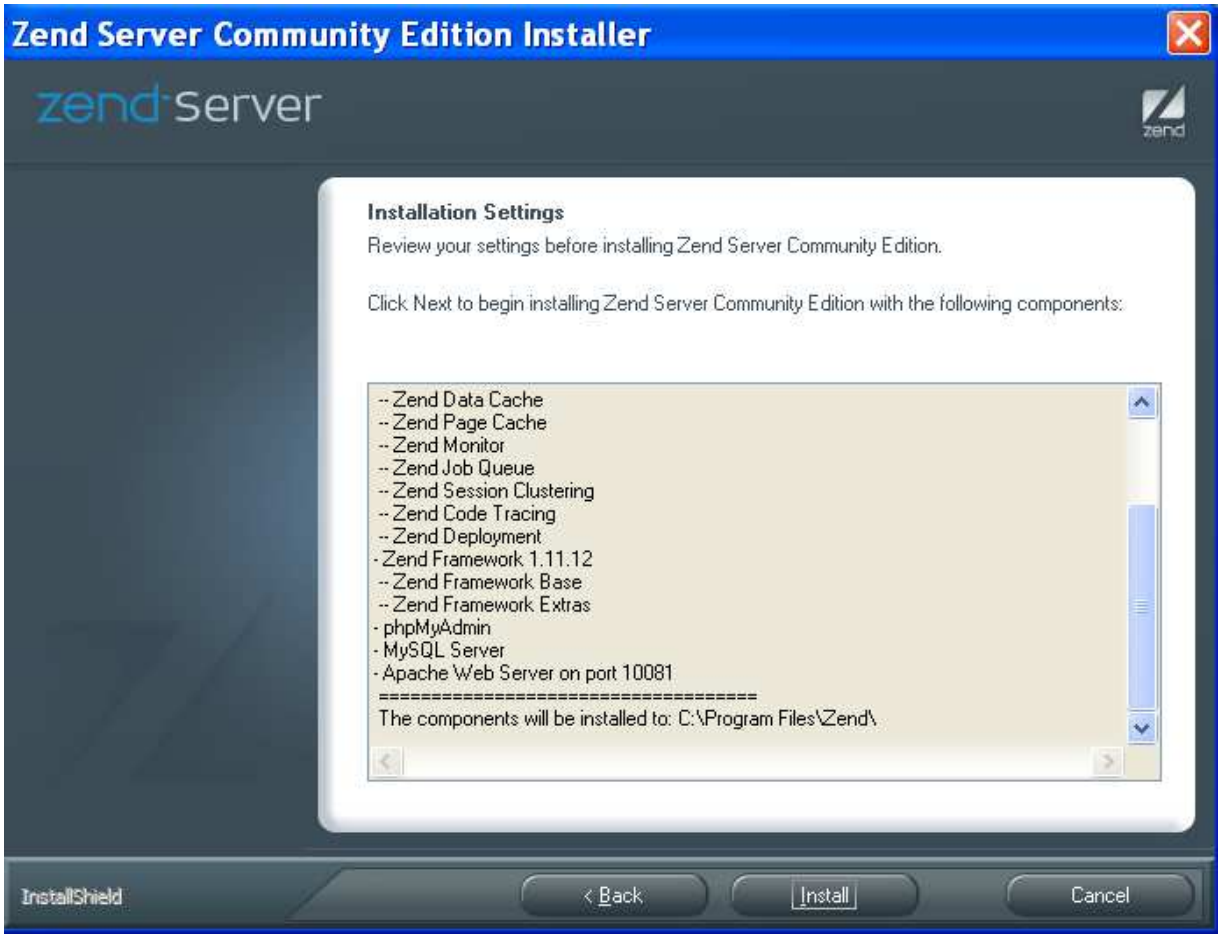


図 1-6.　*Zend Server CE カスタム・セットアップ*

http://localhost:10081/ZendServer/にある手順を実行する。

インストレーションを終えた後、C:\ProgramFiles\Zend\ZendServer\etc\php.ini に行き次をセットする:

```
Display_errors = On

Display_startup_errors = On
```

MySQL Workbench 5.2 CE

Zend Server CE のインストレーションの際、phpMyAdmin を含めた。以後この本では MySQL Workbench CE 版 5.2 を MySQL データベースのインターフェイスとして話していく。君の好きなものを使ってもよい。
phpMyAdmin は非常に用途が広く使いやすい。しかし、Zend Server CE でインストールしてログ・インしようとすると、ときどき問題があることがわかった。ユーザネームとパスワードを C:\ProgramFiles\Zend\phpMyAdmin\config.inc で変更できるが、ユーザネームを”admin”とし、パスワードをブランクにしておくならアクセスできるはずだ。

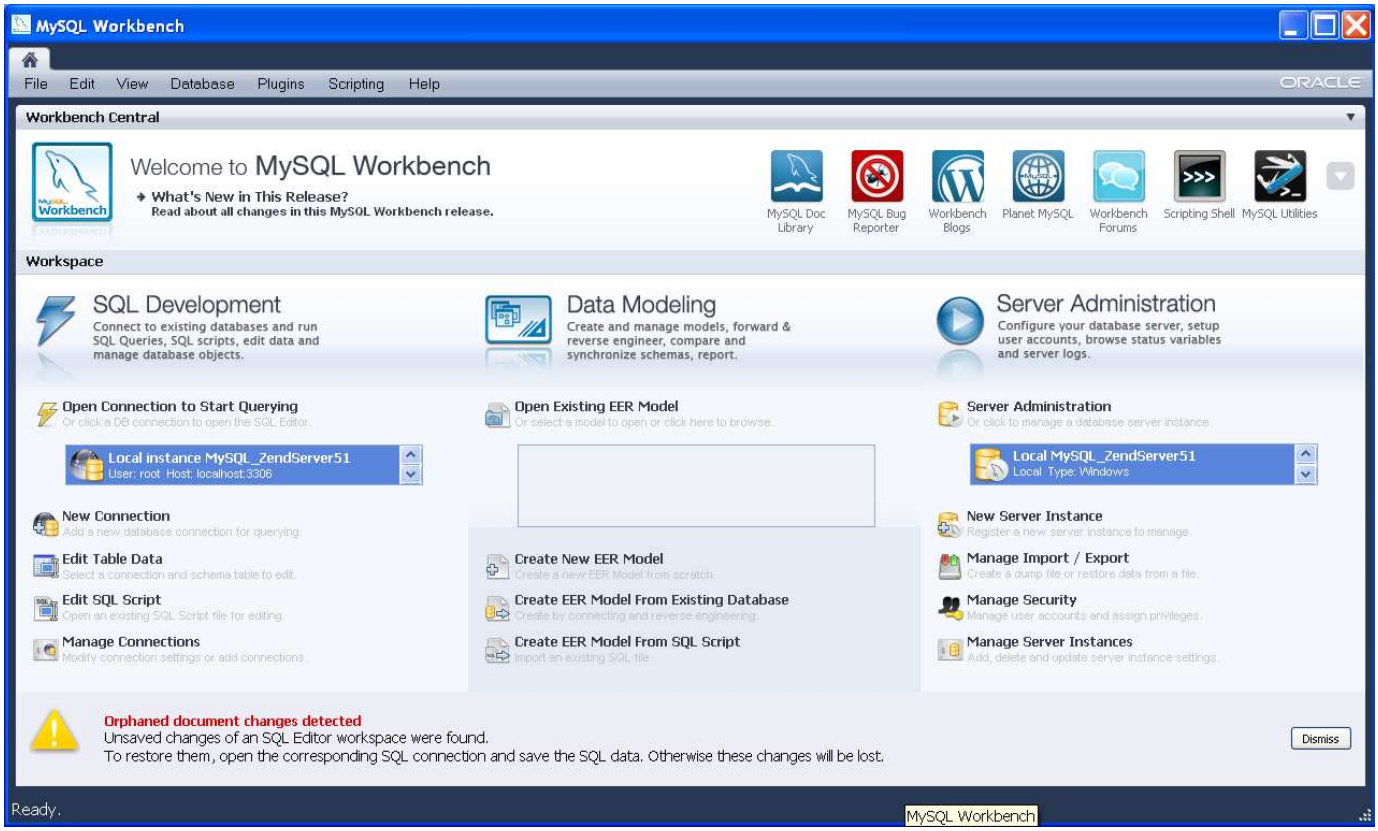


図 1-7.　*MySQL Workbench 5.2 CE ワークベンチ・コントロール・パネル*

MySQL Workbench 5.2 CE は包括的でオープン・ソースのデータベースの設計、開発、管理ツールで、次でダウンロードできる：

http://dev.mysql.com/downloads/workbench/5.2.html

統合開発環境の設定

この本で以下の章の例にあるコードの断片をランさせるため、Eclipse for PHP Developers Version:Helios Release を使うことにしている。しかし Eclipse の使用は一つの選択肢に過ぎない。君はテスト用 PHP のスクリプトを Apache のパス: http://localhost/mytestfile.php を使って好みのブラウザーから実行できる。君のソース・ファイルは Apache のディレクトリーにコピーする必要がある。この本で説明した設定ではそのパスは：C:\Program Files\Zend\Apache2\htdocs だ。

Eclipse を使うのなら、次からダウンロードできる：

http://www.eclipse.org/downloads/packages/eclipse-php-developers/heliosr

Eclipse はウインドウズ、ライナックス、マックのいずれでも使用できる。ダウンロードのページの右側で君のプラットフォームに対応したリンクをクリックするとよい。zip ファイルをダウンロードし、Eclipse ファイルを便利なディレクトリーに展開するとよい。

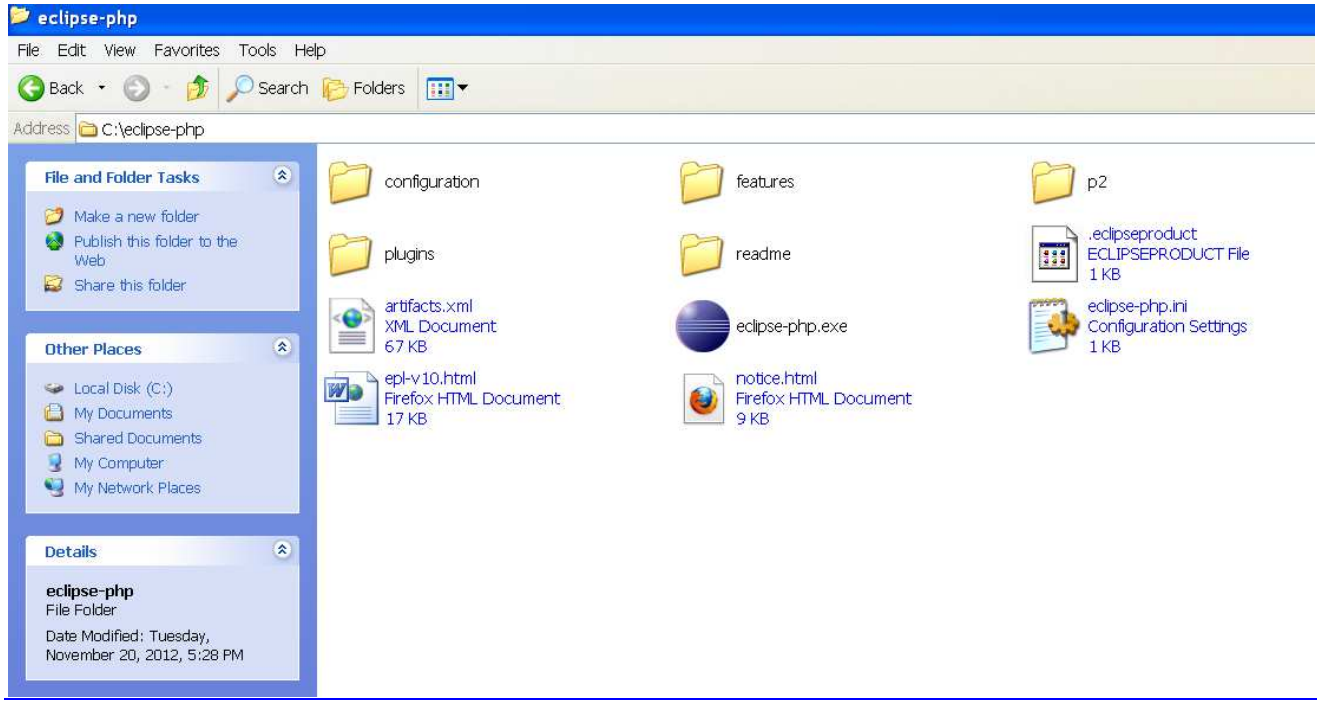


図 1-8. *Eclipse Helio PDT 起動アイコン*

Eclipse-php.exe のアイコンをダブル・クリックし、ポップ・アップ画面で”run”をクリックする：

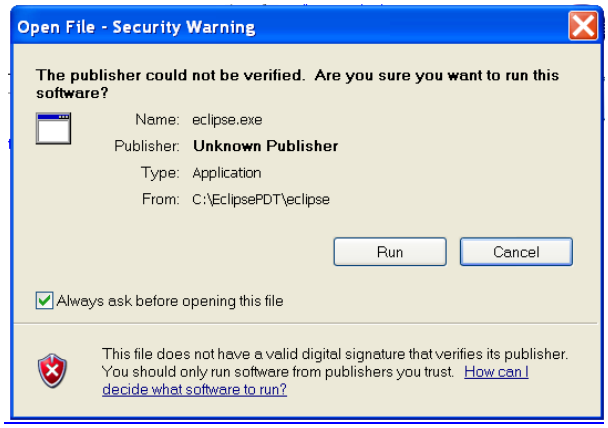


図 1-9. *Eclipse Helio PDT*

タイトル画面でそれが Helios Version であることを確認する:

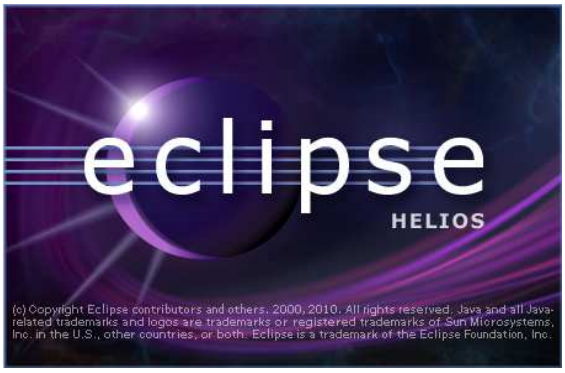


図 1-10. *Eclipse Helio PDT タイトル画面*

さてこれで Eclipse でのコーディングを開始できる。PHP のスクリプトを実行するにはツール・バーにある緑色の”run”の矢印をクリックし、さらに次を選択する：

Run As -> PHP Script

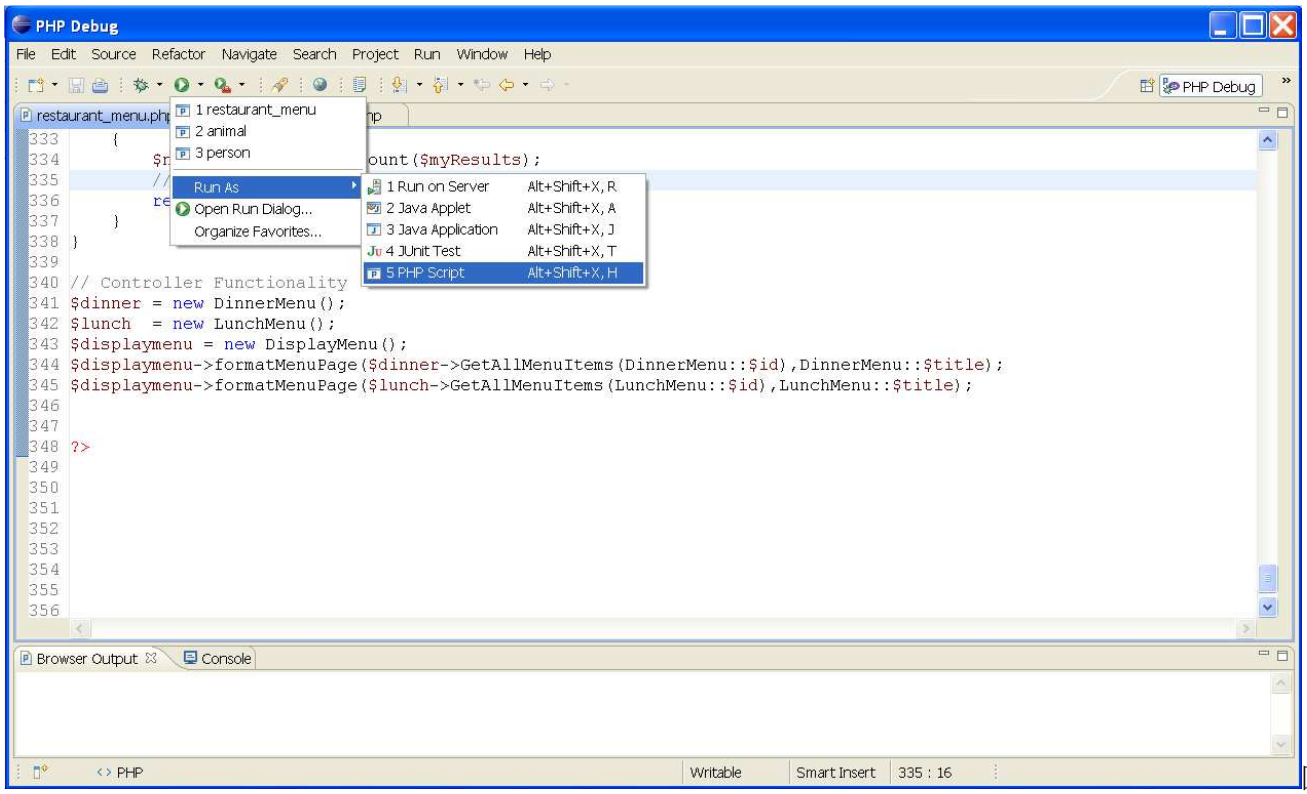


図 1-11. *Eclipse PHP デバッグ画面：Run As -> PHP Script*

第２章　PHP オブジェクトとクラス

最初の困惑

オブジェクト指向プログラミングを説明するのが難しいのは、そのための一本のあるいは順序立てた確実な方法がなく、核となるコンセプトがどれも全体にからむものなのだ。男は線状の思考、女は大局的思考をすると言われる。男はスタート・ポイントを探し、物事を順序立てて考える。女は大局的思考だから全体を見、すべてを一括して考える。手続き的プログラミングが明らかに線状なのに対して、オブジェクト指向の核となる概念は大局的なのだ。

しかしこのことがオブジェクト指向プログラミング(OOP)の利点なのだ。だからと言ってまだそれを知らない者に説明しやすいわけではない。たいていの教師は OOP の３つの優位なところすなわち：カプセル化(encapsulation)、インヘリタンス(inheritance)とポリモルフィズム(polymorphism)から始め、そして詳細に移る。しかし、これでは分りづらいのだ。

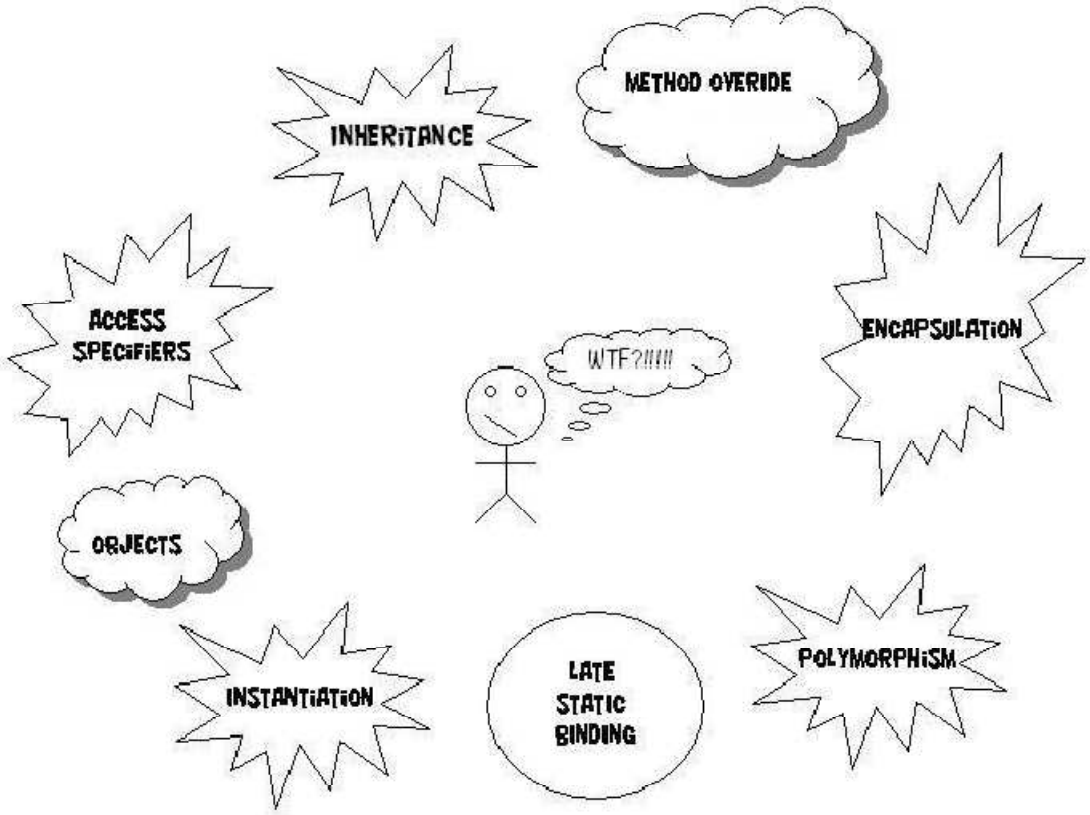


図 2-1. 最初の困惑

PHP オブジェクトとクラス

ここでオブジェクトとは何かから始めようと思う。もっともクラスなしではオブジェクトはあり得ない。クラスの本質はオブジェクトの設計図だ。それは C-struct に似てユーザが定義するデータ型だが、もう少し複雑だ。

クラスはオブジェクトの(一般に複数の)プロパティと呼ぶ特徴を定義するものだ。各プロパティはオブジェクトを記述するもので値を割り当てられる。例えば、色はプロパティで、[赤]は色に割り当てられた値だ。

１つのクラスはメソッドと呼ばれるもののメンバーとなるいくつかの関数を持てる。これらの関数はクラスの内部でプロパティに割り当てられた値のデータを操作しクラスでできているオブジェクトの行動を制御する。例えば、１つのメソッドがオブジェクトの色のようなプロパティの値をデータベースに書き込むことができる。

オブジェクトはオブジェクト指向プログラミングのまさに核心部分だからコンピュータ・プログラミングの観点から説明しよう。**オブジェクトは、クラス・関数で定義されるユーザ定義のテンプレートで構造化された、データである。**技術的に言えば、クラスはその構成要因を定義し、自身のインスタンスを作るのに使われる構造である。簡単に言えば、クラス・関数は基本的にオブジェクトのプロパティ、プロパティ、オブジェクトの行動方法、そしてそのメソッド・関数を定義する設計図である。プロパティはオブジェクトを記述し、メソッドはプロパティに割り付けられたデータを処理する方法である。

オブジェクトは想像できるあらゆるもの、鳥、机、家、あるいは人、でありうる。もし、オブジェクトが人であるなら、その人を記述するいくつかのよく使われる特徴は、名前、性別、目の色、髪の色、などである。

あるオブジェクトが存在するためには、それを定義するクラスがなければならない。ここで取り上げる単純な Person のクラスは PHP で次のようなものから始められる:

```
<?php

class Person

{

}

?>
```

これは全くの基本で単にクラスを定義するだけだ。この段階ではクラスは空である。オブジェクトを記述してそれを作り出す何のプロパティも持っていないし、オブジェクトが動作し、データを処理するためのメソッドも持っていない。少し上記の人を記述する基本的な特徴をつけくわえよう：

```
class Person
{
        $name;
        $gender;
        $haircolor;
        $eyecolor;
}
```

これで人を記述できるいくつかのプロパティを定義した。しかし、これらプロパティに対して何もすることができない。というのは、これらのプロパティに値を設定したり獲得したりするメソッドをまだ持っていないからだ。では、プロパティを扱う設定と獲得の関数を加えよう。また__construct と呼ぶメソッドも加えよう。ま

だこの時点ではこれで何かができるわけではない。それはまだ先の話で、第３章で PHP のマジック・メソッドのところで詳しく話そう。

```
<?php

class Person
{
	$name;
	$gender;
	$haircolor;
	$eyecolor;


	function setName($name)
	{
		$this->name = $name;
	}
	function getName()
	{
		return $this->name;
	}




	function setGender($gender)
	{
		$this->gender = $gender;
	}
	function getGender()
	{
		return $this->gender;
	}



	function setHairColor($haircolor)
	{
		$this->haircolor = $haircolor;
	}
	function getHairColor()
	{
		return $this->haircolor;
	}



	function setEyeColor($eyecolor)
	{
		$this->eyecolor = $eyecolor;
	}
	function getEyeColor()
	{
		return $this->eyecolor;
	}

	function __construct(){}

}
?>
```

さて上の Person クラスは人を記述するいくつかのプロパティを持っていてその情報を設定したり獲得したりするいくつかのメソッドも持っている。ここで使われている $this 変数に注目しよう。これについてはすぐ後で説明する。

さてこれで person のクラスができたので、このクラスからある person のオブジェクトを作ろう。ここでこれから登場する友達 Mindi と Billy を紹介する。彼らは Person クラスの２つのインスタンス（２つのオブジェクト）になれるのだ。Mindi と Billy を造るために、彼らをインスタンシエート(instantiate)しなければならない。

インスタンシエーション

最初にインスタンシエート(instantiate)という用語から始めよう。あるクラスをインスタンシエートするということはそのクラスに基づいた一つのオブジェクトを造ること、あるいはそのクラスのインスタンスを造ることを意味する。PHP でこれを行うには **new** というキィーワードを用いてこのようにする：

```
$Mindi = new Person()
$Billy = new Person()
```

$Mindi と$Billy はクラス Person の新しいインスタンス(オブジェクト)なので、彼らのプロパティには特定の値を割り付けることができる。Person のあとに括弧を付けたのに注意だ。これは Person クラスの定義でプロパティの定義の直後に定義した construct 関数による。この関数はこのサンプルではまだ実装されていないが第３章で PHP のマジック・メソッドのところで詳しく話そう。

さて Mindi から始めよう。皆彼女の入れ墨のことを知っているが、まず、名前や性別その他の属性から設定しよう：

```
$Mindi->setName (‘Mindi’);
$Mindi->setGender (‘female’);
$Mindi->setHairColor (‘light brown’);
$Mindi->setEyeColor (‘blue’);
```

つぎに Billy には：

```
$Billy->setName (‘Billy’);
$Billy->setGender (‘male’);
$Billy->setHairColor (‘brown’);
$Billy->setEyeColor (‘brown’);
```

$this 変数

$this 変数は関数・コールをする時のオブジェクトへのポインターである。といってもあまりピンと来ないかもしれない。Mindi の使用例で説明しよう。映画“Weird Science”で２人の変人が抽象的な娼婦を作り出す。それは$Mindi をインスタンシエートした時にしたことと同じだ。彼女の設計図は Person のクラスだった。われわれは彼女に魅惑的な青い目を次のメソッドであたえた：

```
$Mindi -> setEyeColor(‘Blue’);
```

このメソッドは Person クラスでは次のように定義されている：

```
public function setEyeColor($eyecolor)
(
       $this -> eyeColor  =  $eyeColor;
)
```

$Mindi がインスタンシエートされたとき$this が $Mindi に入れ替わった。なぜなら$Mindi は

$this の様な一般的なものの代わりに特定の属性を持っている以外は Person クラスのコピーだ。このため $Mindi オブジェクトでは setEyeColor(‘blue’) メソッドがこのようになっている：

```
public function setEyeColor(‘blue’)
{
	$Mindi -> eyeColor  =  ‘blue’;
}
```

アクセス・モディファイアー

アクセス・モディファイアー(access modifier)はクラスのプロパティとメソッドの露出度を制御するキィーワードだ。PHP では次の３段階がある:

1. **Private　（非公開）**
2. **Protected　（限定公開）**
3. **Public　　（公開）**

Mindi をそのクラスから見て記述しよう。例えば Mindi が腕に入れ墨をしているとしよう。このことを PHP では次のように書ける：

```
class Mindi(
$tattoo_location;
functiuon  setlocation() (   )
)
```

このようにすると、入れ墨の既定値は public にある。すなわち、プログラムのどの箇所からでも見ることが出来る。

次のように書いても同じことになる：

```
class Mindi (
	public $tattoo_location;
)
```

これは、public のアクセス・モディファイアーで設定され、public の露出度を持っているので、Billy のクラス、Alex のクラス、父のクラスのいずれも全て Mindi の入れ墨を見ることが出来るということだ。

さて、Mindi が母のクラスや父のクラスには知られたくなくて、友達には見せたい入れ墨があるとしよう。入れ墨を背中にして、ビキニ・トップを着れば見せることが出来る。

```
class Mindi (
	protected $tattoo_location;
	protected function setLocation() (  )
)
```

Mindi は友達との中ではビキニを着、母や父の前では着ない。このことを PHP では extends のキィーワードを用いて、人々への招待を彼女が希望する人々に拡張することで記述できる：

```
class Billy extends Mindi (
	function viewTattoo() (  )
)
```

```
class Alex extends Mindi ....

class Anna extends Mindi….

class MathProffessor extends Mindi ….
```

これらのクラスは Mindi のクラスへ拡張しているので、彼女の入れ墨を見ることが出来る。これはインヘリタンス(相続)され、Mindi に拡張した全てのクラスは彼女のインヘリタンスの階層に組み込まれている。

さて、彼女が入れ墨をもっと秘密の場所に（想像に任せる）していて、彼女のボーイフレンドの Billy だけに見せ、他には見せたくないとしよう。PHP では：

```
class Mindi (
private $tattoo_location;
private setLocation () ( )
private letBillySee () ( )
)
```

３つの PHP アクセス・モディファイアーを形式的にまとめると：

```
class Student  (
        private $name;
        public $age;
        public  function  __construct ($name, $age) (
                $this->name  =  $name;
                $this->age  =  $age;
        )
)
$tmpStudent  =  new Student  (“Mindi”. “22”);
echo “Name  :  “  ,  $  tmpStudent->name;  //エラーが起こる
```

メンバーのデータが **private** になっているのは、外部のプログラムが必要な検証を経ないで意図せずに値を変更することを避けるためである。メンバーのデータを **private** にした場合、このデータの値にアクセスするには public な設定/獲得用のメソッドを提供する必要がある。アクセス・モディファイアーの **protected** では自身のクラスとその全てのサブ・クラスからそのメンバー・データとメンバー・関数にアクセスができる。グローバルなアクセスは拒絶される。

```
class Person (
        protected $name;
)
class Student extends Person (
        function setName ($name) (
                //これで  $name は Person で保護される
                $this->name  =  $name;
                )
)
$tmpStudent  =  new Student ();
$tmpStudent->setName (“mindi”);
$tmpStudent->name  =  “mindi”;  //これは$name が保護されていて public ではないのでエラーとなる
```

public ではどのコードからも名称によってメンバーにアクセスできる。

```
class Student (
        private $name;
        public $age;
        public function  _construct ($name, $age) (
                $this->name  =  $name;
                $this->age  =  $age;
        )
)
```

```
$tmpStudent  =  new  Student (“Mindi”, “22”);
echo “Age  :  “  , $c->age;  // 22 が印刷される
```

さて以前に述べたようにオブジェクト指向の原理は”大局的(global)”であると開発者は考えるべきだということに立ち戻る。Mindi の入れ墨をアクセス・モディファイアの説明に用いたが、カプセル化(encapsulation)とインヘリタンス(inheritance)についても説明する必要がある。

インヘリタンスとカプセル化はクラスで定義されたオブジェクトの機能を規制する方法である。

インヘリタンス

Mindi の入れ墨を思い出そう。特に背中のを？背中の入れ墨は PHP の用語で protected と記述された。これは Mindi のクラスと Mindi のクラスに extend したクラスの中で見ることができる。PHP5 でインヘリトするには、クラスの定義で extends のキィーワードを使う必要があった。

```
class Parent
(
)
class Child extends Parent
(
)
```

インヘリタンスによって「知識」は受け継がれる。これはコードと機能性を新しいプログラムとパッケージに拡張する構造化された方法である。クラスは階層の中に造られ、インヘリタンスは構造とメソッドを階層で一段下ったクラスに受け渡す。これは複雑なシステムで機能を追加するときに要求されるプログラミングが少なくて済むことを意味する。もし、１つのステップが階層の最下段で追加されるなら、そのステップに関連する処理とデータのみが追加される。そのステップのその他の一切はインヘリトされている。既存のオブジェクトの再利用ができることはオブジェクト指向プログラミングの主要な利点と考えられている。

メソッドのオーバーライド

メソッドのオーバーライドは元になるクラスの関数が、派生したクラスで同じ名称、関数宣言とアクセス・モディファイアー（public も protected も）で再定義されることをいう。

メソッドのオーバーライドをする理由は、ベースとなるクラスで定義されていたものに、それを書き換える形で機能の追加をできるようにすることだ。「飲み物」という名前のクラスがあって、２つの子供のクラス、「マルティーニ」と「スコッチ」、を派生させるとしよう。飲み物のクラスには「かき回す」とか「混ぜる」とかのメソッドがすでに定義されている。しかし、「マルティーニ」と「スコッチ」には独自のかき回し方、混ぜ方があるから、「かき回す」とか「混ぜる」の機能にオーバーライドする必要が出てくる。

「飲み物」Beverage の例を見よう。

```
class Beverage {
```

```
        public function stir() {
                echo "Stir method of Beverage Class called";
        }

        public function mix() {
                echo "Mix method of Beverage Class called";
        }

}

class Martini extends Beverage {
        public function stir() {
                echo "Stir method of the Martini Class called";
        }
        public function mix() {
                echo "Mix method of Martini Class called";
        }

}

class VodkaTonic extends Beverage {
        public function stir() {
                echo "Stir method of the VodkaTonic Class called";
        }
        public function mix() {
                echo "Mix method of VodkaTonic Class called";
        }
}

$m = new Martini();
$s = new VodkaTonic();

$m->stir();
echo "\n";
$vt->stir();
```

Output:

```
Stir method of the Martini Class called
Stir method of the VodkaTonic Class called
```

親のメソッドの呼出し

先ず初めに”invoke”は、呼出し、何かの行動を要求する意味である。この短い説明をこれでおいておきたい。さて、元のクラスのメソッドをオーバーライドするときに、その機能は子供のクラスから明確にオーバーライドされない限り隠されている。親のクラスのメソッドをインボークするには、 parent のキィーワードに続いて scope resolution operator “::”とメソッドの名称を書く必要がある。

```
        parent::function_name();
```

次の例を見よう：

```
class Person {
        public function showData() {
                echo "This is Person's showData()\n";
        }
}

class Employee extends Person{
        public function showData() {
                parent::showData();
                echo "This is Employee's showData()\n";
        }
}

$c = new Employee();
$c->showData();

Output:

This is Person’s showData()
This is Employee’s showData()
```

上の例で Employee の子供のクラスにある showData() が親の Person クラスの showData() 関数をインボークする様子を見よう。Employee クラスが呼び出され showData() メソッドを実行するときには、さらに親クラスの showData() 関数が呼び出される。親のクラスの showData() が終了すると、Employee クラスの showData() 関数の残りのコードが実行される。

水平インヘリタンス　—　trait の使用

PHP は複数からのインヘリタンスをサポートしていない。クラスは１つの親クラスにだけ extend できる。もし１つより多くのクラスからのコードを必要としたらどうする？最近まで、開発者はこれを親以外のクラスからコピー&ペーストというスマートとは言えない方法で実現してきた。今では、PHP5.4 で導入された trait（訳注:遺伝形質の意）と呼ぶ新しい機能で実現でき、これは水平インヘリタンスを可能にする。

trait は構造はクラスのようだが、インスタンシエートされることができない。プロパティやメソッドは trait の構造の中で定義できるが、それらを使うにはクラスから **use** キィーワードと共に組み込んで呼び出す必要がある。

```
trait Math
{
	public function add($num1,$num2)
	{
		return $num1 + $num2;
	}

	public function multiply($num1, $num2)
	{
		return $num1 * $num2;
	}
}

trait SalesTicket
{

	private $ticket = array();

	public function add($item)
	{
		array_push($this->ticket,$item);
	}

	public function getTicket()
	{
		return $this->ticket;
	}
}

class FoodOrder
{
	use Math,SalesTicket;
	{
	//Give priority to the add method from the Math trait
		Math::add insteadof SalesTicket;
	}

}

$order = new FoodOrder();
echo $order->add(5,7), "<br />";

Output:
12
```

カプセル化(encapsulation)

カプセル化はその単語の意味する通りカプセルに入れること、あるいは機能を１つのパッケージにしまうことである。あるオブジェクトがカプセルあるいは箱であるとしよう。アクセス・モディファイアーは誰が中を見ることが出来るかを制御する窓の様なものである。それが public であれば窓は大きく開いていて誰もが内部を見ることが出来、private であれば隠ぺいされていて誰にでも内部を見せるわけではない。

オブジェクトの内容を見れる程度の制御はカプセル化がどう使えるかの重要なポイントだ。

あるオブジェクトの内容はプロパティとメソッドで、それらの全てがオブジェクトの内部に詰め込まれているので、その内容の見れる程度を制御する一種の小さなプログラムが用意された、それがカプセル化だ。

技術的に言えば、カプセル化はデータに対して処理関数を結びつける内蔵型のモジュールを生成することだ。カプセル化によってコードのモジュール化が良くなる。複数のルーティンを分離し、互いに矛盾が起きることの無いように保ってくれる。繰り返すが、各々のオブジェクトは、自身の役割をあたかも自身の内蔵型カプセルあるいは錠剤として扱うミニ・プログラムであると考えることができる。

ポリモルフィズム(Polymorphism)

ここまで来てクラスが設計図でありオブジェクトはクラスに基づいて構成されていることは分っているだろう。また同じクラスから多くの異なったオブジェクトがつくられることを理解できるだろう。机の１つのクラスがあるとしよう。すると丸い机、四角い机、長い机、短い机も作ることが出来る。青い机、茶色の机もありうる。樫の木で作った机、プラスティックの机；すべて同じ基本のクラスから作れるのだ。

ポリモルフィズムは多くの形態を意味する。コンピュータプログラミングに於いてポリモルフィズムは必要性に応じて適応できるよう**１つより多くの形態を持つ能力を持つ変数、関数、オブジェクト**のことを言う。異なったタイプのオブジェクトが一つのインタフェイスを通じてデータの処理ができる仕組みのことだ。ポリモルフィズムによってアプリケーションがよりモジュール構成をとれ拡張しやすくなる。異なったコースをたどる処理を記述するためのごちゃごちゃになった読みづらい条件判定文の代わりに、必要性に応じて選択する交換可能なオブジェクトを造ればよいのだ。これがポリモルフィズムの基本的な目的だ。

ポリモルフィズムは異なったタイプのオブジェクトが一つのインタフェイスを通してデータを処理できる仕組みだ。

よくわかる例を使って説明しよう。あるプログラムが speak() と呼ぶ関数を含む動物と呼ぶオブジェクトを定義するとしよう。すると、インヘリタンスを用いて牛や犬などの子供のオブジェクトを派生でき、それぞれに特有の speak() 関数を造りこめる。ポリモルフィズムの仕組みによってそのプログラムは牛のためか、犬のためか、その他の派生した動物のためか知らなくても speak() 関数を呼び出すことが出来、しかも特定の動物のための正しい関数が実行されるのだ。

ポリモルフィズムはメソッドの多重定義やメソッドのオーバーライドとは異なるのでこれらと混同することはない。オブジェクト指向プログラミングでは、メソッドはクラスに属する関数であるが、クラス変数はそのメンバーを参照する。

メソッドの多重定義(Method Overloading)

*メソッドの多重定義*はクラスが同じ名称の２つ以上の関数を持ち、その各々のパラメータが異なった数であったり、型であったりすることを許していることを言う。

メソッドのオーバーライド

*メソッドのオーバーライド*はインヘリタンスによって派生した子供のクラスが親のクラスにある関数と同じ名称、同じパラメータの関数を親のものとは異なる処理内容で定義するときに起こる。これはポリモルフィズムとは後からのバインディング(late-binding)を必要としない点で異なるが、オーバーライドはポリモルフィズムが達成できる仕掛けの一部である。

これは最初は分りにくい点かもしれないが、次の例を見れば分かるだろう。

```
class Animal {
   protected $name;

   public function __construct($name)
   {
      $this->name = $name;
   }

   public function getName()
   {
      return $this->name;
   }
}

class Cow extends Animal
{
   public function getName()
   {
      return 'Cow: '.parent::getName();
   }
}

$a = new Cow("Elsie Mae Mae");
echo $a->getName();
```

これはオーバーライディングの例である。この場合はポリモルフィズムとは異なる。なぜなら Cow を宣言し、Cow の関数を　Cow のインタフェイスで直接呼び出すからである。しかし、PHP は型指定の緩い言語である。というのは、ほとんどのデータの型の処理はプログラマーがするのではなく PHP によって間接的に行われるからである。このため PHP におけるオーバーライディングとポリモルフィズムの違いはどちらかというとわずかなものに見える。少し異なる次の例を見よう：

```
<?php

class Animal {
   protected $name;

   public function __construct($name)
   {
      $this->name = $name;
   }
```

```
   public function getName()
   {
      return $this->name;
   }

   public function speak()
   {
      return $this->getName().' is speaking <br />';
   }
}

class Cow extends Animal
{
   public function getName()
   {
      return 'Cow: '.parent::getName();
   }
}


$a = new Cow("Elsie Mae Mae");
echo $a->speak();
$b = new Animal("something");
echo $b->speak();

?>
```

