23 VZOROVÝCH POSTUPŮ PRO RYCHLEJŠÍ VÝVOJ

Návrhové vzory v PHP

Marian Böhmer

Stručný přehledný výklad
Ukázky v UML a příklady z praxe
Alternativní řešení a postupy

computer press

Marian Böhmer

Návrhové vzory v PHP

Computer Press
Brno
2012

Návrhové vzory v PHP

Marian Böhmer

Překlad: Lukáš Krejčí
Obálka: Martin Sodomka
Odpovědný redaktor: Martin Herodek
Technický redaktor: Jiří Matoušek

Objednávky knih:
http://knihy.cpress.cz
www.albatrosmedia.cz
eshop@albatrosmedia.cz
bezplatná linka 800 555 513
ISBN 978-80-251-3338-5

Vydalo nakladatelství Computer Press v Brně roku 2012 ve společnosti Albatros Media a. s. se sídlem Na Pankráci 30, Praha 4. Číslo publikace 15 935.

1. vydání

ALBATROS MEDIA a.s.

Obsah

ČÁST II
TVOŘIVÉ VZORY

Úvodem

Návrhové vzory nabízejí postupy řešení často se vyskytujících problémů při návrhu softwaru. Často jsou považované za velmi komplikované, asociované s komplexními diagramy jazyka UML a obtížně pochopitelnými architekturami. Tato kniha vám ukáže, že návrhové vzory a jejich implementace v jazyce PHP nejsou až tak komplikované, jak se mohou někomu jevit po prvním přečtení jejich definice. Na praktických příkladech se seznámíte s různými návrhovými vzory, které vám pomohou při každodenní práci a kvůli nimž už nebudete muset organizovat velká pracovní střetnutí nebo vytvářet vícestranné diagramy. Pomocí návodů v této knize se pro vás návrhové vzory stanou nástrojem, který vám ulehčí a obohatí chvíle při vývoji aplikací v jazyce PHP.

Jazyk PHP 5 přinesl množství novinek, například kompletně přepracovanou podporu pro XML nebo jednodušší použití webových služeb přes protokol SOAP, který je automaticky podporovaný v jazyce PHP. Přístup k různým typům databází lze nyní díky PHP 5.1 a PDO provádět přes jednotné rozhraní.

Naproti tomu tato aplikační rozhraní sama o sobě zatím neumožňovala realizovat profesionální aplikace. O to se postaral až Zend Engine 2 s kompletně přepracovaným objektovým modelem, který nabízí viditelnost pro atributy a metody, rozhraní, výjimky, ale i abstraktní a finální třídy. To vám, jakožto vývojářům v jazyce PHP, umožňuje navrhovat softwarové architektury, které nebudou v ničem zaostávat za architekturami vývojářů v jazyce Java. Tato kniha vám ukáže, jak můžete při návrhu softwaru využít prvky jazyka PHP 5.3 k tomu, aby váš software splňoval moderní standardy a bylo jej možné bez problémů rozšířit v případě, že se změní požadavky na aplikaci.

Dále se seznámíte s pravidly, která byste měli dodržovat při návrhu aplikace. Kromě toho vám návrhové vzory nabízejí jazyk, pomocí něhož můžete řešit problémy s ostatními kolegy vývojáři, aniž abyste museli každý detail podrobně vysvětlovat.

Struktura knihy

Celá kniha je rozdělená do pěti částí a lze ji číst dvěma způsoby. Buď ji budete číst postupně, od začátku do konce, a přitom se naučíte krok za krokem jednotlivé návrhové vzory, nebo ji použijete jako katalog, v němž si vyhledáte přesně ten návrhový vzor, jehož implementací se chce zabývat. Každý návrhový vzor představuje jednu kapitolu. V některých kapitolách se setkáte s odkazy na jiné návrhové vzory – můžete je však použít i nezávisle na sobě. Jiné kapitoly naopak nemusí být rozepsané dopodrobna, a to zvláště v případě, že popisovaný návrhový vzor lze nahradit jiným návrhovým vzorem, který řeší stejný problém.

V úvodní části (kapitola 1 – Základní pravidla při návrhu softwaru) si na příkladu vyzkoušíte, jakým chybám byste se měli vyhnout při návrhu architektury softwaru. Vyvodí se z nich všeobecně platná pravidla, jež se stanou základem návrhových vzorů probíraných v této knize.

Druhá část (kapitoly 2–6) pojednává o návrhových vzorech týkajících se tvorby objektů.

