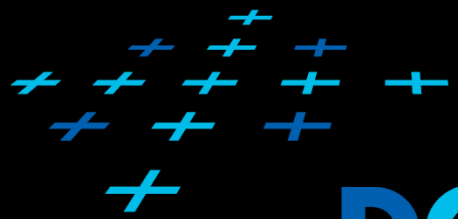


DCGI

KATEDRA POČÍTAČOVÉ GRAFIKY A INTERAKCE

Jazyk PHP pokr.

Martin Klíma

OOP & PHP 5

- V PHP 5 konečně značné rozšíření OOP
- Blíží se to moderním programovacím jazykům jako Java nebo C#
- Výhody OOP
  - Zapouzdření (nové modifikátory public, protected, private)
  - Dědičnost
  - Polymorfismus

Konstruktor / destruktor

- V PHP4 má konstruktor stejné jméno jako třída
- V PHP5 existuje speciální metoda se jménem **__costruct()** a **__destruct()**
- Nicméně PHP5 je zpětně kompatibilní, takže podporuje obojí

konstrutor/destruktor

```php
<?php
// php 4
class A {
      var $x;

      function A($hodnota) // konstrutor podle PHP4
     { $this->x = $hodnota;}
}
//php 5
class B {
      private $x;

      function __construct($hodnota) // konstrutor podle PHP5
     { $this->$hodnota = $hodnota;}
}
?>
```


Konstruktor / destruktor (PHP5)

- Odvozená třída by měla zavolat konstruktor

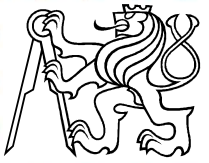
```php
<?php

class A {
      protected $x;    // instanční proměnná

      function __construct($hodnota) {// konstrutor
         $this->x = $hodnota;
      }
}

class B extends A {
      function __construct($hodnota) {
       parent::__construct($hodnota); // volání konstruktoru nadtřídy
      }
}
?>
```


Public, Private, Protected

- Modifikátory viditelnosti – nově v PHP5

- Public – kdokoli může vidět a modifikovat
- Private – je vidět jen uvnitř třídy
- Protected – jako private a také z rozšiřujících tříd
- Vztahuje se na proměnné i metody

- U proměnných se už nepoužívá klíč. slovo **var**



Ukázka PPP

```php
class A {
      public $x = 1;
      private $y = 2;
      protected $z = 3;
      public function p1() {echo $this->x.$this->y.$this->z;}
}

class B extends A {
     function p2 () { echo $this->x.$this->y.$this->z;} // y není vidět
}
$a = new A();
$b = new B();
$a->p1();
$b->p1();
$b->p2();
// ale pozor
$b->y = "Y"; // toto je programatorska chyba
$b->p2();
$a->y = "Y"; // toto je chyba
```


Statické proměnné a metody

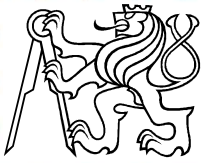
- Klíčové slovo **static**
- Bude vyrobena jen jednou pro všechny instance dané třídy
- Metody nesmí být závislé na proměnných instance
- Dají se volat přes název třídy **trida::metoda();**

Statické proměnné a metody

```php
<?php

class A {
      protected static $pocitadlo = 0;

      public static function stavPocitadla() {
         echo self::$pocitadlo;
               // self nas oprostuje od $this, tj od instance
      }

      public function __construct() { self::$pocitadlo++;}
}

$a = new A();
$b = new A();

A::stavPocitadla(); // volani staticke metody

?>
```


Final

- Třídy a metody mohou být final
- Třída final nemůže být rozšířena
- Metoda final nemůže být přetížena v potomkovi



Final

```php
<?php

final class A {
  function delejNeco() {}
}

class B extends A {} // fatal error

class C {
  final function foo() {}
  function bar() {}
}

class D extends C { function bar() {} } // pretizeni metody bar
je OK
class E extends C { function foo() {} } // chyba!!!
?>
```