OUTPUT:

```
Cow: Elsie Mae Mae is speaking
something is speaking
```

実行時結合(Late Binding)（ダイナミック結合）

実行時結合はダイナミック結合(dynamic Binding)とも呼ばれ、ポリモルフィズムが達成される仕組みである。上の例で、 speak() 関数は親のクラスで定義されているが、ポリモルフィズムの関数であり、子供のクラスで getName() を呼び出すことで特定の処理が行われる。実行時結合とは要求に対して特定のデータ型が結合されるのがその関数が呼び出されて実行されるときであるということである。簡単に言えば、 speak() は一般的な Animal クラスで定義されるが、getName の子供に特定の処理を呼び出すことになる。この実行時結合は PHP の比較的最近の特徴でこれで本来のポリモルフィズムが可能になった。

```
Another example:

class Animal {
   protected $name;

   public function __construct($name)
   {
      $this->name = $name;
   }

   public function getName()
   {
      return $this->name;
   }
}

class Cow extends Animal
{
   public function getName()
   {
      return 'Cow: '.parent::getName();
   }
```

```
}

function Name(Animal $a)
{
   echo $a->getName();
}

$b = new Cow("Elsie Mae Mae");
Name($b);
```

この例はポリモルフィズムによる。というのは Name() 関数は Animal の型のオブジェクトを受け取ることになっているが、それが牛なのか、犬なのか、他の動物なのかは知らないからだ。 Animal という特定の型が Name() 関数のパラメータに指定されているところに注目しよう—関数のパラメータのデータの型を指定するのは型指定の厳密なプログラミング言語で通常要求されることだが、ここでの型のヒントは PHP ではオプションである。型のヒントの指定は、もしその関数にオブジェクトでないものや、異なる型のオブジェクトを渡した場合に重大なエラーを引き起こす。

PHP は、他の言語ならコンパイル時に行うような多くの一般的な処理を実行時に行うため、ダイナミックなプログラミング言語だと言える。それは型指定の緩い、あるいはダイナミックな言語と言われ、大半の型のチェックがコンパイル時ではなく実行時に行われる。ダイナミックな型指定の値は型を持つが、変数は持たない、すなわち、変数はいかなる型のいかなる値も取りうる。

第３章　マジック・メソッド

マジック・メソッドは PHP の特別な機能を実現するためにユーザがクラスの中で使うことのできる予約されたメソッドの総称である。マジック・メソッドによって PHP の特別な働きを使うことが出来る。マジック・メソッドの名前は２つの下線に続いて小文字で表記するのが慣例となっている。

_construct()

コンストラクタ(construct)はマジック・メソッドの１つでオブジェクトのインスタンスを造る際に呼ばれる。これは普通クラスの宣言をする最初にすることであるが、必ずしもそうでなくてもよい。他のメソッドと同様クラスのどこで宣言してもよい。Mindi のインスタンスを造ったとき、Person のクラス名称のあとに付けた括弧のことを思い出そう。

```
$Mindi  =  new Person();
```

Person() クラスへ参照をしているのは $Mindi オブジェクトのための construct である。この例では括弧の中になにも無いので construct は使わなかった。このことは construct を呼び出したときに $Mindi のプロパティは何も初期化しないことを意味する。あるクラスで construct 関数を定義しなかった時には PHP のエンジンは実装していない construct を使うと想定している。すなわち：

```
public function __construct() ()
```

Construct メソッドは他のメソッドと同様にインヘリトをする。そのため前のオブジェクト指向プログラミングへの紹介のところでのインヘリタンスの例について考えるなら、次のように Animal クラスへ construct を加えてもよいのだ：

```
// animal.php
class Animal
(
        Public function   __construct() (
                $this->created  =  time() ;
                $this->logfile_handle  =  fopen( ‘ /path/to/log.txt’, ‘w’);
        )
)
```

これで Animal クラスからインヘリトしてクラスを造ることが出る。Cow のクラスの中になにも追加しなくても、それを宣言し Animal からインヘリトすることが出来る。

```
class Cow extends Animal (
)
$milky  =  new Cow();
echo $milky->created;
```

もし Cow クラスの中に __construct メソッドを定義するなら、Cow オブジェクトはインスタント化されたときにそれを実行することになる。上の例では定義していないので、PHP は親のクラスの記述の中に情報を求めそれを使うことになる。すなわち、新しいクラスでオーバーライドすることもでき、しないこともできる―便利だ。

親のコンストラクタとデストラクタを使う

親のメソッドを行使するのと同様に親の PHP5 コンストラクタと PHP5 デストラクタを行使されるようにすることが出来る。次の例を見よう：

```
class Person{
	public function __construct() {
		echo "This is Person's __construct()\n";
	}

	public function __destruct() {
		echo "This is Person's __destruct()\n";
	}
}

class Employee extends Person{
	public function __construct() {
		parent::__construct();
		echo "This is Employee's __construct()\n";
	}

	public function __destruct() {
		parent::__destruct();
		echo "This is Employee's __destruct()\n";
	}
}

$c = new Employee();

Output:
This is Person’s __construct()
This is Employee’s __construct()
This is Person’s __destruct()
This is Employee’s __destruct()
```

__destruct()

コンストラクタの部分にあったファイル・ハンドルに気付いていたかい？オブジェクトの使用が終了した時周りにものを残して置きたくない。そこでデストラクタ(__destruct) メソッドがコンストラクタの逆をしてくれる。これはオブジェクトが滅却されたときに実行される。明示的にそうしてもよいが、もう使わなくなったときに PHP が掃除してくれる。Animal についてデストラクト・メソッドは次のようなものになる：

```
class Animal{

  public function __construct() {
    $this->created = time();
    $this->logfile_handle = fopen('/path/to/log.txt', 'w');
  }

  public function __destruct() {
    fclose($this->logfile_handle);
  }
}
```

__get(), __set(), __isset()

クラスの内部で明示的な宣言のないプロパティの値の設定と取得処理は PHP ではプロパティのオーバーロードによる。__set()関数はクラスの明示的に宣言されていないプロパティに値をアサインしようとするごとに PHP のエンジンが自動的に起動する。__get()関数はスクリプトがクラスで宣言されていないプロパティの値を取得しようとするときに起動される。__isset() 関数は未定義のプロパティが要求されたときに起動される。__isset() 関数の目的はプロパティに働きかける前にテストすることである。

“User”の例１

“User”と呼ぶベースとなるクラスがあるとしよう。それはソフトウエア上で実世界のあるユーザを表現している。この例のクラスの定義は次のようなものだ：

```
class User {
    private $firstname = 'John';
    private $lastname = 'Doe';
    private $email = 'janedoe@somedomain.com';

//コンストラクタ（実装されていない）

    public function __construct(){}

// user のファースト・ネームをセットする
    public function setFirstName($firstname)
    {
            $this->firstname = $firstname;
    }

// user のファースト・ネームを獲得する
    public function getFirstName()
    {
            return $this->firstname;
    }
// user の苗字をセットする
    public function setLastName($lastname)
    {
            $this->lastname = $lastname;
    }

// user の苗字を獲得する
    public function getLastName()
    {
    return $this->lastname;
    }
// user の e-メール・アドレスをセットする

    public function setEmail($email)
    {
            $this->email = $email;
    }

// user の e-メール・アドレスを獲得する
    public function getEmail()
    {
            return $this->email;
    }
}
```

これで分かるように、上記の“User”クラスはしごく簡単に理解できるだろう。いくつかの set 関数と get 関数で構成されていて、ファースト・ネーム、苗字、e-メールの３つの宣言されたプロパティの値を設定あるいは取得するように使われている。これなら特に変わったことがないだろう？次はこのクラスをどう使うかの例だ：

```
$user = new User();
$user->setFirstName('John');
$user->setLastName('Doe');
$user->setEmail('john@domain.com');

//user のデータを表示する
echo 'First Name: ' . $user->getFirstName() . '<BR> Last Name: ' . $user->getLastName() . '<BR> Email: ' .
$user->getEmail();
```

```
// 次の表示が行われる
displays the following:
First Name: John
Last Name: Doe
Email: john@domain.com
```

“User”の例２

前の例では、基本的な“User”の例を示した。そのアプリケーション・インタフェイスでは宣言されたプロパティの値を設定あるいは取得するのが容易だった。しかし、前に述べたように、この定義をもっと短くすることが可能で、__set()と __get()のメソッドを具体的に実装するだけでより「ダイナミックな」ものにすることができる。

この考えをさらに示すために、このサンプル例のクラスを再定義して、新クラスのプロパティを実行時に造るだけでなく、各々の値を至極簡単に取得できるようにして見せよう。

これから次の数行でやろうとしていることは分っていると思うが、変更したバージョンの”User”クラスをよく調べてほしい。それはこんな感じになる：

```
class User
{
//コンストラクタ（実装されていない）
    public function _construct(){}

//宣言されていないプロパティをセットする
    function __set($property, $value)
    {
            $this->$property = $value;
    }

//プロパティを定義させる
    function __get($property)
    {
            if (isset($this->$property))
            {
                    return $this->$property;
            }
    }
}
```

最初のケースでは、 __set()関数が宣言されていないプロパティを造りそれに値を割り付ける。それに対し後半のケースでは、相手役の __get()がプロパティが既にセットされているときにその値を取得する。

これら２つの方法での実装は非常に簡単な方法でプロパティをオーバーロードさせてくれる。さらにクラスのソース・コードをより短く、より簡潔にしてくれる。しかしこのプロセスは考慮に入れておくべき短所がある。第１に、そして重要なことだが、クラスの各プロパティの宣言が明示的にされていないのでコードの読解が難しくなるということだ。第２に、Eclipse PDT (エクリプス PHP 開発環境)の様な統合開発環境がこれらのプロパティについてコメントしたり記述したりして把握しておくことが、事実上不可能だということだ。

```
// “User”クラスのプロパティ・オーバーロードを用いる例
$user = new User();
$user->firstname = 'Jane';
$user->lastname = 'Doe';
$user->email = 'janedoe@mydomain.com';
$user->address = 'My address 1111';
```

```
// user データの表示
echo 'First Name: ' . $user->firstname . ' Last Name: ' . $user->lastname . '<BR> Email: ' . $user->email .
'<BR>Address: ' . $user->address;
```

OUTPUT:

```
First Name: Jane
Last Name: Doe
Email: janedoe@mydomain.com
Address: My address 1111
```

__call()

もしあるクラスが__call() を実装しているとすると、そのクラスのあるオブジェクトが存在しないメソッドで呼び出されると、__call() が代わりに使われる。例えば：

```
<?php
//このクラスは __call() マジック・メソッドがどのように使われるかを示す
class CallTest {
  private $v = array(1, 7, 9);

  public function __call($method, $args) {
    echo "The method \"$method\" was called.\n";
    var_dump($args);
    return $this->v;
  }
}

// test() は存在しない
$example = new CallTest();
$a = $example->test('f', 'o', 'o');
var_dump($a);
?>
```

OUTPUT:

```
The method "test" was called.
array(3) {
  [0]=>
  string(1) "f"
  [1]=>
  string(1) "o"
  [2]=>
  string(1) "o"
}
array(3) {
  [0]=>
  int(1)
  [1]=>
  int(7)
  [2]=>
  int(9)
}
```

では test() と呼ばれるメソッドを加えてもう一度実行してみよう。今度は __call() の代わりに test() が呼び出されることに注目しよう：

```
<?php
//このクラスは __call() マジック・メソッドがどのように使われるかを示す

class CallTest {
  private $v = array(1, 7, 9);
```

```
  public function __call($method, $args) {
    echo "The method \"$method\" was called.\n";
    var_dump($args);
    return $this->v;
  }

  public function test($args) {
    echo "The method \"test\" exists now!\n";
    return NULL;
  }
}

// test() が存在する
$example = new CallTest();
$a = $example->test('f', 'o', 'o');
var_dump($a);
?>
```

OUTPUT:

```
The method "test" exists now!
NULL
```

_sleep()

マジック・メソッドの__sleep()はあるクラスのオブジェクトがバイナリー・コードになるとき(serialized)に呼び出される。バイナリー・コードはオブジェクトが記憶装置に記憶するかあるいはネットワーク経由で他の場所に転送するためバイトの１次元連続並びに変換されたものである。このマジック・メソッドの__sleep()はパラメータは受け取らず、アレイを戻す。そのアレイはバイナリー・コードになるべきクラスのメンバーのリストを含んでいる。このことから、ある特定のクラスメンバーをバイナリー・コードにしたくないなら、それをアレイに入れてはいけない。次の例を見よう：

```
class Employee {
          private $fname;
          private $date_of_birth;

          public function setFirstName($fname) {
                    $this->fname = $fname;
          }

          public function getFirstName() {
                    return $this->fname;
          }

          public function setBirthDate($dob) {
                    $this->date_of_birth = $dob;
          }

          public function getBirthDate() {
                    return $this->date_of_birth;
          }

          public function __sleep() {
                    return array("fname"); //このため、各状態がバイナリー コードになる
          }

}

$e = new Employee();
$e->setFirstName("Marsha");
$e->setBirthDate("09-12-1983");

$data = serialize($e)."\n";
echo $data."\n";
```

Output:

```
O:8:"Employee":1:{s:15:" Employee fname";s:6:"Marsha";}
```

上の例で、バイナリー・コード化されたデータは Employee オブジェクトの fname を含んでいるだけであることが分かる。その理由は、マジック・メソッドの__sleep() は‘fname’のデータ・メンバーのみを含んだアレイを戻すからである。

_wakeup()

マジック・メソッドの__wakeup() は __sleep() メソッドの反対であり、パラメータは受け入れず何も戻さない。オブジェクトがバイナリー・コードから戻って通常のストリングになり、ファイルに保存したり、ＵＲＬに送ったりできるようになった後はセットアップ操作がこれを行うことになる。次の例を見よう：

```
class Employee {
          private $fname;
          private $date_of_birth;

          public function setFirstName($fname) {
                    $this->fname = $fname;
          }

          public function getFirstName() {
                    return $this->fname;
          }

          public function setBirthDate($dob) {
                    $this->date_of_birth = $dob;
          }

          public function getBirthDate() {
                    return $this->date_of_birth;
          }

          public function __sleep() {
                    return array("fname");
          }

           public function __wakeup() {
                if($this->fname == "Marsha") {
                       $this->date_of_birth = "09-12-1983";
                }
           }
}

$e = new Employee();
$e->setFirstName("Marsha");
$e->setBirthDate("09-12-1983");

$data = serialize($e)."\n";
var_dump(unserialize($data));

OUTPUT:
object(Employee)#2 (2) {
  ["fname:private"]=>
  string(6) "Marsha"
  ["date_of_birth:private"]=>
  string(10) "09-12-1983"
}
```

上の例では、$e オブジェクトがバイナリー・コードされそのアウトプットが$data 変数に記憶された後、$data 変数を用い、それを unserialize() に渡す。オブジェクトがバイナリー・データから戻って生成される前に、__wakeup() メソッドが呼び出される。

_clone()

PHP では、オブジェクトは常に参照によって割り当てられ渡されていく。簡単な例で説明しよう：

```
 Class CopyClass
```

```
{
   public $name;
}

$first = new CopyClass();
$first->name="Number 1";
$second = $first;

echo "first = " . $first->name . "<BR><BR>";

$second->name="Number 2";

echo "first = " . $first->name . "<BR>";
echo "second = " . $second->name . "<BR>";
```

OUTPUT:

```
first = Number 1
first = Number 2
second = Number 2
```

$first オブジェクトのコピーを造って$second と呼び、$second->name=”Number 2”と値をセットした後、$First->name プロパティの値が“Number 1”から“Number 2”に変わったことに注目しよう。これは$first と $second のどちらのオブジェクトも実際には同じオブジェクトを参照し、$second オブジェクトは参照によって割り当てられているからである。

マジック・メソッドの__clone はオブジェクトのコピーを造るのに使うことが出来る。そのオブジェクトはコピーされたオブジェクトを参照することなく、代わりに新しく完全に区別できるオブジェクトを造り出す。前の例で$second = $first と設定した代わりに__clone を使うとそれがどのように働くかを示そう:

```
Class CopyClass
{
   public $name;
}

$first = new CopyClass();
$first->name="Number 1";
$second = clone $first;
echo "first = " . $first->name . "<BR><BR>";

$second->name="Number 2";

echo "first = " . $first->name . "<BR>";
echo "second = " . $second->name . "<BR>";
```

OUTPUT:

```
first = Number 1
first = Number 1
second = Number 2
```

今回は$first->name の値は$second->name に値をセットした後でも変わらなかった。それは clone が$first のコピーを全く別のものとして造ることを可能にしているからである。このため、前のように同じオブジェクトを参照するのではない２つのオブジェクトを造れたのである。

第４章　抽象的クラスとメソッド

ポリモルフィズムに不可欠な部分は共通インタフェイスであり、共通インタフェイスとは要素間の相互作用が*行われる所である。PHP で共通インタフェイスを定義する方法は２通りあり、**インタフェイスと抽象クラス***である。それぞれ使い道があり、それらをクラスの階層の中にはめ込むときに取捨選択すればよい。

これらの特徴を説明するには何通りかの方法があるが、ある人は抽象クラスをインタフェースとクラスの混ざったものと説明するのが良いという。しかし、私は PHP のインタフェースについての議論は第５章に後送りし、ここでは抽象クラスをまず説明したい。

PHP の抽象クラスは、もっぱら基本のクラスとしてだけ使われる汎用のクラス、すなわち、インスタンス化されることの無い抽象的なクラスである。抽象クラスはインヘリタンスによってより具体的なクラスを構築する土台となる物である。言い換えると、抽象クラスのメソッドとプロパティは抽象クラスからインヘリトされた子のクラスで実装される。抽象クラスからオブジェクトを造ることはできないが、その全てのメソッドとプロパティは抽象クラスを基にしたクラスから作られたオブジェクトで使われるべきものなのだ。

君をさらに煙に巻く前に、Mindi と彼女の友達の Vallery について話そう。娘たちはと浜辺でセクシーで高価な水着でその肢体を誇らしげにさらすのを好む。

誰がその小さな着衣がそんなに高くつくと考えたことがあるだろうか？Mindi も Vallery もはでなルックスを保つために仕事が必要になってくるので、娘たちにウエイトレスの仕事を与えて助けるとしよう。

ウエイターやウエイトレスとして容姿が良いのは大きな助けになるが、容姿が全てではない。良いウェイターやウエイトレスは徹底的にメニューを頭に入れなくてはならない。そこでさらに進んで彼女たちを助けるためにメニューを造って仕事ができ、お金を稼ぎ、新しいビキニを買い、浜辺に戻ることが出来るようにしてあげよう！メニューはメニュー品目から出来ているから、メニュー品目から始めるとしよう。

娘たちがどこで働くのか、どのような料理をサービスするのかの選択の自由度を与えたいので、必要などのような種類のメニューでも作れるようにしたい。そこでできる限り一般的、**抽象的な**もので始めたい。

abstruct のキィーワード

抽象クラスあるいはメソッドは abstruct のキィーワードを使って宣言する。最大限の自由度を得て欲しいだけ多くの種類のメニューとメニュー品目を造れるようにするため、抽象 Menu クラスから始める。

```
abstruct class Menu
{

}
```

さて、メニュー品目のためにある abstruct クラスができたが、その中に何も入れていないから、何もしてくれない。Mindi と Vallery のためにこれを考えていこう。最終的にはデータベースを持つことになるだろうと仮定して、メニューの各々とメニュー品目の各々にユニークな識別子が要る。そのためには数通りの方法があるがここではメニュー発生器を設計しメニュー品目の一般的なリストを持ち、１つ以上のメニュー(ランチ・メニュー、ディナー・メニューなど)に使えるようにしよう。このことは、１つのメニュー品目 ID に対し

て複数のメニューID が存在する 1 対多の関係があることを意味する。このことから、メニュー・クラスはメニュー品目 ID のフィールドも必要とすることになる。名称と記述のフィールドも加えよう。

```
abstract class Menu
{
	private $menuid;
	private $menuitemid;
	private $menuname;
	private $description;

	public function setMenuID($menuid){$this->menuid = $menuid;}
	public function getMenuID(){return $this->menuid;}

	public function setMenuItemID($menuitemid){$this-> menuitemid = $ menuitemid;}
	public function getMenuItemID(){return $this-> menuitemid;}

	public function setMenuName($menuname){$this->menuname = $menuname;}
	public function getMenuName(){return $this->menuname;}

	public function setDescription($description){$this->description= $description;}
	public function getDescription(){return $this->description;}
}
```

多分 MenuItem のクラスを abstruct にしないのがベストだ。というのは全て同じ構造を持つメニュー品目の一般リストあるいはテーブルを造ろうとしているのなら、MenuItem のクラスのインスタンシエートをしたくなるからだ。各々のメニュー品目が持つべき基本的プロパティとして、ユニークな ID、名称、記述、価格、提供サイズ、それに写真へのリンクを与えよう。それにこれらのプロパティに値を設定、獲得するメソッドも与えよう。

```
class MenuItem
{
	private $menuitemid;
	private $itemname;
	private $description;
	private $price;
	private $servingsize;
	private $picture;

	public function setID($menuitemid){$this-> menuitemid = $ menuitemid;}
	public function getID(){return $this-> menuitemid;}

	public function setPrice($price){$this->price = $price;}
	public function getPrice(){return $this->price;}

	public function setPicture($picture){$this->picture = $ picture;}
	public function getPicture (){return $this->picture;}

	public function setItemName($itemname){$this->itemname = $itemname;}
	public function getItemName(){return $this->itemname;}

	public function setDescription($description){$this->description= $description;}
	public function getDescription(){return $this->description;}

	public function setServingSize($servingsize){$this->servingsize = $servingsize;}
	public function getServingSize(){return $this->servingsize;}
}
```

さてこれで元になるメニューのクラスが出来た。特定のメニューのクラスをいくつか構成し、それをインスタンシエートして実際のメニュー・オブジェクトを造るとしよう。

```
class MainMenu extends Menu
{
```

```
//...
}

class DrinkMenu extends Menu
{
//...
}

class LunchMenu extends Menu
{
//...
}

class KidsMenu extends Menu
{
//...
}

class DessertMenu extends Menu
{
//...
}
```

最後の２つのメニュー・クラス, KidsMenu と DesertMenu のクラスをチェックしよう。多分これら２つのクラスはさらに子のクラスを生成してインヘリトさせることで拡張できるようにしておくことはしたくない。その必要性がないからだ。デザート・メニューは通常誰もがレストランで見る最後のものだし、子供用メニューは見てすぐ分かる物でないといけない。そこでこれら２つのクラスを最終のものとして拡張性に蓋をしてしまおう。そうすれば Mindi や Vallery がこの先さらに混乱することがないだろう。このためには final というキィーワードを使えばよい。

抽象メソッド

第２章のポリモルフィズムの議論でオーバーライドの話が出たのを思い出そう。そう、抽象メソッドは、直接呼び出されることはなく、オーバーライドされることで初めて使われる。直接呼び出すとエラーを引き起こす。抽象メソッドは、抽象クラスの中で定義される。抽象クラスでないクラスで抽象メソッドを定義しようとすると、コンパイル・エラーを引き起こす。

次の例で Dog と名付ける抽象クラスを造り、その中で bark() と呼ぶ抽象メソッドを造る。

```
<?php

abstract class Dog {

    private $name = null;
    private $gender = null;

    public function __construct($name) {
        $this->name = $name;
    }

    public function getName() {return $this->name;}
    public function setName($name) {$this->name = $name;}

    abstract public function bark();

}
```

```
// Override the abstract bark() method of the abstract Dog class

class Rottweiler extends Dog {

    public function bark() {
        echo "WOOOF!! WOOOF! WOOOF!" . "<br>";
    }

}

class Chihuahua extends Dog {

    public function bark() {
        echo "yip! yip! yip!" . "<br>";
    }
}


// This class causes a compilation error, because it fails to implement bark().
class Mutt extends Dog {
    // this will cause an error because this dog didn't override bark()
}


?>
```

OUTPUT:
Fatal error: Class Dog contains 1 abstract method and must therefore be declared abstract or implement the remaining methods (Dog::bark)

final キィーワード

```
 final Class KidsMenu extends Menu
 {
 //...
 }

 final Class DessertMenu extends Menu
 {
 //...
  }
```

インヘリタンスはクラス階層の中で柔軟性を生む。クラスやメソッドはオーバーライドされるので、クライアントのメソッドの中の call はどのクラスのインスタンスにそれが渡されるかによって異なった働きをする。ときには、子供メニューやデザート・メニューの場合のように、クラスやメソッドは変わることがない。Final のクラスは子のクラスを持つことがないし、final のメソッドはオーバーライドされない。

第５章　PHP インタフェース

インタフェイスは単にテンプレートなのだ。インタフェイスの目的はクラスに複数のインタフェイスを持たせることで柔軟性を持たせることだ。ただし、多重インヘリタンスは許さない。第２章で述べたように、PHP に多重インヘリタンスはない。インタフェイスは契約のようなもので、インタフェイスを組み込むときにはそれが定義している全てのメソッドを実装しなくてはならない。言い換えると、インタフェイスを組み込むクラスはインタフェイスで定義された全てのメソッドを実装しなくてはならない。

PHP は型指定の緩い言語である。クラスは拡張するクラスと同様に取り入れるインタフェイスの型を受け入れることになる。

インタフェイスを使用すると「インタフェイスに向けたプログラミング」と呼ばれるスタイルでプログラミングをすることが出来る。その意味するところはプログラミング論理の基本を内部の詳細な実装ではなく、オブジェクトのインタフェイスに置くことだ。インタフェイスに向けたプログラミングをすれば実装の詳細に依存する分が減り、コードが再利用可能になる。そしてプログラマーは後ほど同じインタフェースで実装された他のオブジェクと入れ替えることで簡単にシステムの行動を変えることが出来る。

インタフェイスはコードを含んでいないことを除くと、クラスに似たようなものだ。インタフェイスはメソッドの名称や引数を定義することが出来るが、メソッドの中身は定義できない。１つのクラスは複数のインタフェイスを実装することが出来る。

キィーワード interface と implements

インタフェースは interface というキィーワードを用いて宣言され、implements というキィーワードを用いて実行される。

```
interface MyInterface {

}


class MyClass implements MyInterface {

}
```

メソッドはインタフェースの中で定義できる。本体（括弧の中）が無いことを除いては
クラスと同様だ。

```
interface MyInterface1 {
       public function doSomething();
}

interface MyInterface2 {
      public function doSomethingElse();
```

```
}

interface MyInterface3 {
        public function setName($name);
}
```

複数のインタフェイスをコンマで分離して列挙することで作りこめる。使用するクラスではここで定義したメソッドの全てを書かれた通りに含んでおくことが必要だ。

```
// 有効な例
 class MyClass implements MyInterface1, MyInterface2, MyInterface3 {
     protected $name;
     public function doSomething() {
                //something を行うコード

     }
     public function doSomethingElse() {
              //something else を行うコード
     }
     public function setName($name) {
         $this->name = $name;
     }
 }

//無効な例
 class MyClass implements MyInterface1, MyInterface2, MyInterface3 {
     // doSomething() が抜けている!
     private function doSomethingElse() {
         // これは public でないといけない!     }
     public function setName() {
         // name 引数が抜けている!     }
 }
```

インタフェイスに向けたプログラミング

もし、C#を経験していれば、このことには慣れているだろう。もしそうでないなら、これで一体何の利益があるのか疑うだろう。インタフェイスによって「インタフェイスに向けたプログラミング」(Coding-to-the-Interface)と呼ばれるスタイルでプログラミングをすることが出来、また PHP ではインタフェイスは「契約」の様な働きをすると言ったのを思い出してほしい。さて、君の職歴の中でドメイン駆動（domain driven）設計だとか契約プログラミング(design-by-contract)と呼ばれるソフトウエア構築の設計技法が必要とされる場面に出会う可能性が高いと言っておこう。このような設計技法の目的はソフトウエア・コードの断片をモジュール化し、できるだけ結合がゆるくなるようにして、コードセグメント相互の依存性を最少に抑えることである。これによって１つのプロジェクトを複数の開発者で開発するときに、人の領域に入り込まずソフトウエアの矛盾を起こさないで、仕事をずっと容易にすることができる。大きなプロジェクトでのコード変更の混乱を減らし、プロジェクトの他の場所でのプログラムの流れを破壊するリスクも減らすことが出来る。また、ユニット・テストを容易にしクラスを独立の小さなプログラムとして単体でテストできることになる。

実際にいかに PHP インタフェイスが役に立つかを示すため、レストラン・メニューの応用を続けよう。すでに開発した MenuItem クラスを思い出そう：

```
class MenuItem
{
      private $menuitemid;
      private $itemname;
      private $description;
```

```
	private $price;
	private $servingsize;
	private $picture;

	public function setID($menuitemid){$this-> menuitemid = $ menuitemid;}
	public function getID(){return $this-> menuitemid;}

	public function setPrice($price){$this->price = $price;}
	public function getPrice(){return $this->price;}

	public function setPicture($picture){$this->picture = $ picture;}
	public function getPicture (){return $this->picture;}

	public function setItemName($itemname){$this->itemname = $itemname;}
	public function getItemName(){return $this->itemname;}

	public function setDescription($description){$this->description= $description;}
	public function getDescription(){return $this->description;}

	public function setServingSize($servingsize){$this->servingsize = $servingsize;}
	public function getServingSize(){return $this->servingsize;}
}
```

そして抽象 Menu クラスを作成した。

```
abstract class Menu
{
	private $menuid;
	private $parentid;
	private $menuitemid;
	private $menuname;
	private $description;

	public function setMenuID($menuid){$this->menuid = $menuid;}
	public function getMenuID(){return $this->menuid;}

	public function setParent($parentid){$this->parentid = $parentid;}
	public function getParent(){return $this->parentid;}

	public function setMenuItemID($menuitemid){$this-> menuitemid = $ menuitemid;}
	public function getMenuItemID(){return $this-> menuitemid;}

	public function setMenuName($menuname){$this->menuname = $menuname;}
	public function getMenuName(){return $this->menuname;}

	public function setDescription($description){$this->description= $description;}
	public function getDescription(){return $this->description;}
}
```

次に Menu クラスを拡張するいくつかのクラスを作成した：

```
class MainMenu extends Menu
{
//...
}

class DrinkMenu extends Menu
{
//...
}

class LunchMenu extends Menu
{
//...
}

final class DinnerMenu extends LunchMenu
```

```
{
//...
}

final class KidsMenu extends Menu
{
//...
}

final class DessertMenu extends Menu
{
//...
}
```

さて、ここで Mindi と Vellery はジレンマに陥る。ランチ・メニューと Dinner Menu には多くの共通のメニュー品目があるが、提供サイズと価格がディナーとランチで異なっている。これをあるインタフェースを作成してディナーがランチと異なってボリュームが大きく、価格も高いことを確実にしよう。

さて、ディナー用のメニューがランチ用と同じメニューで、ディナー用には提供サイズと価格を調節することにしよう。まずランチ・メニューを拡張してディナー・メニューとすることから始めよう：

```
class DinnerMenu extends LunchMenu
{
//...
}
```

次にすべきことはインタフェイスを作成してディナーの価格とボリュームをランチの品目と異なるように決めるビジネス・ルールを設定するようにしよう。さらに働いている間のサービス・タイム Happy Hour のインタフェイスも作成するとしよう。

```
interface DinnerPortion {
      public function setDinnerPortion();
}

interface DinnerPrices {
      public function setDinnerPrices();
}

interface HappyHourDrinkPrices {
      public function setHappyHourDrinkPrices();
}
```

さて、これらを全部合体させよう：

```
final class DinnerMenu extends LunchMenu implements DinnerPortion, DinnerPrices
{
      public function setDinnerPortion($menuitemObject)
       {
             $adjusted_servingsize = 1;
              $base_servingsize = $menuitemObject->getServingSize();

              //Make the dinner portion 50% bigger than the lunch portion.
             $adjusted_servingsize = $base_servingsize * 1.5;

                          return  $adjusted_servingsize;

       }

       public function setDinnerPrices($menuitemObject)
       {
             $adjusted_price = 1;
              $base_price = $menuitemObject->getPrice();
```

```
                //Make the dinner price 25% more than the lunch price.
               $adjusted_price = $base_price * 1.25;

                              return  $adjusted_price;

        }
}
```

さらにDrinkMenu に HappyHourDrinkPrices のインタフェイスを作成しよう。

```
final class HappyHourMenu extends DrinkMenu implements HappyHourDrinkPrices
{
       public function setHappyHourDrinkPrices($drinkObject)
       {
              $adjusted_price = 1;
              $base_price = $drinkObject->getPrice();

              //Make the happy hour drink prices 30% less than regular price
              $adjusted_price = $base_price * 0.7;

                              return  $adjusted_price;

       }
}
```

PHP インタフェイスは結局規則を守らせる仕組みなのだ。インタフェイスを契約と捉えるだけでなく、言語の構造を多くの関係の無いオブジェクトのクラスが持っている性質や動作で共通なものを分類する手段として使うのだと捉えることが重要だ。この点をよく理解してもらう試みとして、*インタフェイスに向けたプログラム*の意味するところをもう１つの例で示すとしよう。今回はあるゲーム応用を構築することにしよう。そのゲームは運転手が恐ろしい道路を突っ走るのだが、他にも２つの全く異なるものにも脅されるのだ。

例えば：

```
class GrizzlyBear extends Animal {
   public function GrowlAtYou(){}
   public function ChaseYou(){}
   public function PounceOnYourCar(){}
   public function EatYou(){}
 }

 class Cop extends Person {
   public function FollowYou(){}
   public function PullYouOver(){}
   public function ArrestYou(){}
   public function BeatYouSenseless(){}
 }
```

明らかに、これら２つのオブジェクトには直接のインヘリタンスとしては何も共通のものはなにもない。しかし、これらはどちらも脅威だと言える。

このゲームはプレイヤーが恐ろしい道路を突っ走るときプレイヤーを襲うある種のランダムな出来事を必要とする。それは灰色熊(Grizzly Bear)かもしれないし、警官か、さらに両方かもしれない。しかし、この両者をどうして１つの関数で扱うのか？どのようにして異なる種類のオブジェクトに同じ方法で「脅しをかけろ」と要求することができるのか？

これらを実現するキィ―は灰色熊も警官もそのモデルについては全然似ていなくても共通の漠然と解釈される行動を共有するようにすることなのだ。そのために両者が実装できるインタフェイスを造るとしよう：

```
interface IThreat {
    public function BeThreatening();
 }


class GrizzlyBear extends Animal implements IThreat {
   function GrowlAtYou() { echo "Growwwwwwl!!"; }
   function ChaseYou(){}
   function PounceOnYourCar(){}
   function EatYou(){}

   function BeThreatening() {
      GrowlAtYou();
      ChaseYou();
      PounceOnYourCar();
   }
 }

 class Cop extends Person implements IThreat {
      function FollowYou(){}
      function PullYouOver(){}
      function ArrestYou(){}
      function BeatYouSenseless(){}

   function BeThreatening() {
      FollowYou();
      PullYouOver();
      ArrestYou();
   }
 }
```

これで２つのクラスがあり、それぞれの方法で脅しをかけられる。そしてそれらは同じ元のクラスから派生して同じプロパティを持っていなくてもよい。そのかわり IThreat インタフェイスの契約を守る必要がある。契約は簡単で、BeThreatening を持てばよい。これに関しては、次のモデルをつくる：

```
class ScaryRoad {

  public function __construct() {}

   public function DriveCar(array $threatening_object) {
    //恐ろしい道路をドライブしていく間



      foreach ($threatening_object as $threat)
      {
        $threat->BeThreatening();
      }
   }
 }


//$cop1 = new Cop();
$cop1 = new Cop();
$grizzly1 = new GrizzlyBear();
$grizzly2 = new GrizzlyBear();

$road = new ScaryRoad();
$road->DriveCar($threats=array($cop1, $grizzly1, $grizzly2));
```

ここで恐ろしい道路をいくつかの脅威を受け付ける DriveCar() 関数で造ろう。インタフェースの使い方をよく見てほしい。すなわち、我々のシナリオでは脅威のアレイの要素は実際には GrizzlyBear オブジェクトであったり Cop オブジェクトであったりする。

DriveCar メソッドは運転者が恐ろしい道路をドライブしていくときに呼び出される。われわれの応用では、脅威がその働きをするときである。各々の脅威は IThreat インタフェイスの方法に従って脅すように指示される。このようにして GrizzlyBears にも Cop にもそれぞれの方法で脅しをかけさせることが容易にできる。単に ScaryRoad オブジェクトの中の何か脅威である物を持つようにすればよいだけだ。それが何であるかは構わないし、それらが他のものと共通に持つべきものはない。

同様のサンプルをみよう：

Let's look at a similar example:

```
class AttractiveStranger extends Person {
  public function LookAtYou(){}
  public function SmileAtYou(){}
  public function TalkToYou(){}
}

class Pepper extends Vegetable {
  public function BurnYourTongue(){}
  public function CauseBathroomEmergency(){}
}
```

明らかに、これら２つのオブジェクトには直接インヘリタンスによる共通のものはなにもない。

しかし、どちらも汗をかくはめにしてくれると言えるだろう。

ゲームはそのプレイヤーがバーにいるときに汗をかかせる何らかのランダムな事象を持っているとしよう。それは魅力的な見知らぬ人（AttractiveStranger）かもしれないし、胡椒か、それら両方かもしれない。しかし、どうすれば１つの関数でどちらも起こさせることが出来るだろうか？いかにして異なるタイプのオブジェクトの各々に「汗をかかせることを実行しろ」と要求できるのだろうか？

それを実現させるキィ―は AttractiveStranger と Pepper がモデルにおいて全く似ていなくても漠然と解釈される行動を共有することである。そういうことだからどちらもが実装できるインタフェイスを１つ造ろう：

```
interface ISweat {
    function MakeYouSweat();
}


class AttractiveStranger extends Person implements ISweat {
  public function LookAtYou(){}
  public function SmileAtYou(){}
  public function TalkToYou(){}

  public function MakeYouSweat() {
    LookAtYou();
    SmileAtYou();
    TalkToYou();
  }
}

class Pepper extends Vegetable implements ISweat {
  public function BurnYourTongue();
  public function CauseBathroomEmergency();

  public function MakeYouSweat() {
    BurnYourTongue();
    CauseBathroomEmergency();
  }
}
```

これで２つのクラスがそれぞれの方法で汗をかかせることが出来ようになった。しかも、その２つは同じ元のクラスから派生して共通の生来のプロパティを持っている必要はない―単に ISweat の契約を守ればよいのだ。このことを注意して、次のモデルを造ろう:

```
class CollegeBar implements ISweat{

   __construct() { ... }

   void SitAtBar() {
    // バーに座っているときに


   $attractivestranger = new AttractiveStranger();
   $hotpepper = new Pepper();

   $attractivestranger->MakeYouSweat();
   $hotpepper-> MakeYouSweat();
   }

 }
```

ここで、幾人かの男といくつかの汗をかかせるものを受け入れているカレッジのバーの登場だ。インタフェイスの使用法に注目しよう。われわれのシナリオでは、thingsThatMakeYouSweat のアレイの要素が実際には AttractiveStranger のオブジェクトであっても Pepper のオブジェクトであってもよい。

SitAtBar メソッドが男がバーに座っているときに呼び出される。われわれの応用では、thingsThatMakeYouSweat がその働きをするときである。thingsThatMakeYouSweat の各々はバーに座っている男に ISweat のインタフェイスの方法で汗をかかせるように指示される。このようにして AttractiveStranger と Pepper のどちらにもそれぞれに固有の方法で汗をかかせることが容易にできるのだ。単に CollegeBar オブジェクトの内部にある thingsThatMakesYouSweat である何かがあるようにすればよい。それが実際何であっても構わない、それらが互いに共通のものを持っていなくてもよいのだ。

これが PHP インタフェイスの必要性と使用法を明らかにする助けになり、また「インタフェイスに向けたプログラム」の意味するところを説明する助けにもなって欲しい。

第６章　Static メソッドとプロパティ

辞書を引くと‘static’という語は、固定したあるいは安定した状態、変化が少ないか全くないことに関係する、あるいは特徴づけられるという意味とされている。オブジェクト指向 PHP における‘static’という単語は同様に static のプロパティやメソッドがダイナミックな方法では決して使われないということを意味する。Static なプロパティやメソッドはクラスで定義されるが、クラスから派生されるオブジェクトにカプセル化されることはない。繰り返そう。Static なプロパティとメソッドはもし public ならその static プロパティとメソッドが宣言されているクラスからオブジェクトを生成することなくプログラムのどの場所からでもアクセスが可能である。キィーワード static はクラス・レベルの共用メソッドを作成したり、同じ型のオブジェクトの間でデータを共用したりするのに便利である。

キィーワード static の使用は初めて学ぶ人には混乱のもとになるかもしれない。よくある最初の質問は：static なプロパティは global な変数と同じなのですか？static メソッドの中のキィーワード self と static の違いはなんですか？static なプロパティやメソッドを呼びだす方法は？static なメソッドは拡張できますか？static プロパティやメソッドの実用的な利用法はなんですか？これらは全て良い質問なので、この短い章の中で答えていくことにする。まず最初に、static プロパティとインスタンスで生成されたクラスのダイナミックなプロパティとを比較できる例を示そう。もう一度 Person クラスをつかって Mindi と Vallery の新しい友達を生成しよう：

$tmpPerson = new Person();
$tmpPerson->Name = “Marsha”

Marsha と名付けた Person クラスのインスタンスを生成して Marsha という名の新しい娘をつくったところだ。次に tmpMarsha オブジェクトの”name” プロパティの値に”Marsha”を設定した。さて、PHP にキィーワード static を導入すると、そのオブジェクトだけでなくクラスの前後関係を通してメソッドやプロパティへアクセスできる（注：そのメソッド・プロパティは最初に static と宣言して置く必要がある）。すなわち、オブジェクトを生成しなくても Name プロパティにアクセスできるのだ。例えば:

```
class Person {

    static public $Name = "Marsha";

    static public function helloWorld() {

        print "Hello world from " . self::$Name;

    }

}
```

メソッドやプロパティを static であると宣言するにはキィーワード static を使わなければならない。クラスの領域の外部からプロパティにアクセスするには、クラスの名称に続けて２つのコロン(::)、そしてプロパティの名称をつなげて用いる。２つのコロンの使用は領域解決オペレータ**(SCOPE RESOLUTION OPERATOR)**と呼ばれる。例えば：

```
print Person::$Name . "\n";

Person::helloWorld();


Output:
Marsha
Hello world from Marsha
```

２つ上のサンプルで self　のキィーワードをクラスの内部からメンバーである変数やプロパティにアクセスするために使ったのを見たはずだ。

static メソッドはクラスの領域の中にあって、通常のメソッドやプロパティ（すなわち static　と宣言していない）にはアクセスできない。というのはそれらはオブジェクトに属するべきもので、クラスにではないからである。その結果 static メソッドの内部からは$this 変数を使うことはできない。それは static メソッドはオブジェクトのつながりの中で呼び出されるのではないからだ。その代りに self を使うことになるのだ。

