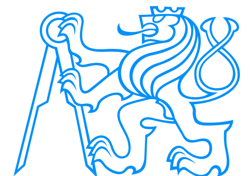
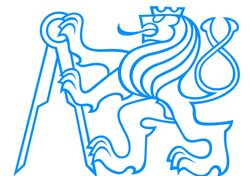
Objektově orientované programování v PHP 5

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

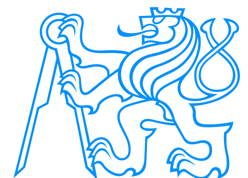


OOP = změna v myšlení

- E-shop
  - uživatel si může přidávat zboží do košíku
  - chceme mít přehled o množství zboží, celkové ceně, výrobci, názvu, …

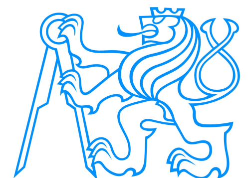
Implementace bez OOP => kuk (kosik_procedural.php)

Implementace pomocí OOP => kuk (kosik_objektovy1.php)



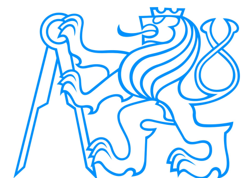
OOP v PHP 5

- Změny oproti PHP 4
  - **přepsaný OOP model**
  - objekty předávány odkazem, nikoli hodnotou
  - nové funkce



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 4
class A {
      var $x;

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

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

Computer Graphics Group

kuk const_dest.php

Konstruktor / destruktor

- Odvozená třída by měla zavolat konstruktor původní třídy

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

      function A($hodnota) {
         $this->x = $hodnota;
      }
}
class B extends A {
      function __construct($hodnota) {
       parent::__construct($hodnota);
      }
}
?>
```

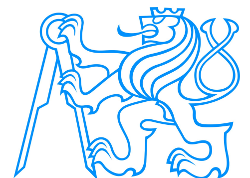
Computer Graphics Group

kuk const_dest2.php

Public, Private, Protected

- Modifikátory viditelnosti

- 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**



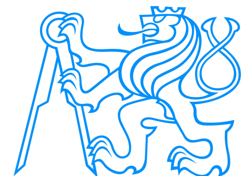
Ukázka PPP

```
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
```

kuk ppp1.php

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();**



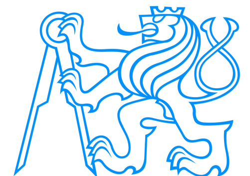
Statické proměnné a metody

```
<?php

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

?>
```

kuk static.php

Final

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



Final

```
<?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
- …konečně ☺

```
<?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;
?>
```

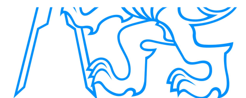
Computer Graphics Group

Zpracování chyb v PHP4 a 5

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

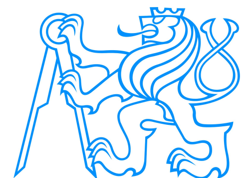
```
<?
error_reporting(E_ALL);
function my_error_handler ($severity, $msg, $filename,
$line_num) {
        // dostanu info o chybe a muzu si s ni delat co chci
        echo "Zavaznost: $severity <br>Hlaska: $msg <br> Soubor:
$filename <br> Cislo radku: $line_num <br>";
}
set_error_handler("my_error_handler");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

Computer Graphics Group



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.



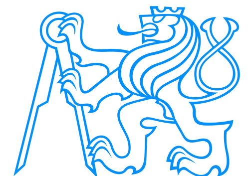
Vyjímky

```
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";
//             return false;}
//      catch (NegativValueException $e2) {echo "negative value
odchyceno v ramci funkce"; return false;}
}
deleni(1,2);
deleni(1,0);
deleni(-1,5);
```

Autoload

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

Zde se můžeme situaci zachránit.



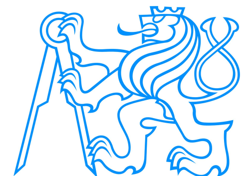
Využití

- 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 struturu našeho projektu takto

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

Autoload

Soubor trida1.php, uložen v adresáři /class

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

Nějaký skript

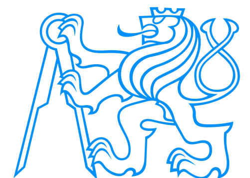
kuk definiceA.php

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

// definici tridy A zatim neni, proto bude zavolana funkce
__autoload.
$a = new Trida1();
echo $a->x;
?>
```


kuk autoload.php

Dynamické metody

- Můžeme řešit volání neexistujících metod pomocí metody __call()
- Této metodě je předáno jméno a pole argumentů