V třetí části knihy (kapitoly 7–13) se představí návrhové vzory, které se zabývají kompozicí různých objektů. V této části se navíc seznámíte se strategiemi, které umožňují rozšiřitelnost daných kompozic.

Čtvrtá část (kapitoly 14–24) se zabývá poslední skupinou klasických návrhových vzorů. Tyto vzory řeší problémy, které se mohou často vyskytnout při interakci různých objektů.

V poslední, páté části (Příloha) najdete krátký úvod do jazyka UML, díky němuž nebudete mít problém pochopit diagramy jazyka UML v jednotlivých kapitolách. Kromě toho v této části najdete i techniky nazývané vkládání závislostí (angl. Depencency Injection), které staví na filozofii převráceného řízení (angl. Inversion of Control – IoC), a také se naučíte vytvářet aplikační rozhraní s Fluent Interfaces.

Ve všech výpisech v knize budou vynechané počáteční a koncové značky (`<?php` a `?>`) a bude uvedený jen kód jazyka PHP. Pro označení názvů jednotlivých tříd, metod a atributů budeme používat anglické pojmy, což je způsob, který doporučuji i pro vaše projekty, protože tyto pojmy se lépe hodí k nativním názvům tříd a funkcí jazyka PHP.

Pro lepší přehlednost nebudou uváděné komentáře. U reálných projektů byste však měli myslet alespoň na popis aplikačního rozhraní pomocí nástroje PHPDoc (*http://www.phpdoc.org*).

Všechny příklady jsou napsané pro jazyk PHP ve verzi 5.3. Chcete-li je použít se starší verzí jazyka PHP, musíte je patřičně upravit – vzdát se používání jmenných prostorů a odstranit z výpisů klíčová slova `namespace` a `use`.

Komu je kniha určena

Tato kniha je určená vývojářům, kteří už mají zkušenosti s programováním v jazyce PHP. V ideálním případě se už při realizaci projektu střetli s objektově orientovaným programováním (OOP) v jazyce PHP 4 nebo PHP 5, avšak žádné hluboké znalosti OOP v jazyce PHP 5 nejsou podmínkou.

Pro vysvětlení a pochopení jednotlivých návrhových vzorů probíraných v této knize nejsou vyžadované žádné hluboké znalosti jazyka UML (Unified Modeling Language). Kromě krátkého úvodu do tohoto jazyka, který najdete v příloze, nabízí tato kniha diagramy, jejichž obsah pro vás bude srozumitelný i se základy jazyka UML.

Kniha Návrhové vzory v PHP je napsaná pro programátory, kteří chtějí při vytváření profesionálních architektur využívat vlastnosti OOP. Jejím těžištěm je přitom architektura softwaru. Věci, které v knize nebudou probírané, se týkají oblasti nízkoúrovňové logiky, jako je například přístup k souborům, analyzování dokumentů XML nebo přístup k údajům v databázi pomocí jazyka SQL.

Máte-li málo zkušeností s jazykem PHP, avšak pojem návrhové vzory vám je známý z jiných programovacích jazyků, můžete se pomocí této knihy seznámit s implementací různých vzorů, jejichž realizace uvedeným způsobem je možná jen v jazyce PHP.

Zpětná vazba od čtenářů

Nakladatelství a vydavatelství Computer Press, které pro vás tuto knihu připravilo, stojí o zpětnou vazbu a bude na vaše podněty a dotazy reagovat. Můžete se obrátit na následující adresy:

redakce PC literatury
Computer Press
Spielberk Office Centre
Holandská 3
639 00 Brno

nebo

sefredaktor.pc@cpress.cz

Computer Press neposkytuje rady ani jakýkoli servis pro aplikace třetích stran. Pokud budete mít dotaz k programu, obraťte se prosím na jeho tvůrce.

Errata

Přestože jsme udělali maximum pro to, abychom zajistili přesnost a správnost obsahu, chybám se úplně vyhnout nedá. Pokud v některé z našich knih najdete chybu, ať už chybu v textu nebo v kódu, budeme rádi, pokud nám ji nahlásíte. Ostatní uživatele tak můžete ušetřit frustrace a pomoci nám zlepšit následující vydání této knihy.

Veškerá existující errata zobrazíte na adrese *http://knihy.cpress.cz/K1886* po klepnutí na odkaz Soubory ke stažení.