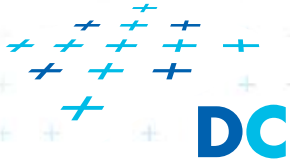
Konstanty tříd

- Konstanty spojené s danou třídou

```php
<?php
class A {
   const HOST = "localhost";
   const USER = "Franta";
}

class B {
   const HOST = "faraon.felk.cvut.cz";
   const USER = "xklima";
}

echo A::HOST;
echo B::HOST;
?>
```


Jmenné prostory

- jednoznačné pojmenování tříd
  - zabraňuje konfliktům jmen
- Od PHP 5.3.0
- klíčové slovo ***namespace***
- v jednom souboru může být definováno více jmenných prostorů (ne jako v Javě)
- namespace se vztahuje na
  - konstanty
  - funkce
  - třídy



Ukázka

```php
<?php
// nas CVUT namespace
namespace Cz\Cvut\Fel\Dcgi\Php;

class Person {
        protected $name;
        protected $surname;

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


// nejaky jiny namespace
namespace Cz\Zend\Php;

class Person {
        protected $name;
        protected $surname;

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


Ukázka

```php
// plne kvalifikovana jmena
$p1 = new \Cz\Cvut\Fel\Dcgi\Php\Person("Frantisek", "Vomacka");
$p2 = new \Cz\Zend\Php\Person ("Jaroslav", "Pazout");


// $p1 a $p2 jsou instance ruznych trid! zde je dukaz:
if ($p1 instanceof \Cz\Cvut\Fel\Dcgi\Php\Person) {
          echo ("\$p1 je instanci tridy \\Cz\\Cvut\\Fel\\Dcgi\\Php\\Person \n");
} else {
          echo ("\$p1 neni instanci tridy \\Cz\\Cvut\\Fel\\Dcgi\\Php\\Person \n");
}

if ($p1 instanceof \Cz\Zend\Person) {
          echo ("\$p1 je instanci tridy \\Cz\\Zend\\Person \n");
} else {
          echo ("\$p1 neni instanci tridy \\Cz\\Zend\\Person \n");
}
?>
```

Magické metody

některé metody jsou tzv. magické

__construct	konstruktor
__destruct	destruktor
__call	je zolána v případě, že voláme neexistující metodu
__callStatic	
__get	je zolána v případě, že voláme neexistující vlastnost
__set	
__isset	
__unset	

__sleep	Řízení serializace
__wakeup	
__toString	Serializace do řetězce
__invoke	třída se může chovat jako fce
__set_state	
__clone	volána před klonováním

OOP v PHP5 - klonování objektů

- v php 4 není, klonují se předáváním hodnotou
- v php 5 speciální metoda

Klonování v praxi

```php
<?php
class A {
      var $x = 1;
}

$instance1 = new A();
$instance1->x = 10;

$instance2 = clone $instance1;
$instance2->x = 20;

echo $instance1->x; // 10;
echo $instance2->x; // 20;

?>
```


Klonování zpětná kompatibilita

```php
<?php

if (version_compare(phpversion(), '5.0') < 0) {
    eval('
    function clone($object) {
      return $object;
    }
    ');
}
?>
```


Klonování

- Můžeme definovat methodu __clone(), která bude volána při klonování
- Například můžeme sledovat, zda daný objekt je klon

Klonování

```php
<?php
class A {
      var $x = 1;
      var $is_clone = false;

      function __clone()
     {
          $this->is_clone = true;
     }
}

$instance1 = new A();
$instance2 = clone $instance1;

echo $instance1->is_clone?"inst1 je klon":"inst1 neni klon";
echo $instance2->is_clone?"inst2 je klon":"inst2 neni klon";
?>
```


Dynamické metody

- Pokud zavoláme metodu, která neexistuje:
  - bude zavolána funkce __call(), pokud existuje
  - pokud neexistuje funkce __call(), dojde k chybě
- Můžeme řešit volání neexistujících metod pomocí metody __call()
- Této metodě je předáno jméno a pole argumentů

Dynamické metody

```php
<?php
class math {
    function __call($name, $args)
    {
        switch ($name) {
            case 'add':
                return array_sum($args);
            case 'divide':
                $val = array_shift($args);
                foreach ($args as $v) $val /= $v;
                return $val;
        }
    }
}

$m = new math();
echo $m->add(1,2); //  3
echo $m->divide(8,2); //  4
?>
```

Dynamické vlastnosti

- Některé vlastnosti mohou být definovány za běhu programu.
- Sada magických metod __get, __set, __isset, __unset jsou v takovém případě zavolány a mohou situaci vyřešit.