Dynamické metody

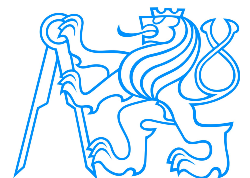
```php
<?php
class math {
    function __call($name, $args)
    {
        switch ($name) {
            case 'add':
                return array_sum($args);
            case 'divide':
                $val = array_shift($args);
                foreach ($args as $v) $val /= $v;
                return $val;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

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

Computer Graphics Group

kuk dynamic.php

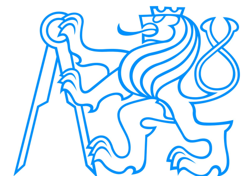
Abstraktní třídy a metody

- 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

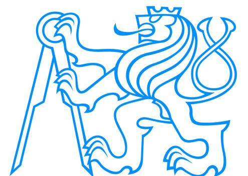
- Například udělám nákupní košík, který bude umět skoro vše, ale bude nezávislý na použité DB.
- GenericBasket
  - add
  - remove
  - abstract save
  - abstract load



Definice abstraktní třídy

```
abstract class AbstractBasket {
    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();

}
```

Computer Graphics Group



Implementace abstraktní třídy

```php
class FileBasket extends AbstractBasket {

  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);
  }
}
```

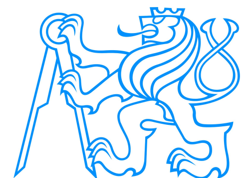
Použití

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


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

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

var_dump($kosik2);
```

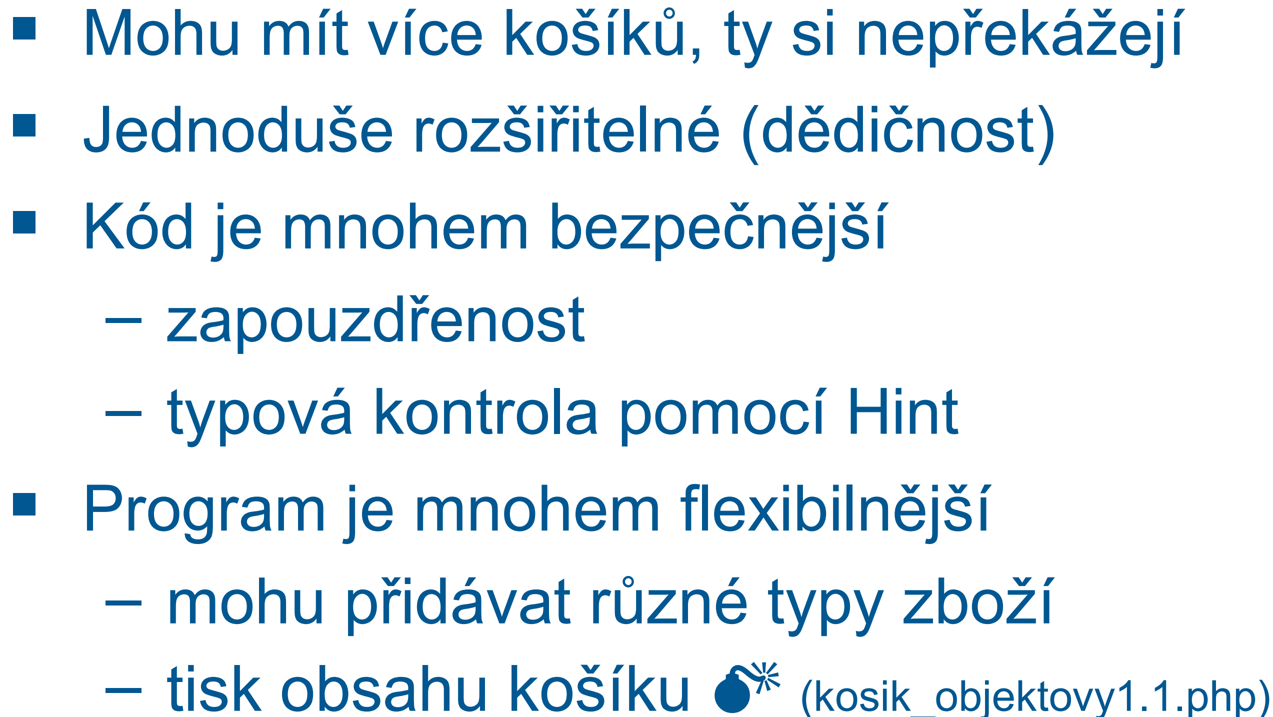
kuk abstract_basket.php

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



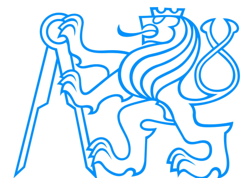
Výhody objektového přístupu



- Mohu mít více košíků, ty si nepřekážejí
- Jednoduše rozšiřitelné (dědičnost)
- Kód je mnohem bezpečnější
  - zapouzdřenost
  - typová kontrola pomocí Hint
- Program je mnohem flexibilnější
  - mohu přidávat různé typy zboží
  - tisk obsahu košíku 💣 (kosik_objektovy1.1.php)