レストランのメニューへの応用に戻って、有用な static プロパティとメソッドを付け加えよう。特定のメニューのクラスを生成するために abstruct Menu クラスを拡張したのを思い出そう。それらのクラスの各々にそれら特定のメニューを発生させたときにタイトルをセットするように static プロパティを加えよう。同時に今の応用に追加する Database クラスの中に static メソッドを採用しよう。

Static プロパティ

下に掲げた MainMenu, DrinkMenu, LunchMenu, KidsMenu, DessertMenu, DinnerMenu そして HappyHourMenu を見よう。Static で public な$title と名付けるプロパティを加えよう。この static なプロパティはメニューのオブジェクトを生成しなくてもそのテキストをメニューの見出しとして使うことを可能にする。

```
<?php
class MenuItem
{
                private $menuitemid;
                private $itemname;
                private $description;
                private $picture;
                private $servingsize;
                private $price;

                public function setID($menuitemid){$this-> menuitemid = $menuitemid;}
                public function getID(){return $this-> menuitemid;}

                public function setPrice($price){$this->price = $price;}
                public function getPrice(){return $this->price;}

                public function setPicture($picture){$this->picture = $picture;}
                public function getPicture (){return $this->picture;}

                public function setItemName($itemname){$this->itemname = $itemname;}
                public function getItemName(){return $this->itemname;}

                public function setDescription($description){$this->description= $description;}
                public function getDescription(){return $this->description;}

                public function setServingSize($servingsize){$this->servingsize = $servingsize;}
                public function getServingSize(){return $this->servingsize;}
}


abstract class Menu
{
                private $menuid;
                private $menuitemid;
                private $menuname;
                private $description;

                public function setMenuID($menuid){$this->menuid = $menuid;}
                public function getMenuID(){return $this->menuid;}
```

```
		public function setMenuItemID($menuitemid){$this-> menuitemid = $menuitemid;}
		public function getMenuItemID(){return $this-> menuitemid;}

		public function setMenuName($menuname){$this->menuname = $menuname;}
		public function getMenuName(){return $this->menuname;}

		public function setDescription($description){$this->description= $description;}
		public function getDescription(){return $this->description;}
}



class MainMenu extends Menu
{
		static public $title="<b><font color=blue>Main Menu</font></b>";
}

class DrinkMenu extends Menu
{
		static public $title="<b><font color=yellow>Drink Menu</font></b>";
}

class LunchMenu extends Menu
{
		static public $title="<b><font color=green>Lunch Menu</font></b>";
}

Class KidsMenu extends Menu
{
		static public $title="<b><font color=orange>Kids Menu</font></b>";
}

Class DessertMenu extends Menu
{
		static public $title="<b><font color=red>Dessert Menu</font></b>";
}



interface DinnerPortion {
		public function setDinnerPortion($itemobject);
}

interface DinnerPrices {
		public function setDinnerPrices($itemobject);
}

interface HappyHourDrinkPrices {
		public function setHappyHourDrinkPrices($itemobject);
}



final class HappyHourMenu extends DrinkMenu implements HappyHourDrinkPrices
{
		static public $title="<b><font color=orange>Happy Hour Drink Menu</font></b>";

		public function setHappyHourDrinkPrices($menuitemObject)
		{
			$adjusted_price = 1;
			$base_price = $menuitemObject->getPrice();

			//Make the dinner price 30% less than the normal price.
			$adjusted_price = ($base_price * 0.7);

			return  round($adjusted_price,2);
		}
}
```

```
final class DinnerMenu extends LunchMenu implements DinnerPortion, DinnerPrices
{
		static public $title="<b><font color=blue>Dinner Menu</font></b>";

		public function setDinnerPortion($menuitemObject)
		{
			$adjusted_servingsize = 1;
			$base_servingsize = $menuitemObject->getServingSize();

			//Make the dinner portion 50% bigger than the lunch portion.
			$adjusted_servingsize = $base_servingsize * 1.5;

			return  $adjusted_servingsize;
		}

		public function setDinnerPrices($menuitemObject)
		{
			$adjusted_price = 1;
			$base_price = $menuitemObject->getPrice();

			//Make the dinner price 25% more than the lunch price.
			$adjusted_price = ($base_price * 1.25);

			return  round($adjusted_price,2);
		}
}
```

さて、第２、４，５とこの章で議論した概念を簡単なテストで適用しよう。ディナー・メニューはランチ・メニューから拡張したものだった。そしてランチの価格とディナーの価格の間に区別を設けるために DinnerPrice インタフェイスを実装した。さらに、次のテスト・コードの最後の部分で DinnerMenu のタイトルを調節した、すなわち、オブジェクトを生成することなく static public な $title プロパティを変更していることに注目しよう。

```
// テスト
// メニュー品目のオブジェクトを生成する
$tmpMenuItem = new MenuItem();
$tmpMenuItem->setItemName('Atlantic Yellow Fin Tuna');
$tmpMenuItem->setPrice('14.75');

echo LunchMenu::$title . "<BR>";
echo $tmpMenuItem->getItemName() . ' - $' . $tmpMenuItem->getPrice();
echo "<BR><BR>";

// DinnerMenu で実装した setDinnerPrice メソッドを使う
// DinnerPrice インタフェイスで宣言したクラス
$tmpDinnerMenu = new DinnerMenu();
echo DinnerMenu::$title . "<BR>";
echo $tmpMenuItem->getItemName() . ' - $' . $tmpDinnerMenu->setDinnerPrices($tmpMenuItem);
echo "<BR><BR>";

//DinnerMenu クラスの static $title 変数を変えることでタイトルを変える。
DinnerMenu::$title="<font color=orange><i><b>Dinner</b></i></font>";
echo DinnerMenu::$title . "<BR>";
echo $tmpMenuItem->getItemName() . ' - $' . $tmpDinnerMenu->setDinnerPrices($tmpMenuItem);
echo "<BR><BR>";
?>
```

図６－１． Eclips PDT のデバッグ出力

簡潔であることはしばしば複雑な問題への最良の解決策になる。わざと他の人に解読不可能にするためにコードをわざと巨大なスパゲッティのもつれた塊にしてしまうプログラマーに出会ったことがあった。もちろんそういうプログラマーは自身の価値を高くし仕事を確保しようとしてそうしているのである。君がこの本を読んで、得た知識を設計のパターンやより進化したソフトウエア・アーキテクチャーに生かしていくのなら、そのような離れ業をする必要はないと断言する。君の価値は、ビジネス・モデルと要求を理解し、何もないところから開発する能力にカプセル化されるのだ。それは決してまるで底引き網のように長年たったプログラムを守って開発することではないのだ。

static メソッド

ということなので、static メソッドの話に戻ろう。static メソッドはオブジェクトではなくクラスから直接に呼び出すことが出来る。Static プロパティと同じだ。この章の初めに述べたように、static 修飾子にまつわるよくある質問は static なメソッドやプロパティはどう役に立つのか？具体的にどう使うのか？というものだ。

一つ言えることは、データベース機能が static 機能の良い利用方法ということだ。データベース全体に直接アクセスする必要はない、次のように結合資源にアクセスするだけでよいのだ。

```
$result = StaticClass::getDatabaseConnection()->query();
```

しかしのちほど保存をするためにクラスにアクセスする必要があったり、同じオブジェクトの複数のインスタンス化をするなら、static にはしない方が良いだろう。クラスはグローバルな領域にあり、したがってコードがどの領域にあっても、コードのどの場所にあってもアクセスできるのだ。

```
function getUsers()
(
        $users  =  StaticClass::getDatabaseConnection()->query( ‘SELECT ・FROM users’ );
)
```

例：異なる引数をとるかも知れないオブジェクトの複数のインスタンスを生成するのに static メソッドを用いる。

```
Class DBConnection
{
    public static function createFromConfiguration(Configuration $config)
    {
        $conn = new DBConnection();
```

```
        $conn->setDsn($config->getDBDsn());
        $conn->setUser($config->getDBUser());
        $conn->setPassword($config->getDBPass());

        return $conn;
    }

    public static function newConnection($dsn, $user, $password)
    {
        $conn = new DBConnection();
        $conn->setDsn($dsn);
        $conn->setUser($user);
        $conn->setPassword($password);

        return $conn;
    }
}
```

レストラン・メニューの応用でデータベースが必要となってくるので、次の方式を持つものを 1 つ生成し、データベースとのやり取りの全てのを取り扱う Database クラスを生成しよう。私はデータベースの管理を行うのに MySQL Workbench Community Edition を気に入っている。最新のリリースは version 5.2 だ。MySQL Workbench は次のサイトから無料でダウンロードしてインストールできる。
http://dev.mysql.com/downloads/workbench/

MySQL WB 5.2 CE をインストールすると、”New Connection” のリンクをクリックして Set Up New Connection のウィザードを開き、 “localhost” connection を生成する。

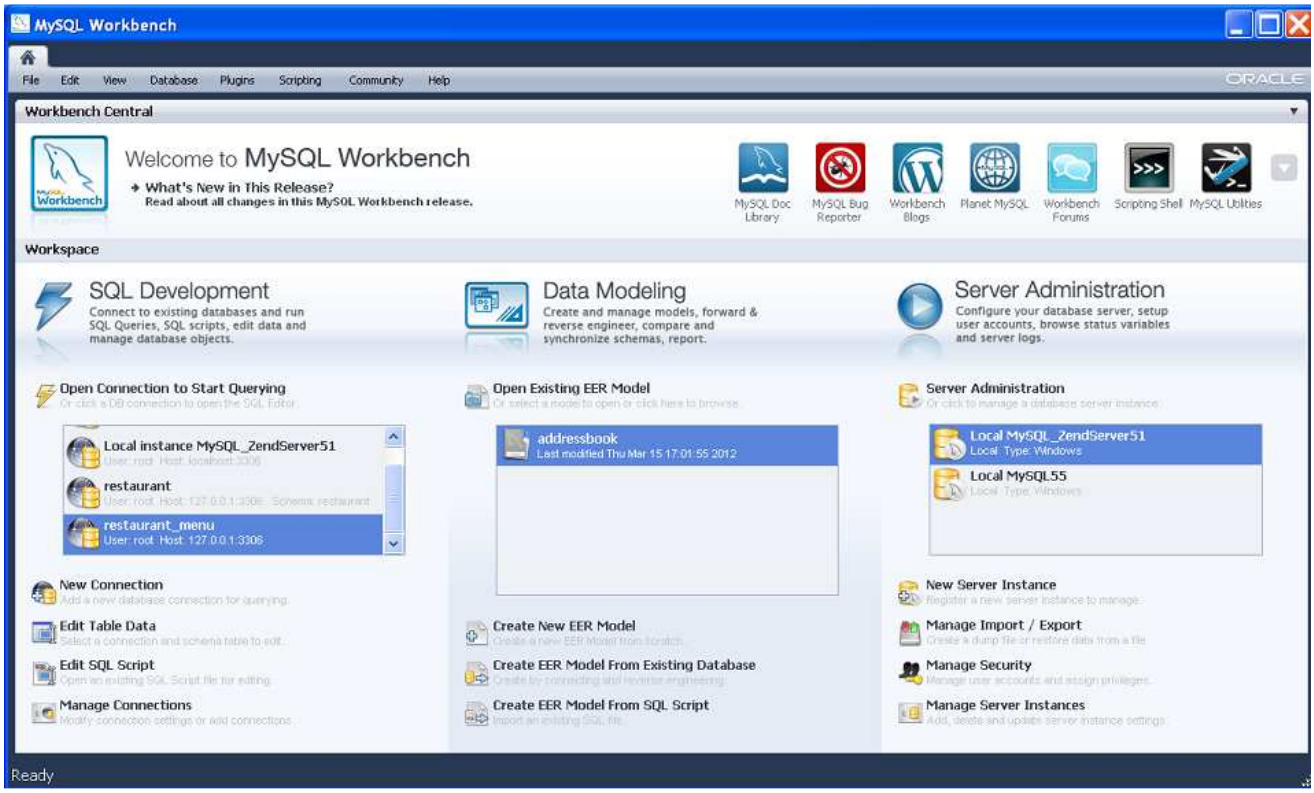


図 6-2. MySQL Workbench 5.2 CE Workbench コントロール・パネル

```
CREATE DATABASE `restaurant` /*!40100 DEFAULT CHARACTER SET utf8 */

CREATE TABLE `menuitem` (
```

```
  `menuitemid` int(11) NOT NULL AUTO_INCREMENT,
  `parentid` int(11) NOT NULL,
  `itemname` varchar(255) NOT NULL,
  `description` varchar(255) DEFAULT NULL,
  `servingsize` varchar(255) DEFAULT NULL,
  `picture` varchar(255) DEFAULT NULL,
  `price` varchar(45) NOT NULL,
  PRIMARY KEY (`menuitemid`),
  UNIQUE KEY `menuitemid_UNIQUE` (`menuitemid`)
) ENGINE=MyISAM AUTO_INCREMENT=17 DEFAULT CHARSET=utf8$$


CREATE TABLE `menuitemtomenu` (
  `id` int(11) NOT NULL AUTO_INCREMENT,
  `menuitemid` int(11) NOT NULL,
  `menuid` int(11) NOT NULL,
  PRIMARY KEY (`id`,`menuitemid`,`menuid`),
  UNIQUE KEY `idmenuitemtomenu_UNIQUE` (`id`)
) ENGINE=InnoDB AUTO_INCREMENT=10 DEFAULT CHARSET=utf8$$
```

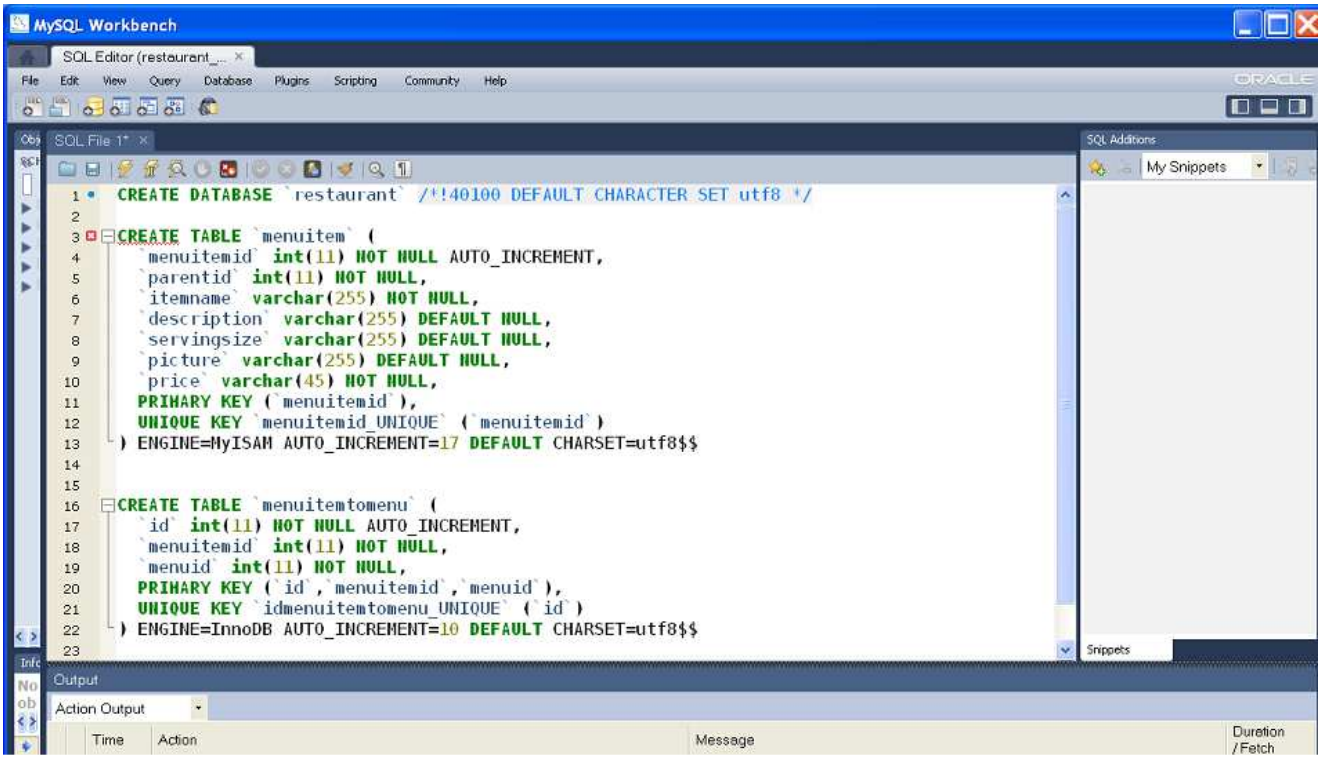


図 6-3. MySQL Workbench 5.2 CE での RestaurantMenuManager.sql ファイル

```
<?php
//Database.php
require_once('config.php');

class Database
{
    public $connection;
                private function Database()
                {
                   $databaseName = $GLOBALS['configuration']['db'];
                   $serverName = $GLOBALS['configuration']['host'];
                   $databaseUser = $GLOBALS['configuration']['user'];
                   $databasePassword = $GLOBALS['configuration']['pass'];
                   $databasePort = $GLOBALS['configuration']['port'];

                   $this->connection = mysql_connect ($serverName.":".$databasePort, $databaseUser,
$databasePassword);
                        mysql_set_charset('latin1',$this->connection);
```

```
                if ($this->connection)
                {
                        if (!mysql_select_db ($databaseName))
                        {
                        throw new Exception('Cannot find: "' . $databaseName . '"');
                        }
                }
                else
                {
                        throw new Exception('Cannot connect to the database.');
                }
        }

        public static function Connect()
        {
                static $database = null;
                if (!isset($database))
                {
                        $database = new Database();
                }
                return $database->connection;
        }

        public static function Reader($query, $connection)
        {
                $cursor = mysql_query($query, $connection);
                return $cursor;
        }

        public static function Read($cursor)
        {
                return mysql_fetch_assoc($cursor);
        }

        public static function Query($query, $connection)
        {
                $result = mysql_query($query, $connection);
                return mysql_num_rows($result);
        }

        public static function InsertOrUpdate($query, $connection)
        {
                $result = mysql_query($query, $connection);
                return intval(mysql_insert_id($connection));
        }
}
?>
```

ここで、メニュー品目のクラスを「再分解」し、第３章で説明した__set(),__get() や__isset() といったマジック・メソッドを用いて抽象化しなくても抽象化の本質を与えるとしよう。

```
Class MenuItem
{
        // コンストラクタ(実装されていない)
                public function _construct(){}

        // 宣言なしのプロパティの設定
                function __set($property, $value)
                {
                $this->$property = $value;
                }

        // 定義されたプロパティの獲得

                function __get($property)
                {
                        if (isset($this->$property))
```

```
                {
                        return $this->$property;
                }
        }
}
```

これでMenuItem クラスはすっきりしてより柔軟になり、上記のレストランのデータベースとの連携がスムーズになった。メニューの表を必要としないことに注目したい。メニュー品目の表と特定のメニューをメニュー品目に関係づける MenuItemToMenu と呼ぶ橋渡しの表とがあればよい。この設計で個別のメニュー品目が複数のメニューに使われることが可能になり、第４章でしたようにインタフェイスの使用でメニュー品目の提供サイズと価格を特定のメニューに応じて変えることも可能にする。第７章でもう一度このレストランのメニューへの応用に戻り例外の扱いを可能にしたうえで全てをまとめるとしよう。

シングルトン設計パターン

Static メソッドのもう一つの実用的な利用方法はシングルトン(Singleton)設計パターンの活用だ。シングルトン・パターンはクラスのインスタンス化を１つのオブジェクトだけに限定する設計の形態である。

シングルトンは数通りの方法で有用である。第１に、シングルトンはグローバル変数に代わってデータをプログラム全体で使えるようにする方法としてよく採用される。その理由はいくつかある。最初に、グローバル変数に頼っているクラスはある応用から他の応用へと再利用する際、新応用に於いて同じグローバル変数を定義することを確実にする必要があってこれが面倒になってくる。第２の理由はグローバル変数は通常リソースを食うが、シングルトン・パターンなら適当な割当てと初期化でもよいのだ。第３の理由はグローバルではグローバルの名称スペースあるいはPHP 5.3+ の名称スペースの割当ての煩雑さを免れない。名称スペースの問題は第８章で取り上げる。

シングルトン・パターンはまた、抽象工場、ビルダー、プロトタイプさらに外観設計への活用が可能なのだ。設計パターンの問題はこの本の範囲を超えているが、いくつかの設計パターンへの有用な参考文献を付録に入れておく。これらの参考文献を読んでみることを特に勧める。設計パターンはソフトウエア開発の統合部分だから。

シングルトン・パターンの実装は、クラスを生成し、もしまだないならば新しいそのインスタンスを造るメソッドを造りこんでおくことで行われる。すでに１つのインスタンスが存在するのなら、そのオブジェクトへの参照を戻すだけとなる。他の方法でオブジェクトが生成されることの無いよう、コンストラクターはprivate にされる。また$instance プロパティはprivate かつ static に設定することにも注意したい。このサンプルのクラスでは、$instance プロパティがシングルトン・オブジェクトのインスタンスを保持し、戻り値とする。

シングルトン：

```
class SingletonClass {
    private static $instance;
    private function __construct() { }
    public function __clone() {
        trigger_error('Clone is not allowed.', E_USER_ERROR);
    }
    public static function init() {
        if (!isset(self::$instance)) {
            $c = __CLASS__;
            self::$instance = new $c;
        }
        return self::$instance;
```

```
    }
// その他のシングルトンのための public, dynamic メソッド
}

$singleton = SingletonClass::init();
```

いかに static なプロパティとメソッドが有用かを示す次のサンプルでは、クラスのオブジェクトあるいはインスタンスの数を調べる。このサンプルは非常に基礎的なもので、することと言えば public で static なプロパティ$instance をクラスのコンストラクターがコールされる毎に１づつカウント・アップするというものだ。同様に、デストラクターがコールされる毎に１づつカウント・ダウンする。これが可能なのは Static の$instance はそのクラスから生成されたオブジェクトの中ではなく CountInstanceOfMyself クラスの中に留まっているからである。このようにして public で static なプロパティがあたかもグローバル変数であるかのようにして使われるのである。レストラン・メニューの応用においてメニューのクラスで public で static な変数の場合もそうだった。

```
Class CountInstancesOfMyself {
    public static $instances = 0;
    public function __construct() {
        CountInstancesOfMyself::$instances++;
    }
    public function __destruct() {
        CountInstancesOfMyself::$instances--;
    }
}
$a = new CountInstancesOfMyself();
$b = new CountInstancesOfMyself();
$c = new CountInstancesOfMyself();

echo CountMe::$instances;
```

Output:
3

実行時 static 結合

実行時 static 結合は PHP 5.3 で使える特徴で、static なインヘリタンスの状況に応じて呼び出されたクラスへの参照を行うものである。結局、実行時 static 結合は簡単に キィーワード static 対キィーワード self の違いで説明できる。

どちらのキィーワードも似ているが、self はそれを含んでいるクラス、static は呼び出されたクラスを指すところが違う。簡単な例で説明しよう。空っぽの家あるいはアパートがあり、そこに何か物を置きたいとしよう。空の場所に持ち込みたい物を１つの abstract な HousholdObject であるとして、そこから独身者の部屋に必要な基本的な物を与えるいくつかの子のクラスをを生成する：

```
abstract class HouseholdObject{

}


class Couch extends HouseholdObject {
      Public static function create() {
            return new Couch();
      }
}
```

```
class FlatScreenTV extends HouseholdObject {
      Public static function create() {
            return new FlatScreenTV ();
      }
}

class Refrigerator extends HouseholdObject {
      Public static function create() {
            return new Refrigerator ();
      }
}
```

create() 関数は HousholdObject の元のクラスから拡張された全てのクラスに共通なので、create() 関数を親のクラスに移しておいて、子のクラスに拡張するのが実用的である。そうすれば 1 回タイプするだけでよい。create() 関数の中でキィーワード　self() をクラスへの参照として使っていることに注目したい：

```
abstract class HouseholdObject{
      Public static function create() {
            return new self();
      }

}

class Couch extends HouseholdObject {

}

class Table extends HouseholdObject {

}

class Refrigerator extends HouseholdObject {

}
```

Self() をクラスへの参照として戻すことの問題点は、create() 関数が子のクラスから呼び出されたときに、領域の解決操作をするとエラーとなることだ。

```
Couch::create();
Table::create();
```

Output:
Fatal error: Cannot instantiate abstract class HouseholdObject

これはキィーワード self を使い、それが親のクラス abstruct　HouseholdObject クラスを参照するからだ。Creat()関数が Couch、Table、そして Refrigerator　のクラスに拡張し、Couch や Table や Refrigerator を生成　するために呼び出すとき、それは HouseholdObject を代わりに生成するからだ。この create 関数はそれを呼び出すクラスを参照するのではなく、その代わりに abstruct クラスを呼び出すのだ。思い出してくれるとよいが、abstruct クラスはインスタンス化されないので、PHP エンジンはエラーを出すのだ。

さて、実行時 static 結合を用いてこれを再度試してみよう。今回は create 関数は static な参照を戻す。この static な参照は親の HouseholdObject クラスではなく、create() 関数を呼び出すクラスを参照する。

```
abstract class HouseholdObject{
      Public static function create() {
            Return new static();
      }
}

class Couch extends HouseholdObject {

}

class Table extends HouseholdObject {

}

class Refrigerator extends HouseholdObject {

}
```

第 7 章　PHP のエラー制御と例外の取り扱い

エラーと例外の取り扱いに関する話題はこの本で一番楽しくない話題だ。しかし、もっとも重大な意味を持つものなのだ。なぜなら、それは君がそれにふさわしい配慮をするなら開発者として最悪のミスを救ってくれるからだ。エラーは起こるもので、避け難い。しかし、心構えがあれば助けられることもあり、開発者として最も起こりそうな問題を注意深く考慮すれば、ソフトウエアの開発に成功する可能性を高めることができるのだ。Javascript form validation のような有用なツールと連携すれば、誘発するエラーを予防する助けになる。予想されるエラーと例外を扱うコードを開発しておけば大惨事を予防する助けになる。PHP 5 ではエラー報告と設定指示に加えて好都合な例外取扱い機能が含まれている。まず例外処理機能から始めよう。

多くの設定指示が PHP のエラー報告処理を定める。そのいくつかを最後に紹介するが、まず話を進めよう。退屈だと思ったら、最低限それがどんなものか、この本のどこに書かれているかだけでも思い出すとよい。そうすれば必要となった時に見ることができる。

組み込み例外クラス

PHP 5 以降では例外は特別なオブジェクトで PHP の Exception クラスあるいは PHP Exception クラスに拡張するクラスからインスタンス化される。Exception の型のオブジェクトの目的はエラーに関する情報を保持し報告することである。Exception クラスのコンストラクターはメッセージ文字列とエラー・コードの 2 つの引数を受け取るオプションがある。Exception クラスは数個の public なメソッドを含んでいて、エラーの状況を判断するのに役立つ。Exception クラスを簡単に紹介し、そしてそれぞれのメソッドの使用法を示すとしよう。

```
<?php
class Exception
{
    protected $message = 'Unknown exception';   //  例外のメッセージ
    private   $string;                          //  __toString のキャッシュ

    protected $code = 0;                        //  ユーザ定義の例外コード

    protected $file;                            //  例外のソースのファイル名
    protected $line;                            //  例外のソースの行
    private   $trace;                           //  遡ってのトレース
    private   $previous;                        //  入れ子の例外では前の例外

    public function __construct($message = null, $code = 0, Exception $previous = null);

    final private function __clone();           //  例外の複製の抑止

    Final public  function getMessage();        //  コンストラクタに渡されたメッセージの文字列を得る

    final public  function getCode();           //  コンストラクタに渡されたエラー・コードを得る
    final public  function getFile();           //  ソースのファイル名を得る
    final public  function getLine();           //  ソースの行を得る
    final public  function getTrace();          //  遡ってのトレースの多次元行列を得る
    final public  function getPrevious();       //  前の例外を得る
    final public  function getTraceAsString();  //  トレースで戻ったデータのフォーマットされた文字列を得る

    //*  オーバーライド可能  */
    public function __toString();               //  表示のためのフォーマットされた文字列}
?>
```

先ずデータベースとのやり取りをするためにレストラン・メニューの応用で作った Database クラスを見てみよう。データベースへの接続を試みたが、設定ファイルで指定した名前のデータベースが見つからなかったり、同じく設定ファイルにあったその他の接続の資格認定情報が間違っていたりして失敗した時に、Exception クラスを使うのに注目されたい。

例外の発生

```
<?php
//Database.php

require_once('config.php');

class Database
{
	public $connection;

	private function Database()
	{
		$databaseName = $GLOBALS['configuration']['db'];
		$serverName = $GLOBALS['configuration']['host'];
		$databaseUser = $GLOBALS['configuration']['user'];
		$databasePassword = $GLOBALS['configuration']['pass'];
		$databasePort = $GLOBALS['configuration']['port'];

		$this->connection = mysql_connect ($serverName.":".$databasePort, $databaseUser,
$databasePassword);
		mysql_set_charset('latin1',$this->connection);
		if ($this->connection)
		{
			if (!mysql_select_db ($databaseName))
			{
				throw new Exception('I cannot find the specified database "'.$databaseName.'".
Please edit configuration.php.');
			}
		}
		else
		{
			throw new Exception('I cannot connect to the database. Please edit configuration.php with
your database configuration.');
		}
	}

	public static function Connect()
	{
		static $database = null;
		if (!isset($database))
		{
			$database = new Database();
		}
		return $database->connection;
	}

	public static function Reader($query, $connection)
	{
		$cursor = mysql_query($query, $connection);
		return $cursor;
	}

	public static function Read($cursor)
	{
		return mysql_fetch_assoc($cursor);
	}

	public static function NonQuery($query, $connection)
	{
		mysql_query($query, $connection);
		$result = mysql_affected_rows($connection);
		if ($result == -1)
		{
			return false;
		}
		return $result;
```

```
        }

        public static function Query($query, $connection)
        {
                $result = mysql_query($query, $connection);
                return mysql_num_rows($result);
        }

        public static function InsertOrUpdate($query, $connection)
        {
                $result = mysql_query($query, $connection);
                return intval(mysql_insert_id($connection));
        }
}
```

図 7-1. トライ—キャッチ—投入

通常例外処理は try と catch のブロックに造りこむだろう。実行したいことは try のブロックの中にあり、コードの正常な実行に対する例外の処理は catch ブロックに入れるだろう try と catch のキィーワードはマーカー節と呼ばれ例外処理の領域を定義する。catch ブロックの特別な特徴は Exception オブジェクトを入力変数として受入れ次のようにException クラスのメソッドを使うことが出来ることだ：

```
class MyClass
{
    public function doSomething()
    {
        //...
        try
        {
            // 何かをトライする
        }
        catch (Exception $e)
        {
            // 例外を処理/報告する
        }
    }
}
```

さて、さらにfinally ブロックを加えてこの例の関数の出力を助け、またこの構造にthrow と呼ぶ例外を起こすものを追加しよう：

```
class MyClass
{
    public function doSomething()
    {

        $filename = '/path/to/test.txt';
```

```
        try
        {
           if !(file_exists($filename))
           {
                throw new Exception("file does not exist.");
           }
        }
        catch (Exception $e)
        {
               echo "Exception caught: " . $e->getMessage() . “<\ br>”;
               echo "Exception code: " . $e->getCode() . “<\ br>”;
        }
        finally
        {
                        // ファイルがなければそれを生成する

               $handle = fopen($filename, 'w');
               fclose($handle);
        }
    }
}
```

エラーへの感度のレベルを適度に設定する

次を注意深く読んでほしい。次のサンプルはエラー報告の設定をする方法を示している。次の 2 行をコード本体の先頭に入れる。2 行目は error_reporting 指示と呼ばれる。error_reporting 指示は報告の感度レベル(sensitivity level)を決めるのに使われる。

```
<?php

ini_set('display_errors',1);
error_reporting(E_ALL);

// コード本体
...

?>
```

もう一つの方法は php.ini ファイルを編集して次の 1 行を入れることだ：

```
error_reporting  =  E-ALL
```

ある 1 つのコードについてエラー報告をやめるには、次の 1 行を入れる：

```
error_reporting(0);
```

14 の異なったレベルが利用でき、その組み合わせもまた可能である。次のこれらのレベルのすべてを列挙したテーブルを示す。各レベルはそれより下の全てのレベルを含んでいることに注意したい。例えば、E_ALL のレベルはテーブルでその下にある他の 13 レベルから出る全ての メッセージを報告する。

エラー報告感度レベル：

エラーレベル	記述
E_ALL	全てのエラーと警告
E_COMPILE_ERROR	致命的コンパイル時エラー

E_COMPILE_WARNING	コンパイル時警告
E_CORE_ERROR	PHP 初期スタート中に発生した致命的エラー
E_CORE_WARNING	PHP 初期スタート中に発生した警告
E_ERROR	致命的実行時エラー
E_NOTICE	実行時注意
E_PARSE	コンパイル時文法エラー
E_RECOVERABLE_ERROR	致命的に近いエラー(PHP5.2 で導入)
E_STRICT	PHP バージョン移行可能性に関する忠告 (PHP 5.0 で導入)
E_USER_ERROR	ユーザで引き起こしたエラー
E_USER_NOTICE	ユーザで引き起こした注意
E_USER_WARNING	ユーザで引き起こした警告
E_WARNING	実行時警告

表 7-1. PHP のエラー報告感度レベル

PHP 5 で導入された E_STRICT は正しいコーディング方法かどうかのコア開発者の判定に基づいてコード変更の提案をし、また PHP のバージョン間の移行性を確認しようとする。もしルールを無視した関数やシンタックスを使用したり、誤った参照をしたり、クラスのフィールドに領域のレベルでなく変数定義を用いたり、その他のスタイル上の矛盾を持ち込んだりすると、E_STRICT はそのことに注意を喚起する。

PHP 6 では E_STRICT が E_ALL に統合されているので error_reporting を E_ALL に設定してユーザが引き起こす警告を見る必要がある。

開発フェーズで便利な全てのエラーの報告をするためには、指示の設定を次のようにする：

error_reporting = E_ALL

実行時の致命的エラー、シンタックス・エラー、コアのエラーだけを気にかけるのであれば、論理演算子を用いて指示を次のようにする：

error_reporting E_ERROR| E_PARSE | E_CORE_ERROR

ユーザが引き起こすエラー以外の全てのエラーを報告するのであれば、次のようにする：

error_reporting E_ALL & ~(E_USER_ERROR | E_USER_WARNING | E_USER_NPTICE)

error_reporting の指示でチィルダー文字(~)を論理演算子 NOT に使うわけではない。

立ち上げ時のエラーの表示

display_startup_errors　の指示をオンに設定すると PHP エンジンの初期化中に起きたエラーを全て表示してくれる。display_erros と同様テスト中はこの指示をオンにし、サイトが公開中にはオフとする。

エラーのログ

エラーは常にログ(経時記録)されるべきである。なぜならその記録は応用と PHP エンジンに特定の問題を決定するために最も価値ある手段を提供してくれるからだ。だから、log_errors を常に可能にしておくべきだ。この記録がどこに保存されているかは error_log の指示による。

ログを見つける

エラーはシステムの syslog に送られるか、error_log 指示で管理者が設定した特定のファイルにおくられる。もし syslog に設定されていれば、エラー情報は Linux の syslog あるいは Windows の event log に送られる。

syslog に慣れていなければ、それは Linux のログ設備で API にシステムと応用の実行に適切なメッセージを記録する。Windows の event log は本質的には Linux の syslog と等価である。それらの記録はいずれも Event Viewer を用いてみることが出来る。

ログの行の最大長さの設定

log_errors_max_len 指示は各ログの最大長をバイトで設定する。デフォルト値は 1,024 バイトである。これを 0 とすると、最大長の設定が無いことを意味する。

繰り返しエラーの無視

ignore_repeated_errors を可能に設定すると、PHP は同じファイルの同じ行で発生する繰り返しのエラーメッセージを無視する。

同じ場所から発生しているエラーの無視

ignore_repeated_source を可能に設定すると PHP は異なったファイルからの、あるいは同じファイルの異なった行から発生している繰り返しエラーを無視する。

変数の最近のエラーの記録

track_errors を可能に設定すると PHP は変数$php_errormsg に最近のメッセージを記録する。一旦記録されると、自由にその変数データを出力するなり、データベースに格納するなり、変数に適切な操作を行うことが出来る。

もしエラーを別途のテキスト・ファイルにログすることにしていれば、ウエブ・サーバーのプロセス・オーナーはこのファイルに書き込むことに適切な許可を得ていなければならない。さらに、このファイルは文書のルートの外側にあることを確認するべきだ。それはアタッカーが入り込み密かにサーバーに入り込むために情報有効な情報をほりだす可能性があるからだ。

define_syslog_variables() 関数が、openlog()、 closelog() と syslog() 関数を使用するために必要な定数の初期化を行う。次のようなものだ：

void define_syslog_variables(void)

ログに関するどの関数の使用より前にこの関数を実行しておくことが必要だ。

ログのための接続をする

openlog() 関数はプラットフォームのシステムのログとの接続を開き、ログの際に必要な数個のパラメータを指定し、一つ以上のメッセージをシステムログに挿入するための場を設定する。次のようなものだ：

int openlog(string ident, int option, int facility)

次を含むいくつかのパラメータがサポートされている。

ident: メッセージを同定する。その値は各記録の先頭に付けられる。通常この値はプログラムの名称になるので、PHP 関係のメッセージは"PHP"あるいは "PHP5"となる。

option: メッセージを生成するときのオプションを決める。選択可能なオプションは表 8-2 に示されている。1 つ以上のオプションを指定するときには、各々を縦棒 | で分離する。例えば、3 つのオプションを次のように指定できる：

LOG_ODELAY | LOG_PERROR | LOG_PID

Facility: どのカテゴリーのプログラムがメッセージをログしているかを決めるのを助ける。次のようにいくつかのカテゴリーがある：LOG_KERN, LOG_USER, LOG_MAIL, LOG_DAEMON, LOG_AUTH, LOG_LPR, それに LOG_LOCAL*N* 、最後の *N* は 0 から 7 までの値をとる。指定のプログラムがメッセージの行先を決めることに注意。例えば、LOG_CRON を指定すると以降のメッセージは cronlog　に提出されるし、LOG_USER を指定するとメッセージを転送してメッセージ・ファイルにしまう。PHP がコマンド・ラインのインタプリータとして使われていない限り、LOG_USER に指定するだろう。crontab からの PHP スクリプトを実行しているときには LOG_CRON を使うことが多い。この件の詳細は syslog の説明書を見るとよい。

ログのオプション

オプション	記述
LOG_CONS	syslog に書き込み中にエラーが起こった時には、システム・コンソールに出力する。
LOG_NDELAY	直ちに syslog に接続を開く。
LOG_ODELAY	最初のメッセージがログに提出されるまで接続を開かない。これがデフォルトになっている。

LOG_PERROR	ログされたメッセージを syslog と標準エラーの両方に送る。
LOG_PID	各メッセージにプロセス ID (PID) を付ける。

表 7-2. エラー・ログのオプション

ログの接続を閉じる

closelog() 関数が openlog() で開いた接続を閉じる。次のようなものだ。

int closelog(void)

syslog() 関数が特別のメッセージを syslog に送る役目を負っている。次のようなものだ。

int syslog(int priority, string message)

最初のパラメータ、priority、は syslog における優先順位を指定する。以下にその値を重大性の順で示す：

LOG_EMERG: 重大なシステムの問題、クラッシュを知らせるなど。

LOG_ALERT: システムの完全さを脅かすのを回避するため直ちに解決しなければならない状態

LOG_CRIT: 危機的なエラー。サービスを使用不能に追い込むかもしれないが、必ずしもシステムを危険にさらすものではない。

LOG_ERROR: 一般的なエラー

LOG_WARNING: 一般的な警告

LOG_NOTICE: 正常だが注意するべき状態

LOG_INFO: 一般的な情報を与えるメッセージ

LOG_DEBUG: たいていは応用をデバッグしているときにだけ役立つ情報

2 番目のパラメータのメッセージにはログしたいメッセージのテキストを指定する。PHP エンジンが出すエラー・メッセージをそのままログしたいときには、文字列　%min をメッセージに含めておくとよい。この文字列は実行時エンジンによって提供されるエラー・メッセージの文字列(sterror)に置き換えられる。

これで適切な関数のことが分かったと思う。次に使用例を示す：

SHORT CODE HERE starting with “<?php define_syslog_variables(); “ ending with “closelog(); ?>”

レストランの応用に戻って、全てをつないで、さらに例外処理のコードも加えよう。すでに Database.php と呼ぶ分離したファイルも持っていて、それは Database クラスを含み　もちろん“restaurant”データベースとその他の応用との間のやり取りを取り扱ってくれる。残りの応用コードは別のファイルに分離されていて、さらに応用は 3 つの主要な部分；”Model”, “View”そして “Controller”として分離されている。