```php
class GetSetClass {

        // zde si budeme udrzovat vlastnosti tridy
        private $hodnoty = array();

        // tato vlastnost je definovana a pro pristup
        // k ni nebude volana zadna magicka funkce
        public $jmeno = "GestSetTrida";

        // tato funkce bude zavolana v okamziku, kdy se pokusime
        // nastavit vlastnost, ktera neni dostupna (neexistuje)
        public function __set($klic, $hodnota) {
                // zapiseme klic=>hodnota do pole $values
                $this->hodnoty[$klic] = $hodnota;
        }

        // tato funkce bude zavolana v okamziku, kdy se pokusime
        // precist vlastnost, ktera neni dostupna (neexistuje)
        public function __get($klic) {
                // pokud klic zname, vratime hodnotu
                if (array_key_exists($klic,$this->hodnoty)) {
                        return $this->hodnoty[$klic];
                } else {
                        // jinak vyvolame lehkou chybu
                        trigger_error("Pokus o přečtení nedefinované vlastnosti
$klic", E_USER_NOTICE);
                        return null;
                }
        }
```

```php
// dotaz na existenci vlastnosti
        public function __isset($klic) {
                return isset($this->hodnoty[$klic]);
        }

        //zruseni vlastnosti
        public function __unset($klic) {
                unset($this->hodnoty[$klic]);
        }
}

// otestovani funkcnosti
// fragment kodu

$gsc = new GetSetClass();

$gsc->adresa = "Karlovo nam.";  // pouzije se magicka metoda
$gsc->prijmeni = "Klima";  // pouzije se magicka metoda
$gsc->jmeno = "Martin";  // jmeno je klasicky dostupna vlastnost, bude primo
nastavena

// zkusime se zeptat na nejakou vlastnost
echo ($gsc->adresa); // tato byla drive dynamicky nastavena, hodnota je Karlovo
nam.
// zkusime se zeptat na neco, co jeste nastaveno nebylo
echo ($gsc->PSC); // vyvola nami definovanou chybu.
```

Autoload

Jestliže nebude nalezena definice dané třídy, bude jako zavolána funkce __autoload().

Zde se můžeme situaci zachránit.

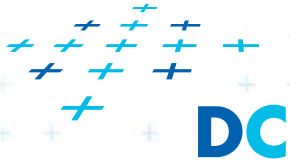
Využití

- Využijeme funkci __autoload k vkládání souborů s definicí tříd v okamžiku jejich potřeby.

- Nahradíme tím instrukci *include_once*
  - include_once je značně náročná

- Můžeme si zorganizovat strukturu našeho projektu takto

```
/class
    trida1.php
    trida2.php
    tridaš.php
```



Autoload

Soubor trida1.php, uložen v adresáři /class

```php
<?php
class Trida1 {
  public $x = 1;
}
?>
```

kuk definiceA.php

Nějaký skript

```php
<?php
function __autoload($classname) {
    $file_name = strtolower($classname).".php";
    include("/class/".$file_name);
}

// trida A zatim nebyla definovana, bude zavolana fce __autoload
$a = new Trida1();
echo $a->x;
?>
```


kuk autoload.php

Typová kontrola (hinting)

- PHP 5 zavádí možnost kontroly typu složitých dat. typů (tedy objekty, pole)
- Lze použít jen u parametrů funkcí/metod

```php
// mejme tridu A
class A {
        private $text = null;

        // text je typu string, nemuzeme pouzit hinting, je to jednoduchy dat. typ
        public function __construct($text) {
                $this->text = $text;
        }
        public function getText(){
                return $this->text;
        }
}

// mejme jinou tridu
class B {
        private $messages = array();
        public function __construct() {
        }

        // zde je typova kontrola, neni mozne volat jinak nez s instanci
        // tridy A jako param
        public function addMessage(A $message) {
                $this->messages[] = $message;
        }
        public function getMessages ($delimiter) {
                return $this->messages;
        }
}