ČÁST I

Úvod do návrhu softwaru

KAPITOLA 1

Základní pravidla při návrhu softwaru

Máloco podléhá tak častým změnám jako právě software. Ve většině projektů se na něj nakládají stále nové požadavky a stejně rychle odpadají požadavky dosud implementované – nebo v lepším případě jsou často pozměněné. Jako softwarový vývojář musíte jednotlivé komponenty své aplikace navrhnout takovým způsobem, abyste mohli relativně rychle a flexibilně reagovat na měnící se požadavky. V této kapitole se dozvíte, jak lze vyvíjet flexibilní aplikace, a také to, že je důležité zamyslet se nad návrhem aplikace ještě před vlastní implementací. Tím se vyhnete situaci, kdy náhle stojíte před problémem, kdy musíte přepsat velké části aplikace, aby bylo možné reagovat na změněné požadavky. Jestliže tomu nezabráníte, může každý změněný požadavek na software znamenat zvýšené náklady, čímž se prodlouží čas nutný k dokončení projektu nebo se do softwaru nestihnou zapracovat všechny požadavky.

Na příkladu v této kapitole si osvojíte základy návrhu softwaru a naučíte se, jak sestavit aplikaci skládající se z více tříd, jež splňují požadavky na rozšiřitelnost a flexibilitu.

Na konci kapitoly se dozvíte, jak lze co nejrychleji najít vhodný návrhový vzor pro řešení budoucích problémů. Vyzbrojeni těmito vědomostmi můžete použít následující kapitoly jako encyklopedii, když narazíte v jednom ze svých projektů na problémy, které budete chtít vyřešit pomocí návrhových vzorů.

Protože hlavním cílem této knihy je blíže vás seznámit se správnou architekturou aplikací, výklad nebude zacházet až do detailů a nebude se věnovat perzistentnímu ukládání údajů nebo grafickému uživatelskému rozhraní. Místo toho budeme klást důraz na rozhraní, která vám dané třídy nabízejí, a na to, jak jednotlivé třídy mezi sebou komunikují.

Pravidla softwarového návrhu

K objasnění základních pravidel softwarového návrhu využijete následující příklad. Představte si, že byste měli implementovat aplikaci, pomocí níž bude knihovna spravovat své publikace. Na konci této kapitoly budete mít za sebou vývoj funkční aplikace, s níž bude možné tyto publikace půjčovat a každé takovéto půjčení přesně protokolovat. V reálné knihovně bude správce chtít jistě ukládat o každé publikaci více informací a půjčování nebude nefungovat přesně tak, jako to bude uvedené zde, ale jako příklad probírané problematiky to stačí.

Pokud jste se už nějakým způsobem zabývali objektově orientovaným programováním, jsou vám pojmy *třída* (angl. *Class*), *objekt* (angl. *Object*) a možná i rozhraní (angl. *Interface*) docela dobře známé. Koncepce tříd představuje pokus přenést věci z reálného světa, s nimiž má aplikace něco společné, do světa programování.

Každý typ publikace má samozřejmě jiné vlastnosti, ale určitým způsobem se všechny vzájemně podobají, z čehož lze odvodit následující rozhraní:

```
namespace cz\k1886\example;

interface Publication {
    public function open();
    public function close();
    public function setPageNumber($page);
    public function getPageNumber();
}
```

Reprezentace libovolné knihy by potom mohla vypadat následovně:

```
namespace cz\k1886\example;

class Book implements Publication {
    protected $category;
    protected $pageCount;
    protected $pageNumber = 0;
    protected $closed = true;
```

```
    public function __construct($category, $pageCount) {
        $this->category = $category;
        $this->pageCount = $pageCount;
    }
    public function open() {
        $this->closed = false;
    }
    public function setPageNumber($page) {
        if ($this->closed !== false || $this->pageCount < $page) {
            return false;
        }
        $this->pageNumber = $page;
        return true;
    }
    public function getPageNumber()
    {
        return $this->pageNumber;
    }
    public function close() {
        $this->setPageNumber(0);
        $this->closed = true;
    }
    public function __destruct() {
        if (!$this->closed) {
            $this->close();
        }
    }
    // další metody pro čtení atributů
}
```

Reprezentace knihy má čtyři atributy: kategorii, do níž náleží, počet stran, konkrétní stranu, na níž je právě otevřená, a příznak udávající, zda je otevřená nebo zavřená.