“Model” を構成しているコードは MenuModel.php にある。このファイルは欲しいと思う特定のメニュー・オブジェクト(すなわち：DrinkMenu, KidsMenu, DinnerMenu, など)のためのモデルを定義する全てのクラス、特定のメニューのクラスの親のクラスである abstruct なメニューのクラス、そしてメニュー品目のクラスを含んでいる：

```
<?php
//MenuModel.php

require_once('/path/to/Database.php');

class MenuItem
{
        // constructor (not implemented)
        // コンストラクター（ここには実装されない）

        public function _construct(){}

        // set undeclared property
        //  未宣言のプロパティを設定する

        function __set($property, $value)
        {
        $this->$property = $value;
        }

        // get defined property
        //  定義済みプロパティの値を得る
        function __get($property)
        {
                if (isset($this->$property))
                {
                        return $this->$property;
                }
        }
}


abstract class Menu
{
        private $menuid;
        private $parentid;
        private $menuitemid;
        private $menuname;
        private $description;

        public function setMenuID($menuid){$this->menuid = $menuid;}
        public function getMenuID(){return $this->menuid;}

        public function setMenuItemID($menuitemid){$this-> menuitemid = $menuitemid;}
        public function getMenuItemID(){return $this-> menuitemid;}

        public function setMenuName($menuname){$this->menuname = $menuname;}
        public function getMenuName(){return $this->menuname;}

        public function setDescription($description){$this->description= $description;}
        public function getDescription(){return $this->description;}

        function getAllMenuItems($menuid)
        {
                $connection = Database::Connect();
                $this->query = "select mitm.*, mi.itemname, mi.description,
                                        mi.price, mi.picture, mi.servingsize
                                        from `menuitemtomenu`
                                        as mitm LEFT JOIN `menuitem`
                                        as mi on mitm.menuitemid = mi.menuitemid
                                        where mitm.menuid = $menuid";

                $menuitemList = Array();
                $cursor = Database::Reader($this->query, $connection);
                while ($row = Database::Read($cursor))
                {
                        $menuitem = new MenuItem;
                        $menuitem->menuitemid = $row['menuitemid'];
```

```
                        $menuitem->itemname = $row['itemname'];
                        $menuitem->price = $row['price'];
                        $menuitem->servingsize = $row['servingsize'];
                        $menuitem->description = $row['description'];
                        $menuitemList[] = $menuitem;
                }
                return $menuitemList;
        }

}

class MainMenu extends Menu
{
        static public $title="<b><font color=blue>Main Menu</font></b>";
        const id = 1;

        static public function menutime()
        {
                return  time();
        }
}

class DrinkMenu extends Menu
{
        static public $title = "<b><font color=lightblue>Drink Menu</font></b>";
        const id = 2;
}

class LunchMenu extends Menu
{
        static public $title="<b><font color=green>Lunch Menu</font></b>";
        const id = 3;
}

class KidsMenu extends Menu
{
        static public $title="<b><font color=yellow>Kids Menu</font></b>";
        const id = 5;
}

class DessertMenu extends Menu
{
        static public $title="<b><font color=red>Dessert Menu</font></b>";
        const id = 6;
}

class AppetizerMenu extends Menu
{
        static public $title="<b><font color=purple>Appetizer Menu</font></b>";
        const id = 7;
}


interface AdjustPortion {
        public function setDinnerPortion($itemobject);
}

interface AdjustPrice {
        public function setDinnerPrices($itemobject);
        public function setHappyHourDrinkPrices($itemobject);
}



final class HappyHourMenu extends DrinkMenu implements AdjustPrice
{
        static public $title="<b><font color=orange>Happy Hour Drink Menu</font></b>";

        public function setHappyHourDrinkPrices($menuitemObject)
        {
                $adjusted_price = 1;
                $base_price = $menuitemObject->getPrice();

                // Make the dinner price 30% less than the normal price
                //  ディナーの価格を通常価格の 30% 引きにする。
                $adjusted_price = ($base_price * 0.7);

                return  round($adjusted_price,2);
        }

        public function setDinnerPrices($menuitemobject) {  }
}


final class DinnerMenu extends LunchMenu implements AdjustPortion, AdjustPrice
{
        static public $title="<b><font color=blue>Dinner Menu</font></b>";
        //DinnerMenu inherits ID from LunchMenu
```

```
        //  DinnerMenu は ID を LunchMenu からインヘリトする。


        public function setDinnerPortion($menuitemServingSize)
        {
                $adjustment = 1;
                $adjusted_servingsize = "";
                $portion = explode(" ", $menuitemServingSize);

                // Make the dinner portions 50% bigger than the Lunch portion
                //  ディナーのサイズをランチのサイズの 50% 増しにする。
                foreach ($portion as $subportion)
                {
                        if (is_numeric($subportion))
                        {
                        $adjustment = $subportion * 1.5;
                        $adjusted_servingsize = $adjusted_servingsize . " " . round($adjustment,2);
                        }
                        else
                        {
                        $adjusted_servingsize = $adjusted_servingsize . " " . $subportion;
                        }
                }

                return  $adjusted_servingsize;
        }


        public function setDinnerPrices($menuitemPrice)
        {
                $adjusted_price = 1;

        // Make the dinner price 25% more than the Lunch price.
        //  ディナーの価格をランチの価格の 25% 増しにする。
           try
           {
                 if ($menuitemPrice != 0)
                 {
                    $adjusted_price = ($menuitemPrice * 1.25);
                    return  round($adjusted_price,2);
                 }
                 else
                 {
                    throw new Exception('MenuItem Price is $0.0');
                 }
           }
           catch (Exception $e)
           {
                 echo "Caught exception: " .  $e->getMessage() . "\n";
           }
        }

        // override the base method to use the AdjustPrice interface
        // AdjustPrice のインタフェイスを用いるので親のメソッドにオーバーライドする
        public function GetAllMenuItems($menuid)
        {
                $connection = Database::Connect();
                $this->query = "select mitm.*, mi.itemname, mi.description,
                                        mi.price, mi.picture, mi.servingsize
                                        from `menuitemtomenu`
                                        as mitm left join `menuitem`
                                        as mi on mitm.menuitemid = mi.menuitemid
                                        where mitm.menuid = $menuid";

           try
           {

// Note: Recall that the throw new Exception(…) statement is in the Database class in Database.php.
// 注意 throw new Exception(…)ステートメントはDatabase.php のDatabase にあることを思い出そう。

                $menuitemList = Array();
                $cursor = Database::Reader($this->query, $connection);
                while ($row = Database::Read($cursor))
                {
                        $menuitem = new MenuItem;
                        $menuitem->menuitemid = $row['menuitemid'];
                        $menuitem->itemname = $row['itemname'];
                        $menuitem->price = self::setDinnerPrices($row['price']);
                        $menuitem->description = $row['description'];
                        $menuitem->servingsize = self::setDinnerPortion($row['servingsize']);
                        $menuitemList[] = $menuitem;
                }
                return $menuitemList;
           }
           catch(Exception $e)
           {
```

```
                echo 'Caught exception: ',  $e->getMessage(), "\n";
            }
        }

        public function setHappyHourDrinkPrices($menuitemObject){}
}


// DBMapper handles most of the communication between the model and the database
// Class. DBMapper extends Menu
//  DBMapper はモデルとデータベースの間のほとんどのやり取りを取り扱う。

{

        public function Erase($menuitemid)
        {
            try
            {
                $connection = Database::Connect();
                $this->query = "DELETE mi.*, mitm.* FROM `menuitem` AS mi
                                LEFT JOIN `menuitemtomenu` AS mitm ON mitm.menuitemid = mi.menuitemid
                                WHERE mi.menuitemid = $menuitemid";

                $rows = Database::Query($this->query, $connection);
            }
            catch(Exception $e)
            {
                echo "Caught exception in Erase Method: " . $e->getMessage(), "\n";
            }
        }


        public function Save($data) {

            try
            {

                $connection = Database::Connect();

                if ( (isset($data["menuid"])) && (isset($data["menuitemid"])) )
                {

                // If the record doesn't exist then we will INSERT it.
                //  もしレコードがなければ、それを挿入する。

                $this->query = "select `menuitemid`, `menuid`
                                            from `menuitemtomenu`
                                            where `menuitemid`= " . $data['menuitemid'] .
                                            " and `menuid`= " . $data['menuid'] . " LIMIT 1";

                // Note: Recall that the throw new Exception(...) statement is already in the
                // Database class in Database.php
                //  注意：throw new Exception(…) のステートメントは既に Database.php の中の Database クラスにある。


                $rows = Database::Query($this->query, $connection);
                if ($rows == 0)
                        {
                                $this->query = "insert into `menuitemtomenu` (`menuitemid`,`menuid`) values (
                                ". $data['menuitemid'] . ", " . $data['menuid'] . " )";
                        }
                        $insertId = Database::InsertOrUpdate($this->query, $connection);
                }
                else
                {

                        $menuitem = new MenuItem;
                        $menuitem->itemid = $data['menuitemid'];
                        $menuitem->itemname = $data['itemname'];
                        $menuitem->description = $data['description'];
                        $menuitem->servingsize = $data['servingsize'];
                        $menuitem->picture = $data['picture'];
                        $menuitem->price = $data['price'];

                // Check to see if the record already exists in the menuitem table.
                // If so we will UPDATE it, if not we will INSERT it.
                // レコードがすでにメニュー品目のテーブルにあるか調べる
                //  もしあるのならそれを更新し、ないならそれを挿入する。


                        $this->query = "select `menuitemid` from `menuitem`
                                        where `menuitemid`='" . $menuitem->itemid ."' LIMIT 1";

                // Note: Recall that the throw new Exception(...) statement is already in the
                // Database class in Database.php.
                //  注意：throw new Exception(…) のステートメントは既に Database.php の中の Database クラスにある。
```

```
                $rows = Database::Query($this->query, $connection);

                if ($rows > 0)
                {
                        $this->query = "update `menuitem` set
                        `itemname`='" . mysql_real_escape_string($menuitem->itemname) . "',
                        `description`='" . mysql_real_escape_string($menuitem->description) . "',
                        `servingsize`='" . $menuitem->servingsize . "',
                        `picture`='" . $menuitem->picture . "',
                        `price`='" . $menuitem->price . "' where `menuitemid`='".$menuitem->itemid."'";
                }
                else
                {
                        $this->query = "insert into `menuitem`
                        (`itemname`,`description`,`servingsize`,`picture`,`price`) values (
                        '".mysql_real_escape_string($menuitem->itemname)."',
                        '".mysql_real_escape_string($menuitem->description)."',
                        '".$menuitem->servingsize."',
                        '".$menuitem->picture."',
                        '".$menuitem->price."' )";
                }
                $insertId = Database::InsertOrUpdate($this->query, $connection);
            }

        }
        catch (Exception $e)
        {
            echo "Caught exception in Save method: " . $e->getMessage(), "\n";
        }
    }


    public function getMenuItemTable()
    {
            $connection = Database::Connect();
            $this->query = "select menuitemid, itemname, description,
                                price, picture, servingsize
                                from `menuitem`";

            $menuitemList = Array();
            $cursor = Database::Reader($this->query, $connection);
            while ($row = Database::Read($cursor))
            {
                    $menuitem = new MenuItem;
                    $menuitem->menuitemid = $row['menuitemid'];
                    $menuitem->itemname = $row['itemname'];
                    $menuitem->price = $row['price'];
                    $menuitem->servingsize = $row['servingsize'];
                    $menuitem->description = $row['description'];
                    $menuitemList[] = $menuitem;
            }
            return $menuitemList;
    }

}
?>
```

ここで応用全体のためにエラー報告を設定しよう。次の 2 行を index.php ファイルに挿入するだけだ。次のコマンドが全てのエラーを報告してくれる：

```
<?php
//index.php

ini_set('display_errors',1);
error_reporting(E_ALL);

//...more code

?>
```

第 8 章　モデル–ビュウ–コントローラ

モデル–ビュウ–コントローラによる設計方式

モデル–ビュウ–コントローラ(MVC) 設計方式はソフトウエア構築の上で、概念設計，提示、ユーザ入力に対する行動を 3 つの区別できる分野に分離する：

モデル

これは基本になるデータの構成のことだ。モデルは応用の行動とデータを管理する。

ビュウ

これはユーザや顧客に提示するもののことだ。ビュウは、モデルの状態の情報を要求してその表示を管理しモデルを表現するものである。

コントローラ

コントローラはユーザの入力に基づく処理を行う要素である。コントローラはモデルの状態を変える指示をする。

これら 3 つの要素を分離することで緩やかな結合を実現するのが容易になり、コントローラが複数の異なるモデルとビュウ要素と働くことが可能になる。ビュウもコントローラもモデルに依存する。一方モデルは、ビュウとコントローラに依存しない。この特別な関係が分離することの主要なメリットの一つだ。またこの分離によってモデルを視覚表示と独立に開発及びテストすることが出来る。

PHP にはいくつかのよく知られた高品質な MVC のフレームがあって、それぞれのさまざまな長所、欠点を考慮して選ぶことができる。しかし、無用な混乱を避けるため、ここではその 1 つに首を突っ込むことはしないでおこう。その代り専用の MVC フレームワークを開発しよう。第 7 章で我々は既にレストラン・メニュー管理の応用のためのデータ・モデル MenuModel.php を開発済みで、そこには元にする Menu クラスとそれを拡張した MenuItem クラス、DBMapper クラス、さらにそれらから派生したサブ-メニューのクラスが含まれていた。ここでこれらを MVC フレームワークに統合し、ビュウとコントローラの要素を加えることにしよう。

MVC URL 構造と URL マッピング

始めるにあたって、君は MVC フレームワークで応用開発が働く場所としてわかりやすい次の URL の構造を見たことがあるだろう：

Http://domain/controller/action/id

レストラン・メニュー管理(MVC フレームワーク)の応用でそうするためには Apache で.htaccess URL の書き直しを利用する。

.htaccess と呼ぶファイルを造る必要がある(先頭にピリオッド”.”がついているのを見落とさないよう)。このファイルはサイトのルート・ディレクトリに入っているだろう。このファイルの目的は Apache とそのモジュールをサイト／ディレクトリの表記に関して設定することだ。この場合 .htaccess は次のようになるだろう。

```
Options +FollowSymLinks
RewriteEngine on
RewriteRule ^([a-zA-Z]*)/?([a-zA-Z]*)?/?([a-zA-Z0-9]*)?/?$
index.php?controller=$1&action=$2&id=$3 [NC,L]
```

最初の 2 行は Apache を我々の新しい書き換えルール用に準備するもので、そのルールを次の行の RewriteRule 命令を使って定義する。このコマンドでは最初レギュラー表現があり、そして何か変数が入った標準 URL が続く。このルールが表現しているのは、ユーザが要求したどんな URL であってもこのレギュラー規則にあっていればウエブ・サーバのレベルで指定の実際的 URL に変換されるということだ。ユーザがこの変換を目にすることはない。レギュラー表現の丸括弧の中に一致があれば左から右に変数が与えられる。最初の件数が$1、2 番目が$2 など。この特別なルールはまたコントローラが URL の左側を削り取った後のどんなものでも受け入れる（疑問符?がこれを扱う）。

index.php ファイル

我々のウエブ・サーバはルート・ディレクトリの index.php を指し、それが全ての要求の唯一の入口になる。index.php ファイルは応用のエラー報告の設定をするところでもある。また index.php ファイルは、解釈”interpreter” オブジェクトを造り、要求された URL を適切なコントローラ・クラスのインスタンスに翻訳し、要求された動作(コントローラ・クラスのメソッド)をそのインスタンスの上で実行する。いくつか他のことを説明した後で、もう一度 index.php ファイルに戻ることにする。

```
<?php
//index.php

ini_set('display_errors',1);
error_reporting(E_ALL);


//一般クラスへの参照

require("helperclasses/Interpretor.php");
require("helperclasses/BaseController.php");
require("helperclasses/Template.php");
require_once("helperclasses/Database.php");

//データ・モデル・クラスへの参照

require("models/MenuModel.php");

//モデル・クラスへの参照

// コントローラ・クラスへの参照

// コントローラをつくり、行動を実行する。

$interpreter = new Interpreter($_GET);
$controller = $interpreter->CreateController();
$controller->ExecuteAction();

?>
```

MVC フォルダーの構造

全てのコントローラ、モデル、ビュウのファイルは同じ名前のフォルダーに整理されている個別のファイルである。ビュウ・フォルダーは個別のコントローラのサブ・フォルダーを持っている。コントローラとモデルのファイルはそれぞれ PHP のクラスである。ビュウのファイルは PHP であるが、この例では全くの HTML

である。基本的なフォルダーの構造は次のようになっている：

```
/htdocs
      /models
      /views
      /controllers
      /helperclasses
      /images
      /database
```

当然、MenuModel.php ファイルをモデルのフォルダーに置く。Database.php ファイルは MySQL データベースとのやり取りを手助けする Database クラスを含んでいるので helperclasses のフォルダーに入る。データベースのディレクトリはデータベースを造り、データベース構造（いくつかのテストデータとともにロードする）もつくるのに使われる実際の.sql ファイルを保存するために使われる。

.sql ファイルは既にいくらかのテストデータを含んでいるので、最初にすることはそのデータを提示あるいは見る方法を開発することである。ということで、レストラン・メニューを見せるために必要なモデル、コントローラとビュウのファイルを造ろう。これをしておけば、同じパターンでメニュー品目の追加や編集ができるようになる。

ユーザは応用とウエブ・ブラウザーを通じて対話する。ウエブ・ブラウザーは要求を URL に送る。MVC

URL はこのようなはずだ。

http://domain/controller/action/id

あるいは、我々の例では：

http://localhost/show/Display

コントローラは URL からの入力を解釈したものを受け取り、それに従ってコマンドを実行する。コントローラに編集と追加の処理をさせるため URL からの入力を受け付け、拡張されることのあるコマンドを中継するベース・クラスが必要になる。URL のコマンドの解釈はベース・コントローラのするべき仕事ではないのでもう一つ利便クラスを造ってこの仕事をやらせよう。このクラスを Interpreter と名付け Interpreter.php と呼ぶ利便クラスのファイルに入れる。

先ず Interpreter クラスから始めよう：

```
class Interpreter {
       private $controller;
       private $action;
       private $urlvalues;
       // URL の値をオブジェクトの生成の際に保存する


       public function __construct($urlvalues)
       {
              $this->urlvalues = $urlvalues;
```

```
		if ($this->urlvalues['controller'] == "")
		{
			$this->controller = "show";
		}
		else
		{
			$this->controller = $this->urlvalues['controller'];
		}
		if ($this->urlvalues['action'] == "")
		{
			$this->action = "index";
		}
		else
		{
			$this->action = $this->urlvalues['action'];
		}
	}
```

このクラスからインスタンス化した１つのオブジェクトが URL コマンドを砕き(.htaccess からの援助を得て)、その解釈からどの特定のコントローラが要求されているのか、そのコントローラのクラスの中でどの行動（メソッド）が要求されているかを求める。前章でここで質問を出してもよいと言っていた。コントローラのクラス（異なった動作をする複数のコントローラを生成しようとしているならうまくいかないが）から static なメソッドを呼ぼうとしているのでないなら、コントローラのオブジェクトのインスタンスを造る必要がある。従って、URL コマンドに応じて特定のコントローラオブジェクトをインスタンス化できることが要求される。
そこで、Interpreter クラスは次の様なメソッドを必要とする：

```
	// 要求されたコントローラをオブジェクトとしてインスタンス化する

	public function CreateController()
	{
		// そのクラスはそんざいするか？

		if (class_exists($this->controller))
		{
			$parents = class_parents($this->controller);
			// そのクラスはコントローラのクラスを拡張するか？

			if (in_array("BaseController",$parents))
			{
				// そのクラスは要求されたメソッドを含んでいるか？

				if (method_exists($this->controller,$this->action))
				{
					return new $this->controller($this->action,$this->urlvalues);
				}
				else
				{
					throw new Exception("Bad Method: " . $this->urlvalues);
				}
			}
			else
			{
				throw new Exception("Bad Controller: " . $this->urlvalues);
			}
		}
		else
		{
			throw new Exception("Bad Controller: " . $this->urlvalues);
		}
	}
```

以上を全て一つの Interpret.php と呼ぶ利便クラスにまとめ、 helperclasses フォルダーに保存しよう。

```
<?php
//Interpreter.php

class Interpreter {
	private $controller;
	private $action;
	private $urlvalues;
```

```
        //store the URL values on object creation


        public function __construct($urlvalues)
        {
                $this->urlvalues = $urlvalues;
                if ($this->urlvalues['controller'] == "")
                {
                        $this->controller = "show";
                }
                else
                {
                        $this->controller = $this->urlvalues['controller'];
                }
                if ($this->urlvalues['action'] == "")
                {
                        $this->action = "index";
                }
                else
                {
                        $this->action = $this->urlvalues['action'];
                }
        }


        //instantiate the requested controller as an object
        public function CreateController()
        {
           try
           {
                //does the class exist?
                if (class_exists($this->controller))
                {
                        $parents = class_parents($this->controller);
                        //does the class extend the controller class?
                        if (in_array("BaseController",$parents))
                        {
                                //does the class contain the requested method?
                                if (method_exists($this->controller,$this->action))
                                {
                                        return new $this->controller($this->action,$this->urlvalues);
                                }
                                else
                                {
                                        return new Exception("Bad Method: " . $this->urlvalues);
                                }
                        }
                        else
                        {
                                return new Exception("Bad Controller: " . $this->urlvalues);
                        }
                }
                else
                {
                        return new Exception("Bad Controller: " . $this->urlvalues);
                }
           }
           catch (Exception $e)
           {
                echo 'Caught exception: ',  $e->getMessage(), "\n";
           }
        }
}

?>
```

さて URL から interpreter を経由して入ってくるコマンドを中継する仕事をする仕事をするベース・コントローラのクラスを書いて、それを利便クラスのファイルに入れよう：

```
<?php
//BaseController.php

abstract class BaseController {
        protected $urlvalues;
        protected $action;
```

```
        public function __construct($action, $urlvalues) {
                $this->action = $action;
                $this->urlvalues = $urlvalues;
        }

        public function ExecuteAction($input) {
                return $this->{$this->action}($input);
        }

        protected function ReturnView($viewmodel) {
                $viewlocation = 'views/' . get_class($this) . '/' . $this->action . '.php';
                require($viewlocation);
        }
}

?>
```

利便クラスの議論のさいちゅうだが、必要な最後のものを紹介しよう。これはダイナミックなデータを、ダミーの html テンプレートを見てそのファイルのキィーをモデルからくるダイナミックなデータに置き換えて表示ファイルにに書き込むのを支援する。このクラスを Template と呼び、そのファイルを自身の PHP ファイルにいれる：

```
<?php

class Template
{
public $file;
        public $vars=array();

        public function set($key, $value)
        {
          $this->vars[$key] = $value;
        }

        public function display()
        {

           try
           {
               if(file_exists($this->file))
               {
                 $output = file_get_contents($this->file);
                 foreach($this->vars as $key => $value)
                 {
                   $output = preg_replace('/{'.$key.'}/', $value, $output);
                 }
                 echo $output;
               }
               else
               {
                 throw new Exception("Missing template --" . $this->file);
               }

             }

   catch (Exception $e)
   {
                echo "Exception caught: " . $e->getMessage();
           }

        }
}

?>
```

ここで display() メソッドが PHP のレギュラー表現関数 preg_replace()を使って丸括弧のなかの HTML ビュウのファイルにあるどのようなテキストでも置きなおすのに注目したい：

```
$output = preg_replace('/{'.$key.'}/', $value, $output);
```

この$key と対応する$Value はアレイ vars の中に保存される。 Key / 値のペアは set($key,$value) 関数を通ってアレイ vars に加えられる。これがまだ納得できないとすればビュウの案を見た後にしよう。メニューを”show” する簡単な HTML ページを書こう。

カスタム MVC 設計　ー　レストラン・メニュー管理応用

メニューを見せる

```
<!-
メニュー・データを見せるためのビュー・コード案
<--
Header
-->
<html>
<head>
</head>
<body>
<img src="images/logo.jpg">
<table width='900'>

< !-
この部分ではサブ・メニューのオブジェクト毎に 1 回印刷する。もし DrinkMenu, AppetizerMenu, LunchMenu, そして DessertMenu を生成するのならこの部分は対応するメニューのタイトルを表示するため 4 回印刷されることになる。
-->
<tr><td colspan='3'><center><h1><font style='Georgia'><i>{Title}</i></font></h1></center></td></tr>
<tr>
<th width='*' align='left'><b>Menu Item</b></th>
<th width='250' align='left'><b>Serving Size</b></th>
<th width='40' align='left'><b>Price</b></th>
</tr>
<tr><td colspan='3'> </td></tr>

< !-
この部分はループで使用され、メニュー品目オブジェクト毎に 1 回印刷される。
-->
<tr>
<td width='*'><b>{itemname}</b></td>
<td width='250'>{servingsize}</td>
<td width='40'>${price}</td>
</tr>
<tr><td colspan='3'><div style='word-wrap: break-word;'>{description}</div></td></tr>
<tr><td colspan='3'> </td></tr>
</table>
<br><br>

<!-
Footer
-->
</body>
</html>
```

中括弧でかこまれた”keys” すなわち {Title}, {itemname}, {servingsize}, {price} と{description}に注目しよう。Template クラスでの Display 関数がこの文字列パターンを見つけデータ・モデルから得る実際の関係する値に置きなおす preg_replace メソッドを使う。そのデータ・モデルはデータベースから値を得る。

ビュウ

PHP をビュウのファイルに含めることは可能だが、できればそうせずに、純粋にマークアップ言語だけで構成されるすっきりしたプレゼンテーション層にしておくのがよい。MVC の本質は応用の主な領域を完全に分離して置くことなのだ。レストラン・メニュー管理の応用でそれを達成するために上記の HTML を分解して分離したテンプレート・ファイルにする必要がある。というのは HTML でのコメントが示していたようにこのページの 2 つほどの部分が異なった繰り返し呼び出しに使われるからだ。メニューを”show”するこのビュウについていえば、このマークアップを次のように 4 つの分離した HTML ビュウ・ファイルに分解する必要がある:

```
<!-
header_template.html
-->
<html>
<head>
</head>
<body>
<img src="images/logo.jpg">
<table width='900'>
<!- ------ -->


<!-
label_template.html
-->
<tr><td colspan='3'><center><h1><font style='Georgia'><i>{Title}</i></font></h1></center></td></tr>
<tr>
<th width='*' align='left'><b>Menu Item</b></th>
<th width='250' align='left'><b>Serving Size</b></th>
<th width='40' align='left'><b>Price</b></th>
</tr>
<tr><td colspan='3'> </td></tr>
<!- ------ -->


<!-
menu_template.html
-->
<tr>
<td width='*'><b>{itemname}</b></td>
<td width='250'>{servingsize}</td>
<td width='40'>${price}</td>
</tr>
<tr><td colspan='3'><div style='word-wrap: break-word;'>{description}</div></td></tr>
<tr><td colspan='3'> </td></tr>
</table>
<br><br>
<!- ------ -->


<!-
footer_template.html
-->
</body>
</html>
<!- ------ -->
```

これらのビュウ・ファイルをビュウ・ディレクトリの”show/template” と呼ぶフォルダーに保存する。その結果フォルダーの構成は次のようになる：

```
/htdocs
      /models
      /views
            /show
```

```
            /templates
    /controllers
    /helperclasses
    /images
    /database
```

Display クラス

これでプレゼンテーション・テンプレートが設定できたが、ショウの特徴となるビュウを完成させるためにもう一つ必要なのが、Display() と呼ばれる利便クラスである。Display のクラスはテンプレート・ファイルを組立てて、要求される場所に対応するモデルからくるダイナミックなデータを書くことのがその役割だ。Display クラスは Template クラスからインスタンス化されたオブジェクトを用いてこのダイナミックなデータをスタティックなテンプレート・ファイルに書き込む。このクラスを Display.php と呼ぶファイルに入れる。

```
<?php


$display = new Display();
$display->Header();
$display->Body($viewmodel);
$display->Footer();

class Display
{

        public function Header()
        {
                require_once('views/show/templates/header_template.html');
        }

        public function Body($viewmodel)
        {
            foreach ($viewmodel as $menu)
            {
                $template_header= new Template();
                $template_header->file = "views/show/templates/label_template.html";
                $template_header->set('Title', $menu['title']);

                $template_header->display();

                $template_body = new Template();
                $template_body->file = "views/show/templates/menu_template.html";

                foreach ($menu['data'] as $menuitem)
                {
                        $template_body->set('itemname', strtoupper( $menuitem->itemname) );
                        $template_body->set('description', $menuitem->description);
                        $template_body->set('servingsize', $menuitem->servingsize);
                        $template_body->set('price', $menuitem->price);

                        $template_body->display();
                }
            }
        }

        public function Footer()
        {
                        require_once('views/show/templates/footer_template.html');
        }
}

?>
```

見て分かるように Show 特徴のための Display() クラスは Header(), Body()と Footer()の 3 つのメソッドだけでできている。レストラン・メニューをユーザに提示するのにデータ・モデルから必要とする情報の全ては入

力変数$viewmodel の中にあり、ここでの目的に役立つよう Body() メソッドに渡される。そこでこの特別なシナリオで発生するダイナミックなフォーマッティングの全てが行われる。

さてメニューを見せるためにモデルとコントローラもすべてを結合する段階にきた。コントローラは URL のユーザによって与えられた行動に直接対応するメソッドを含んでいる。ここで URL の行動とは実際にはコントローラの中の呼び出されるべきメソッドの名称である。呼び出されたコントローラの名称とは、行動(メソッド)を含んでいるクラスであり、やはり URL にある。コントローラの行動/メソッドは適切なモデルのインスタンスを生成する役目があり、さらに対応するビュウにデータを転送するのを助ける。コントローラとその関連モデルは同じ名称を持つファイルに保管するべきである。メニューデータを表示するという場合には show.php と呼ぶコントローラのディレクトリに保存されるコントローラ・ファイルを持ち、show.php と呼ぶモデルのディレクトリに保存されるモデルのファイルも持つ。

コントローラ

コントローラ(controller)はその名が示すように、MVC 応用の主要な要素の間の通信を制御する。それは「電話交換手」モデルとビュウの間の要求と応答を中継する。レストラン・メニューの応用に関しては、”Show” コントローラは BaseController クラスの延長であり、Interpreter からの行動を受け取り、ExecuteAction()関数でその行動を実行する。

メニューを”Show” する URL コマンドは次のものだ：

http://localhost/show/Display

Interpreter がこの URL を評価し Show コントローラのクラス Display() メソッドが起動される必要があることを判断し、Show クラスのインスタンスを /htdocs/controllers/show.php に生成し Display() メソッドを実行する。Display() メソッドは 3 つのことをする。まず最初に ShowModel に対応するインスタンスを生成し $viewmodel と呼ぶ。次に$viewmodel オブジェクトの Display() メソッドを呼び出す。3 番目に ShowModel から対応するビュウを BaseController クラスで定義された ReturnView() メソッド経由で view/show/Display.php ファイルに転送する。上記のように Display.php ファイルでは Show 特徴のための対応するビュウが Display.php ファイルの中の Display() クラスのインスタンスを生成することで設定され、さらに Header(), Body(), と Footer() メソッドを起動する。

次が”Show” のコントローラだ：

```
<?php
/* *** controllers/show.php *** */

class Show extends BaseController {

        public function Display() {

                $viewmodel = new ShowModel();
                $this->ReturnView($viewmodel->Display());

        }
}

?>
```

さて上記のコントローラに対応する ShowModel と呼ぶモデルを造る必要がある。

モデル

次の例でみるように、”Show”コントローラに対応するモデルは ShowModel と呼ぶクラスの中で定義され、モデルのディレクトリの中の show.php と呼ぶファイルに入る：

```
<?php
/* *** models/show.php *** */


class ShowModel {

 public function Display() {

  $drinks = new DrinkMenu();
  $appetizers = new AppetizerMenu();
  $lunch = new LunchMenu();
  $dinner = new DinnerMenu();

  $arr_drinks=array("title"=>DrinkMenu::$title,"data"=>$drinks->getAllMenuItems(DrinkMenu::id));
  $arr_appetizers=array("title"=>AppetizerMenu::$title,"data"=>$appetizers->getAllMenuItems(AppetizerMenu::id));
  $arr_lunch=array("title"=>LunchMenu::$title,"data"=>$lunch->getAllMenuItems(LunchMenu::id));
  $arr_dinner=array("title"=>DinnerMenu::$title,"data"=>$dinner->getAllMenuItems(DinnerMenu::id));

  return array($arr_drinks, $arr_appetizers, $arr_lunch, $arr_dinner);

  }

}
?>
```

これが最終的にデータ・モデルを置くところであり、abstruct な Menu クラスに拡張するいくつかのサブ・メニュー・オブジェクトを生成して使う。abstruct な Menu クラスは getAllMenuItems と呼ぶメソッドを持ちそれが MenuItem オブジェクトを生成し、データベースのテーブルからの適切なデータと共に取り上げる。DinnerMenu クラスではこのメソッドにオーバライドするのを思い出そう。

我々のビジネス設計によれば、このビジネスの規則を守らせるインタフェイスを用いているので LunchMenu と DinnerMenu は同じ MenuItem を使う。このインタフェイスは同じメニュー品目を持つランチとディナーのメニューで価格と盛り付けサイズを調整するために使うということを思い出そう。その結果、DinnerMenu クラスに入っている abstruct Menu クラスの getAllMenuItem($menuitemid) メソッドにオーバライドする。このように設計していなければ通常 getAllMenuItems() メソッドをベースの Menu クラスに入れずに DBMapper クラスに入れたであろう。インタフェイスの使用を実例で示すためにそうしたのだ。

先ず手始めに、ドリンク、アペタイザー、ランチ、ディナーのサブ・メニューを DrinkMenu,, AppetizerMenu, LunchMenu, DinnerMenu のクラスのそれぞれのインスタンスを生成して加える。これらのオブジェクトのそれぞれは getAllMenuItems() メソッドを呼び出し各オブジェクトのクラスの const id に関連した適切なデータをを検索する。$dinner オブジェクトは getAllMenuItems() メソッドを呼び出すが、DinnerMenu クラスから参照されるインタフェイスを実施するためにそれにオーバライドする。出力を見るとき、対応するメニュー品目の間で価格と盛り付けサイズを比較するのを忘れないように。

全てがうまくいく前に最後にすべきことがある。Controllers/show.php と models/show.php のファイルへの参照を index.php ファイルに付け加える必要がある。index.php は URL 要求の入口であるとともに全てを結びつける組立てファイルでもある。データ・モデルとヘルプ・ファイルもまたここから結ばれる。Interpreter クラスのインスタンスを生成し、それに URL の query のストリングを含む$_GET アレイを渡す。Interpreter

クラスのインスタンスは URL の入力に基づいて適切なコントローラ・オブジェクトを生成する。index.php ファイルが最後にすることは、コントローラ・オブジェクトの ExecuteAction メソッドを起動することで、それが URL の行動を実行する。

```
<?php
ini_set('display_errors',1);
error_reporting(E_ALL);


// 一般クラスへの参照
require("helperclasses/Interpreter.php");
require("helperclasses/BaseController.php");
require("helperclasses/Template.php");
require_once("helperclasses/Database.php");

// データ・モデルへの参照
require("models/MenuModel.php");


// モデルのクラスへの参照
require("models/show.php");


// コントローラ・クラスへの参照
require("controllers/show.php");


// コントローラを生成し行動を実行する
$interpreter = new Interpreter($_GET);
$controller = $interpreter->CreateController();
$controller->ExecuteAction();

?>
```

これで全部だ！出力を見てみよう：

WILDOCEAN

Fresh Exquisite Seafood & Fine Dining

Drink Menu

Menu Item	Serving Size	Price
CHERRY SODA Refreshing soda blended with candied cherry syrup	16 oz	$1.59
FRESH BREWED COFFEE Fresh Ground, Breshly Brewed Coffee	8 oz	$0.59
ICE WATER Ice Cold Water with Lemon	12 oz	$0.29
JALAPEÑO-LIME SPRITZER A refreshing chilled spritzer with a kick	16 oz	$1.99

Appetizer Menu

Menu Item	Serving Size	Price
OYSTERS ROCKEFELLER 6 oysters baked, topped with Pernod flavored spinach and Parmesan cheese.	6 oysters	$12.55
CHILLED SHUCKED OYSTERS Our fresh live oysters are from the Lousiana Gulf Coast and are the best live oysters to be found	12 oysters	$13.99
GRILLED LOUISIANA OYSTERS 6 oysters on the half shell, topped with Parmesan cheese and butter, char-grilled until plump.	6 oysters	$9.99
OYSTERS BIENVILLE 6 oysters baked, topped with Bay shrimp in a rich white wine cream sauce.	6 oysters	$9.99
PEEL-N- EAT SHRIMP Large Gulf shrimp steeped in beer, fresh lemon and country herbs. Chilled and served with our homemade red sauce.	12 shrimp	$9.99
GULF SHRIMP SKEWER Seven large shrimp skewered and prepared to your liking. Grilled or Blackened	7 shrimp	$8.99
CRAB BISQUE Sherry infused cream and crab stock with lump crab meat. Served with fresh made buttery corn bread.	1 bowl	$7.50

Lunch Menu

Menu Item	Serving Size	Price
GULF AMBERJACK SANDWICH Served seasoned and grilled or pan-blackened on a toasted buttery brioche bun with lettuce, tomato, red onion and sliced pickles. Served with your choice of a side item.	10 oz fillet	$11.95

HAWAIIAN COCONUT SHRIMP 8 shrimp $14.25
Butterflied jumbo shrimp hand dipped in our sweet coconut milk batter, then rolled in toasted coconut flakes and deep-fried golden. Served with apricot dipping sauce. Served with hot roll, butter, & your choice of a side item.

NEW YORK STRIP STEAK 12 oz $19.95
Our hand-cut 12 ounce New York Strip, seasoned just right! Cooked by our grillmaster to your specification. Served with hot roll, butter, & your choice of a side item.

SOUTHERN STYLE CRABCAKES 2 cakes $13.95
Three golden brown crabcakes served over a garnish of rice pilaf with homemade remoulade sauce. All entrees served with hot roll, butter, & your choice of a side item.

OYSTERS BENEDICT 8 oysters $10.95
Fried gulf oysters, sliced tomatoes and poached eggs on toasted English muffins, topped with creamy hollandaise sauce. All Benedicts served with your choice of home-style potatoes, buttery grits or fresh fruit.

TWIN JUMBO SOFT SHELL CRAB 2 crabs $20.95
Two soft shell crabs dipped in our Cajun breading and fried golden. Served over Creole shrimp stuffing & topped with basil buerre blanc. This comes with a hot roll, butter & your choice of a side item.

ALASKAN KING CRAB LEGS 1.5 lbs $36.35
Fresh, lushes Alaskan Red King Crab Legs, served with drawn butter, lemon and crab splitters. All entrees served with hot roll,butter & your choice of a side item.

TILAPIA ROYALE 14 oz tilapia fillet $13.95
Pan-blackened tilapia served over fresh spinach cakes topped with sauteed crawfish and hollandaise sauce. Served with hot roll & butter and your choice of a side item.

PREMIUM STEAK BURGER 0.75 lb $7.95
Served on a toasted buttery brioche bun with lettuce, tomato, red onion and sliced pickles. This is served char-grilled or blackened. Served with your choice of side item.

SOFT SHELL CRAB PO-BOY 2 crabs $10.95
Served on a toasted hoagie roll with shredded lettuce, or Cajun cole slaw, sliced tomatoes and homemade remoulade sauce. Served with your choice of a side item. Add $.50 for a Baked or Stuffed Potato.

OLD FASHIONED MONTE CRISTO 1 lb sandwich $9.95
Pit smoked ham, Hickory smoked turkey and sliced provolone, egg-battered and fried golden. Dusted with powdered sugar and served with honey mustard sauce for dipping. Served with your choice of a side item.

BACON CHEESEBURGER .75 lb, 2 sides $8.50
Served on a buttery toasted brioche bun with lettuce, tomato, red onion, bacon and sliced pickles.Your choice of cheddar,provolone,swiss, or pepperjack cheese. This comes with your choice of a side item.

SWORDFISH MAQUE CHOUX 12 oz swordfish steak $32
Fresh swordfish pan seared and served with Maque Choux fresh corn, Tasso ham, roasted peppers, garlic, tomatoes and green onions served with grits and micro arugula

Dinner Menu

Menu Item	Serving Size	Price

GULF AMBERJACK SANDWICH 15 oz fillet $14.94
Served seasoned and grilled or pan-blackened on a toasted buttery brioche bun with lettuce, tomato, red onion and sliced pickles. Served with your choice of a side item .

HAWAIIAN COCONUT SHRIMP 12 shrimp $17.81
Butterflied jumbo shrimp hand dipped in our sweet coconut milk batter, then rolled in toasted coconut flakes and deep-fried golden. Served with apricot dipping sauce. Served with hot roll, butter, & your choice of a side item.

NEW YORK STRIP STEAK 18 oz $24.94
Our hand-cut 12 ounce New York Strip, seasoned just right! Cooked by our grillmaster to your specification. Served with hot roll, butter, & your choice of a side item.

SOUTHERN STYLE CRABCAKES 3 cakes $17.44
Three golden brown crabcakes served over a garnish of rice pilaf with homemade remoulade sauce. All entrees served with hot roll, butter, & your choice of a side item.

OYSTERS BENEDICT 12 oysters $13.69
Fried gulf oysters, sliced tomatoes and poached eggs on toasted English muffins, topped with creamy hollandaise sauce. All Benedicts served with your choice of home-style potatoes, buttery grits or fresh fruit.

TWIN JUMBO SOFT SHELL CRAB 3 crabs $26.19
Two soft shell crabs dipped in our Cajun breading and fried golden. Served over Creole shrimp stuffing & topped with basil buerre blanc. This comes with a hot roll,

ALASKAN KING CRAB LEGS 2.25 lbs $45.44
Fresh, lushes Alaskan Red King Crab Legs, served with drawn butter, lemon and crab splitters. All entrees served with hot roll,butter & your choice of a side item.

TILAPIA ROYALE 21 oz tilapia fillet $17.44
Pan-blackened tilapia served over fresh spinach cakes topped with sauteed crawfish and hollandaise sauce. Served with hot roll & butter and your choice of a side item.

PREMIUM STEAK BURGER 1.13 lb $9.94
Served on a toasted buttery brioche bun with lettuce, tomato, red onion and sliced pickles. This is served char-grilled or blackened. Served with your choice of side item.

SOFT SHELL CRAB PO-BOY 3 crabs $13.69
Served on a toasted hoagie roll with shredded lettuce, or Cajun cole slaw, sliced tomatoes and homemade remoulade sauce. Served with your choice of a side item. Add $.50 for a Baked or Stuffed Potato.

OLD FASHIONED MONTE CRISTO 1.5 lb sandwich $12.44
Pit smoked ham, Hickory smoked turkey and sliced provolone, egg-battered and fried golden. Dusted with powdered sugar and served with honey mustard sauce for dipping. Served with your choice of a side item.

BACON CHEESEBURGER 1.13 lb, 3 sides $10.63
Served on a buttery toasted brioche bun with lettuce, tomato, red onion, bacon and sliced pickles.Your choice of cheddar,provolone,swiss, or pepperjack cheese. This comes with your choice of a side item.