$b = new B();
//$b->addMessage("vzkaz");  // toto je chyba, neprojde pres type hinting
$b->addMessage(new A("vkaz")); // toto je OK, kontrola typu v poho
```

Zpracování chyb v PHP4 a 5

- error_reporting()
- set_error_handler()
- kuk chyby_4.php

```
<?
error_reporting(E_ALL);
function my_error_handler ($severity, $msg, $filename,
$line_num) {
        // dostanu info o chybe a muzu si s ni delat co chci
        echo "Zavaznost: $severity <br>Hlaska: $msg <br> Soubor:
$filename <br> Cislo radku: $line_num <br>";
}
set_error_handler("my_error_handler");
echo $xxx;
?>
```

Vyjímky v PHP5

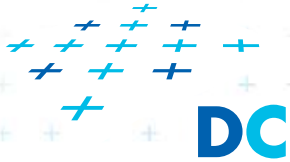
- Je zde zaveden lepší způsob ošetřování vyjímek.
- Podobnost s Javou.
- Jestliže je vygenerována vyjímka (chyba), je vyroben nový objekt.
- Každá vyjímka je rozšířením třídy Exception.
- Odvozením nové třídy lze vyrábět vlastní vyjímky.

Vyjímky

```
class DevZeroException extends Exception {}
class NegativValueException extends Exception {}

function deleni ($a, $b) {
        try {  if ($b == 0) throw new DevZeroException();
               if ($a<0 || $b<0) throw new
NegativValueException();
               return $a/$b;
        }
        /*catch (Exception $e) {
               echo "doslo k nejake vyjimce!!!!";
               return false;
        }*/
        catch (DevZeroException $e) { echo ("nulou nelze
delit\n");
               return false;}
        catch (NegativValueException $e2) {echo ("negative
value\n");    return false;}
}
deleni(1,2);  // zadna vyjimka, vse OK
deleni(1,0);  // DevZeroException
deleni(-1,5); // NegativValueException
```

Abstraktní třídy a metody

- Abstraktní metoda definuje jméno a parametry, žádnou implementaci
- Třída, která má alespoň jednu abstraktní metodu je také abstraktní
- Hodí se tehdy, když chci skoro všechno udělat za koncového uživatele, jenom nějakou maličkost nechat na něm.

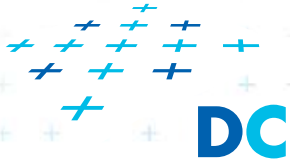


Abstraktní metody a třídy

- Například udělám nákupní košík, který bude umět skoro vše, ale bude nezávislý na použité DB.
- GenericBasket
  - add
  - remove
  - abstract save
  - abstract load

Definice abstraktní třídy

```
abstract class AbstractCart {
    protected $obsah = array();
    public function add ($zbozi) {
        $this->obsah[] = $zbozi;
    }

    public function remove ($zbozi) {
        foreach ($this->obsah as $klic => $polozka) {
            if ($polozka == $zbozi) unset($this->zboz[$klic]);
        }
    }

    public abstract function load();
    public abstract function save();

}
```

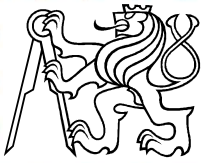


Implementace abstraktní třídy

```
class FileCart extends AbstractCart {

  public function load(){
    $file = fopen("kosik.txt","r");
    $this->obsah = array();
    while ($radek = fgets($file)) {
      $this->obsah[] = $radek;
    }
    fclose($file);
  }

  public function save() {
    $file = fopen("kosik.txt","w");
    foreach ($this->obsah as $polozka) {
      fputs($file, $polozka."\r\n");
    }
    fclose($file);
  }
}
```

Použití

```php
$kosik0 = new AbstractCart(); //  toto nelze!!! chyba


$kosik = new FileCart();
$kosik->add("Brambory");
$kosik->add("Jablka");
$kosik->save();