Zapouzdření údajů

Třídy a objekty vám umožňují omezit přístup k údajům, čehož byste měli využívat. Díky tomu totiž můžete změnit vnitřní strukturu třídy, aniž by se to dotklo jiných tříd nebo metod, které dotyčnou třídu používají. Takové změny mohou být kromě jiného způsobené i následujícími důvody:

- Změnily se požadavky na aplikaci, přičemž je nutné změnit algoritmy. Pokud se změny provedou v rámci jedné třídy, neovlivní to žádnou z tříd, jež upravenou třídu volají.
- Implementované algoritmy jsou velmi pomalé, protože aplikaci používá stále větší počet uživatelů. Tyto algoritmy můžete přepsat, povedou-li ke stejnému výsledku.
- Je zapotřebí změnit úložiště údajů. K tomu může dojít například v situaci, když zjistíte, že aktuální úložiště je vzhledem k objemu údajů nevhodné, a je tedy nutné použít místo souboru databázi.

Pro objasnění znovu využijeme příklad knihovny. Vzhledem k tomu, že půjčování publikací není zadarmo, musí být někde uložená denní sazba pro jednu publikaci. Je-li tato hodnota zpočátku stále stejná, může vás napadnout použít globální proměnnou nebo nějaký podobný způsob konfigurace:

```
$dailyRate = 10;
```

Co se ale stane, přijde-li požadavek, aby se denní sazba pro dětské knihy lišila od sazby pro knihy vědecké nebo aby celková částka závisela na délce vypůjčení. Tím by velmi vzrostl počet konfiguračních možností a museli byste v příslušných místech své aplikace implementovat logiku, pomocí níž byste načítali správnou konfiguraci.

Z těchto důvodů je lepší zapouzdřit přístup k denní sazbě hned od začátku. Protože se předpokládá, že výška této sumy se bude lišit v závislosti na vypůjčené publikaci, představuje rozhraní `Publication` dobré místo pro tuto logiku. Toto rozhraní totiž implementují všechny publikace (knihy, časopisy apod.). Rozšiřte tedy toto rozhraní o jednu metodu a tu pak implementujete v příslušných třídách. Příklad metody pro třídu `Book` by mohl vypadat následovně:

```
namespace cz\k1886\example;

class Book implements Publication {
    // ... atributy a metody

    public function getDailyRate() {
        return 10;
    }
}
```

Má-li být pro jiný typ knihy účtovaná jiná denní sazba, pak stačí jen přepsat tuto metodu a nový požadavek máte implementovaný. Tím jste se seznámili s prvním pravidlem návrhu softwaru u objektově orientované architektury:

PRAVIDLO

Přístup k údajům vždy v rámci třídy zapouzdřete a poskytněte metody, pomocí nichž lze dané údaje získat.

Stejně jednoduchá je implementace požadavku, aby denní sazba byla nižší v případě vypůjčení knihy na déle než dva týdny. V tomto případě stačí také změnit jen jednu metodu:

```
namespace cz\k1886\example;

class Book implements Publication {
    // ... atributy a metody

    public function getDailyRate($days = 1) {
        if ($days >= 14) {
            return 7.50;
        }
        return 10;
    }
}
```

S tím také samozřejmě souvisí úprava tříd, jež volají metodu `getDailyRate()`, protože tato metoda musí od nynějška předávat počet dní. Této dodatečné změně se můžete vyhnout, pokud jste se při návrhu rozhraní pokusili stanovit, jaké údaje by byly eventuálně potřebné při výpočtu denní sazby. Nikdy však nebude možné zohlednit všechny budoucí požadavky hned při první implementaci rozhraní. V takovém případě byste museli metodě předávat všechny dostupné údaje, čímž byste rozbili zapouzdření údajů. Pokuste se najít zlatou střední cestu a metodu navrhnout tak, aby byla v budoucnosti jednoduše rozšiřitelná – jak jsme si to ukázali v tomto příkladu. Tím jste se naučili už druhé pravidlo objektově orientovaného návrhu softwaru:

PRAVIDLO

Svá rozhraní navrhujte tak, aby je bylo možné později rozšířit.

Na začátku návrhu softwaru provedete vždy stejné kroky: musíte se zamyslet nad spravovanými údaji a nad tím, jak lze tyto údaje zahrnout do entit. Z toho pak vyplynou jednotlivé třídy a metody.