SWORDFISH MAQUE CHOUX 18 oz swordfish steak $40
Fresh swordfish pan seared and served with Maque Choux fresh corn, Tasso ham, roasted peppers, garlic, tomatoes and green onions served with grits and micro arugula

View Menu Edit Menu Add New MenuItem Assign Item to Menu

図 8-1. http://localhost/show/Display からのブラウザー出力

新メニュー品目を追加する機能の開発

レストラン・メニュー管理応用にさらに能力を付け加えるには、同じパターンで行えばよい。今の応用は追加・編集がないし、メニュー品目ををメニューに割り付ける方法もない。データ・モデルの DBMapper クラスにある Save()メソッドは開発済みなのを思い出そう。Save()メソッドは$_POST のアレイを入力として受け、受け取った入力がまだデータベースになければ挿入し、すでにあるときには更新する。またどのデータベース・テーブルとやり取りをするのかを決定する。われわれのデータベースは非常に単純である。その図式は 2 つのテーブルを持つだけである：

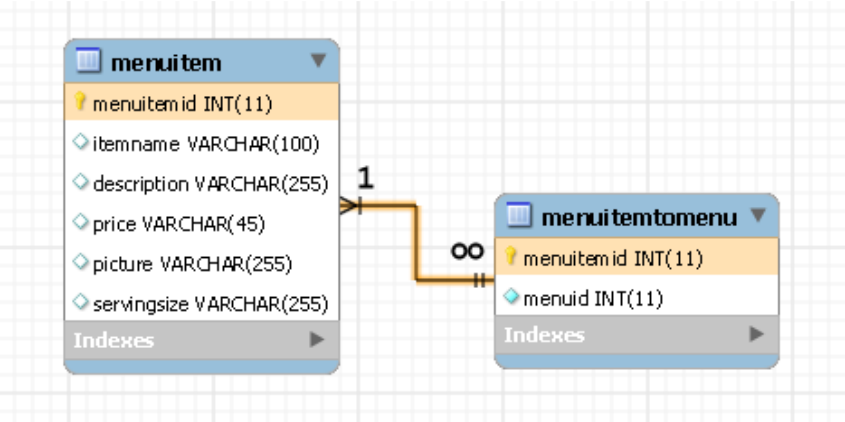


図 8-2. レストランのデータベースの構成要素関係ダイアグラム

新しいメニュー品目を追加する機能を開発するには、ユーザから獲得したいデータに気付く必要がある。大きめの応用でドメイン駆動型設計に固執するとデータ・モデルかから決定される。われわれの目的には、データ

ベース図式を見るだけでよい。このことをわきまえて、HTML 文書を書き上げよう。そこではユーザが新しいメニュー品目を menuitem テーブルに追加することが出来る。

```
<html>
<head>
<title>Add New Menu Item</title>
</head>
<body>
<form method="POST" action="index.php/add/Save">
<table>
<tr><td>Item Name:</td><td><input type="text" name="itemname"></td></tr>
<tr><td>Description:</td><td><input type="text" name="description"></td></tr>
<tr><td>Serving Size:</td><td><input type="text" name="servingsize"></td></tr>
<tr><td>Picture:</td><td><input type="text" name="picture"></td></tr>
<tr><td>Price:</td><td><input type="text" name="price"></td></tr>
<tr><td><input type="submit" name="submit" value="submit"></td></tr>
</table>
</form>
</body>
</html>
```

フォームを開くタグラインに注目しよう：

```
<form method="POST" action="index.php/add/Save">
```

このフォームからのユーザ入力データを index.php に投函し、それは適切なコントローラに$_POST アレイを使えるようにする。フォームのタグラインの行動に基づいて、この新しいコントローラを”Add” と名づけ、これが Save()と呼ぶメソッドを起動する。こうして次にするべきこと、すなわち新しいコントローラを生成することに繋げていこう。最初に BaseController クラスを拡張してそのメソッドを使えるようにする必要がある。Show コントローラでしたように、モデルとビュウの間の連絡訳をする Display()と呼ぶメソッドを生成する。このコントローラの Save() メソッドはユーザ入力を持っている$_POST アレイを受け取ってそれを MenuModel.php の DBMapper クラスにある Save() メソッドに渡す。それは入力がデータベースに書かれることを確認する。Save()はユーザをまだ開発していないが編集のページへ導く。このコントローラを含むファイルを add.php と名づけコントローラのディレクトリに保存する。

メニュー品目追加機能のためのコントローラ

```
<?php
// controller/add.php

class Add extends BaseController {

	protected function Display() {

		$viewmodel = new AddModel();
		$this->ReturnView($viewmodel->Display());
	}

	protected function Save()
	{
		$saveitem = new DBMapper();
		$saveitem->Save($_POST);
		header('Location: index.php/edit/Display');
	}
}
```

```
?>
```

この機能のためのモデルは単純だ。なぜなら「メニュー品目追加」のフォームはデータ・モデルから何も必要としないのでビュウに何のデータも送らないからだ。我々の名付け規則によって AddModel クラスを生成しよう。Show 機能にした方法と統一的であるために AddModel クラスに Display()メソッドを与え、このシナリオではビュウは何も要求しないので、戻り値も無しとする：

メニュー品目追加機能のモデル

```
<?php
// models/add.php

class AddModel {

        public function Display()
        {
                return;
        }
}

?>
```

メニュー品目追加機能のビュウ

ビュウを Header, Body, Footer に分けたのと統一するため、新メニュー品目追加の HTML 文書を 3 つの別々のファイルに分離する。またフッターにはブラウザーで見たいところを選べるようナビゲーションのリンクを追加しよう。新しい追加コントローラと ShowAddForm() の行動/メソッドがメニュー品目追加のページを開くナビゲーションのリンクにも使われるのに注目しよう。これらのナビゲーションのリンクをコピー/ペーストで次のファイルにも入れる。

Views/show/templates/footer_template.html

メニュー品目追加機能のビュウ・テンプレート

```
<!-
header_addmenuitem.html
-->
<html>
<head>
<title>Add New Menu Item</title>
</head>
<!- ------ -->



<!-
body_addmenuitem.html
-->
<body>
<form method="POST" action="index.php/add/Save">
<table>
<tr><td>Item Name:</td><td><input type="text" name="itemname"></td></tr>
<tr><td>Description:</td><td><input type="text" name="description"></td></tr>
<tr><td>Serving Size:</td><td><input type="text" name="servingsize"></td></tr>
<tr><td>Picture:</td><td><input type="text" name="picture"></td></tr>
<tr><td>Price:</td><td><input type="text" name="price"></td></tr>
<tr><td><input type="submit" name="submit" value="submit"></td></tr>
</table>
</form>
<!- ------ -->
```

```
<!-
footer_addmenuitem.html
-->
</ br>
</ br>
<center>
<table width="100%">
<tr>
<td><a href="index.php/show/ShowMenu">View Menu</a></td>

<td><a href="index.php/add/ShowAddForm">Add New MenuItem</a></td>

</tr>
</table>
</center>
</body>
</html>
<!- ------ -->
```

て、ビューのディレクトリーに”add/templates”のフォルダー・パスを造って、この３つのファイルをそこにしまおう。

メニュー品目追加機能のための表示管理

メニュー表示機能のためにしたのと同じで、メニュー品目追加機能のためのビューの構成を管理する Display()クラスを構築しよう。この Display()クラスは add/views フォルダーの中の Display.php と呼ぶファイルの中に含まれる。今までに述べてきたように、このメニュー品目追加機能のビューには入力フォームを構築するのにデータ・モデルから何も必要としない。だから、この表示のクラスは簡単だ。

Display() と呼ぶクラスを造って３つのメソッド：Header()、 Body() と Footer() を与えよう。このクラスは Display.php と呼ぶ PHP ファイルに置いて、このクラスのインスタンスを造り、そのメソッドを起動して views/add フォルダーに保存しよう。するとこのファイルがコントローラから呼びだされたときにビューを構成することになる：

```
<?php
// views/add/Display.php

$display = new Display();
$display->Header();
$display->Body();
$display->Footer();


class Display
{
        public function Header()
        {
                require_once('views/add/templates/header_addmenuitem.html');
        }

        public function Body()
        {

                require_once('views/add/templates/body_addmenuitem.html');
        }

        public function Footer()
        {
                require_once('views/add/templates/footer_addmenuitem.html');
        }
}

?>
```

あと残っていることは新しいモデルとコントローラに index.php ファイルへの参照を加えることだ。

```
<?php
ini_set('display_errors',1);
error_reporting(E_ALL);


// 一般クラスへの参照
require("helperclasses/Interpreter.php");
require("helperclasses/BaseController.php");
require("helperclasses/Template.php");
require_once("helperclasses/Database.php");

// データ・モデルへの参照

require("models/MenuModel.php");

// モデルのクラスへの参照
require("models/show.php");
require("models/add.php");

// コントローラ・クラスへの参照
require("controllers/show.php");
require("controllers/add.php");

// コントローラを生成し行動を実行する
$interpreter = new Interpreter($_GET);
$controller = $interpreter->CreateController();

$controller->ExecuteAction();

?>
```

これでユーザ・インタフェイスから新メニュー品目をデータベースに追加できるようになった。新しいメニュー品目をいくつかの使えるメニューに入れる必要があるので、ユーザがこれを行えるようあるページの HTML を書き上げよう。

```
<html>
<head>
<title>Assign Menu Items to Menus</title>
</head>
<body>
<form method="POST" action="index.php/assign/Save">
<table>
<tr><td>Select a Single Menu Item:</td></tr>
<tr>
<td>
<select name="menuitemid">

<option value="{menuitemid}">{itemname}</option>

</select>
</td>
</tr>
<tr><td> </td></tr>
<tr><td>Choose the Menus you want to add the Menu Item to</td></tr>
<tr><td>
<select name="menuid" multiple="multiple">

<option value="{menuid}">{menuname}</option>

</select>
</td>
</tr>
</table>
<br>
<br>
```

```
<input type="submit" value="UPDATE">
</form>
<br>
<br>
<br>
<br>
<br>
<br>
<center>
<table width="100%">
<tr>
<td><a href="index.php/show/ShowMenu">View Menu</a></td>
<td><a href="index.php/add/ShowAddForm">Add New MenuItem</a></td>
<td><a href="index.php/assign/ShowAssignForm">Assign Item to Menu</a></td>
</tr>
</table>
</center>
</body>
</html>
</body>
</html>
```

上記の HTML 文書でしたことは全ての使えるメニュー品目の選択リストを生成し、また全ての使えるメニューの選択リストも生成することだ。最初の選択リストからの値はユーザが選んだメニュー品目の menuitemid である。変動するデータの選択リストを使うのでどちらの選択リストもループを使うことになるのでこのページの部分を分離して繰り返し使えるようにし、ヘッダーやフッターなど他の HTML ページの部分も分離する。その結果ビュウは 5 つのテンプレートのファイルに分かれ、それらを views/assign ディレクトリの中のテンプレートのフォルダーに保存する。

メニュー品目をメニューに割り当てる機能のためのビュウ・テンプレート

```
<!- header_assignmenuitem.html -->
<html>
<head>
<title>Assign Menu Items to Menus</title>
</head>
<body>
<form method="POST" action="index.php/assign/Save">
<table>
<tr><td>Select a Single Menu Item:</td></tr>
<tr>
<td>
<select name="menuitemid">
<!- ----------------- -->


<!- body_assignmenuitem1.html -->
<option value="{menuitemid}">{itemname}</option>
<!- ----------------- -->


<!- body_assignmenuitem2.html -->
</select>
</td>
</tr>
<tr><td> </td></tr>
<tr><td>Choose the Menus you want to add the Menu Item to</td></tr>
<tr><td>
<select name="menuid" multiple="multiple">
<!- ----------------- -->
```

```
<!- body_assignmenuitem3.html -->
<option value="{menuid}">{menuname}</option>
<!- ----------------- -->


<!- footer_assignmenuitem1.html -->
</select>
</td>
</tr>
</table>
<br>
<br>
<input type="submit" value="UPDATE">
</form>
<br>
<br>
<br>
<br>
<br>
<br>
<center>
<table width="100%">
<tr>
<td><a href="index.php/show/Display">View Menu</a></td>
<td><a href="index.php/add/Display">Add New MenuItem</a></td>
<td><a href="index.php/assign/Display">Assign Item to Menu</a></td>
</tr>
</table>
</center>
</body>
</html>
</body>
</html>
<!- ----------------- -->
```

フッターはメニュー品目のメニューへの割り当てのページへのナビゲーションのリンクを含んでいるのに注目しよう。さて割当て機能のコントローラを書き下してそれを assign.php と名付けるファイルに入れコントローラのディレクトリに保存しよう。

メニュー品目をメニューに割り当てる機能のためのコントローラ

```
<?php
// controllers/assign.php

class Assign extends BaseController {

        protected function Display() {

                $viewmodel = new AssignModel();
                $this->ReturnView($viewmodel->Display());
        }

        protected function Save()
        {
                $assignitem = new DBMapper();
                $assignitem->Save($_POST);
                header('Location: index.php?controller=show&action=Display');
        }
}

?>
```

我々のデザインの性質上、データベースにメニュー・テーブルを持っていない。その代わり、abstruct メニュー・クラスへ拡張したサブ・メニュー・クラス（すなわち:LunchMenu, DinnerMenu, KidMenu など）を生成して、データ・モデルでのサブ・メニューを生成する。このため、サブ・メニューとその ID を Display() メソッドのアレイに手作業で追加しなければならない。そうすればメニューを列挙する選択リストでの表示に渡される。ShowAssignForm() はデータ・モデルのデータベースからメニュー品目を得る。Save() メソッドは基本的には”add” コントローラにおけるのと同様、ビュウを編集フォームに送りなおすだけである。このフォームは AssignModel を作成し、Assign コントローラとモデルの参照を index.php を加えた後で造ることにする。

ということで、AssignModel の作成に入ろう。ShowModel と Addmodel と同様に AssignModel のクラスは Display() メソッドを含んでいる。この Assign MenuItem-to-Menu 機能のための Display() メソッドは、データ・モデル中に造った使えるサブ・メニュー・クラスに対応するアレイを得るとともに、データベースからメニュー品目のテーブルを取り出す必要がある。この機能のビュウはこの情報をユーザ・インタフェイスの選択リストに並べることが必要だ。

メニュー品目をメニューに割り当てる機能のためのモデル

```
<?php

class AssignModel {

        public function Display()
        {
                $loadtable = new DBMapper();
                $menuitems=$loadtable->getMenuItemTable();
                $arr_menuitems = array("menuitem"=>$menuitems);

                $menus = array("2"=>"Drinks","3"=>"Lunch & Dinner","5"=>"Kids","6"=>"Dessert","7"=>"Appetizers");
                $arr_menus = array("menu"=>$menus);

                return array($arr_menuitems, $arr_menus);
        }

}

?>
```

AssignModel クラスの Display() メソッドは MenuModel.php 中の DBMapper クラスの getMenuItemTable() メソッドを使ってメニュー品目テーブルの全体を得る。メニュー品目テーブルのデータとメニューの名称と ID を持ったアレイは別々のアレイに梱包され、両者をアレイにした梱包で Display() メソッドの戻り値としてビュウに送られる。コントローラがこの戻り値を$viewmodel と呼ぶ変数としてビュウに渡す。

メニュー品目をメニューに割り当てる機能のための表示マネージャ

対応するビュウの Display クラスの Display() メソッドは AssignModel からの多重アレイをほどき、このデータが対応するテンプレート・ファイルの HTML で設定されたフォーマットに従って提示されるように、それぞれの(foreach)ループで分離する。Show と Add の機能と同様に views/assign/Display.php ファイルにある Display() クラスをインスタンス化し、Body() メソッドがこの AssignModel からの梱包されたデータを入力として開梱を行う:

```
<?php
```

```
$display = new Display();
$display->Header();
$display->Body($viewmodel);
$display->Footer();


class Display{

	public function Header()
	{
		require_once('views/assign/templates/header_assignmenuitem.html');
	}

	public function Body($viewmodel)
	{

		$template_body1 = new Template();
		$template_body1->file = "views/assign/templates/body_assignmenuitem1.html";

		foreach ($viewmodel as $selectlist)
		{
			foreach ($selectlist['menuitem'] as $menuitem)
			{
			   $template_body1->set('menuitemid', $menuitem->menuitemid);
			   $template_body1->set('itemname', $menuitem->itemname);
			   $template_body1->display();
			}
		}


		require('views/assign/templates/body_assignmenuitem2.html');


		$template_body3 = new Template();
		$template_body3->file = "views/assign/templates/body_assignmenuitem3.html";

		foreach ($viewmodel as $selectlist)
		{
			foreach ($selectlist['menu'] as $menuid => $menuname)
			{
			   $template_body3->set('menuname', $menuname);
			   $template_body3->set('menuid', $menuid);
			   $template_body3->display();
			}

		}
	}

	public function Footer()
	{
		require_once('views/assign/templates/footer_assignmenuitem.html');
	}
}
?>
```

ここではTemplateクラスからインスタンス化したオブジェクトをBody()メソッドの中で2回使う。その都度変化する選択リストが2つあるからだ。Header(), Body(), と Footer()の全てがデータベースからの適切な情報を合体した完全なビュウを組み立てる。後はモデルとコントローラを含んだファイルを要求することで、これをindex.phpファイルを通じて応用に結びつけるだけだ。

```
<?php
ini_set('display_errors',1);
error_reporting(E_ALL);
```

```
// 一般クラスへの参照
require("helperclasses/Interpreter.php");
require("helperclasses/BaseController.php");
require("helperclasses/Template.php");
require_once("helperclasses/Database.php");

// データ・モデルへの参照
require("models/MenuModel.php");

// モデルのクラスへの参照
require("models/show.php");
require("models/add.php");
require("models/assign.php");

//コントローラ・クラスへの参照
require("controllers/show.php");
require("controllers/add.php");
require("controllers/assign.php");


// コントローラを生成し行動を実行する
$interpreter = new Interpreter($_GET);
$controller = $interpreter->CreateController();

$controller->ExecuteAction();

?>
```

メニュー品目の編集あるいは削除

最後にメニュー品目を編集あるいは削除することを可能にしよう。再び ModelMenu.php ファイルの DBMapper クラスに戻って参考にしよう。われわれは既にデータベースにあるレコードを更新する能力を持った Erase() メソッドと Save() メソッドを開発したことを知っている。従って”Edit”コントローラの EditModel クラスが DBMapper メソッドとさらにモデルとビュウの間の通信を中継する Display() メソッドと共に使うメソッドを必要とする。

今回は、ある応用での新しい機能を開発するときに必ずしもビュウから始めなくてもよいことを示す。今回は何をするべきか今した仮定に基づいてコントローラを最初に構築する。BaseController のクラスに拡張することから始めるとして、Display(),Save()と Erase()のメソッドを生成しよう。ユーザは直ちに編集のページに戻って編集や削除の操作が行われたか確認したいだろうと仮定し、Save()と Erase() のメソッドに戻りのコマンドを置こう。

メニュー品目の編集のためのコントローラ

```
<?php
// controllers/edit.php

class Edit extends BaseController {

        public function Display()
        {
                $viewmodel = new EditModel();
                $this->ReturnView($viewmodel->Display());
        }

        public function Save()
        {
                $updateitem = new DBMapper();
                $updateitem->Save($_POST);
```

```
                header('Location: index.php/edit/Display');
        }

        public function Erase()
        {
                $eraseitem = new DBMapper();
                $eraseitem->Erase($_GET['id']); //menuitemid passed through the URL
                header('Location: index.php/edit/Display');
        }

}

?>
```

次に編集機能のためのモデルが必要だ。メニュー品目の全てのプロパティを編集可能としたいということ、全ての使えるメニュー品目のプロパティの現在の値が何かを見たいということが分かっている。そこでメニュー品目をメニューに割り当てる機能でしたように、メニュー品目テーブルを引き出す必要がある。

メニュー品目の編集のためのモデル

```
<?php

class EditModel {

        public function Display()
        {

            $loadtable = new DBMapper();
            $data=$loadtable->getMenuItemTable();

            return($data);
        }

}


?>
```

EditModel は DBMapper クラスの getMenuItemTable() メソッドから受け取るアレイを戻し、Edit コントローラがそれを Edit ビュウに渡し、そこでおそらく Display 管理オブジェクトが受け取る。Display クラスを構築する前にこのビュウの HTML を書き下せば、Display()クラスでのフォーマッティングの設定の必要性がよりよく理解できる。

メニュー品目の編集のための一般的 HTML ページ：

```
<html>
<head>
<title>Edit Menu Items</title>
</head>
<body>
<table>
<tr>
<th>ID</th>
<th>Item Name</th>
<th>Description</th>
<th>Price</th>
<th>Serving Size</th>
```

```
<th> </th>
<th> </th>
<form method="POST" action="index.php/edit/Save">
<tr>
<td><input type="hidden" name="menuitemid" value="{menuitemid}">{menuitemid}</td>
<td><textarea cols="35" rows="1" name="itemname">{itemname}</textarea></td>
<td><textarea cols="55" rows="1" name="description">{description}</textarea></td>
<td><input type="text" name="price" value="{price}"></td>
<td><input type="text" name="servingsize" value="{servingsize}"></td>
<td><input type="submit" value="UPDATE"></td>
<td><a href="index.php/edit/Erase/{menuitemid}">DELETE</a></td>
</tr>
</form>
</table>
</ br>
</ br>
<center>
<table width="100%">
<tr>
<td><a href="index.php/show/Display">View Menu</a></td>
<td><a href="index.php/edit/Display">Edit Menu</a></td>
<td><a href="index.php/add/Display">Add New MenuItem</a></td>
<td><a href="index.php/assign/Display">Assign Item to Menu</a></td>
</tr>
</table>
</center>
</body>
</html>
</body>
</html>
```

この中にメニュー編集ページへのリンクがあることに注目したい。このフッター部は統一性のためにコピーして他のビュウのフッター部にペーストするべきものである。上記 HTML の中にキィーを置き更新と削除のコントロールを設定する。更新の機能はユーザの変更を index.php に投函し、それが$_POST データを編集コントローラが使えるようにする。削除の機能は index.php ファイルを呼び、編集コントローラが Erase() メソッドを起動するよう要求する。必要な menuitemid の情報は URL を通じて渡す。

メニュー品目の編集のためのビュウ・テンプレート

編集ビュウのための HTML を 3 つのテンプレート・ファイルに分割し views/edit/templates に保存しよう。

```
<!-- HEADER TEMPLATE -->
<!-- header_editmenuitem.html -->
<html>
<head>
<title>Edit Menu Items</title>
</head>
<body>
<table>
<tr>
<th>ID</th>
<th>Item Name</th>
<th>Description</th>
<th>Price</th>
<th>Serving Size</th>
<th> </th>
<th> </th>
</tr>
<!-- HEADER TEMPLATE -->
```

```
<!-- BODY TEMPLATE -->
<!-- body_editmenuitem.html -->
<form method="POST" action="index.php/edit/Save">
<tr>
<td><input type="hidden" name="menuitemid" value="{menuitemid}">{menuitemid}</td>
<td><textarea cols="35" rows="1" name="itemname">{itemname}</textarea></td>
<td><textarea cols="55" rows="1" name="description">{description}</textarea></td>
<td><input type="text" name="price" value="{price}"></td>
<td><input type="text" name="servingsize" value="{servingsize}"></td>
<td><input type="submit" value="UPDATE"></td>
<td><a href="index.php/edit/Erase/{menuitemid}">DELETE</a></td>
</tr>
</form>
<!-- BODY TEMPLATE -->


<!-- FOOTER TEMPLATE -->
<!-- footer_editmenuitem.html -->
</table>
</ br>
</ br>
<center>
<table width="100%">
<tr>
<td><a href="index.php/show/Display">View Menu</a></td>
<td><a href="index.php/edit/Display">Edit Menu</a></td>
<td><a href="index.php/add/Display">Add New MenuItem</a></td>
<td><a href="index.php/assign/Display">Assign Item to Menu</a></td>
</tr>
</table>
</center>
</body>
</html>
</body>
</html>
<!-- FOOTER TEMPLATE -->
```

では編集機能のための表示管理に移ろう。

編集ビュウのための表示マネージャ

```
<?php


$display = new Display();
$display->Header();
$display->Body($viewmodel);
$display->Footer();


class Display {

        public function Header()
        {
                require_once('views/edit/templates/header_editmenuitem.html');
        }

        public function Body($viewmodel)
        {

                $template_body = new Template();
                $template_body->file = "views/edit/templates/body_editmenuitem.html";

                foreach ($viewmodel as $menuitem)
                {
```

```
                        $template_body->set('menuitemid', $menuitem->menuitemid);
                        $template_body->set('itemname', $menuitem->itemname);
                        $template_body->set('description', $menuitem->description);
                        $template_body->set('servingsize', $menuitem->servingsize);
                        $template_body->set('price', $menuitem->price);

                        $template_body->display();
                }
        }

        public function Footer()
        {
                require_once('views/edit/templates/footer_editmenuitem.html');
        }
}


?>
```

ここで index.php file ファイルでの controllers/edit.php と models/edit.php へのリンクを更新すれば、それでもうビジネスに使えるのだ:

```
<?php
ini_set('display_errors',1);
error_reporting(E_ALL);


//require the general classes
require("helperclasses/Interpreter.php");
require("helperclasses/BaseController.php");
require("helperclasses/Template.php");
require_once("helperclasses/Database.php");

//require the data model
require("models/MenuModel.php");

//require the model classes
require("models/show.php");
require("models/add.php");
require("models/assign.php");
require("models/edit.php");

//require the controller classes
require("controllers/show.php");
require("controllers/add.php");
require("controllers/assign.php");
require("controllers/edit.php");

//create the controller and execute the action
$interpreter = new Interpreter($_GET);
$controller = $interpreter->CreateController();

$controller->ExecuteAction();

?>
```

レストラン・メニュー管理応用ソース・コードのダウンロード

ここまで開発してきたレストラン・メニュー管理の応用のための全てのコードとデータベースのファイルは次からダウンロードができる。

http://www.phpthis.com/downloads/restmenuapp_mvc

検証コード：ST32919

ダウンロードできるコードベースの中には Integrate_Display.php ファイルを view のフォルダーに含めてあり、テンプレートの使用をバイパスしビュウを合体し、PHP を HTML と統合し 1 つのファイルにしてある。テンプレートを用いた純粋なメソッドは正統な方法であり、さらに MVC 設計を順守した最も純粋な方法であるが、統合したビュウのアプローチも必要かも知れない。

この MVC 応用がいかに広範囲に及ぶものか理解するのをできたと期待する。従業員やチケット販売、在庫を管理する能力を追加することが出来る。要点は、この設計方式を用いてすでにここで作り上げた仕事に侵入することなく、この枠組みに簡単にデータベースや、データ・モデルを加えてレストランの全体管理システムを造り上げることが出来るということだ。

第 9 章　リフレクション API

リフレクション・クラス

PHP リフレクション API は一組のメソッドで構成されていて、クラスの内部構造を調べることを可能にし、インタフェイスを非常に簡単にするものだ。これを使えば、リフレクトされたクラスがあるメソッドを持っているかどうか、あるいはそのクラスがあるプロパティを宣言しているかどうか、さらにそのプロパティが public, protected, private のいずれか、と言ったことを調べるのを非常に簡単にしてくれる。

```
// 'DinnerMenu' のクラスのインスタンスを生成する
$dinner = new DinnerMenu();

// リフレクション・クラスのインスタンスを生成し、'user' クラスに引数として渡す
$reflect_dinner = new ReflectionClass('DinnerMenu');

// リフレクトされたクラスの名称を得る
echo $reflect_dinner->getName();

OUTPUT:
DinnerMenu
```

上で見るように、最初に例示のために DinnerMenu のクラスのインスタンスを生成した。そして 2 番目に、元からある"ReflectionClass" クラスから新しいリフレクター・オブジェクトを発生させた。ReflectionClass は全てのリフレクション API の基幹である。それはリフレクトされるべきクラスの名称を引数として取り入れる。

この例では、"DinnerMenu" の文字列が"ReflectionClass" クラスのコンストラクタに渡される。このプロセスはリフレクタ・オブジェクトを戻し、それが"DinnerMenu" についての価値ある情報を引き出すことを可能にしている。

いまはリフレクション API の紹介を始めたばかりだから、上の例で最初にしたことは"getName()" を起動してリフレクトされたクラスの名称を得ることだった。リフレクション・クラスの getName() メソッドを使うことは、リフレクトされるクラスの名称を知っているので、無意味なことと見えるかも知れない。しかしリフレクトされるクラスの名称があらかじめ分かっていない状況もありうるのだ。この例ではリフレクション・クラスのいろいろなメソッドの導入例として getName() を使った。クラスについてさらに他の情報を得るメソッドを見てみよう。次のセクションで論議するのは"DinnerMeniu" クラスの中で定義された定数の名称を戻すことだ。
さらにリフレクトされるクラスで宣言された定数の名称と値を得る 2 つほどのメソッドの使い方を見てみよう。

```
// 'DinnerMenu' のクラスのインスタンスを生成する
$dinner = new DinnerMenu();

// リフレクション・クラスのインスタンスを生成し、'user' クラスに引数として渡す
$reflect_dinner = new ReflectionClass('DinnerMenu');

// リフレクトされたクラスの定数を得る
echo $reflect_dinner->getConstant('id');

OUTPUT:
3
```

```
// リフレクトされたクラスの定数のアレイを得る
print_r($reflect_dinner->getConstants());
```

OUTPUT:

```
Array ( [id] => 3 )
```

上の例で明らかなように、与えられたクラスで定義された定数に割り当てられた値を引き出すことはリフレクション API の”getConstant()” と”getConstants() のメソッドを呼ぶだけのことである。最初のメソッドは構文解析される定数の名称を獲得し、２番目のは対応する値で埋まった定数のアレイを戻す。

```
// ‘DinnerMenu’ のクラスのインスタンスを生成する
$dinner = new DinnerMenu();


// リフレクション・クラスのインスタンスを生成し、’user’ クラスに引数として渡す
$reflect_dinner = new ReflectionClass('DinnerMenu');


Reflection::export($reflect_dinner);
```

OUTPUT:

```
Class [ <user> final class DinnerMenu extends LunchMenu implements AdjustPortion, AdjustPrice ] {
  @@ C:\temp\restaurant.php 222-269

  - Constants [1] {
    Constant [ integer id ] { 3 }
  }

  - Static properties [1] {
    Property [ public static $title ]
  }

  - Static methods [0] {
  }

  - Properties [0] {
  }

  - Methods [12] {
    Method [ <user, prototype AdjustPortion> public method setDinnerPortion ] {
      @@ C:\temp\restaurant.php 227 - 235

      - Parameters [1] {
        Parameter #0 [ <required> $menuitemServingSize ]
      }
    }

    Method [ <user, prototype AdjustPrice> public method setDinnerPrices ] {
      @@ C:\temp\restaurant.php 237 - 245

      - Parameters [1] {
        Parameter #0 [ <required> $menuitemPrice ]
      }
    }

    Method [ <user, overwrites Menu, prototype Menu> public method GetAllMenuItems ] {
      @@ C:\temp\restaurant.php 248 - 265

      - Parameters [1] {
        Parameter #0 [ <required> $menuid ]
      }
    }

    Method [ <user, prototype AdjustPrice> public method setHappyHourDrinkPrices ] {
      @@ C:\temp\restaurant.php 267 - 267

      - Parameters [1] {
        Parameter #0 [ <required> $menuitemObject ]
      }
    }
```

```
    Method [ <user, inherits Menu> public method setMenuID ] {
      @@ C:\temp\restaurant.php 124 - 124

      - Parameters [1] {
        Parameter #0 [ <required> $menuid ]
      }
    }

    Method [ <user, inherits Menu> public method getMenuID ] {
      @@ C:\temp\restaurant.php 125 - 125
    }

    Method [ <user, inherits Menu> public method setMenuItemID ] {
      @@ C:\temp\restaurant.php 127 - 127

      - Parameters [1] {
        Parameter #0 [ <required> $menuitemid ]
      }
    }

    Method [ <user, inherits Menu> public method getMenuItemID ] {
      @@ C:\temp\restaurant.php 128 - 128
    }

    Method [ <user, inherits Menu> public method setMenuName ] {
      @@ C:\temp\restaurant.php 130 - 130

      - Parameters [1] {
        Parameter #0 [ <required> $menuname ]
      }
    }

    Method [ <user, inherits Menu> public method getMenuName ] {
      @@ C:\temp\restaurant.php 131 - 131
    }

    Method [ <user, inherits Menu> public method setDescription ] {
      @@ C:\temp\restaurant.php 133 - 133

      - Parameters [1] {
        Parameter #0 [ <required> $description ]
      }
    }

    Method [ <user, inherits Menu> public method getDescription ] {
      @@ C:\temp\restaurant.php 134 - 134
    }
  }
}
```

さてこれと同じことをリフレクション・クラス自身で行ってみよう。そうすれば PHP5 のリフレクション・クラス自身で行ってみよう。そうすれば PHP5 のリフレクション API が提供している残りのメソッドをよく見ることが出来る：

```
$see_inside_reflection_class = new ReflectionClass(ReflectionClass);
Reflection::export($see_inside_reflection_class);
```

OUTPUT:

```
Class [ <internal:Reflection> class ReflectionClass implements Reflector ] {

  - Constants [3] {
    Constant [ integer IS_IMPLICIT_ABSTRACT ] { 16 }
    Constant [ integer IS_EXPLICIT_ABSTRACT ] { 32 }
    Constant [ integer IS_FINAL ] { 64 }
  }

  - Static properties [0] {
  }

  - Static methods [1] {
```

```
  Method [ <internal:Reflection> static public method export ] {

    - Parameters [2] {
      Parameter #0 [ <required> $argument ]
      Parameter #1 [ <optional> $return ]
    }
  }
}

- Properties [1] {
  Property [ <default> public $name ]
}

- Methods [40] {
  Method [ <internal:Reflection> final private method __clone ] {
  }

  Method [ <internal:Reflection, ctor> public method __construct ] {

    - Parameters [1] {
      Parameter #0 [ <required> $argument ]
    }
  }

  Method [ <internal:Reflection> public method __toString ] {
  }

  Method [ <internal:Reflection> public method getName ] {
  }

  Method [ <internal:Reflection> public method isInternal ] {
  }

  Method [ <internal:Reflection> public method isUserDefined ] {
  }

  Method [ <internal:Reflection> public method isInstantiable ] {
  }

  Method [ <internal:Reflection> public method getFileName ] {
  }

  Method [ <internal:Reflection> public method getStartLine ] {
  }

  Method [ <internal:Reflection> public method getEndLine ] {
  }

  Method [ <internal:Reflection> public method getDocComment ] {
  }

  Method [ <internal:Reflection> public method getConstructor ] {
  }

  Method [ <internal:Reflection> public method hasMethod ] {

    - Parameters [1] {
      Parameter #0 [ <required> $name ]
    }
  }

  Method [ <internal:Reflection> public method getMethod ] {

    - Parameters [1] {
      Parameter #0 [ <required> $name ]
    }
  }

  Method [ <internal:Reflection> public method getMethods ] {
```

```
  - Parameters [1] {
    Parameter #0 [ <optional> $filter ]
  }
}

Method [ <internal:Reflection> public method hasProperty ] {

  - Parameters [1] {
    Parameter #0 [ <required> $name ]
  }
}

Method [ <internal:Reflection> public method getProperty ] {

  - Parameters [1] {
    Parameter #0 [ <required> $name ]
  }
}

Method [ <internal:Reflection> public method getProperties ] {

  - Parameters [1] {
    Parameter #0 [ <optional> $filter ]
  }
}

Method [ <internal:Reflection> public method hasConstant ] {

  - Parameters [1] {
    Parameter #0 [ <required> $name ]
  }
}

Method [ <internal:Reflection> public method getConstants ] {
}

Method [ <internal:Reflection> public method getConstant ] {

  - Parameters [1] {
    Parameter #0 [ <required> $name ]
  }
}

Method [ <internal:Reflection> public method getInterfaces ] {
}

Method [ <internal:Reflection> public method getInterfaceNames ] {
}

Method [ <internal:Reflection> public method isInterface ] {
}

Method [ <internal:Reflection> public method isAbstract ] {
}

Method [ <internal:Reflection> public method isFinal ] {
}

Method [ <internal:Reflection> public method getModifiers ] {
}

Method [ <internal:Reflection> public method isInstance ] {

  - Parameters [1] {
    Parameter #0 [ <required> $object ]
  }
}
```

```
    Method [ <internal:Reflection> public method newInstance ] {

      - Parameters [1] {
        Parameter #0 [ <required> $args ]
      }
    }

    Method [ <internal:Reflection> public method newInstanceArgs ] {

      - Parameters [1] {
        Parameter #0 [ <optional> array $args ]
      }
    }

    Method [ <internal:Reflection> public method getParentClass ] {
    }

    Method [ <internal:Reflection> public method isSubclassOf ] {

      - Parameters [1] {
        Parameter #0 [ <required> $class ]
      }
    }

    Method [ <internal:Reflection> public method getStaticProperties ] {
    }

    Method [ <internal:Reflection> public method getStaticPropertyValue ] {

      - Parameters [2] {
        Parameter #0 [ <required> $name ]
        Parameter #1 [ <optional> $default ]
      }
    }

    Method [ <internal:Reflection> public method setStaticPropertyValue ] {

      - Parameters [2] {
        Parameter #0 [ <required> $name ]
        Parameter #1 [ <required> $value ]
      }
    }

    Method [ <internal:Reflection> public method getDefaultProperties ] {
    }

    Method [ <internal:Reflection> public method isIterateable ] {
    }

    Method [ <internal:Reflection> public method implementsInterface ] {

      - Parameters [1] {
        Parameter #0 [ <required> $interface ]
      }
    }

    Method [ <internal:Reflection> public method getExtension ] {
    }

    Method [ <internal:Reflection> public method getExtensionName ] {
    }
  }
}
```

これら使えるメソッドをスクロールしてずっと見ると先ほど使った getName(), getConstant(), getCOnstants() メソッドも含まれている。これらのメソッドを自分で試してみるとよく知ることになって良い。それらがユニット・テストで非常に助けになり、パワフルなツールであることが分かるだろう。

第 10 章　PEAR ライブラリ

PEAR の概要

PEAR は PHP Extension and Application Repository（PHP 拡張機能と応用リポジトリ、訳注:集積所と言った意味）の頭文字をとっている。それは再利用可能な PHP の要素やクラスの枠組みであり、配布システムである。DB 抽象パッケージを別にしても、PEAR ライブラリは別にしても、PEAR ライブラリは XML や CURL と併せて使うのに便利な膨大なクラスがある。PEAR の使用によって、他の人々がすでにコードを組み、テストし、使用した何かをコードを組む時間を大幅に節約してくれる。例えば、HTML フォームの検証ルーティンを必要としているなら、PEAR は Validation Package の中にそれを持っている。PEAR リポジトリは http://pear.php.net に集約して管理されているので、公式の PEAR パッケージを使うときにはその品質と信頼性について確信を持てる。

維持されているパッケージの全リストをここに示す。

Windows Zend Framework Environment に PEAR をインストールする

1．手持ちのブラウザーで http://pear.php.net/go-pear のページを開き、それを PHP ファイルとして C:\Program Files\Zend\Apache2\htdocs\go-pear.php に保存する。
2．ブラウザーで http://localhost/go-pear.php に行く

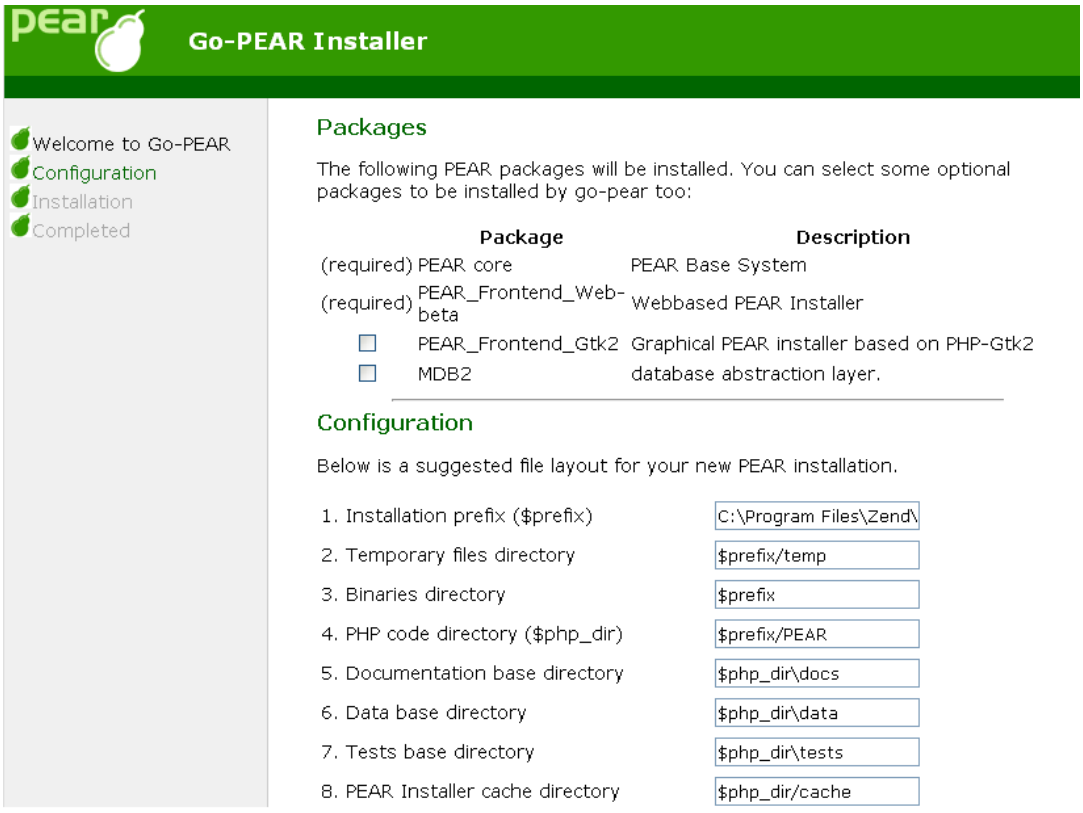


図 10-1. Go-PEAR のインストレーション設定

３． PEAR をインストールするとき必ず次の設定をしておく：

“1. Installation Prefix ($prefix)” = “C:\Program Files\Zend\ZendServer\bin”

４． PEAR のインストレーション・パス(C:\Program Files\ZendServer\bin\PEAR) を”Path” を”Path” 環境変数に加える。その場所は MyComputer （右クリック）->Properties -> Advanced System Settings->Environment Variables にある。その変数を”PEAR”と名付ける。