$kosik2 = new FileCart();
$kosik2->load();

var_dump($kosik2);
```

Interfaces

- Řekněme, že máme 2 zcela různé typy zboží
  - Knihy
  - Parní lokomotivy
- Tyto typu spolu nemají nic společného a proto nemá smysl zavádět dědičnost
- Řešení?
  - Zavedeme rozhraní (interface)
  - Rozhraní je způsob, jak vytvořit vícenásobnou dědičnost

Statické třídní proměnné a metody

- Static = patří k celé třídě, nikoli k instanci
- Co z toho plyne:
  - Existuje jenom jedna (proměnná || metoda) v systému
  - Metody nesmí používat standardní proměnné třídy
  - Metody mohou pracovat používat jen parametricky zadané informace
  - Klíčové slovo **self** místo **this**
    - **this** ukazuje na instanci, tu u statické metody nemáme, proto **self** jakožto ukazatel na třídu

Příklad použití statické proměnné

- Úkol:
  - kolik bylo v systému vytvořeno instancí košíků?
  - kuk (kosik_objektovy1.3.php)

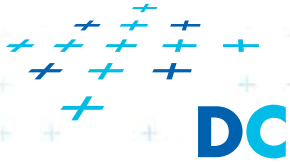
```
class BetterBasket extends Basket {
        protected static $num_of_baskets = 0;

        public function __construct() {
                parent::__construct();
                self::$num_of_baskets++;
        }

        public static function getNumBaskets() {
                return self::$num_of_baskets;
        }
}
```

Vždy volám konstruktor nadtřídy

Statické metody

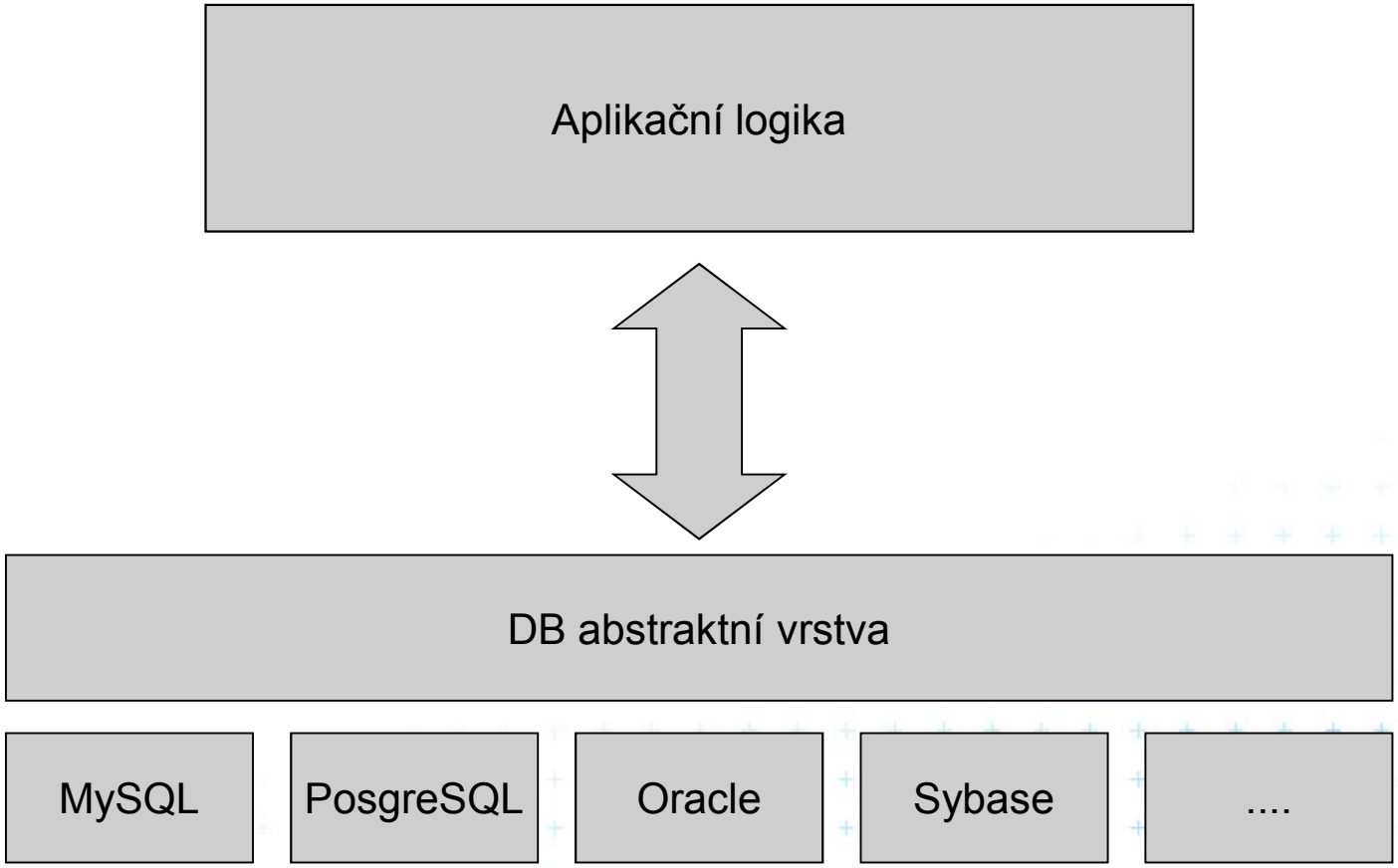
- Dají se volat bez nutnosti vyrobit instanci objektu
- Vše, co potřebují k životu jsou vstupní parametry
- kuk (matematika.php)