Aktéři

V příkladu knihovny se nacházejí následující aktéři, kteří musí být částí vaší aplikace:

- jednotlivé knihy, které jsou částí knihovny a lze je půjčovat,
- členové knihovny, kteří si knihy půjčují,
- samotná knihovna, která spravuje jednotlivé publikace a stará se o půjčování.

Z těchto představitelů můžete hned odvodit potřebné třídy a rozhraní. Pro knihy jste to už provedli, zůstávají už jen členové a knihovna. Implementujte proto třídu `Member`, která bude reprezentovat jednoho člena knihovny:

```
namespace cz\k1886\example;

class Member {
    protected $id;
    protected $name;

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

    public function getId() {
        return $this->id;
    }

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

Daný člen se skládá z jedinečného identifikátoru, jímž se v systému identifikuje, a ze svého jména, což pro jednoduchou demonstraci úplně stačí. Informace, které jsou přiřazené členovi, se předají prostřednictvím konstruktoru a uloží se do atributů objektu. Přístup k těmto údajům je možný pomocí metod `getId()` a `getName()`.

Po vytvoření tříd pro členy a knihy nastal čas věnovat se samotné knihovně, kterou reprezentuje třída `Library`:

```
namespace cz\k1886\example;

class Library {
    protected $library = array();

    public function addToLibrary($id, Publication $p) {
        $this->library[$id] = $p;
    }

    public function rentPublication(Publication $p, Member $m) {
    }

    public function returnPublication(Publication $p) {
    }
}
```

Knihovna má jeden atribut, do něhož se ukládají všechny publikace určené k vypůjčení, a také tři metody:

- Metoda `addToLibrary()` se používá k přiřazení nové publikace do knihovny. Tato publikace se jednoduše uloží do pole `$library`.
- Pomocí metody `rentPublication()` si může člen knihovny vypůjčit konkrétní publikaci. Tuto metodu jste zatím neimplementovali, neboť nejdříve je nutné definovat, v jaké formě se budou údaje ukládat.
- Pro vrácení publikace se používá metoda `returnPublication()`. V tomto případě není nutné předat objekt člena, který publikaci vrací. Koneckonců knihovna přece musí vědět, kdo měl danou publikaci vypůjčenou.

V následujících krocích budete deklarovat metody naplněné skutečnou logikou. Metody `rentPublication()` a `returnPublication()` mají vždy něco společné s vypůjčením publikace. Buď proces vypůjčení začal, nebo byl vrácením publikace zpět do knihovny ukončený.

Tohoto procesu se vždy účastní dva aktéři: publikace, která má být vypůjčena, a osoba, která si ji vypůjčila. Dále jsou důležité časové údaje, kdy byla publikace vypůjčená a také kdy byla vrácená, aby tak bylo možné vypočítat cenu za vypůjčení. Ve skutečné knihovně to většinou funguje jinak, ale jako příklad postačí i tato varianta. Podle pravidel zapouzdřování, která jste se právě naučili, se pokuste dostat tyto informace do jedné třídy.

Implementace procesů půjčování

Nová třída `RentalAction` musí obsahovat následující informace:

- osoba, která si publikaci půjčuje,
- publikace, která se půjčuje,
- datum, kdy byla publikace vypůjčena,
- datum, kdy byla publikace vrácena a proces ukončen.

Implementace této třídy může vypadat následovně:

```
namespace cz\k1886\example;

class RentalAction {
    protected $publication;
    protected $member;
    protected $rentDate;
    protected $returnDate = null;

    function __construct(Publication $p, Member $m, $date = null) {
        $this->publication = $p;
        $this->member = $m;

        // v případě neuvedení data použít aktuální
        if (null === $date) {
            $date = date('Y-m-d H:i:s');
        }
        $this->rentDate = $date;
    }

    public function getPublication() {
        return $this->publication;
    }

    public function getMember() {
        return $this->member;
    }

    public function getRentDate() {
        return $this->rentDate;
    }

    public function getReturnDate() {
        return $this->returnDate;
```

```
    }
}
```

Konstruktoru třídy musíte při vytvoření nového procesu předat jako parametry publikaci, která má být vypůjčena, a dále osobu, která si ji půjčuje. Volitelně můžete uvést datum a čas, kdy byla daná publikace vyzvednuta. V případě neuvedení data se použije aktuální datum. Tyto hodnoty se uloží v příslušných atributech třídy. Dále se ve třídě nacházejí čtyři metody pro čtení atributů, které vám umožňují přistupovat k atributům třídy. Pokud si nyní přijde nějaký člen vaší knihovny vypůjčit určitou publikaci, můžete vytvořit odpovídající proces vypůjčení takto:

```
use cz\k1886\example\Book;
use cz\k1886\example\Member;
use cz\k1886\example\RentalAction;

$book = new Book('PC', 100);
$mabo = new Member(1, 'Marian Böhmer');
$rentalAction = new RentalAction(
    $book, $mabo, '2011-08-22 16:00:00'
);
```

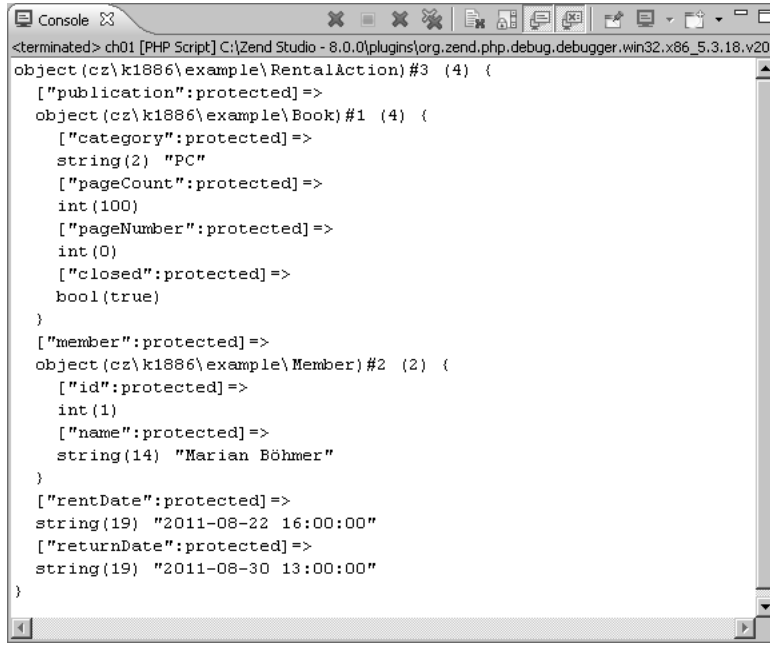
V této chvíli je sice možné knihu vypůjčit, avšak zatím není možné zaznamenat, že už byla i vrácena. Pro tento účel jste si v této třídě už předem rezervovali atribut `$returnDate`. K jeho nastavení je nutné vložit do třídy `RentalAction` následující metodu:

```
namespace cz\k1886\example;

class RentalAction {
    // ... atributy a metody třídy

    public function markPublicationReturned($date = null) {
        // v případě neuvedení data použít aktuální
        if (null === $date) {
            $date = date('Y-m-d H:i:s');
        }
        $this->returnDate = $date;
    }
}
```

Bude-li publikace přinesena zpět do knihovny, pak pro uložení této informace do objektu stačí zavolat příslušnou metodu:

```
$rentalAction->markPublicationReturned('2011-08-30 13:00:00');
```

Výsledek této operace je zobrazený na obrázku 1.1.



Obrázek 1.1: Proces vypůjčení knihy z knihovny

Pokud si chcete ověřit, zda byla kniha vrácena a tím daný proces ukončen, můžete k tomu využít hodnotu, kterou vrací metoda `getReturnDate()`:

```
if (null !== $rentalAction->getReturnDate()) {
    print 'Publikace byla vrácená';
}
```

Tento kód se však neimplementuje jednoduše, a proto je vhodné zapouzdřit jej do další metody:

```
namespace cz\k1886\example;

class RentalAction {
    // ... atributy a metody třídy

    public function isReturned() {
        return null !== $this->returnDate;
    }
}
```

Tuto metodu můžete nyní využít ke zjištění, zda už byla publikace vrácena:

```
if  ($rentalAction->isReturned()) {
    print 'Publikace byla vrácená';
}
```

Tím jste se naučili další pravidlo vývoje softwaru:

PRAVIDLO

V metodách tříd nezapouzdřujte jen údaje, ale také algoritmy, díky čemuž budou komplexní operace implementované centrálně na jednom místě.

Zavedení metody `isReturned()` vám v budoucnosti umožní na základě podmínky určit, zda už byl proces vypůjčení ukončen a kniha vrácena zpět do knihovny.

Pomocí nové třídy `RentalAction` je možné implementovat chybějící metody ve třídě `Library`. Nejdříve je nutné vložit do uvedené třídy nový atribut `$rentalActions`, v němž budou v podobě objektů uložené jednotlivé procesy vypůjčení. Nejjednodušším datovým typem použitelným k tomuto účelu je pole.

Následující implementace metody pro vypůjčení publikace musí splňovat následující kritéria:

- otestovat, zda je požadovaná publikace součástí knihovny,
- otestovat, zda je požadovaná publikace aktuálně vypůjčená (v takovém případě ji nelze znovu vypůjčit),
- pomocí nové instance třídy `RentalAction` vytvořit nový proces vypůjčení.

Implementace metody `rentPublication()` může vypadat následovně:

```
namespace cz\k1886\example;

class Library {
    protected $rentalActions = array();
    // ... atributy a metody třídy

    public function rentPublication(Publication $p, Member $m) {
        $publicationId = array_search($p, $this->library);
        if (false === $publicationId) {
            throw new UnknownPublicationException();
        }
        if (!$this->isPublicationAvailable($p)) {
            throw new PublicationNotAvailableException();
```

```
            }
            $rentalAction = new RentalAction($p, $m);
            $this->rentalActions[] = $rentalAction;

            return $rentalAction;
        }
    }
```

Pomocí funkce `array_search()` jazyka PHP můžete zjistit, zda se publikace nachází v nabídce knihovny. Vrátí-li hodnotu `false`, pak signalizuje chybu vyvoláním výjimky. Test, zda je daná publikace právě vypůjčená, je trochu komplikovanější, a proto je jeho logika přesunuta do další metody. Díky tomu mohou tuto logiku použít i jiné metody třídy. Tímto způsobem opět zapouzdřujete určitý algoritmus do jedné metody. Konkrétní implementace této metody bude probrána později. V případě, že publikace není vypůjčená, vytvoří se nová instance třídy `RentalAction`, které předáte půjčovanou publikaci a člena, který si ji půjčuje. Tuto instanci potom uložíte do k tomu určeného atributu, kde jsou uložené i všechny ostatní procesy vypůjčení. Metoda `rentPublication()` vrací objekt `RentalAction`. Vrácení tohoto objektu sice není v současnosti pro knihovnu důležité, protože neobsahuje žádné další informace než ty, které už obdržela, avšak v budoucnosti může objekt `RentalAction` obsahovat i další informace, například číslo objednávky, které budete chtít poskytnout aplikaci.

Ještě před tím, než začnete ve vypůjčené publikaci listovat, je nutné podívat se na implementaci pomocné metody `isPublicationAvailable()`. Pro zjištění, zda je daná publikace právě vypůjčená, stačí jen otestovat, zda atribut `$rentalActions` obsahuje proces vypůjčení pro hledanou publikaci, který ještě nebyl ukončen. K tomuto účelu vám dobře poslouží metody pro čtení atributů třídy `RentalAction`:

```
public function isPublicationAvailable(Publication $p) {
    foreach ($this->rentalActions as $rentalAction) {
        if  ($rentalAction->getPublication() !== $p) {
            continue;
        }
        if  ($rentalAction->isReturned()) {
            continue;
        }
        return false;
    }
    return true;
}
```

Ve funkci `isPublicationAvailable()` probíhá iterace přes všechny prvky v poli `$rentalActions` (všechny procesy vypůjčení). Pokud aktuální proces nepatří hledané publikaci nebo už je ukončený, pokračuje se s další iterací. Pokud daný proces náleží k hledané publikaci a ještě není ukončen, nemůže být daná publikace znovu vypůjčena, což se signalizuje vrácením hodnoty `false`. Neexistuje-li pro hledanou publikaci žádný proces, který ještě nebyl ukončen, vrátí se hodnota `true`.

V tomto okamžiku lze třídu `Library` použít k půjčování publikací a navíc je zaručené, že jedna publikace může být vypůjčená jen jednou, jak to ukazuje následující příklad:

```
use cz\k1886\example\Library;
use cz\k1886\example\Book;
use cz\k1886\example\Member;

$library = new Library();
$book = new Book('PC', 100);
$mabo = new Member(1, 'Marian Böhmer');
$luigi = new Member(2, 'Luigi Valentino');

$library->addToLibrary('pc1', $book);
$library->rentPublication($book, $mabo);
$library->rentPublication($book, $luigi);
```

Po jeho provedení reaguje knihovna následující výjimkou:

```
Fatal error:  Uncaught exception 'cz\k1886\example\
PublicationNotAvailableException' in ch01\Library.php:24
```

Tuto výjimku můžete ve frontendu své aplikace změnit na chybové hlášení.

Aby bylo možné i vrácení vypůjčených publikací, musíte dále implementovat metodu `returnPublication()`. Tato metoda musí najít aktuální proces vypůjčení, který náleží k dané publikaci, a označit jej jako ukončený. Kód této metody se podobá tomu v metodě `isPublicationAvailable()`. I v tomto případě procházíme přes všechny procesy vypůjčení, dokud nenajdeme proces pro hledanou publikaci, který ještě nebyl ukončen. Tento proces potom označíme pomocí metody `markPublicationReturned()` jako ukončený. Kompletní implementace vypadá takto:

```
namespace cz\k1886\example;

class Library {
```

```
    // ... atributy a metody třídy
    public function returnPublication(Publication $p) {
        foreach ($this->rentalActions as $rentalAction) {
            if ($rentalAction->getPublication() !== $p) {
                continue;
            }
            if ($rentalAction->isReturned()) {
                continue;
            }
            $rentalAction->markPublicationReturned();
            return true;
        }
        return false;
    }
}
```

V tomto okamžiku mohou být vypůjčené publikace znovu vráceny a připraveny na své další vypůjčení:

```
use cz\k1886\example\Library;
use cz\k1886\example\Book;
use cz\k1886\example\Member;

$library = new Library();
$book = new Book('PC', 100);
$mabo = new Member(1, 'Marian Böhmer');
$luigi = new Member(2, 'Luigi Valentino');

$library->addToLibrary('pc1', $book);
$library->rentPublication($book, $mabo);
$library->returnPublication($book);
$library->rentPublication($book, $luigi);
```

Při provedení tohoto kódu nedošlo k vyvolání žádné výjimky. Místo toho se vytvořily dva procesy vypůjčení, z nichž jeden byl i ukončený.

U této implementace vás možná napadne, proč není proces po jeho ukončení z pole vymazán? Jednoduše proto, že byste v takovém případě ztratili historii procesů vypůjčení. Představte si, že by chtěl správce knihovny zjistit, které publikace se nejvíce půjčují nebo kdo je nejaktivnějším členem knihovny. Pomocí procesů uložených v poli `$rentalActions` lze tyto údaje snadno poskytnout.

Je nutné si uvědomit, že v tomto příkladu nešlo o uchovávání údajů ani o jeho výkonnost, ale o vytvoření architektury aplikace. Místo pole s objekty typu `Ren-`

talAction byste v reálné aplikaci použili pro jejich uložení databázovou tabulku. Databázové systémy už nabízejí nástroje, jak lokalizovat záznam k aktuálnímu procesu vypůjčení konkrétní publikace.

Ladění aplikace

V této chvíli máte za sebou základy aplikace a jste schopni půjčovat publikace a přijímat je zpět. V této podkapitole si na příkladu části aplikace ukážeme, jaká další pravidla návrhu softwaru byste měli ve svých aplikacích zohlednit.

Třebaže funkce pro *ladění* (angl. debugging) aplikací v jazyce PHP lze zlepšit pomocí externích nástrojů, skoro každý vývojář v jazyce PHP využívá k výpisu nezbytných informací při tomto kroku příkaz `print` nebo `echo`. Tyto příkazy se během vývoje jednoduše vloží do zdrojového kódu a před tím, než se aplikace nasadí do provozu, se zase odstraní. V rámci této části budete takovýto ladicí kód postupně, kroku za krokem vylepšovat, při čemž se seznámíte s různými pravidly, která byste měli zohlednit i v dalších softwarových projektech.

Při dalším vývoji softwaru knihovny můžete do zdrojového kódu následujícím způsobem vložit hlášení používané při ladění:

```
namespace cz\k1886\example;
use cz\k1886\example\RentalAction;

class Library {
    protected $library = array();
    protected $rentalActions = array();

    public function addToLibrary($id, Publication $p) {
        $this->library[$id] = $p;
        print 'Nová publikace v knihovně: '
              . $p->getCategory() . "\n";
    }

    public function rentPublication(Publication $p, Member $m) {
        $publicationId = array_search($p, $this->library);
        if (false === $publicationId) {
            throw new UnknownPublicationException();
        }
        if (!$this->isPublicationAvailable($p)) {
            throw new PublicationNotAvailableException();
        }
```