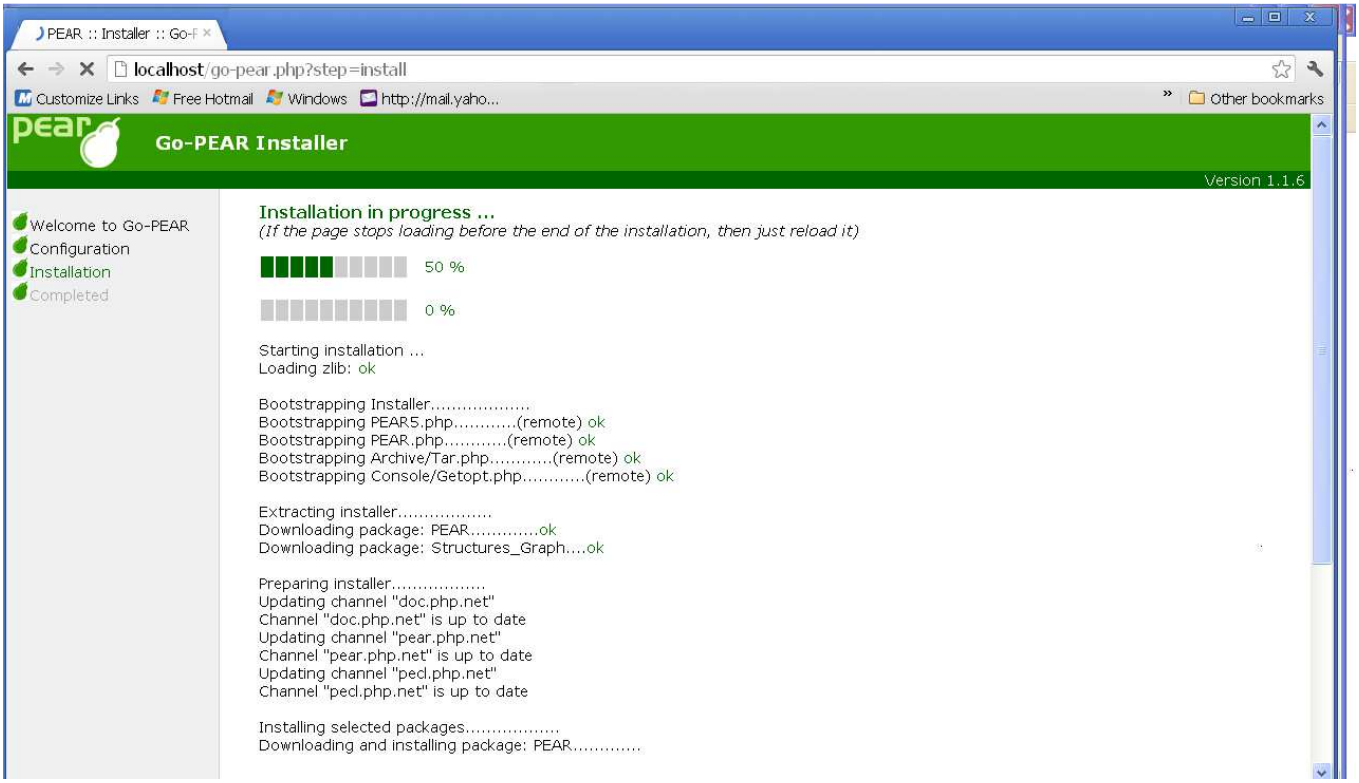


図 10-2. Go-PEAR のインストレーション途中のページ

５． C:\Program Files\Zend\ZendServer\share\etc\php.ini に行き、PEAR のパスを include path に加える。

```
[Zend]
include_path=”.; C:\Program
Files\Zend\ZendServer\share\ZendFramework\library;C:\Program
Files\Zend\zendserver\bin\PEAR”
```

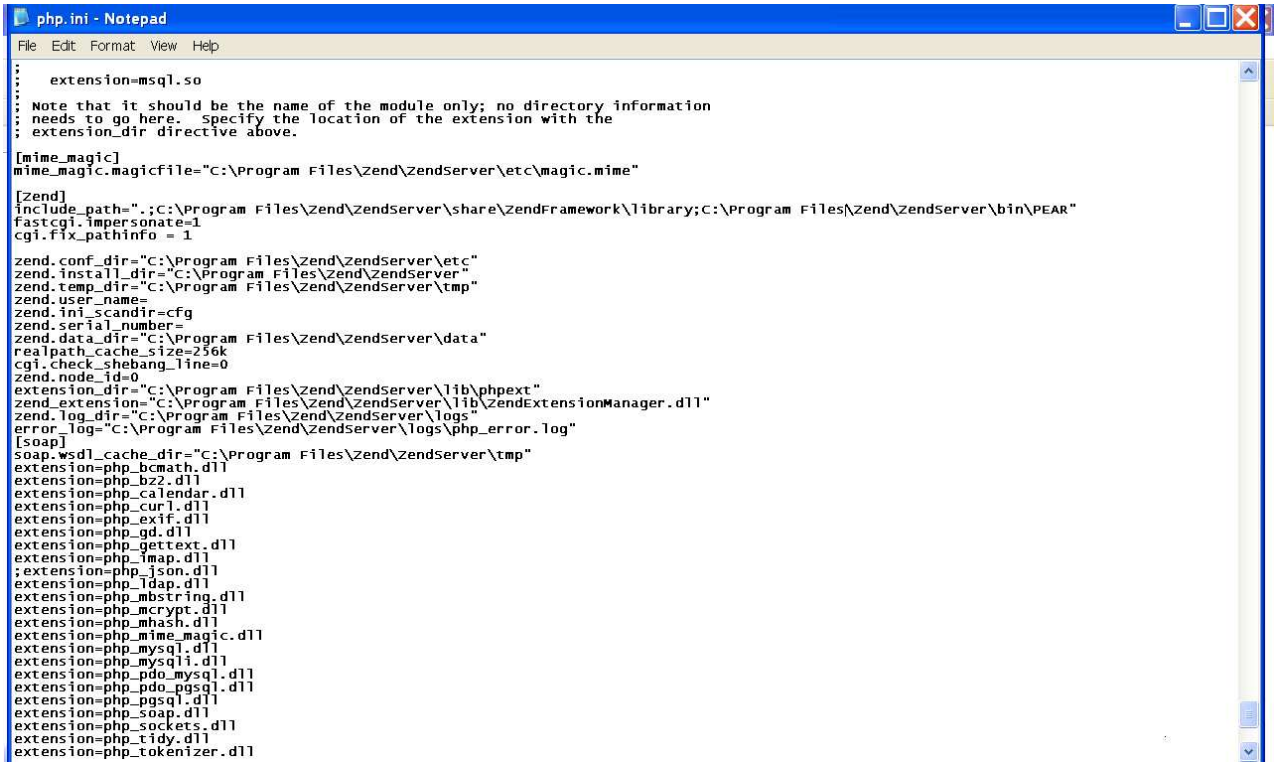


図 10-3. 修正した php.ini ファイル

6． PC をリスタートする。

Windows XAMPP Environment に PEAR をインストールする

1． フォルダー　C:\xampp\php に行きファイル pear.bat を見つける
2． そのファイルをテキスト・エディターで開ける。次の行を見つける.

IF “%PHP_PEAR_INSTALL_DIR%”== “ “　SET “PHP_PEAR_INSTALL_DIR=\xampp\php\pear”

3． その行の上に次の行を挿入する。

SET “PHP_PEAR_INSTALL_DIR=C: \xampp\php\PEAR”

4． 次に C:\xampp\php を PATH environment variable に含める。
5． Windows Vista ではそれを次で見つける

Start->ControlnPanel-> System and Maintenance ->System -> Advanced System Setting->Environment Variable

6． 次にコマンド入力欄に行きコ
7． マンド pear をタイプ・インする。そのコマンドに関する全てのオプションが出てくる。
8． 次のコマンドを使って pear パッケージをインストールする。

Pear install -o ‘path of package’

Windows WAMP Environment に PEAR をインストールする

１．　コマンド入力を開く
２．　PHP フォルダーに行く(例：C:\wamp2\bin\php\php5.9.2-2)
３．　“go-pear” を実行して PEAR をインストールする

Mac OS に PEAR をインストールする

Mac OS X の PEAR インストールをアップ・グレードする代わりに /usr/local/ に独自のインストールする。

/Application/Utilities/Terminal.app を開き次のコマンドを入力する：

$ cd /usr/local

次に、PEAR のインストールを開始する。次のコマンドを入力する

$ curl http://pear.php.net/go-pear | sudo php

要求されたときにはアドミニスターのパスワードを入力し、続く質問に答える。各質問にデフォルトの回答を受け入れるなら問題はないはずだ。スクリプト入力が終わると、PEAR は/usr/local/bin/ にインストールされる。PEAR ライブラリは/usr/local/PEAR/ でアクセスできる。

システム・パスの更新

カスタムな pear をシステム PATH にインストールする。君のプロファイルを vi (あるいは TextMate)にて生成または編集する：

$vi ~/ .bash_profile

このファイルが次のテキストを含んでいることを確認する：

PATH = /usr/local/bin:$PATH

ファイルをセーブし、この変更が効果を出すためにターミナル・アプリケーションをリスタートする。次に、pear が働いていることを確認する。

ターミナルでこのコマンドを入れる：

Which pear

このコマンドは次の応答をするはず：

/usr/local/bin/pear

PHP のインクルード・パスを更新する

次に PHP に PEAR ライブラリがどこにあるかを教えるため　/etc/php.ini ファイルにある PHP のインクルード・パスに PEAR を加える。

OS X 10.6 でデフォルトではこのファイルが存在しない。このファイルを生成するため、ターミナルで次のコマンドを走らせる:

$ sudo cp /etc/php.ini.default /etc/php.ini

次に、vi あるいは TextMate にある/etc/php.ini ファイルを編集し PHP のインクルード・パスを更新する。
/etc/php.ini で次の行を見つける：
　　;include_path = “/php/includes”
先頭の”;”を除き、PEAR ライブラリのパスを加える
　　Include_path = “/usr/local/PEAR: /php/includes”

/etc/php.ini を保存する。System Preference > Sharing > Web Sharing のチェックを外して、再びチェックを入れることで Apache web server をリスタートする。

次に、ウエブ・ブラウザで http://localhost/~[your_user_name]/index.php を開ける。”include_path” を探し、パスが”/usr/local/PEAR” を含んでいることを確認する。これで PEAR はインストールされた。

Linux に PEAR をインストールする

PEAR のパッケージは設定の可能な場所にインストールされる。Linux(あるいは Unix)システムではそれは通常/usr/local/lib/php である。

コマンド・ラインにコマンドをタイプする：
　　$ pear
もし次の応答があれば：
　　$ pear – command not found
それは PEAR が Linux のサーバにまだインストールされていないということだ。インストールするには、コマンド入力で次の手順で行う。

インストール用 php ファイルを http://pear.php.net/go-pear からダウンロードする。
　　$ wget http://pear.php.net/go-pear
php ファイルに名前を付ける
　　$ cp go-pear go-pear.php
php のスクリプトをコマンド・ラインから走らせる。
　　$ php go-pear.php
すると、インストールが始まるのがわかる。バイナリーがインストールされると、php の種々の拡張を次のコマンドでインストールできる。
　　$ pear install package

PEAR を使う- サンプル

Text -> CAPTCHA_Numeral

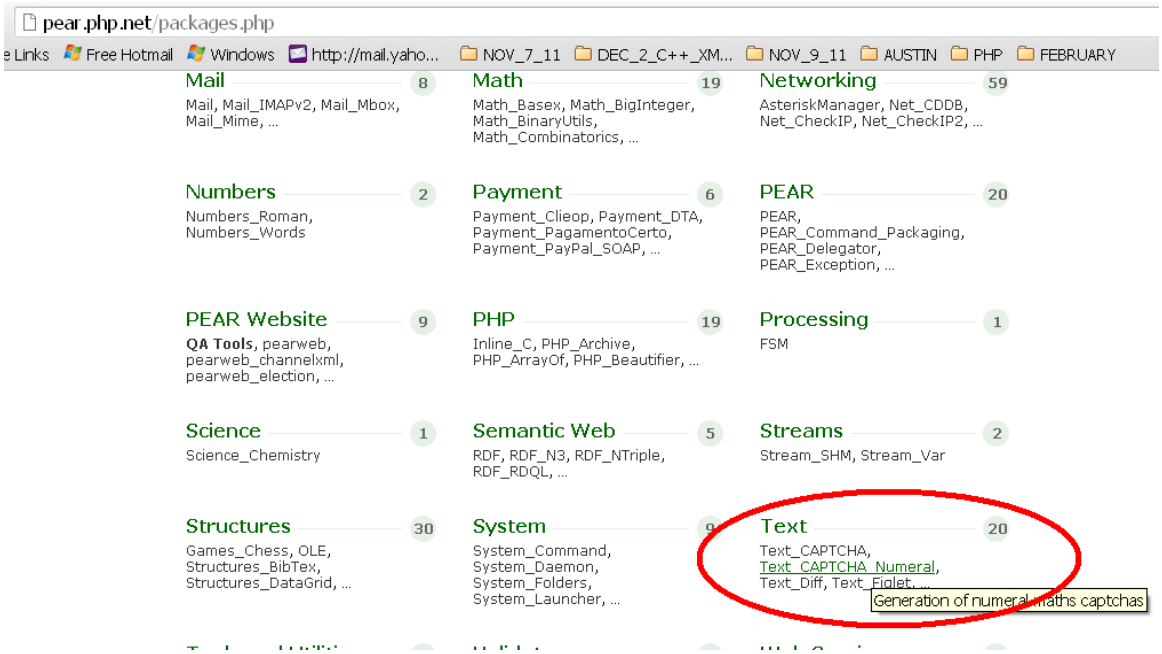


図 10-4. PEAR パッケージのページ

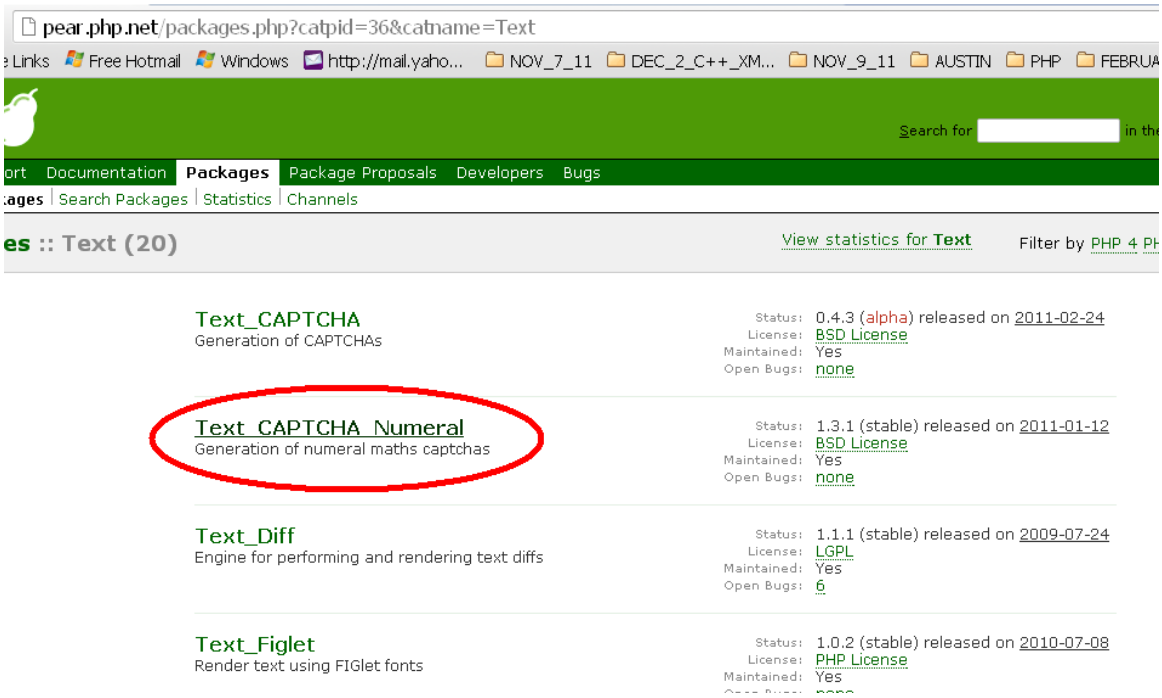


図 10-5. PEAR パッケージ–テキスト・パッケージのページ

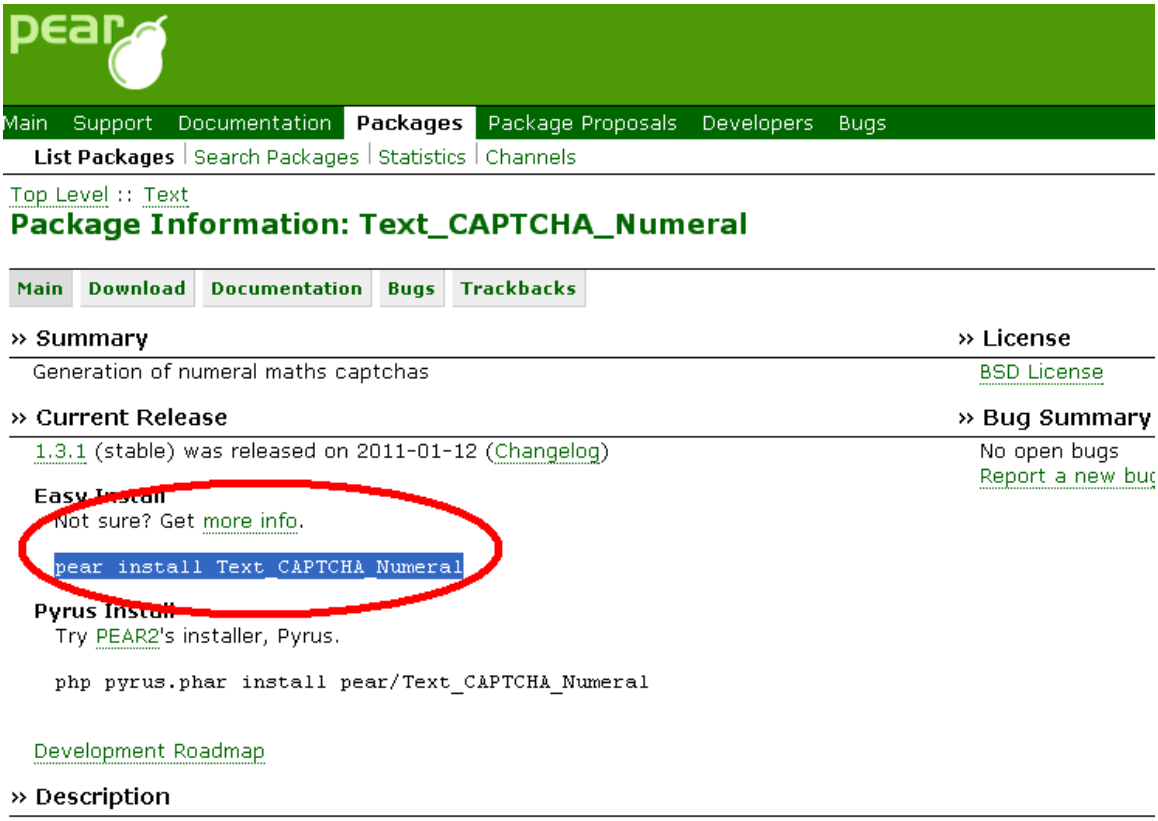


図 10-6. PEAR Text_CAPTCHA_Numeral インストールのページ

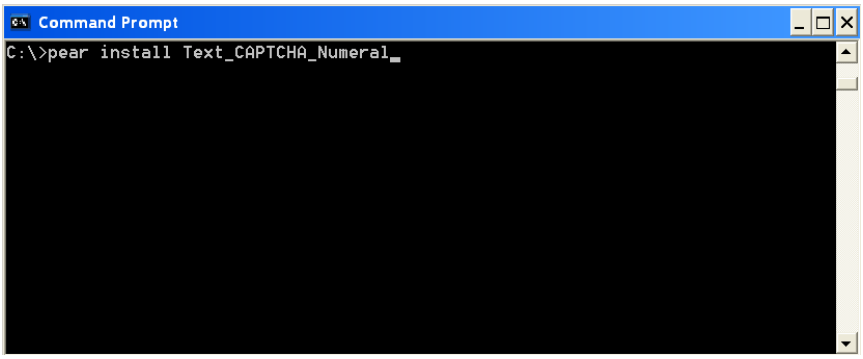


図 10-7. PEAR　Text_CAPTCHA_Numeral インストールのコマンド入力

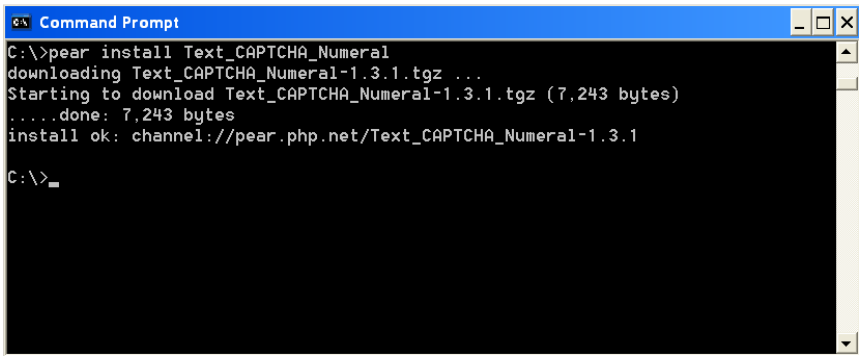


図 10-8. PEAR　Text_CAPTCHA_Numeral インストールのメッセージ

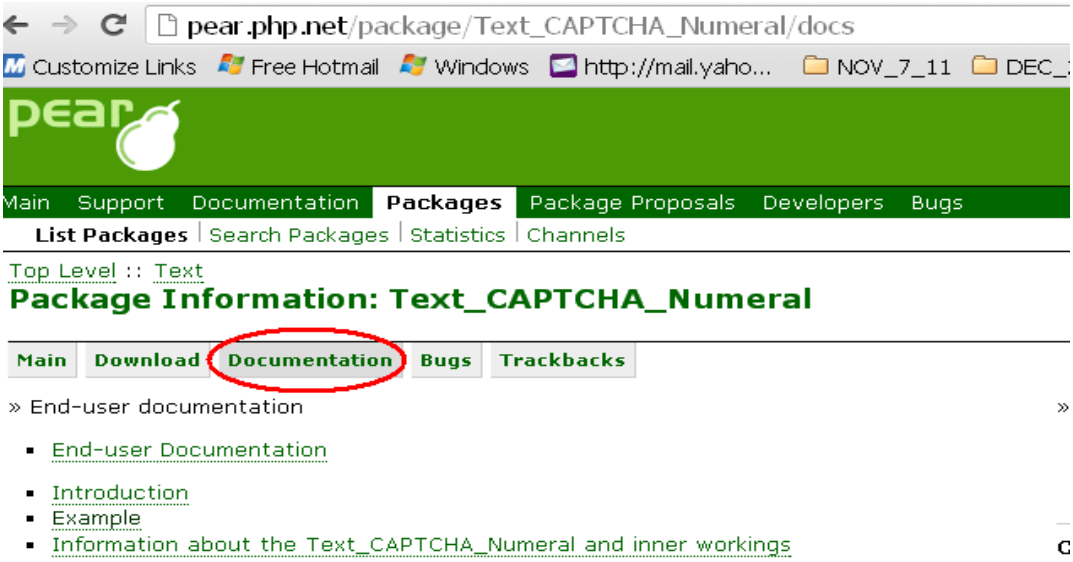


図 10-9. PEAR　Text_CAPTCHA_Numeral 文書類のページ

```
//test_captcha.php --
<?php
require_once 'Text/CAPTCHA/Numeral.php';
$numcap = new Text_CAPTCHA_Numeral;

if (isset($_POST['captcha']) && isset($_SESSION['answer'])) {
if ($_POST['captcha'] == $_SESSION['answer']) {
$errors[] = 'Ok... You might be human...';
} else {
$errors[] = 'You are dumb or not human';
}
}
if (!empty($errors)) {
foreach ($errors as $error) {
print "<h1><font color='red'>$error</font></h1><br />";
}
}


print '
<form name="capter" action="test_captcha.php?page=liveExample" method="post">
<table>
<tr>
<th>What is this result?: '.$numcap->getOperation().'</th>
<td><input type="text" value="" name="captcha" /></td>
</tr>
<tr>
<th/>
```

```
<td><input type="submit" value="Verify I am Human" /></td>
</tr>
</form>
';
$_SESSION['answer'] = $numcap->getAnswer();
?>
```

図 10-10. PEAR　Text_CAPTCHA_Numeral ブラウザーでの表示

第 11 章　PHPUnit

PHPUnit でするユニット・テスト

ソフトウエアを検査することは必須であるが、リリースの前にそれを待つこととそれをすること自体もまた頭に血が昇ることになる。大充血だ。読者の中に『スリー・アミーゴス』のファンがいれば大充血の意味も分かってもらえると思うが。機能テストはリリース直前のテストをするというアプローチで、多くの開発者にはよく分かっているものだ。特にユーザの受け入れテストはユーザ・インタフェイスの機能をやってみることで始まる。ユーザの受け入れテストの最中で見つかった問題は修理が高く掛かり、ソフトウエアのリリースを遅らせることになりかねない。特に密に結合されたコードで構成されたよく整理されていない応用ではその問題は深刻だ。変更の要求は開発者が変化が正しく実装されたかどうかを検証するのに機能テストにたよっている場合に同程度に実施が困難で高くつく。

要点はどのプログラマーもミスをするが、良いプログラマーはできるだけ早くにミスを見つけるためにテストをするということだ。ミスに対するテストを早くするほどそれを見つけるチャンスは大きく、それを見つけるためのコストが少なくて済む。このことから分かると思うが、テストをソフトウエアのリリース直前まで置いておくことやプログラムが期待したように動くことだけチェックするといったことでは不十分で問題を招きやすいのだ。多くのエラーがこの種のテストをすり抜けて生産途上に出現し、開発者に途絶感を与えとんでもない恥をかかせるのだ。

このようにエラーが生産にこっそり入り込むのを最も効果的に防ぐ方法は、ソフトウエアの部品やユニットが正しいことを実行可能なコードの断片を用いて自動的に一連のテストを行うことで開発の初期にエラーを捕まえることだ。これらの実行可能なコードの断片はユニット・テストと呼ばれる。ユニット・テスト・ソフトウエアの枠組みでユニット・テストを自動化することで、コード・モジュールが意図通りにエラーなく動くことを保証してくれる。

PHPUnit は PHP のためのユニット・テスト用ソフトウエアの枠組みで、契約による設計開発の概念と手法を用いてオブジェクト指向 PHP 応用の開発を支援する。これは xUnit ファミリーの枠組みの一つで、Java には JUnit, .NET には NUnit が含まれている。

PHPUnit は Sebastian Bergmann によって開発されたもので、2011 年の 4 月に Ed Heal によって開発された PHP Unit Testing Framework と混同しないよう。他にも PHP のテスト用の枠組みがあるが、PHPUnit は PHP プロジェクトのユニット・テストの事実上の標準となっていて、ここで紹介するただ 1 つのものである。

PHPUnit をインストールする

PHPUnit は PEAR (PHP Extension and Application Repository) インストーラを用いてインストールしなければならない。

Windows Zend Framework Environment で PHPUnit をインストールする

始める前に C:\Users\YOUR_USER_NAME\ppData\Local\Temp にある物を全て削除する。

1．cmd ウインドウを開く。アドミニストレータとして入る必要があるかも知れない。
2．次のコマンドを 1 つづつ分離して実行する(各 1 回だけ！)
   pear channel-discover pear.phpunit.de
   pear channel-discover components.ez.no
   pear channel-discover pear.symfony-project.com
   pear clear-cache

3．次を実行する：

   pear install phpunit/PHPUnit

Windows XAMPP Environment で PHPUnit をインストールする

以下の手順は xampp が C:\xampp にインストールされていると仮定している。

1. コマンド入力画面を開き C:\xampp\php に行く
2. 次のコマンドを別々に実行する(各 1 回だけ)

pear upgrade-channels

pear upgrade

pear channel-discover components.ez.no

pear channel-discover pear.symfony-project.com

pear channel-discover pear.phpunit.de

pear clear-csche

pear install –alldeps phpunit/PHPUnit

注意：

もしなんらかのメモリ・エラーが起こるときには php.ini の”memory_limit” を編集する必要があるかも知れない。それを何か非常に大きい値にセットし、終わった時に戻すようにする。

Mac OS X で PHPUnit をインストールする

PHPUnit を Mac OS X にインストールには第 9 章の PEAR の Mac OS のためのインストールを手順に従って行ったと仮定する。次のコマンドをターミナル画面で入力する：

$ sudo pear channel-discover pear.phpunit.de

次に、これをターミナル画面で入力する：

$ sudo pear install phpunit/PHPUnit

これで終わりだ。PHPUnit はインストールされ、/usr/local/PEAR/PHPUnit/ に入っている。これで PHPUnit を PHP スクリプトに次のラインをインクルードできる：

Require_once 'PHPUnit/Framework.php';

Linux で PHPUnit をインストールする

コマンド・ラインで、次のコマンドを入れる：

```
$  ./bin/pear  channel-discover  pear.phpunit.de
$  ./bin/pear  channel-discover  components.ez.no
$  ./bin/pear  channel-discover  pear.symfony-project.com
$  ./bin/pear  install  phpunit/PHPUnit
```

テスト固定具

テスト固定具(Test Fixtures)とはテストを実行するためのベースラインとして使う固定した状態あるいは前提条件のことを指し、本来テスト目的のためにだけ生成されたデータセットである。それはテスト状況と言ってもよい。開発者はテストの前に分かっている良い状態を設定し、テストの後元の値に戻すべきである。テスト固定具の役割はよくわかった固定の環境をテストに提供し結果が繰り返されるようにする。一般的な固定具の例は、ハードディスクを消去し、よく知られた、クリーンなオペレーティング・システムをインストールすることや、特定の、知られたデータのセットを持つテスト用データベースをロードすることなどである。テストがデータを変える可能性があるので、実際のデータベースをユニット・テストには使用しない。テスト・データベースなどの固定具は各テスト組が実行される前にロードされているべきである。

テストのフェーズ

セットアップ－　テスト固定具の設定

実行　－　テストでシステムとやり取りをする

検証　－　期待した出力が得られたかどうかを決定する

解体　－　テスト固定具を取り壊して最初の状態に戻す。

テストのケースを書く

PHPUnit でテストのケースを書くのは非常に簡単だ。PHPUnit/Framework.php をインクルードし、PHPUnit_Framework_TestCase に拡張するクラスを宣言したうえで、テスト機能を"test"の名前で始めるようにし、条件があっているかどうかチェックするために$this->assertTrue を用いる。次に示すのは PHPUnit を使う非常に簡単なサンプルの全体である。

```
<?php
//test_assertion.php

class SampleTest extends PHPUnit_Framework_TestCase {
  private $x;
  private $y;

  public function setup() {
    $this->x = 4;
    $this->y = '4';
  }

  public function testEqual() {
    if ( ($this->x == $y) && ($this->x!= NULL) && ($this->y!= NULL) ) {
        echo "$x is equal to $y";
        }
  }

  public function testIdentical() {
    if ( ($this->x === $y) && ($this->x!= NULL) && ($this->y!= NULL) ) {
        echo "$x is identical to $y";
        }
  }


  public function teardown() {
    $this->x = '';
    $this->y = '';
  }

}
?>
```

この中で setup() と teardown() メソッドがテスト固定具の初期化と除去の役割をしている。

判定メソッド

判定(assertion)とはシステムの状況についての仮定をチェックするメソッドである。判定を使う典型的な方法はテスト対象のコード文や部分の出力の期待を定義することである。もしテストの出力が期待したものでなければテストは失敗となり、警告メッセージが造られる。判定メソッドへの呼び出しはテストしているシステムの想定される行動を文書化する助けにもなる。

上記のテスト・ケースのサンプルを少し変えて代わりに判定メソッドを使ってみよう：

```
<?php
//test_assertion.php

class SampleTest extends PHPUnit_Framework_TestCase {

  private $x;
  private $y;

  public function setup()
  {
    $this->x = 3;
    $this->y = '7';
  }

  public function testEqual() {
    $this->assertTrue($this->x == $this->y);
  }

  public function testIdentical() {
    $this->assertTrue($this->x === $this->y);
  }

  public function teardown() {
    $x = '';
    $y = '';
```

```
  }
}
?>
```

全ての判定メソッドは最後の引数としてオプションの記述をとることが出来る。これは結果の表示へのラベルとなる。もし省略すれば、既定のメッセージが代わりに出力されるが、通常それでも十分である。この既定のメッセージは独自のメッセージに”%s”を入れておくと、そこにはめ込まれる。この例では全ての判定はパスの場合に true 失敗の場合に false を戻す。

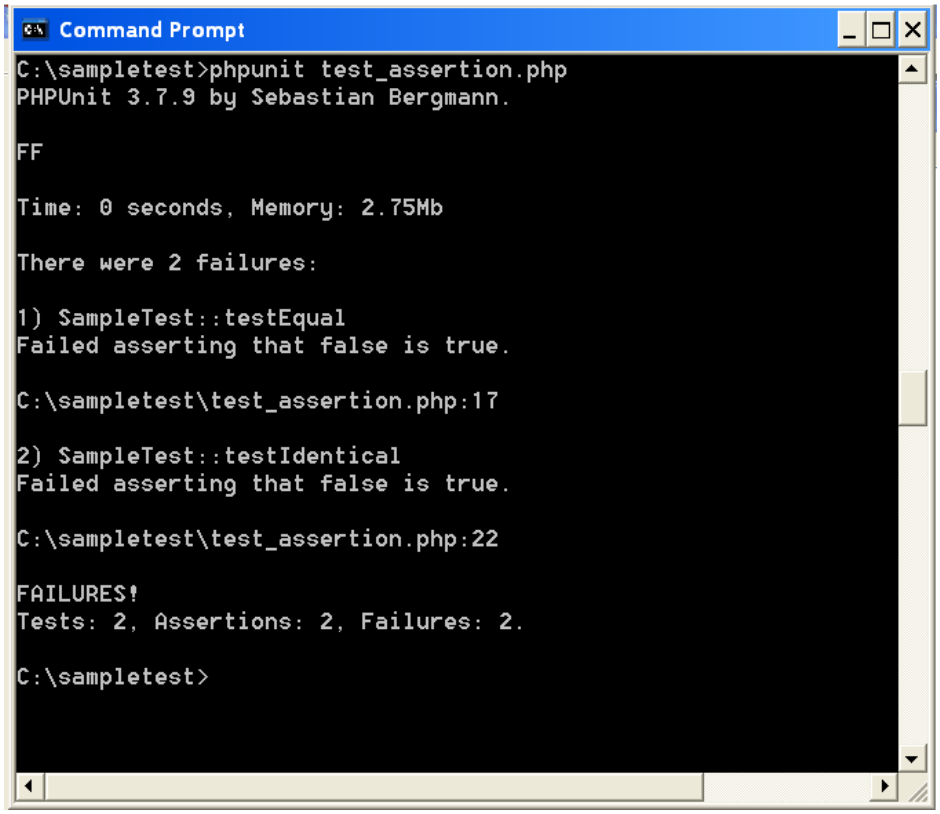


図 11-1. PHPUnit サンプル・テスト：同等の判定に失敗

これと同じ条件を assertEqual() と assertSame() メソッドを使ってテストしよう。

```
<?php
//test_assertion.php

class SampleTest extends PHPUnit_Framework_TestCase {

  private $x;
  private $y;

  public function setup()
  {
    $this->x = 3;
    $this->y = '7';
  }

  public function testEqual() {
    $this->assertEquals($this->x == $this->y, 'Failed' Arguments not equal!);
  }

  public function testIdentical() {
    $this->assertSame($this->x === $this->y, 'Failed' Arguments not identical!);
  }

  public function teardown() {
    $x = '';
    $y = '';
  }
}

?>
```

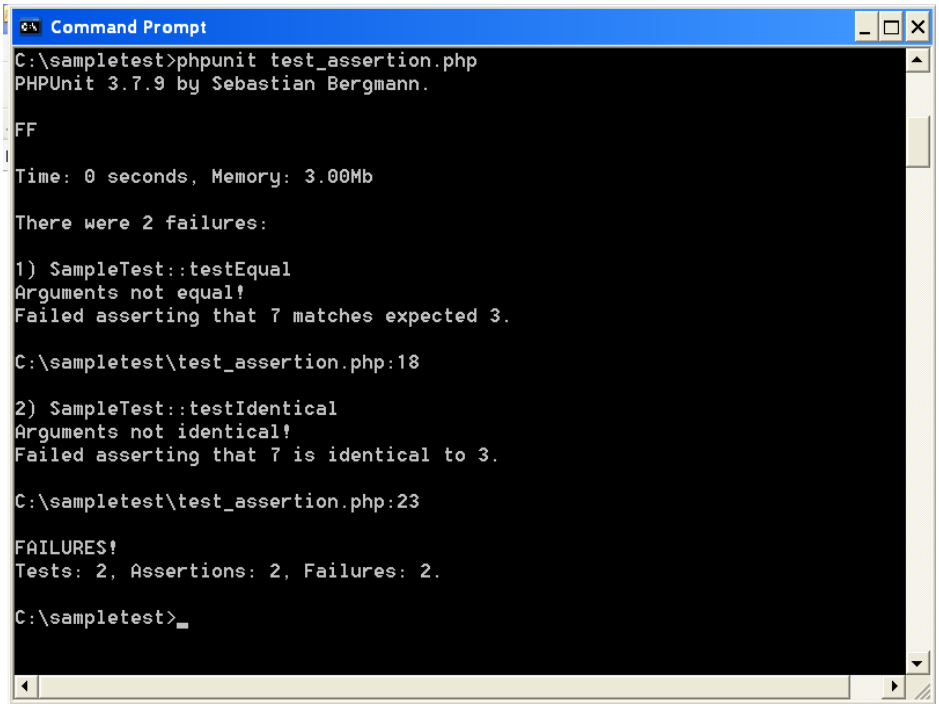


図 11-2. PHPUnit サンプル・テスト：同一の判定に失敗

さて$x と $y を互いに同等であるが異なるタイプとし、それらが同一でないことをテストしよう：

```
<?php
//test_assertion.php

class SampleTest extends PHPUnit_Framework_TestCase {

  private $x;
  private $y;

  public function setup()
  {
    $this->x = 3;
    $this->y = '3';
  }

  public function testEqual() {
    $this->assertEquals($this->x == $this->y, 'Failed' Arguments not equal!);
  }

  public function testIdentical() {
    $this->assertSame($this->x === $this->y, 'Failed' Arguments not identical!);
  }

  public function teardown() {
    $x = '';
    $y = '';
  }
}

?>
```

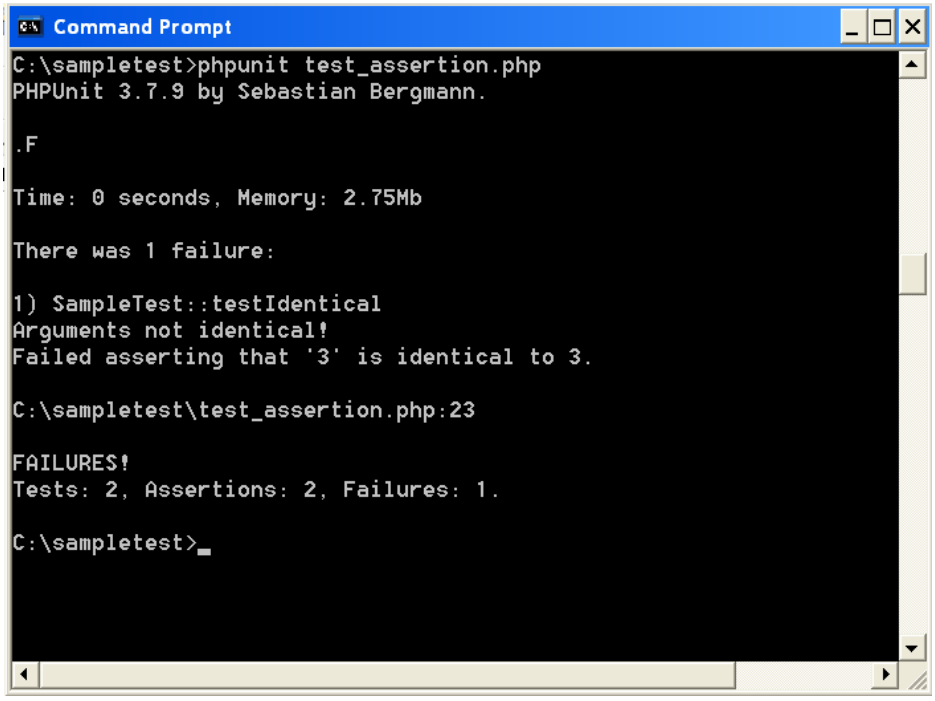


図 11-3. PHPUnit サンプル・テスト：同一の判定に失敗

最後に$x = 3 と $y = 3 としてこれらが同等で同じタイプとして実行してみよう:

```
<?php
//test_assertion.php

class SampleTest extends PHPUnit_Framework_TestCase {

  private $x;
  private $y;

  public function setup()
  {
    $this->x = 3;
    $this->y = 3;
  }

  public function testEqual() {
    $this->assertEquals($this->x == $this->y, 'Failed' Arguments not equal!);
  }

  public function testIdentical() {
    $this->assertSame($this->x === $this->y, 'Failed' Arguments not identical!);
  }

  public function teardown() {
    $x = '';
    $y = '';
  }
}

?>
```