```
<?
class Matematika {
        // vypocita maximum ze dvou zadanych cisel
        public static function maximum ($a, $b) {
                return $a > $b?$a:$b;
        }
}

echo "Vetší z čísel 2 a 3 je číslo:
".Matematika::maximum(2,3);
?>
```

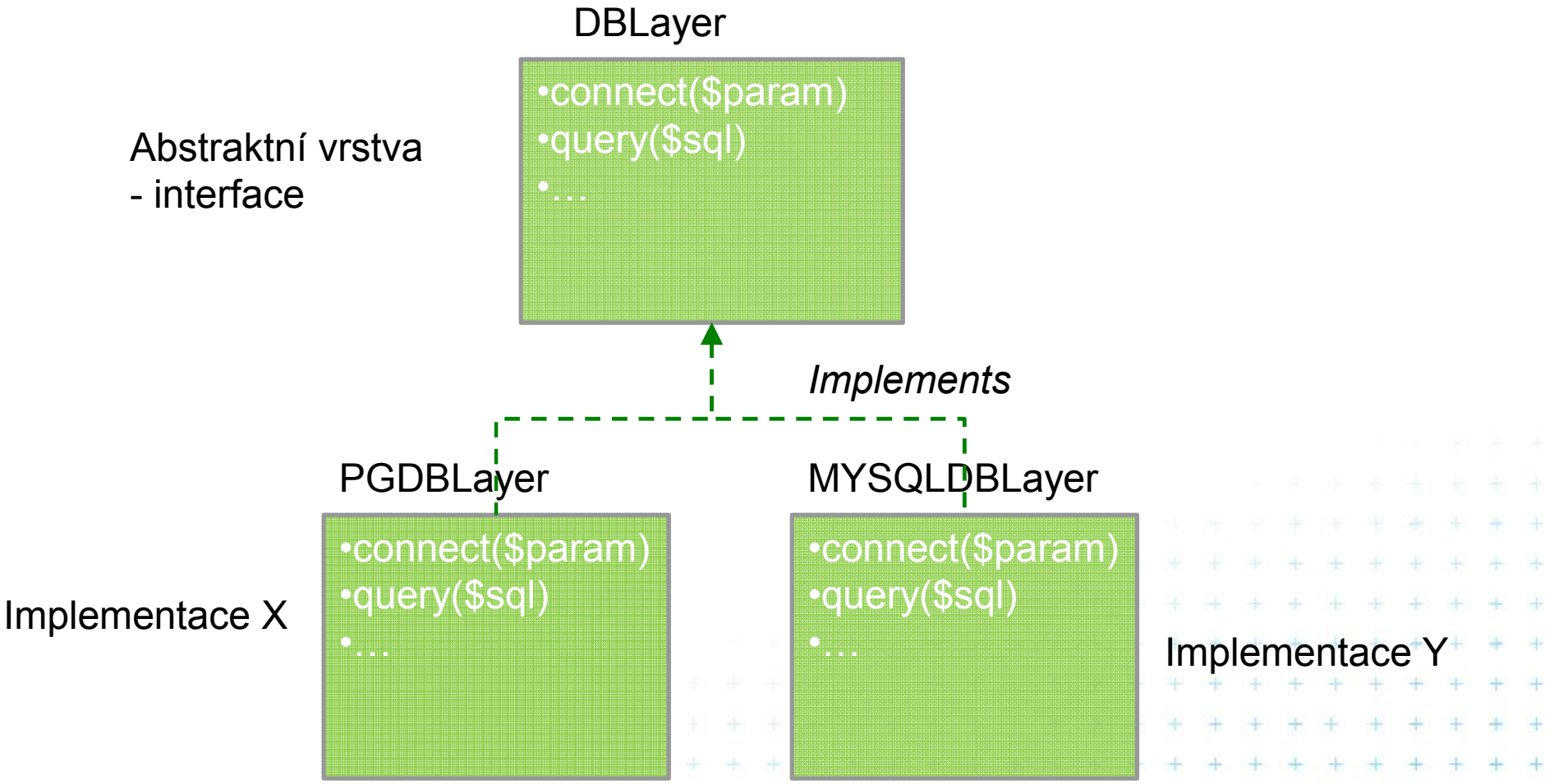


NÁVRHOVÉ VZORY

Vzor Factory – příklad z DB



Vzor Factory



Vzor Factory

- Používá se tehdy, když chceme získat instanci nějakého objektu, ale nechceme se starat o to, jak tento objekt vytvořit
- Příklad:
  - chceme přistupovat k databázi
  - databází je ale mnoho různých druhů (mysql, oracle, …)
  - všechny db implementují stejné rozhraní

Vzor Factory Impl 1/2

```
interface DBLayer {
        public function connect($param);
        public function query($sql);

}

class MySQLDBLayer implements DBLayer {
        public function connect($params) {
                // mysql_connect(....)
        }
        public function query($sql) {
                // mysql_query(...)
        }
}
class PGDBLayer implements DBLayer {
        public function connect($params) {
                //pg_connect(....)
        }
        public function query($sql) {
                // pg_query(...)
        }
}
```

Vzor Factory Impl 2/2

```
// vzor factory = tovarna na objekty
class DBFactory {
  // zde to prijde!
  // vim, ze vratim rozhrani typu DBLayer
  public static function getDBLayer($type){
    switch ($type) {
      case "MySQL": return new MySQLDBLayer(); break;
      case "PG": return new PGDBLayer(); break;
      default: return new MySQLDBLayer();
    }
  }
}
```

Poznámka:
často se setkáme s implementací factory bez parametru. Ten je potom zjištěn v těle metody například z nějakého konfirugračního souboru. Rozhodnutí o použité DB udělá uživatel při instalaci systému.

Jen jedno připojení k databázi

- Vzor Singleton pro připojení k DB
- Připojení je obecně drahá záležitost
- Nepřipojuji se tehdy, když uz spojením mám

```
// jedina trida, ktera umi vsechno s DB
class DB {
        private static $instance = null;
        private $db_link = null;
        private $result = null;

        private function __construct() {
        }
        public static function getInstance() {
                if (self::$instance == null) {
                        self::$instance = new DB();
                        self::$instance->connect();
                }
                return self::$instance;
        }

        public function connect() {
                if ($this->db_link == null) {
                        $link = mysqli_connect(DB_HOST, DB_USER,
DB_PASSWD, DB_NAME);
                        if (!$link) {
                                throw new
DBException(mysqli_errno($link));
                        }
                        mysqli_select_db($link,DB_NAME);
                        $this->db_link = $link;
                }
                return $this->db_link;
        }
```

```
// pokračování

public function query($sql) {
                $this->connect();
                $this->result = mysqli_query($this->db_link,$sql);
                if (mysqli)
                if (!$this->result) {
                        throw new DBException(mysqli_error($this-
>db_link), $sql);
                }
                return $this->result;
        }
}
```

```
// pokračování

public function query($sql) {
                $this->connect();
                $this->result = mysqli_query($this->db_link,$sql);
                if (mysqli)
                if (!$this->result) {
                        throw new DBException(mysqli_error($this-
>db_link), $sql);
                }
                return $this->result;
        }
}
```