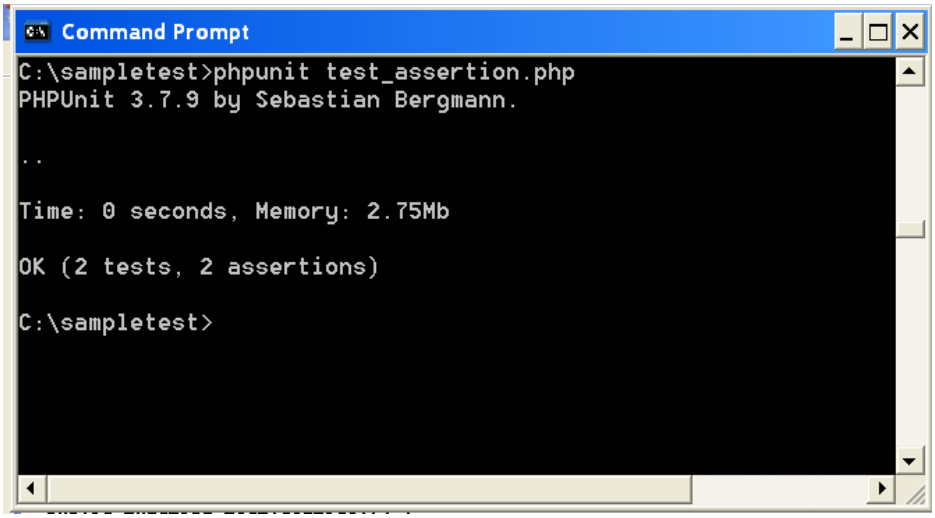


図 11-4. PHPUnit サンプル・テスト：パス

以下は PHPUnit に組み込みの assertion メソッドである：

assertArrayHasKey()
assertClassHasAttribute()
assertClassHasStaticAttribute()
assertContains()
assertContainsOnly()
assertContainsOnlyInstancesOf()
assertCount()
assertEmpty()
assertEqualXMLStructure()
assertEquals()
assertFalse()
assertFileEquals()
assertFileExists()
assertGreaterThan()
assertGreaterThanOrEqual()
assertInstanceOf()
assertInternalType()
assertJsonFileEqualsJsonFile()
assertJsonStringEqualsJsonFile()
assertJsonStringEqualsJsonString()
assertLessThan()
assertLessThanOrEqual()
assertNull()
assertObjectHasAttribute()
assertRegExp()
assertStringMatchesFormat()
assertStringMatchesFormatFile()
assertSame()
assertSelectCount()
assertSelectEquals()
assertSelectRegExp()
assertStringEndsWith()
assertStringEqualsFile()
assertStringStartsWith()
assertTag()
assertThat()
assertTrue()
assertXmlFileEqualsXmlFile()
assertXmlStringEqualsXmlFile()
assertXmlStringEqualsXmlString

表 11-1. PHPUnit の Assertion メソッド

HTML のフォームから提出されたデータをテストする

開発したレストラン・メニューの応用でメニュー品目のデータをデータベースに入れるフォームを生成した。そのときフォームからの値を集めたアレイとしてあらかじめ定義された$_POST 変数をもちいた。

PHP での開発でフォームは必ず扱わなければならないものである。従って、フォームから入力されたデータをテストする必要がある。それはデータを$_GET と　$_POST のアレイで転送する GET メソッドと POST メソッドをテストすることである。ここで「依存性の注入」（Dependency Injection）のアイデアを見てみよう。それはコードにそれが必要とするものを与えるという最良の方法で、必要とするデータを獲得するというのと逆である。サンプルでこの違いを示そう。

依存性の注入の無いサンプル：

```
function AddMenuItem1() {
  foreach($_POST as $form_element => $val) {
    // ここに$val を処理するコード

  }
}

AddMenuItem1();
```

依存性の注入によるサンプル：

```
function AddMenuItem2(array &$formData) {
  foreach($formData as $form_element => $val) {
    // ここに$val を処理するコー
  }
}

AddMenuItem2($_POST);
```

違いがわかるだろうか？ PHPUnit のテストで AddMenuItem2() には選択できる連想配列(訳注 associative array：添え字に文字列も含むスカラー数値以外のデータ型も許す配列)を渡す；依存性を注入している。これに対して AddMenuItem1() は$_POST と結合している。いずれにしても$_POST と $_GET は連想配列であり、コードを開発するにあたって関数に$_POST と $_GET のどちらでも渡せるが、ユニット・テストではいくつかの予想されるデータを確実にコードしなければならない。

ユニット・テストのサンプル：

```
function testAddMenuItem() {
  $fakeFormData = array ('itemname'=>'Ice Tea', 'description'=>'tea with ice', 'servingsize'=>'16 oz',
'price'=>'$1.39');
  AddMenuItem($fakeFormData);
  // 何かを判定する...

}
```

テスト組

テスト組とは同じ固定具を共有する一組のテストである。テストの順序で影響があってはいけない。上記の test_assertion.php のサンプルでは 2 つの簡単なテストが$x と $y の値を割り当てる同じ setu p() の固定具を用いた。これは非常に簡単なテスト組のサンプルだ。

模擬と控え

控え(stubbing) はオブジェクトを設定された戻り値を戻す 2 重テストで置き換える手続きである。

模擬(mocking)はオブジェクトを行動を検定する 2 重テストで置き換える手続きである。模擬オブジェクトはユニット・テストでテスト・ケースの実際のオブジェクトの行動を模擬するのに使われる。これらを使うことで実装しようとしているオブジェクトの機能がより簡単にテストできる。模擬オブジェクトは 1 つあるいは複数の依存性がまだ実現されていない場合や、1 つあるいは複数の依存性がシミュレートの困難な要素を含んでいる場合などに有用である。このような状況の例はデータベースと応用とのやり取りをテストする場合である。よく行われるのはメソッドや何かデータにアクセスするオブジェクトを呼ぶことだが、まだデータベースが設

定されていない場合はどうするかだ。あるいはまだ使えるデータが無いとか、データベースに問い合わせるコードがまだ組まれていない場合もある。模擬データ・アクセスオブジェクトは実際のデータ・アクセスオブジェクトをあらかじめ定めて置いた値を戻すことで模擬する。こうすることでデータベースを設定し、ロードし、テスト・データからロックし、データベースへの問い合わせのコードを書くというトラブルから解放してくれる。架空のデータベースとの接続では模擬コードは接続を用いるオブジェクトが SQL などの問い合わせを正しく作成できているかをテストできる。

模擬クラスを生成する

前の章での抽象メニュー・クラスを思い出そう:

```
<?php
abstract class Menu
{
        private $menuid;
        private $parentid;
        private $menuitemid;
        private $menuname;
        private $description;

        public function setMenuID($menuid){$this->menuid = $menuid;}
        public function getMenuID(){return $this->menuid;}

        public function setMenuItemID($menuitemid){$this-> menuitemid = $menuitemid;}
        public function getMenuItemID(){return $this-> menuitemid;}

        public function setMenuName($menuname){$this->menuname = $menuname;}
        public function getMenuName(){return $this->menuname;}

        public function setDescription($description){$this->description= $description;}
        public function getDescription(){return $this->description;}

        function getAllMenuItems($menuid)
        {
                $connection = Database::Connect();
                $this->query = "select mitm.*, mi.itemname, mi.description,
                                        mi.price, mi.servingsize
                                        from `menuitemtomenu` as mitm
                                        left join `menuitem` as mi
                                        on mitm.menuitemid = mi.menuitemid
                                        where mitm.menuid = $menuid";

                $menuitemList = Array();
                $cursor = Database::Reader($this->query, $connection);
                while ($row = Database::Read($cursor))
                {
                        $menuitem = new MenuItem;
                        $menuitem->menuitemid = $row['menuitemid'];
                        $menuitem->itemname = $row['itemname'];
                        $menuitem->price = $row['price'];
                        $menuitem->description = $row['description'];
                        $menuitemList[] = $menuitem;
                }
                return $menuitemList;
        }

}
?>
```

控えを使って MenuTest() と呼ぶテスト用クラスでこの Menu クラスを模倣しよう：

```
<?php

require_once('/path/to/MenuModel.php');

class MenuTest extends PHPUnit_Framework_TestCase
{
    public function testConcreteMethod()
    {
        $stub = $this->getMockForAbstractClass('Menu');

         $result = array("menuitemid"=>"34",
                "itemname"=>"Nachos",
                "price"=>"$5.99",
                "description"=>"Ground Steak with peppers and onions layered with 5 different cheeses")));

        $stub->expects($this->any())
             ->method('getAllMenuItems')
             ->will($this->returnValue($result));

        $this->assertNotNull($stub->getAllMenuItems('3'));
    }
}
?>
```

上の例では PHPUnit _Framework_Testcase メソッドを getMock() と名付け、模擬したいクラスの名前である”Menu”をそれに渡す。このメソッドはその場で 1 つのクラスを発生させそこからインスタンス化して 1 つのオブジェクトを造る。そしてこの模擬オブジェクトを$stub に保存する。それから getAllMenuItem メソッドにオーバライドする。次の数行は実のキィーである：

```
<?php
$stub->expects($this->any())
               ->method(' getAllMenuItems ')
               ->will($this->returnValue($results));
?>
```

*雄弁なインタフェイス(fluent interface)*と言われるこれらの 3 行は控え(stub)オブジェクトに getAllMenuItems メソッドが呼び出されたときにはいつも、そこから$result にある値を代わりに戻す。PHPUnit によって発生された模擬（と控え）オブジェクトは上のサンプルに見るように expect() メソッドを持っている。このメソッドは照合オブジェクトを要求し、それがメソッドが呼び出されるべき回数、あるいは期待の濃度、を定義する。上の例では any()という便利なメソッドが expect() メソッドに 0 以上のコールが対応するメソッドに行われない限り照合は失敗したことを知らせる。控えオブジェクトが値を返すが呼び出しをテストしないのは便利である。

ここにいくつかの TestCase マッチャー・メソッドのメソッドを挙げる

テスト・ケース・メソッド	記述
`any()`	ゼロあるいはそれ以上の呼び出しが対応するメソッドに行われる
`never()`	対応するメソッドに全く呼び出しがない
`atLeastOnce()`	1 回あるいはそれ以上の呼び出しが対応するメソッドに行われる
`Once()`	1 回の呼び出しが対応するメソッドに行われる
`exactly($num)`	`$num` 回の呼び出しが対応するメソッドに行われる
`at($num)`	1 回の呼び出しが`$num` の番号で対応するメソッドに

行われる

表 11-2. テスト・ケース　照合メソッド

第 6 章から MenuItem のクラスをもう一度見よう：

```
class MenuItem
{
        // コンストラクタ(実装されていない)

        public function _construct(){}

        // 宣言なしのプロパティの設定

        function __set($property, $value)
        {
        $this->$property = $value;
        }

        // 定義されたプロパティの獲得

        function __get($property)
        {
                if (isset($this->$property))
                {
                        return $this->$property;
                }
        }
}
```

さて、このクラスの行動を模擬するためテストのクラスを設定しよう：

```
<?php
class MenuItemTest extends PHPUnit_Framework_TestCase
{
  public function testPHPUnitMock()
  {
    $mock = $this->getMock('MenuItem');

    //NOTE: this uses at() instead of once()
    $mock->expects($this->at(0))
         ->method('__set')
          ->with('itemname', 'Cajun Curly Fries');

    //NOTE: this uses at() instead of once()
    $mock->expects($this->at(1))
         ->method('__set')
          ->with('price', '$2.99');

    $mock->__set('itemname', 'Cajun Curly Fries');
    $mock->__set('price', '$2.99');
  }
}
?>
```

プラグインなしで Eclipse から PHPUnit を走らす

PHPUnit はコマンド・ラインから起動する代わりに Eclips から起動することもできる。多くの開発者はプラグインを使った Eclipse で PHPUnit を設定するのに困難を感じている。ここではプラグインを使う面倒さをなしで Eclipse から PHPUnit を走らせる方法を説明する。

先ず最初に、Eclipse を開く。レストラン・メニューのプロジェクトのためのユニット・テストを設定しよう。レストラン・プロジェクトのためのコードは全てプロジェクトのスペースの中で”source” と呼ぶフォルダーにある。同様にユニット・テストのためのフォルダーも必要なのでプロジェクトのスペースの中で”test” と呼ぶフォルダーを造る。

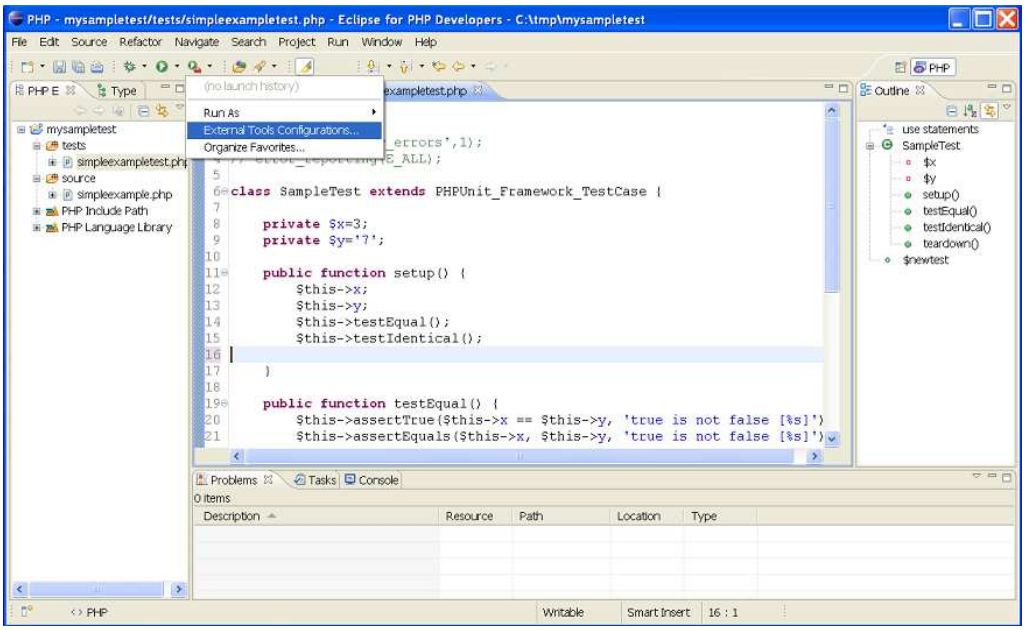


図 11-5. Eclipse 外部ツール　設定メニュー・オプション

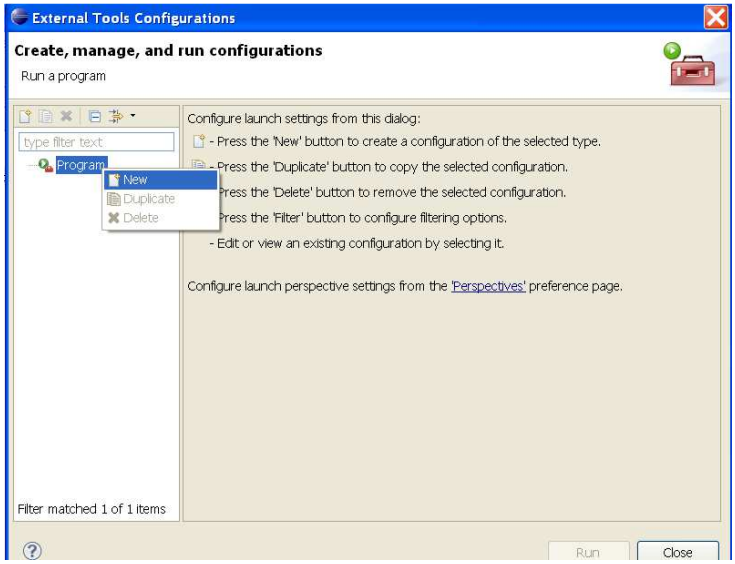


図 11-6. Eclipse 外部ツール設定ー新設定を選ぶ

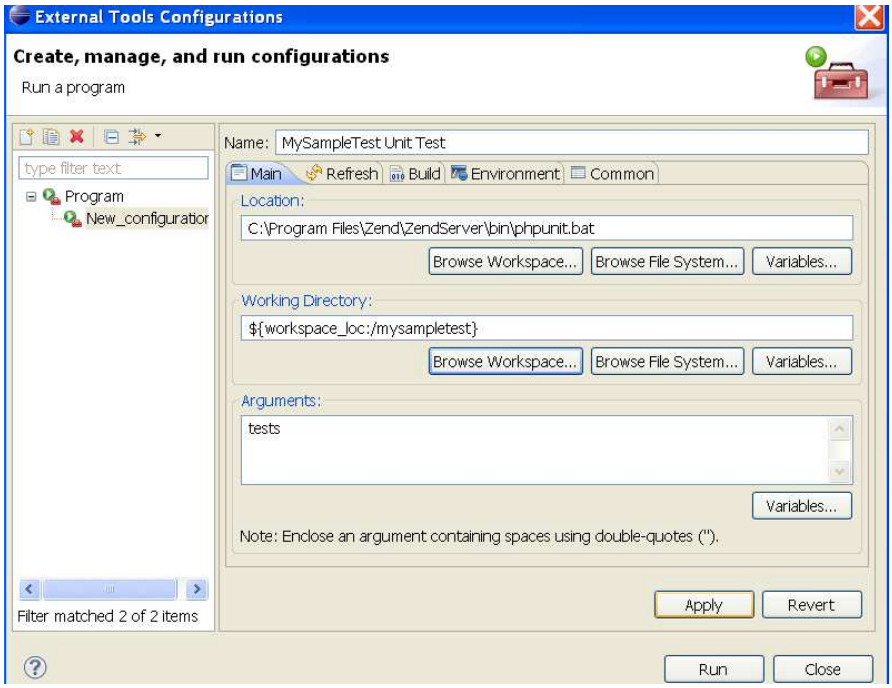


図 11-7. Eclipse 外部ツール設定－PHPUnit テスト・ツールの設定

Mac では phpunit へのパスは通常　/usr/local/bin/phpunit.exe にある。

この章の目的は PHPUnit とその重要な機能について知ってもらうことだった。もっと詳しい説明は PHPUnit の公式文書を次のサイトで参照したい。

http://www.phpunit.de/manual/current/en/phpunit-book.pdf

この文書のフランス語と日本語の版が www.phpunit.de で見つかる。

第 12 章　Subversion を用いたソース・コードの管理

最新のソフトウエア開発プロセスでの重要な要素にソース・コードの管理（バージョンの管理）がある。共同開発者は自分の変更をその都度共通のソースリポジトリ(repository、訳注：集積所)に送る責任がある。そうすることで、原始的なファイル共有システム(共有ドライブ、e-メール、ftp など)に頼ることなく、コード上での共同作業ができる。ソース管理ツールは全てのファイルの全ての前のバージョンを追跡記録しているので、開発者がソフトウエアを前後にサーチしていつ、どこでバグが発生したのかを決めることができる。あまりコミュニケーションの無い 2 人がほぼ同時にした変更が相容れないものだったことを見つけることもできる。自分や他人のソースを盲目的に書き換えることなく、正しい解決に導く。

各々の開発者はデッドラインに間に合わせようと必死で、自分のコードへの変更を急ぐため、コードを壊し、最後のうまくいっていた状態にまで慌てて戻すようなことが起こりやすい。バージョン管理なしでは、応用が大きくなればなるほど、ますますこのような問題が開発者に起こりかねない。特に、“小さな”問題が加わっていって積み重なり、解決に時間がかかり、コード・ベースが大きくなってしまう。この問題は複数の開発者が同じプロジェクトでバージョン管理あるいはソース管理なしで働く状況では、互いの領域を侵害し始める。前日に懸命に組んだコードが見当たらないとか、最後の変更をしたときのものと大幅に変わっていたりしたとき何が起こったのか分からず唖然とし混乱することほど悪い経験はない。

Subversion（よく SVN と略記される）の目的はこのような問題に対処し、比較的簡単であるが実用的で無料のサービスであり、複数の開発者が自分のコードを中央リポジトリの共通コード・ベースから取り出すことができる。Subversion のリポジトリは中央データベースであり、プロジェクトのバージョン管理されたファイルをその完全な履歴と共に保管している。リポジトリは Subversion のサーバを抱えるマシンに存在する。Subversion のサーバは Subversion のクライアント（Tortoise(亀)と呼ばれる）の要求に応じて内容を供給する。

この章は Subversion を紹介し、それを使った基本的な作業を行うための基本的なコマンド・ラインのツール（Unix, Mac,と Windows について）と、優れたグラフィックのツール(Eclipse IDE の共通プラットフォーム版の Subclips プラグインと；Windows グラフィック・シェルの TortoiseSVN 拡張)を紹介する。その基本的作業の概要と SVN の使用法は次のようだ：

- SVN を入手し組み込むこと
- プロジェクトを開始あるいは輸入する
- 基本的な Subversion のライフサイクル（次に示す）

１．リポジトリでプロジェクト(ディレクトリー・パス)を登録する
２．プロジェクト・ディレクトリーで、ファイルとサブ・ディレクトリを生成、編集する
３．リポジトリからのものでローカルのコピーを更新する。最後の更新以降に他のチーム・メンバがした変更を拾う。
４．ステップ 2 に戻る。もし自分の変更を届ける用意ができたらステップ 5 に行く。
５．自分の変更をリポジトリに届ける。ステップ 2 に戻る。

- ファイルとディレクトリを追加・削除する
- リリースを送り出す
- プロジェクトの枝分けをする

この章では Windows,Mac OS, と Unix/Linux システムで上記の 6 項目をいかに行うのかについて細かく述べていく。

Windows での Subversion

Windows でサーバー用、クライアント用の Subversion を立ち上げる

最初にすることは Windows 用最新版バイナリ・インストーラーを次のところからダウンロードすることだ：

http://subversion.apache.org/packages.html

あるいは、そのミラー・サイトでもよい：

http://sourceforge.net/projects/win32svn/

インストーラーの対話ボックスでインストールのパスをセットする：

C:\SVN\

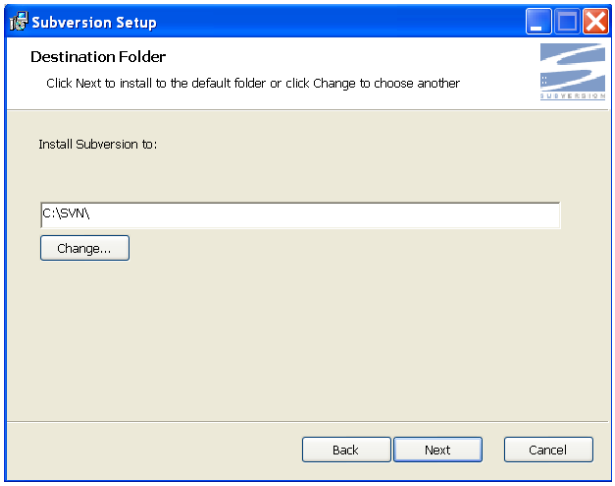


図 12-1. Windows での Subversion の設定―フォルダーの場所

インストーラは指定したパスに C:\SVN\bin を加えるので、コマンド入力を立ち上げ、直ちに使い始めることができる。

最初のソースリポジトリを生成し、それが実行的システムのパスとなる。

Svnadmin create “C:\SVN\repository”

この新しいフォルダーの中で、conf/svnserve.conf ファイルの中の次の 3 行を先頭のパウンド文字”#”を外して、コメントではなくする。

anon-access = none

auth-access = write

password-db = password

次に、何人かのユーザを conf/passwd ファイルに加える。次の既存の Harry と sally のコメント指定を外してもいいし、自身のものを加えてもよい：

harry = harryssrcret

sally = sallyssecret

Subversion 1.4 では、Subversion を Windows Service として簡単にインストールでき、そうすればいつでも使える。次のコマンドを入力すればよい。

sc create svnserver binpath = “C:\SVN\bin\svnserve.exe - - service -r c:\SVN\repository”

displayname= “Subversion” depend= Tcpip start= auto

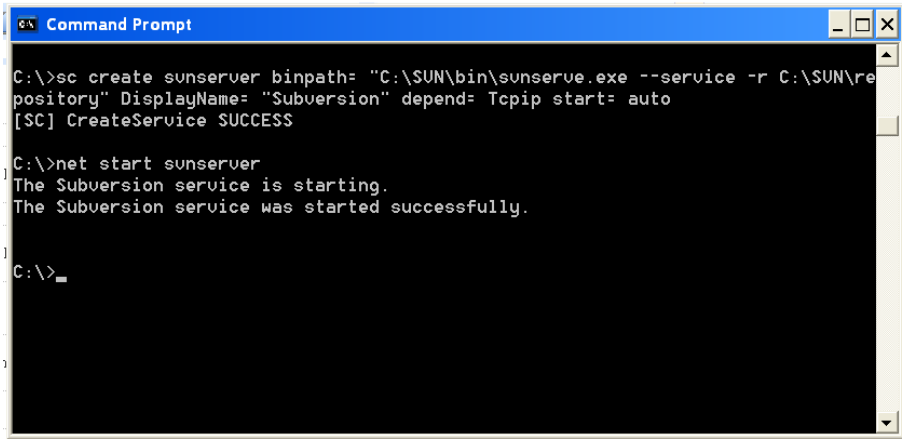


図 12-2. SVN の bin のパスを生成しウインドウズで SVN サービスを開始する

これで、自動開始に設定したので、サーバーが再起動したときには自動的にスタートする。再起動なしで開始するには、次のコマンドを入力する。

net start svnserver

サービスはローカル・システムのアカウントで走っている。通常問題はないが、君が後で Subversion hook script を追加することを計画しているなら、このサービスをアドミニスターのアカウントに移して、もっと広い許可を与えるようにしたいかもしれない。そうすることは簡単で旧来のウインドウズ・サービスの GUI で出来ることだ。

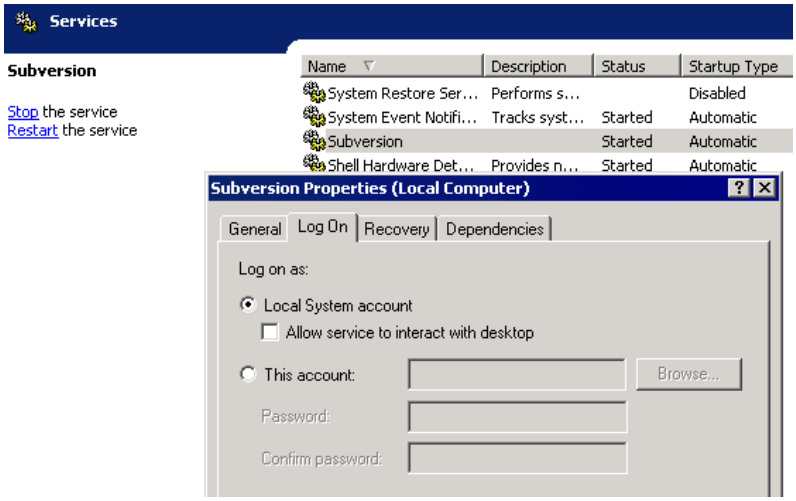


図 12-3. ウインドウズ・サービスのコントロール・パネル

さてローカルで作動していることを検証するため、project_1 と名付けた新プロジェクトのためにルートのフォルダーを加えよう。

set SVN_EDITOR=C:\windows\system32\notepad.exe

svn mkdir svn://localhost/project_1

すると、Subversion はユーザが開始コメントを書くために Notepad のコピーを開く。

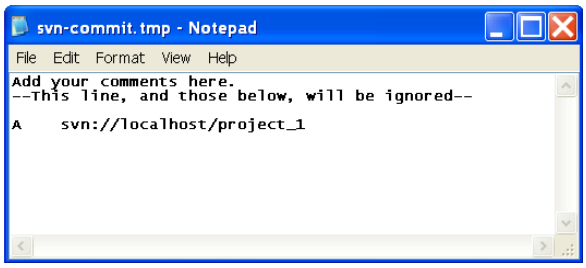


図 12-4.Svn の commit.tmp ファイル―コメントを入力する

何でも好きなコメントを入力すればよい。セーブして閉じる。ここで、資格認証があるが、アドミニストレータの資格認証は無視して enter キィーを押す。資格認証には先ほどの conf/passwd ファイルでセットしたものを使う。

svn mkdir svn://localhost/myproject
Authentication realm: <svn://localhost:3690>
Password for 'Administrator': [enter]
Authentication realm: <svn://localhost:3690>
Username: harry
Password for 'harry': ******

Committed revision 1.

いつも svn://をソース管理用パスの前置語として使う。ということは、Subversion 固有のプロトコル Subversion protocol operates on TCP port 3690 を使うということだ。サーバーのファイヤ・ウォールにクライアントからのアクセスが通過できるようにすることを確認する。そうでなければ、クライアントは接続することができない。

さてこれで、ローカルなウインドウズ・マシンに Subversion のサーバーが走っていることになった。

自分のソースコードを初めて入れる。

Subversion をインストールした後、次のステップはソース・コードをリポジトリに入れることだ。次のコマンドを使って既存のプロジェクトを輸入する。

svn import C://projects/trunk/proj_restaurant file:///usr/local/svnrepos/trunk -m “initial import”

“-m”のフラグは svn コマンドにコメントを付けておくのに用いる。

作業するためのコピーを生成する

コードはリポジトリに登録してあるので、次のステップは修正を加えるためコードのコピーを造ることである。それをワーキング・コピーと呼ぶ。これは svn のチェック・アウト・コマンドで行う。

svn checkout file:///usr/local/svnrepos/trunk C://workingcopyof_myproject

更新とワーキング・コピーの送出し

さて、生成したワーキング・コピーで変更を加え、組み合わせ、テストする。ファイルに加えた変更について Subversion に知らせる必要はない。自動的に検出してくれるからだ。しかし、もしバージョンのついたファイルを追加、コピー、移動、削除するときには、対応する svn の追加、コピー、移動、削除コマンドを使う必要がある。

ワーキング・コピーに変更を加え、すべて正しく動作していることを検証したなら、変更をリポジトリに書く必要があるだろう。そのときには、送り出し(commit) コマンドを使う。このコマンドは、ワーキング・コピーの変化ををリポジトリに書きこむ。

```
Commit  -m  ‘published  changes  made  to  my working  copy’
```

TortoiseSVN の立ち上げ

多くの開発者は Subversion とのやり取りをユニックスのスタイルのコマンドの代わりに TortoiseSVN と呼ぶシェルの拡張を使う。最新の 32 ビットあるいは 64 ビットのウインドウズ用クライアントをつぎのアドレスからダウンロードしインストールする：

http://tortoisesvn.net/downloads.html

インストーラはリブートするように言うが、そうしなくてもよい。

図 12-5. TortoiseSVN 立ち上げウイザードの歓迎ページ

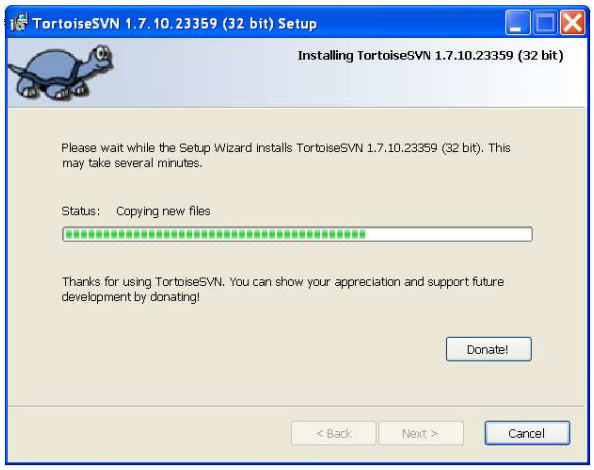


図 12-6. TortoiseSVN 立ち上げウイザードの状況ページ

ここで自分のドライブのどこかにプロジェクト・フォルダーを生成しよう。わたしは C:\project_1 を使った。TortoiseSVN はシェルの拡張なので、それとやり取りするには、エクスプローラで右クリックする。一度プロジェクト・フォルダーを生成した後は、その中で右クリックし、”SVN Checkout”を選ぶ。

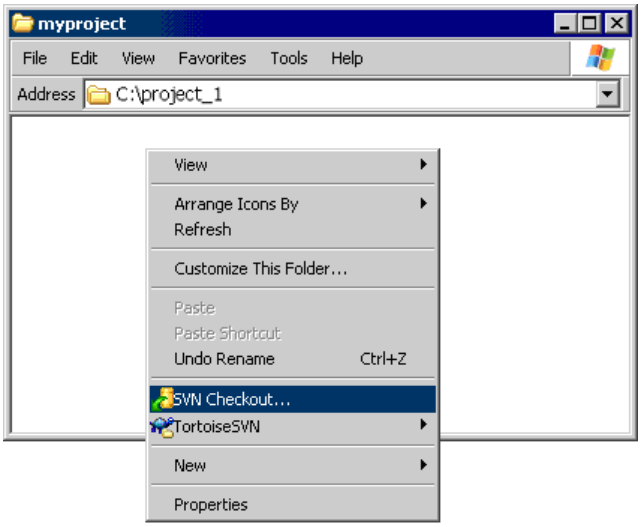


図 12-7.ウインドウズ・エクスプローラ―右クリック・メニュー・オプション

svn://localhost/project_1/ とリポジトリの URL に書き OK をクリックする。

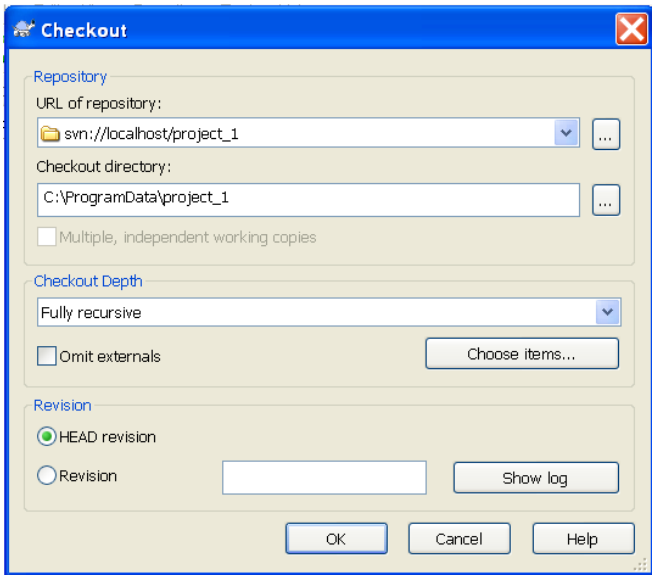


図 12-8 TortoiseSVN チェックアウト対話

Tortoise はソース管理に C:\project_1 フォルダーを svn://localhost/project_1 のパスに関連付けた。ローカルのファイルにしたことは何でもソース管理に反映される。

Subversion ではどのプロジェクトでもそのルートで”TTB　フォルダー“で開始するのが慣例となっている。TTB とは、Trunk, Tags, Branches の 3 フォルダーである。

というのは、subversion は、枝分けとタグ付けにも通常のディレクトリのコピーを用いるので、Subversion の仲間では各プロジェクトのためのリポジトリの場所を、プロジェクトに関するデータを含むディレクトリで最も先頭のルートを選び、そしてそのルートの下に 3 つのサブ・ディレクトリを生成する。それらは：trunk、す

なわちプロジェクトの主な活動が行われるディレクトリ； branches、種々の名前の核となる開発からの枝分かれのためのディレクトリ； tag、下げ札すなわち、断片的で、作っても、多分破棄され、しかし変化することがないものを集めた場所、である。

いくら多くの変更でもそれを束ねて 1 つのユニットにしかつ微細にチェックできることに注目したい。終わったらフォルダーを右クリックして”SVN Commit…”を選ぶ。

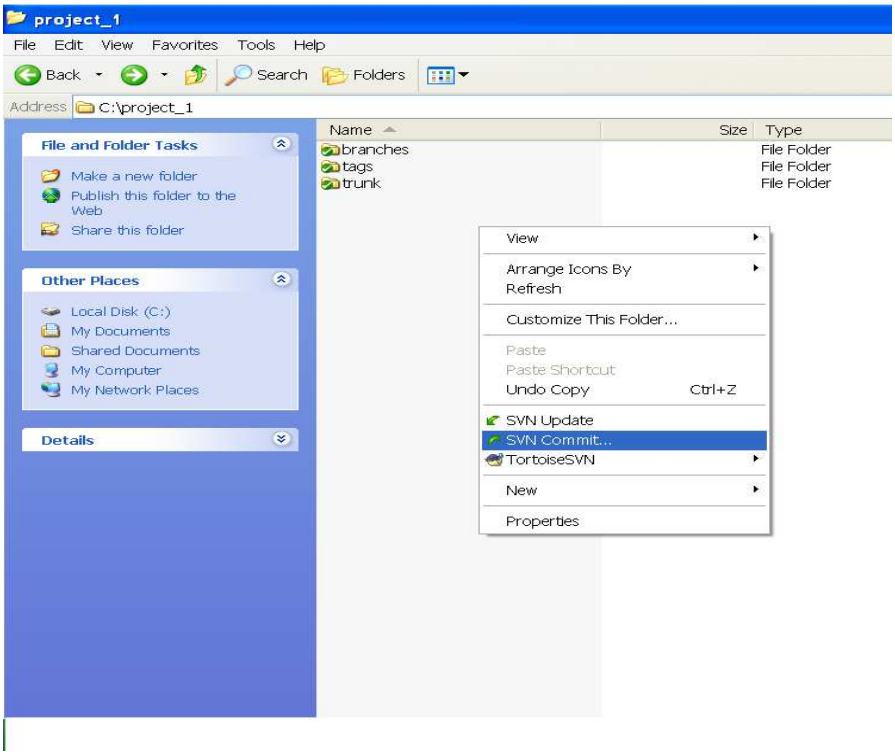


図 12-9. ウインドウズ・エクスプローラで右クリック・メニュー - >SVN Commit

Commit の対話で、yes, we do want to check in these files を選び、コメントを追加する。

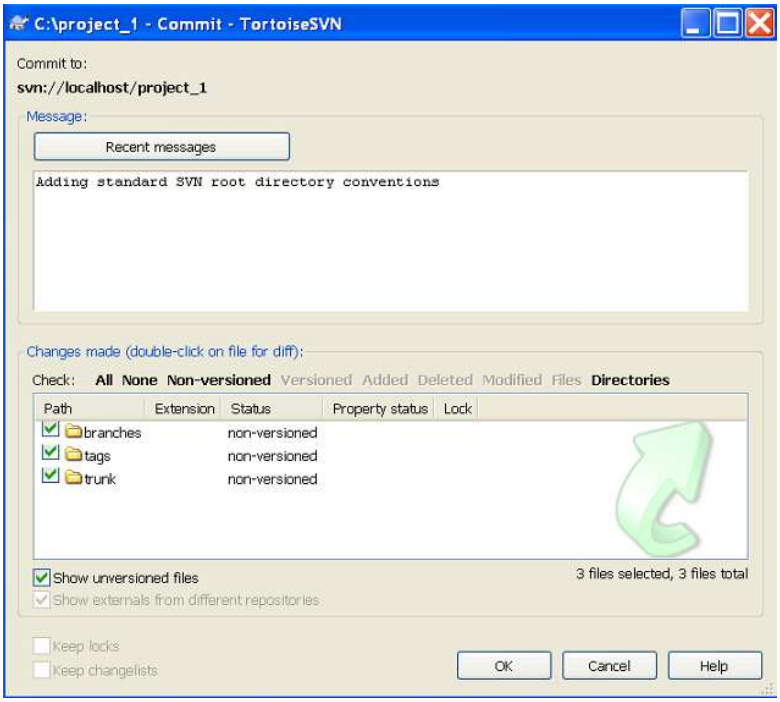


図 12-10. TortoiseSVN の Commit 対話

ここでサーバの資格認証を要求されるが、Tortoise はそれを便利に記憶して置いてくれる。Commit 送り出しが終われば、ファイルがシェルに現れソース管理のアイコンが重なる：

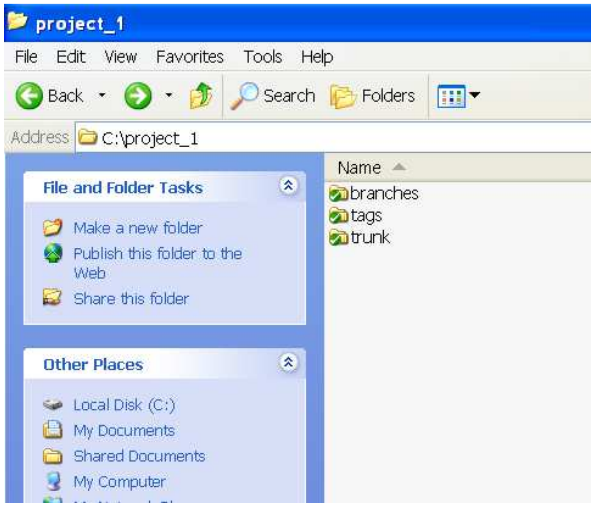


図 12—11. SVN ソース管理アイコンのオーバレイ

さて、右クリックで”TortoiseSVN, Setting”を選ぶ。

注意： Tortoise は各々のフォルダと君のシステムのドライブにソース管理のオーバレイを適用しようとする。このせいで厄介なファイルのロックが起こるかも知れない。Tortoise に特定のフォルダにシェル・コマンドを適用するように指示するには、その限定を”Icon Overlays”で設定する；exclude と include パスの入力を見つけよう。このサンプルでは exclude のパスを全てとし、include のパスを自分のプロジェクトにした。

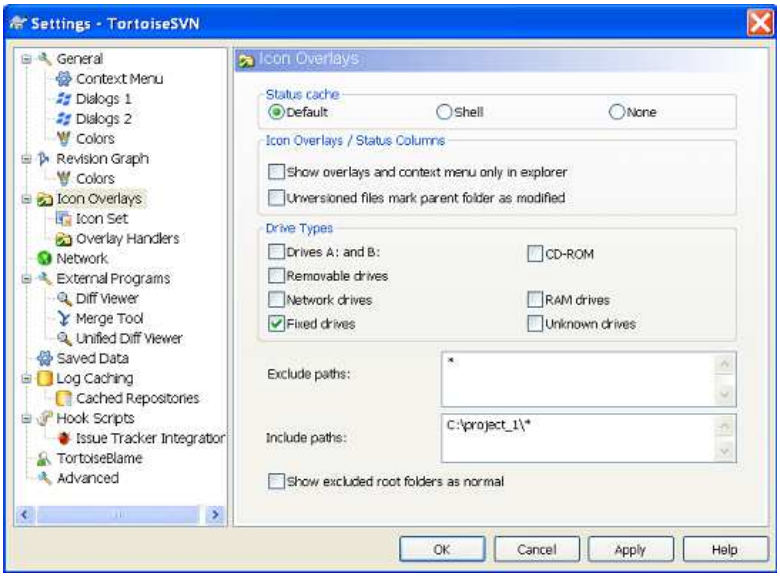


図 12-12.　TortoiseSVN ー設定コントロール・パネル

Tortoise はシェル・エクステンションなので、設定の変更にはリブートが必要であることに注意。

Tortoise で新しいファイルを追加するのは簡単だ。例えば、ノートパッド(あるいは君の好きなテキスト・エディタ)を開いて Employee.php と呼ぶファイルを生成する。新しいファイルで次をタイプする：

```
<?php

Class Employee ()

?>
```

次に Employee.php を C://project_1 にセーブし Notepad を閉じる。ウインドウズ・エクスプローラを用いて C://project_1 フォルダを開き、Employee.php ファイルの上で右クリックし TortoiseSVN のメニューから”add”を選ぶ。

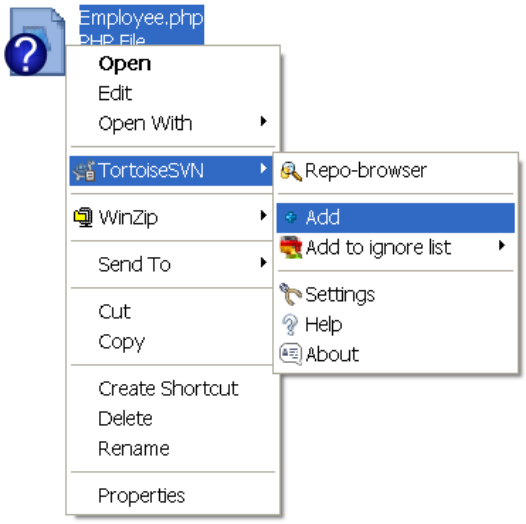


図 12-13.ウインドウズ・エクスプローラでの右クリックのメニュー・オプション->TortoiseSVN->Add

次にポップ・アップする対話でコメント欄に何かコメントを入力し、”OK”をクリックする。

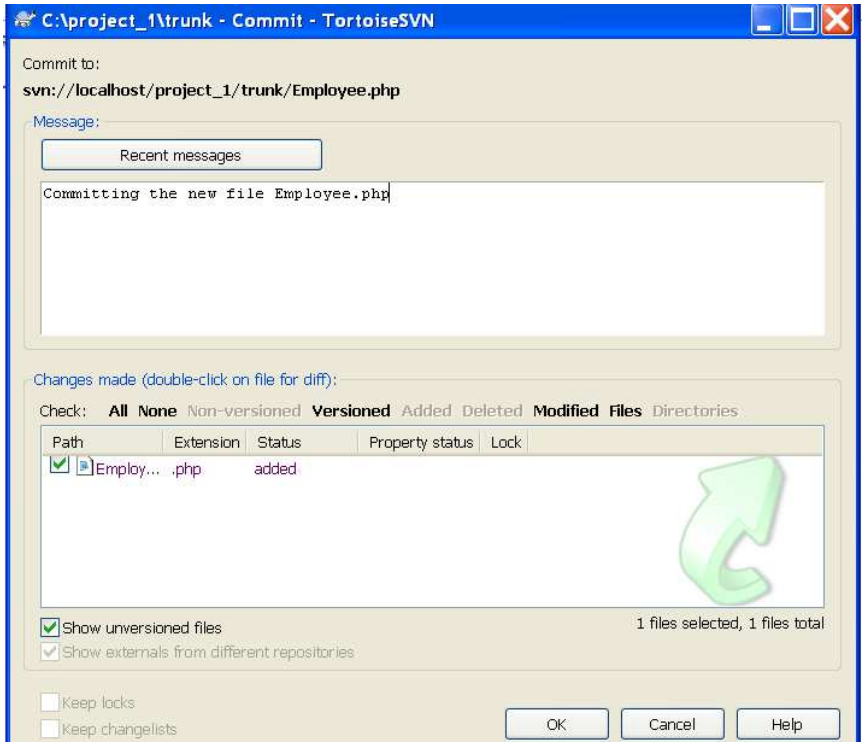


図 12-14.　TortoiseSVN のコミット対話

これで完了だ。新しいファイル Employee.php を SVN リポジトリに加えたところだ。

提出したものの輸出

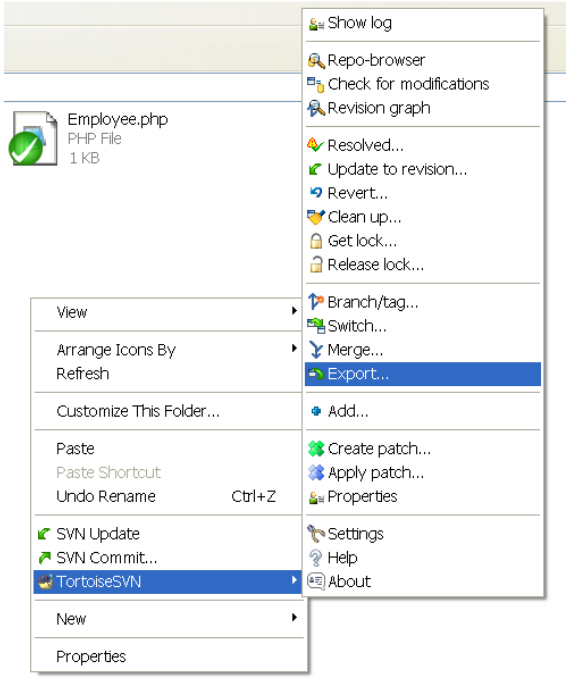


図 12-15.ウインドウズ・エクスプローラでの右クリックのメニュー・オプション->TortoiseSVN->Export オプション

ファイルのタグと枝分け

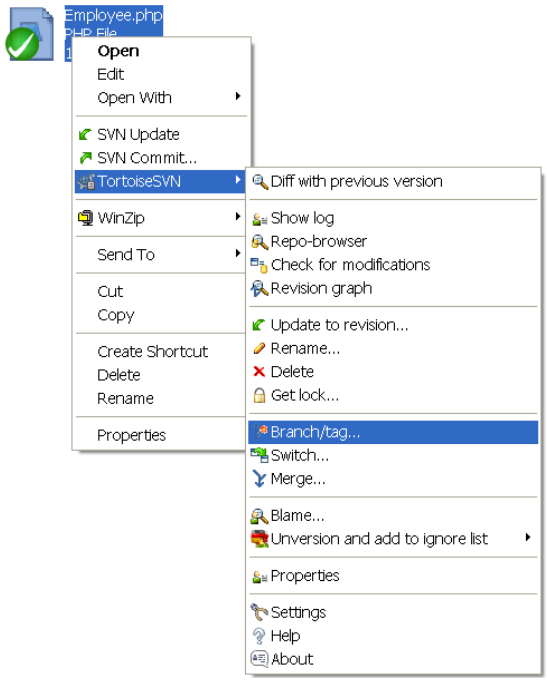


図 12-16.ウインドウズ・エクスプローラでの右クリックのメニュー・オプション->TortoiseSVN->枝分けオプション

Mac OS X

Mac OS X でサーバー用、クライアント用の Subversion を立ち上げる

Subversion のダウンロード

Subversion のクライアント用を http://subversion.apache.org/packages.html からダウンロードし、インストールする。Mac OS X セクションまでスクロール・ダウンしてパッケージ管理システム(Flink, MacPorts, openCollabNet, WANdisco)を選択する。

Subversion の開梱とインストール

ダウンロードの後、.dmg ファイルを開梱し、.pkg-installer をクリックして Subversion をインストールする。Subversion 自身はグラフィカルなユーザ・インターフェイスの機能を持っていないので、インストールの後でもアプリケーション・ディレクトリに新しいファイルを見いだせないだろう。その代りハード・ドライブのディレクトリ :/usr/local/bin にいくつかのコマンド・ラインのコマンドをインストールしている。

Subversion をパスに加える。

ターミナルを開き、/Application/Utilities に移り、svn [enter]とタイプする。これでもし：Type ‘svn help’ usage after the dollar sign($) と出力があれば svn は正しく動作している。どのディレクトリからも Subversion のコマン

ドを使えるようにするため、svn をパスに加える必要がある。その意味が分からなくても気にしなくてよい。次の指示に従うとよい。

コマンド・ラインのテキスト・エディタ pico で‘.profile’と名付けるファイルを生成することから始めよう：

$ pico .profile

次のラインをテキスト・ファイルに加える：

$ export PATH=$PATH: /usr/local/bin

Control を押しながら x を叩く。そしてファイルの保存を y とリターン・キィーで確認する。これで Subversion の位置をパスに追加した。ターミナルにこのファイルを読ませてパスの変化を知らせよう（2 つのドットの間にスペースが必要）：

$ . .profile

もう 1 つ別のターミナルを開き、もう一度次を試してみよう：svn [enter]

Subversion リポジトリの立ち上げ

第 1 に Subversion リポジトリをローカルなコンピュータで立ち上げる必要がある。それはプロジェクトのすべてのバージョンを保存する場所である。このように設定する：

$ svnadmin create svnrepos

これによってホーム・ディレクトリに’svnrepos’と呼ぶリポジトリを生成する。ただし svnadmin は何のフィードバックもくれない。このフォルダーが存在することを確認するためターミナルで’ls’とタイプするか、ファインダーでそれを探すとよい。

Subversion でプロジェクトを生成する

次にサンプルのプロジェクトを生成する。新しいフォルダを生成し、そのフォルダに’Employee.php’と呼ぶ空のファイルを生成する：

$ mkdir proj_restaurant

$ touch proj_restaurant/Employee.php

このプロジェクトをリポジトリに輸入しよう。ターミナルで’your_user_name’を君のユーザ・ネームで置き換えて次をタイプする：

$ svn import proj_restaurant file:///Users/your_user_name/svnrepos/proj_restaurant -m “initial import”

output:

Adding Employee.php

Committed revision 1.

これでフォルダ'proj_restaurant'をリポジトリ'svnrepos'に輸入した。リポジトリでの各バージョンは'revision'と呼ばれ、ユニークな番号で識別される。常に短いメッセージを('-m')リポジトリにした変更に付けるようにしよう。

Subversion からファイルを取り出す

ここで作業するためのファイルを取り出すあるいは"チェック・アウト"しよう：

$ svn checkout file:///Users/your_user_name/svnrepos/proj_restaurant proj_restaurant-copy

すると、ワーキング・コピーに加えられた('A') 全てのファイルが表示出力される：

A proj_restaurant-copy/Employee.php

Cheched out revision 1.

ワーキング・コピーのディレクトリに移るため次をタイプする：

$ cd proj_restaurant-copy

次にワーキング・コピーの内容をチェックする：

$ ls -al

これで、ディレクトリを隠れたファイルを含めて出力する：

.

. .

. svn

Employee.php

隠れていたディレクトリ'.svn'に注意。ここにはリポジトリの名称のような Subversion のメタデータが格納されるので将来タイプする必要がない。

更新と変化の送り出し

ここでファイルに変更を加え、リポジトリに送り返す。気に入りのテキスト・エディタでワーキング・コピーから'Employee.php'を開きファイルに次を加える：

```
<?php

class Employee()

?>
```

そして Subversion にリポジトリと自分のコピーとの違いを見つけるよう問い合わせる：

```
$ svn status
```

すると、自分のファイルに変更があったこと('M') を知らせてくる：

```
M   Employee.php
```

変更した内容でリポジトリを更新する。このことを『送り出し』("Commit")と呼ぶ：

```
$ svn commit -m "Added some comments"
Output:
Sending  Employee.php
Transmitting  file data  .
Committed  revision  2 .
```

ファイルとディレクトリの追加と削除

ファイルの追加

新しい文書を Subversion に追加するのは簡単だ。SVN の add コマンドを使ってそれを送り出す。この例では newfile1.php と呼ぶファイルを touch コマンドで造って svn の add コマンドでリポジトリに追加した。

```
% touch /projects/proj_restaurant/newfile1.php
% svn add /projects/proj_restaurant/newfile1.php
% svn commit –m 'added new file'

Adding            newfile1.php
Transmitting file date ...
Committed revision 3.
```

ファイルの削除

Subversion からファイルを削除するのも追加と同様に簡単だ。remove のサブ・コマンドを使えばよい。

```
% svn remove /projects/proj_restaurant/newfile1.php
% svn commit –m 'removed newfile1.php'

Deleting    newfile1.php
Committed revision 4.
```

ディレクトリの追加

Subversion の標準ディレクトリ構造と統一性を保つため、リポジトリに３つのディレクトリを設定しよう：

```
$ svn add trunk/
$ svn add tags/
$ svn add branches/
```

議論のためダミーのディレクトリも作っておこう：

```
$ svn add dummy/
```

ディレクトリの削除

同様に remove サブ・コマンドを使って Subversion からディレクトリを削除できる：

```
% svn remove dummy

D      dummy

% svn commit –m 'commit to removing the images directory'
```

リリースのタグ付けと輸出

プロジェクトのために３つのディレクトリ trunk, branches, tags を造ってある。これは個別プロジェクトのためのディレクトリの推奨設定である。

trunk ディレクトリは主にそこで仕事をする所だ。ファイルを造り、編集し、チェックし、コンパイルする。だから trunk はプロジェクトの目的のためのベースとするディレクトリである。事実、初心者には多分これが使用する唯一のディレクトリとなるだろう。

branches と tags のディレクトリはより複雑な並行した開発で使いたくなるものだ。例えば、branches は独立で開発するコードのコピーの場である。大きなプロジェクトで複数の参加者がコードに全く異なった変更を加える場合に重要となってくる。このコースではこれを使うことは無いだろう。

tags はそれに対して君が使いそうなものだ。それはある一時のコードの一時保存となる。より人に優しい名前の付け方が許される君のコードのコピーだ。例えば、まだプロトタイプの状態のコードにタグを付ければよく、いつでもこのときのコードに戻って見ることができて便利だ。

```
$ cd proj_restaurant/trunk
$ touch Timecard.php
$ pico Timecard.php
$ svn add Timecard.php
A       Timecard.php
$ svn commit
[エディターでメッセージを入力し、保存し、閉じる]
Sending Timecard.php
Transmitting file data ...
Committed revision 3.
[これは完成した最初のリリースだろう。だから、一時記録を取っておこう。]
$ cd ..
$ ls
branches/      tags/          trunk/
[さて trunk が今存在するので、それを tags の下の新しいスポットにコピーをするため“svn cp”を使う ]
$ svn cp trunk tags/release_1.0.0
A tags/release_1.0.0
$ svn commit trunk
[メッセージを入れておく「提出するリリースの一時保存を取る」]
 we're turning in."]
Sending tags/release_1.0.0...
Committed revision 4.
$ ls trunk/
Timecard.php      # <-- 君の職務コード；ここで開発する
$ ls tags/
```

```
release_1.0.0/
$ ls tags/release_1.0.0/
Timecard.php      #<-- このバージョンはここで凍結された
```

```
$ svn export file:///Users/your_user_name/svnrepos/proj_restaurant proj_restaurant-copy
```

このコピーはウエブに配置しても安全なものだ。しかし、ワーキング・コピーではないので、このフォルダーからリポジトリに変更を送り出すことはできない。

プロジェクトの枝分けと統合

枝分けを生成する

何よりもまず「枝分け」とは何なのかと思うだろう。小さな店舗用システムをオーナーのために構築することを考えよう。システムを提供し、歓迎されて、他のビジネス・オーナーにも見せ、受注に繋がったとしよう。そこで最初のオーナーのための店舗用システムをそのまま提供したが、第２のオーナーはいくつか特別の変更と微調整を要求した。第１のオーナーもまた、細かな追加を要求した。最初は同じだったシステムだったが時がたつにつれ２つの異なった応用を保守していることとなる。

これが基本的な枝分けの概念だ。ずっと前の時に戻れば共通の歴史を持っていても、殻の中では互いに独立な開発の並行線が存在する。Branch は常にリポジトリでの trunk のコピーから生まれ、そしてそこから独自の歴史を刻んでいくのだ。

枝分けにシナリオ的に言って２通りほどのタイプがある。最初のものは**開発者枝分け**と言ってもよい。この場合枝分けすることに利点があり、新しい機能をその中に造りこめる。後ほどそれらを trunk に統合してもよい。このシナリオでよく見られるのは、大きな機能拡張をして、その完成までに少し時間がかかる場合だ。プロジェクトの他の開発を全て停止したり、部分的な変更を送り出したりするわけにはいかない。しかし、他方では、バージョン管理されたファイルとディレクトリのおかげで新しい機能の開発を続けることができ、正常の送り出しができる。

もう一つの枝分けのよくあるケースは、**リリース枝分け**として知られる。これはプロジェクトの主要なリリースの数日前に行われる。リリース枝分けをして、そこで最後の微調整やバグ退治が行われる。このように枝分けで作業することで、チームの他のメンバーはリリース枝分けで行われていることに関係なく、trunk で日々の開発を続けることができる。

枝分けを造るのは非常に簡単である。枝分けは単に trunk のコピーであり、svn cp あるいは svn copy コマンドで簡単にできる。このとき２つの Subversion リポジトリの URL を与えればよい。その第１は枝分けすべきリソースのアドレス、一般的には trunk 、第２は新しい枝分けに希望するアドレスである。

```
% svn cp -m 'Making test branch' svn://localhost/usr/local/svnrepos/projects/trunk
svn://localhost/usr/local/svnrepos/projects/branches/branch_1.0.0
```

```
Committed revision 12.
```

これでいいのだ。君の枝分けが造られ、自動的にコミットされるので、作業するためにチェック・アウトできる。枝分けで作業するのは、Subversion のどのバージョンのリソースで作業するのと何ら変わりはない。

統合

枝分けを造ったのなら、枝分けでの変更を trunk にも戻したくなる時が必ず来るだろう。これは統合(marging)として知られていて、svn merge コマンドの使用で達成される。これは枝分けより少し複雑となるが、やはり簡単だと言える。

君と共に開発に努力している Dan が Employee.php にかなりの変更を加えているとしよう。これらの変更のすべてを一度に統合するには、先に使用した２つの URL （trunk と枝分けの両方の URL）の記録が要る。また trunk のワーキング・コピーも必要とする。ワーキング・コピーが入っているディレクトリへ cd コマンドで移動し、svn merge コマンドで統合を開始する。次の形式に従ってコマンドを作成する：

```
% svn merge svn://localhost/usr/local/svnrepos/projects/branches/branch_1.0.0
svn://localhost/usr/local/svnrepos/projects/trunk
U    Employee.php
% svn status
M    Employee.php
```

上記コマンドは、枝分け _1.0.0 に対して行われた全ての変更を取り上げそれらを trunk に統合する。今回は、何も自動的にはコミットされない。結果となる変更はワーキング・コピーに対して行われているので、まずそれをレビューすることができ、それで満足できるかどうか確認する。そうであれば、これらの変更をその他のワーキング・コピーへの変更と併せて svn commit でコミットをすることができる。

もう一つの統合の方法は、リビジョン番号を指定して行うものである。Dan が有効な変更をしてリビジョン 114 と 115 としてコミットをしているとしよう。それらの変更が trunk あるいはほかの枝分けにも複製されることを望むとしよう。これらのリビジョン番号を–r フラグで svn merge に渡すことができる。

```
% svn merge  -r 113:115 svn://localhost/usr/local/svnrepos/projects/branches/branch_1.0.0
U    Employee.php
% svn status
M    Employee.php
```

Subversion は /branches/branch_1.0.0 のリビジョン 113 と 115 に加えられた全ての変更を取り上げ、それら全てを現在のワーキング・コピーにコピーすることようる指示される。自動的にコミットされるものはない。結果のコードをレビューし、自分でコミットを行う。

グラフィカル・ユーザ・インタフェイス

多くの人はターミナルで仕事をするのを好まない。応用ではボタンを押すのに対して、テキストのコマンドを思い出すのは複雑だと思うだろう。Subversion のコマンドのためのグラフィカル・ユーザ・インタフェイスを持つ有料あるいは無料のソフトが２～３インターネット上にある。

Mac OS X のための無料だが良い GUI のソフトが svnX である。コードを書くのと同じソフトでワーキング・コピーの管理もするテキスト・エディタに **TextMate** はよい選択である。**TextMate** は Subversion のバンドルも含んでいて、メニューから Subversion のほとんどのコマンドを簡単に起動できる。リポジトリの立ち上げと最初

のワーキング・コピーのチェック・アウトだけは一度だけターミナルを使用使う必要がある。その後は、Shift-Control-A を押すことで Subversion バンドルのメニューを開くことができる。

Linux

Subversion のサーバーをライナックスのサーバーに立ち上げる

ユニックス系のオペレーティング・システムには、既に Subversion のクライアントがインストールされ使用できる状態であることが多い。これを確認するには、次のコマンド・ラインをタイプしてみるとよい（ここでは csh シェルを使う）：

```
% svn help
```

もし使用上の情報が現れたらリポジトリの立ち上げに進める。もし現れなかったら Subversion をダウンロードし、インストールしなければならない。Linux か Unix かは君の好みによる。簡単なインストール・プログラムの Yum あるいは Apt を使うとよい。いずれにしても、配布時の説明書を参考にするのが最良の策だ。

Subversion のソース・コードあるいは手持ちの Linux あるいは Unix に専用のバイナリーを次のサイトからダウンロードできる

http://subversion.apache.org/packages.html

ステップ１：

この例ではリポジトリを /usr/local/svnrepos のディレクトリに立ち上げる。このディレクトリに書き込むには、ルート特権が必要だ：

```
% svnadmin create -fs-type fsfs /usr/local/svnrepos
```

このコマンドはコマンド・ラインに何の確認も戻ってこない。しかし、svnrepos のディレクトリが /usr/local ディレクトリに造られたかどうかは簡単にチェックできる。

ステップ２：

君の SVN ユーザを造ろう：リポジトリが立ち上がっているので、svn ユーザを造る。Svnserve.conf ファイルを好みのエディタで開ける：

```
% vi /usr/local/svnrepos/conf/svnserve.conf
```

このファイルで次の３行を入力する：

```
anon-access = none

auth-access = write

password-db = passwd
```

ステップ３：

ここでパスワード・ファイルを作成する：

% vi /usr/local/svnrepos/conf/passwd

そのファイルに君のユーザとして次のフォーマットで１行を追加する：

Exampleuser = examplepassword

ステップ４：

Svn commit を起動すると Subversion は“デフォルト・エディター”をスタートさせ、テキスト・ファイルに何かコメントを付けるよう要求する。Subversion と共に使うデフォルト・エディター―を設定しよう。Vi エディターより pico や emacs の方が便利だろう。ここでの説明には emacs を使う：

% setenv EDITOR emacs

ステップ５：

簡単のため、emacs（あるいは君の好みのエディター）を開き、Employee.php と呼ぶ骨格だけのファイルを造ろう：

```
<?php

Class Employee

{

}


?>
```

ファイルを Employee.php として /project/proj_restaurant/ またはプロジェクトのディレクトリとして好むところにセーブしよう。

また、trunk, tags, branches のディレクトリ構造を設定しよう。これは Subversion で使われる重要な慣例で、これからの説明で出てくる。基本的に、すべての作業は trunk ディレクトリで行う。この３つのフォルダーは君のプロジェクト・フォルダーの下にあるべきで、もし君のプロジェクトのパスが次のようであれば：

/projects

    /proj_restaurant

それはこのようになっているべきだ：

/projects

    /branches

    /tags

/trunk

/proj_restaurant

そのため cd で /projects に行き”TTB”ディレクトリを造る：

Mkdir trunk

Mkdir branches

Mkdir tags

これと同じディレクトリ構造をリポジトリにも設定する（ディレクトリやファイルの追加を後で説明する）

% svn add trunk/

% svn add tags/

% svn add branches/

結果としてディレクトリのトリーはこのようになる：

/svnrepos

/branches

/tags

/trunk

さて、君のプロジェクトを輸入しよう：

この例ではプロジェクトのファイルを /projects/proj_restaurant/ に入れたと仮定している。君はプロジェクトを好きな場所でスタートできる、ただ次のコマンドの輸入のパスは君のプロジェクトのファイルのパスと一致していることが必要だ。

% svn import /projects/trunk/proj_restaurant file:///usr/local/svnrepos/trunk

ステップ６：

svn のサーバーをデーモンとして走らせる：

% svnserve -d

出来た！これで svn のサーバーが proj_restaurant と呼ぶプロジェクトと共に走っている。

ステップ７：

リポジトリでチェック・アウトを試してみよう：

% svn co svn://localhost/usr/local/svnrepos/trunk/proj_restaurant

anon-access を none と設定しているので、ユーザ・ネームとパスワードの入力を要求される;それは /usr/local/svnrepos/conf/passwd のファイルでせっていしたものだ。

更新と変更のコミット

Employee.php ファイルをコメントを加えて更新しよう：

```
<?php

/**

*このファイルは Employee クラスを含んでいる

**/

class  Employee
{
}

?>
```

変更を正式の登録する前にどのファイルに変化があったかを調べるために status コマンドを使う:

```
% svn status -show-updates
```

この status コマンドはローカルで更新の影響を受けたファイルのリストを表示する。さて、変更をコミット (commit) しよう：

```
% svn commit Employee.php
```

するとデフォルトのエディター(この例では pico) がテンポラリー・ファイルで起動される；このファイルに変更の目的について何か簡単なコメントを入れるのがよい。

Subversion はこのチェック・インに対して次のメッセージで応答することを覚えておくとよい。

```
Sending       Employee.php
Transmitting file data ...
Committed revision 2.
```

コミットした変更は連続のリビジョン番号が付けられる。（この例では、その番号は２だった。）過去のどのチェック・インもこのコマンドを使って調べることができる：

```
% svn log -v -r2
```

-r のフラグでどのリビジョンを調べるかを指定できる；-v のフラグはどのファイルに接触があったかを示す。

注意：svn commit コマンドを起動したが、チェック・アウトした全てのファイル、あるいは全ての変更を送りだすつもりはなかったと気が付いたなら、エディターからメッセージを付けることなく出る。すると戻ることができるかも知れない。コミット・メッセージを書いてセーブしてしまうと、その機会は失われる。

ファイルとディレクトリの追加と削除

ファイルの追加

新しい文書を Subversion に追加するのは簡単だ。svn の add コマンドを使ってそれを送り出す。この例では newfile1.php と呼ぶファイルを touch コマンドで造って svn の add コマンドでリポジトリに追加した。

```
% touch /projects/proj_restaurant/newfile1.php

% svn add /projects/proj_restaurant/newfile1.php

% svn commit –m 'added new file'

Adding       newfile1.php
Transmitting file date ...
Committed revision 3.
```

ファイルの削除

Subversion からファイルを削除するのも追加と同様に簡単だ。remove のサブ・コマンドを使えばよい。

```
% svn remove /projects/proj_restaurant/newfile1.php

% svn commit –m 'removed newfile1.php'

Deleting     newfile1.php
Committed revision 4.
```

ディレクトリの追加

Subversion にディレクトリを追加するには add サブ・コマンドを使う：

```
% mkdir images
% touch images/picture1.jpg
% svn add images

A      images
A      images/picture1.jpg
```

ディレクトリの削除

同様に remove サブ・コマンドを使って Subversion からディレクトリを削除できる：

```
% svn remove images

D      images/picture1.jpg
D      images

% svn commit –m 'commit to removing the images directory'
```

リリースのタグ付けと輸出

プロジェクトのために３つのディレクトリ trunk, branches, tags を造ってある。これは個別プロジェクトのためのディレクトリの推奨設定である。

trunk ディレクトリは主にそこで仕事をする所だ。ファイルを造り、編集し、チェックし、コンパイルする。だから trunk をプロジェクトの目的のためのベースとするディレクトリである。事実、初心者には多分これが使用する唯一のディレクトリとなるだろう。

branches と tags のディレクトリはより複雑な並行した開発で使いたくなるものだ。例えば、branches は独立で開発するコードのコピーの場である。大きなプロジェクトで複数の参加者がコードに全く異なった変更を加える場合に重要となってくる。このコースではこれを使うことは無いだろう。

tags はそれに対して君が使いそうなものだ。それはある一時のコードの一時保存となる。より人に優しい名前の付け方が許される君のコードのコピーだ。例えば、まだプロトタイプの状態のコードにタグを付ければよく、いつでもこのときのコードに戻って見ることができて便利だ。

tag の使用を次の例で示したい。

```
% cd /projects/proj_restaurant/trunk
% emacs MyPHPCode.php
% svn add MyPHPCode.php
A       MyPHPCode.php
% svn commit
[エディターでメッセージを入力し、保存し、閉じる]
Sending MyPHPCode.php
Transmitting file data ...
Committed revision 7.
[これは完成した最初のリリースだろう。だから、一時記録を取っておこう。]
% cd ..
% ls
branches/      tags/          trunk/
[さて trunk が今存在するので、それを tags の下の新しいスポットにコピーをするため“svn cp”を使う ]
% svn cp trunk tags/release_1.0.0
A tags/release_1.0.0
% svn commit trunk
[メッセージを入れておく「提出するリリースの一時保存を取る」]
 we're turning in."]
Sending tags/release_1.0.0...
Committed revision 8.
% ls trunk/
MyPHPCode.php     # <-- 君の職務コード；ここで開発する
% ls tags/
release_1.0.0/
% ls tags/release_1.0.0/
MyPHPCode.php     #<-- このバージョンはここで凍結された
```

Now to export a clean release version of your codebase use the `export` subcommand:

```
% svn export svn://localhost/projects/proj_restaurant/tags/release_1.0.0 \ release_1.0.0
```

さてクリーンになったコードのリリース・バージョンを輸出するには export サブ・コマンドを使う。

プロジェクトの枝分けと統合

枝分けを生成する

何よりもまず「枝分け」とは何なのかと思うだろう。小さな店舗用システムをオーナーのために構築することを考えよう。システムを提供し、歓迎されて、他のビジネス・オーナーにも見せ、受注に繋がったとしよう。そこで最初のオーナーのための店舗用システムをそのまま提供したが、第２のオーナーはいくつか特別の変更と微調整を要求した。第１のオーナーもまた、細かな追加を要求した。最初は同じだったシステムだったが時がたつにつれ２つの異なった応用を保守していることとなる。

これが基本的な枝分けの概念だ。ずっと前の時に戻れば共通の歴史を持っていても、殻の中では互いに独立な開発の並行線が存在する。Branch は常にリポジトリでの trunk のコピーから生まれ、そしてそこから独自の歴史を刻んでいくのだ。

枝分けにシナリオ的に言って２通りほどのタイプがある。最初のものは**開発者枝分け**と言ってもよい。この場合枝分けすることに利点があり、新しい機能をその中に造りこめる。後ほどそれらを trunk に統合してもよい。このシナリオでよく見られるのは、大きな機能拡張をして、その完成までに少し時間がかかる場合だ。プロジェクトの他の開発を全て停止したり、部分的な変更を送り出したりするわけにはいかない。しかし、他方では、バージョン管理されたファイルとディレクトリのおかげで新しい機能の開発を続けることができ、正常の送り出しができる。

もう一つの枝分けのよくあるケースは、**リリース枝分け**として知られる。これはプロジェクトの主要なリリースの数日前に行われる。リリース枝分けをして、そこで最後の微調整やバグ退治が行われる。このように枝分けで作業することで、チームの他のメンバーはリリース枝分けで行われていることに関係なく、trunk で日々の開発を続けることができる。

枝分けを造るのは非常に簡単である。枝分けは単に trunk のコピーであり、svn cp あるいは svn copy コマンドで簡単にできる。このとき２つの Subversion リポジトリの URL を与えればよい。その第１は枝分けすべきリソースのアドレス、一般的には trunk 、第２は新しい枝分けに希望するアドレスである。

```
% svn cp -m 'Making test branch' svn://localhost/usr/local/svnrepos/projects/trunk
svn://localhost/usr/local/svnrepos/projects/branches/branch_1.0.0

Committed revision 12.
```

出来上がり！君の枝分けが造られ、自動的にコミットされるので、作業するためにチェック・アウトできる。枝分けで作業するのは、Subversion のどのバージョンのリソースで作業するのと何ら変わりはない。

統合

枝分けを造ったのなら、枝分けでの変更を trunk にも戻したくなる時が必ず来るだろう。これは統合(marging)として知られていて、svn merge コマンドの使用で達成される。これは枝分けより少し複雑となるが、やはり簡単だと言える。

君と共に開発に努力している Dan が Employee.php にかなりの変更を加えているとしよう。これらの変更のすべてを一度に統合するには、先に使用した２つの Subversion URL （trunk と枝分けの両方の URL）の記録が要

る。また trunk のワーキング・コピーも必要とする。ワーキング・コピーが入っているディレクトリへ cd コマンドで移動し、svn merge コマンドで統合を開始する。次の形式に従ってコマンドを作成する：

```
% svn merge svn://localhost/usr/local/svnrepos/projects/branches/branch_1.0.0
svn://localhost/usr/local/svnrepos/projects/trunk
U    Employee.php
% svn status
M    Employee.php
```

上記コマンドは、枝分け branch_1.0.0 に対して行われた全ての変更を取り上げそれらを trunk に統合する。今回は、何も自動的にはコミットされない。結果となる変更はワーキング・コピーに対して行われているので、まずそれをレビューすることができ、それで満足できるかどうか確認する。そうであれば、これらの変更をその他のワーキング・コピーへの変更と併せて svn commit でコミットをすることができる。

もう一つの統合の方法は、リビジョン番号を指定して行うものである。Dan が有効な変更をしてリビジョン 114 と 115 としてコミットをしているとしよう。それらの変更が trunk あるいはほかの枝分けにも複製されることを望むとしよう。これらのリビジョン番号を–r フラグで svn merge に渡すことができる。

```
% svn merge  -r 113:115 svn://localhost/usr/local/svnrepos/projects/branches/branch_1.0.0
U    Employee.php
% svn status
M    Employee.php
```

Subversion は /branches/branch_1.0.0 のリビジョン 113 と 115 に加えられた全ての変更を取り上げ、それら全てを現在のワーキング・コピーにコピーすることようる指示される。自動的にコミットされるものはない。結果のコードをレビューし、自分でコミットを行う。

まとめ

PHP 5 を紹介するに当たっては、オブジェクト指向プログラミングが PHP の世界の中心であった。第２章から第７章までで PHP におけるオブジェクト指向のサポート上の重要な機能について述べた。またオブジェクト指向 PHP の重要な原理として、実装に向けたものでなく”インタフェイスに向けたコーディング”があり、密に結合されたコードを避けることの重要性も説明した。設計パターンについては、あまり深く掘り下げることがなかったが、モデル - ビュウ - コントローラ設計パターンに従って独自の応用を組織する方法について説明した。これは今日 PHP 応用を構築する際の標準となっている。この本のこの版では PHPUnit を用いたテストと SVN によるバージョン管理も紹介した。次回の追加として Zend フレームワークでの PHPUnit と、流通応用のリビジョン管理とソース・コード管理(SCM)のシステムである Git を紹介したい。また第８章で開発した MVC 応用のステップごとの解説と Zend フレームワークへ移入する方法も示したい。今回の版ではオブジェクト - リレーショナル・マッパーを説明しなかったが、これも Doctorie 2 の紹介を含めて次の版での話題である。その他にもこの版で逃したが追加したい話題として PHP の namespace とタイプ・ヒンティングがある。

PHP はいまだ発展を続けている言語であり、今後の PHP 言語にはさらに新しいオブジェクト指向機能が取巻いていくだろう。PHP が PHP 3 以降長い道のりを経てきたことは疑う余地もない。事実、PHP を使っているウエブサイトの 96%以上が PHP 5 を使用していて、わずか 0.1%が PHP 3 を使っている。PHP 5 は PHP の評判ををますます高め、それを学び、進歩についていくことが重要となっている。W3 Tech のウエブ・テクノロジーのサーベイによると PHP はサーバー・サイドの言語として他の言語を抑えてウエブ応用の 78.7%で使われている。

当然 PHP が今後もマーケットのシェアを獲得していくと期待できる。 PHP5+ がよく使われている Drupal, Joomla, WordPress のようなコンテント管理システム、それに Magento の様な e-コマースのプラットフォームの開発言語として使われている。この本の次の版ではこのようなオープン・ソース・システムについて豊富な例を挙げ、それらをどのように使い、カストマイズするのかについて説明しよう。それに加えて、他の独自応用分野である、独自イメージ・ギャラリーの開発もして繰り返し設計パターンを使ったページ制御の開発方法を示そう。

付録 A

参考文献

ウエブ上のリソース

Eclipse: http://www.eclipse.org

JQuery: http://jquery.com

MySQL: http://www.mysql.com

MySQL Workbench 5.2 CE: http://dev.mysql.com/downloads/workbench/5.2.html

Design Patterns: http:// www.ibm.com/developerworks/library/os-php-designptrns/

PEAR: http://pear.php.net

PHP: http://www.php.net

PHPUnit: http://www.phpunit.de

Sourceforge: http://sourceforge.net/projects/win32svn/

Subversion: http://subversion.apache.org

Templates: http://www.smarty.net

TortoiseSVN: http://tortoisesvn.net/downloads.html

W3 Tech Web Technology Surveys : http://w3techs.com/technologies/details/pl-php/all/all

Zend: http://www.zend.com

書籍

Dagfinn Reiersøl with Marcus Baker and Chris Shiflett. ***PHP in Action.*** Greenwich, CT: Manning Publications Co., 2007

Powers, David. PHP ***Object-Oriented Solutions.*** N.Y, N.Y**.** Apress, 2008

Welling**,** Luke and Thompson, Laura. ***PHP and MySQL Web Development, 4th Edition.*** Upper Saddle River, New Jersey: Pearson Education, Inc, 2009

インデックス

$

$_GET · 63, 81, 82
$_POST · 64, 68, 81, 82
$this · 17, 19, 41

.

.htaccess · 56, 57

_

__call · 30, 31
__clone · 32
__construct · 17, 28
__destruct · 28
__get() · 28, 29, 30, 44
__isset · 28, 44
__set() · 28, 29, 30, 44
__sleep · 31, 32
__wakeup · 32

A

abstruct · 33, 34, 42, 46, 53, 62, 63, 66
access modifier · 19
add · 64, 66, 91, 95, 103
Add · 67
Agile Development · 10
AJAX · 10
AMP · 11
anon-access · 87, 101
any() · 83
assertion · 80
at($num) · 83
atLeastOnce() · 83
auth-access · 87

C

CAPTCHA · 75
Coding-to-the-Interface · 10, 37
controller · 53, 62
CSS · 9

D

Database · 42, 43, 48, 53, 54, 58
Dependency Injection · 81
Design-by-Contract · 10
DOM · 10
dynamic Binding · 26

E

Eclipse · 14, 15, 30, 84, 86
encapsulation · 16, 22, 24
error_log · 50
error_reporting · 49
exactly($num) · 83
Exception · 47
extend · 20
extend · 22
extend · 24
Extreme Programming · 10

F

final · 34, 35
FollowSymLinks · 57

G

getName() · 70
Go-PEAR · 72

H

Helios Release · 14

I

ignore_repeated_errors · 51
ignore_repeated_source · 51
implements · 36
index.php · 55, 57, 63, 64, 65, 66, 67, 68
inheritance · 16, 22
interface · 36

J

JQuery · 10
  Core 機能 · 10

L

LAMP · 9, 11
Late Binding · 26
Linux · 50, 74, 79, 87, 99
LOG_ALERT · 53
LOG_CRIT · 53
LOG_DEBUG · 53
LOG_EMERG · 53
LOG_ERROR · 53
log_errors · 50
LOG_INFO · 53
LOG_NOTICE · 53
LOG_WARNING · 53

M

Mac OS · 73, 78, 87, 92, 98
MAMP · 11
Mapper · 56, 63, 64, 66, 67, 68
Mock · 83
mocking · 82
Model · 53, 66, 67
MVC · 56, 58, 60, 61, 62, 69, 107
MySQL · 9, 11, 12, 13, 14, 43, 58

N

never() · 83
new · 18

O

Once() · 83
Overloading · 25

P

password · 87
PEAR · 72, 73, 74, 75
php.ini · 13, 49, 72, 74, 78
PHPUnit · 77, 78, 79, 80, 84
polymorphism · 16
preg_replace · 60, 61
private · 19, 21, 24, 45, 70
protected · 19, 21, 22, 23, 70
public · 19, 20, 21, 22, 23, 24, 41, 42, 45, 47

R

Reflection · 70
repository · 72, 77, 86
Rewrite · 57

S

scope · 23, 41
sensitivity level · 49
Singleton · 44
startup_errors · 50
static · 41, 42, 43, 44, 45, 59
stub · 82
Subversion · 86, 87, 88, 89, 90, 92, 93, 94, 95, 97, 98, 99, 103, 105, 106
SVN · 9, 10, 86, 90, 92, 98, 99, 107
  add · 91, 95
  branch · 90, 96, 97, 98, 103, 104, 105, 106
  commit · 89, 91, 95, 98, 100, 102, 105
  export · 104
  remove · 95, 96, 103
  tag · 103
  tag · 90, 91, 96, 103
  tag · 104
  trunk · 90, 96, 97, 98, 103, 104, 105
  グラフィカル・ユーザ・インタフェイス · 98
  タグ · 103
  削除 · 67, 68, 78, 86, 89, 95, 96, 103
  控え · 82, 83
  更新 · 64, 86, 102
  模擬 · 82, 83, 84
  輸出 · 92, 103, 104
svnserve.exe · 88

T

template · 59, 61, 67
Test Driven Development · 10
Test Fixtures · 79
Tortoise · 86, 90, 91
trait · 24

U

update · 102

W

WAMP · 11, 73

Z

Zend Framework · 72, 78
Zend Server · 11, 12, 13
Zend Server Interface Port · 13
Zend\phpMyAdmin\config.inc · 14

ア

アニメーション · 10

イ

インスタンス化 · 33, 43, 44, 46, 47, 58, 59, 61, 67, 83
インタフェイス · 25, 29, 33, 36, 37, 38, 39, 40, 42, 44, 63, 70, 77, 107
インヘリタンス · 16, 20, 22, 23, 24, 25, 27, 33, 35, 36, 38, 39, 45
インヘリト · 22, 23, 27, 33, 34

エ

エラーへの感度のレベル · 48

オ

オーバライド · 62, 63, 83

カ

カプセル化 · 16, 22, 24, 41, 43

コ

コミット · 97, 98, 102, 105, 106

シ

システム要件 · 12
シングルトン · 44, 45

ダ

ダイナミック結合 · 26

テ

テストのフェーズ · 79
テスト固定具 · 79, 80

ポ

ポリモルフィズム · 16, 24, 25, 26, 33, 34

メ

メソッドの多重定義 · 25

モ

モデル · 56, 58, 59

リ

リフレクション · 70, 71
リポジトリ · 72, 86, 87, 89, 90, 92, 93, 94, 95, 97, 98, 99, 103, 104, 105

ロ

ログ · 50
ログのオプション · 52
ログのための接続 · 51

依

依存性の注入 · 81

実

実行時結合 · 26

拡

拡張 · 20, 23, 34, 36, 38, 41, 42, 46, 47, 56, 58, 62, 64, 66, 67, 80

書

書き換え · 23, 57, 86

最

最近のエラー · 51

枝

枝分け · 86, 90, 92, 97, 98, 104, 105

構

構成要素関係ダイアグラム · 64

水

水平インヘリタンス · 24

親

親のメソッド · 23

領

領域 · 41, 43, 46, 48, 50, 61