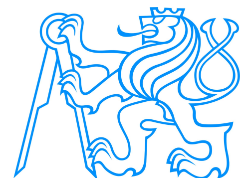
Problém: jak vyrobit nový typ zboží

- Dědičnost
  - Přidání jenom těch vlastností, které jsou nové
  - Ostatní je zděděno
  - Volání konstruktoru mateřské třídy parrent::__construct(….)
- Polymorfismus
  - Dvě různé třídy implementují stejné metody s jinou funkcí (metoda display)



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;
        }
}
```

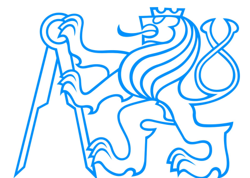
Vždy volám konstruktor nadtřídy

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);
?>
```

Computer Graphics Group

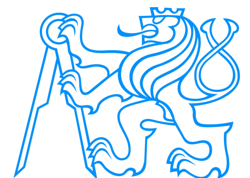


Jen jeden košík v systému?

- Vzor singleton
- kuk singleton.php

```
class SingletonBasket {
		private static $single_basket_instance = null;

		private function __construct() { }

		public static function getInstance() {
				if (self::$single_basket_instance == null) {
						self::$single_basket_instance = new Basket();
				}
				return self::$single_basket_instance;
		}
}
```

Computer Graphics Group



Různé typy zboží v košíku

- 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

Myšlenka rozhraní

Implements

Zboží

- getID( )
- getPrice( )
- getName( )
- display( )

Kniha

- getAuthor( )
- getIssueYear( )
- getISBN( )

Lokomotiva

- getWeight( )
- getWidth( )
- getColor( )

Extends

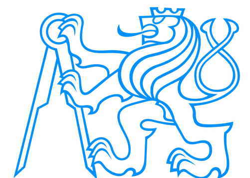
LepsiKniha

- getNumPages( )

kuk kosik_objektovy1.5.php

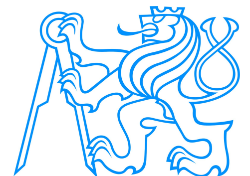
Výhody objektového přístupu

- Mohu mít více košíků, ty si nepřekážejí
- Jednoduše rozšiřitelné (dědičnost)
- Kód je mnohem bezpečnější
  - zapouzdřenost
  - typová kontrola pomocí Hint
- Program je mnohem flexibilnější
  - mohu přidávat různé typy zboží
  - tisk obsahu košíku (kosik_objektovy1.1.php)



Reference

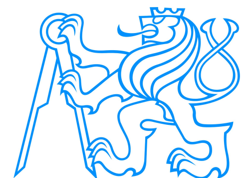
- Odkazem – předává se jen ukazatel do paměti
- Hodnotou – kopíruje se hodnota dat. struktury



```php
<?php
class A {
      var $x = 1;
// pricte hodnotu
     function add($val) {
          $this->x += $val;
     }
}

function zmen_instanci($objekt)
{
     $objekt->add(1);
}

$instance = new A();
zmen_instanci($instance);

echo "Hodnota: " . $instance->x . "<br />"; // PHP4: 1 PHP5: 2

?>
```

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;

?>
```

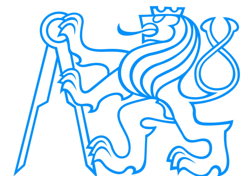

kuk cloning.php

Klonování zpětná kompatibilita

```php
<?php

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

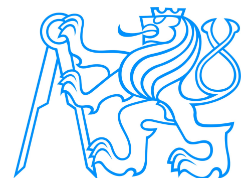
Computer Graphics Group

kuk cloning2.php



Klonování

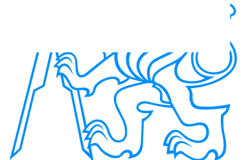
- Můžeme definovat methodu __clone(), která bude volána při klonování
- Například můžeme sledovat, zda daný objekt je klon



Klonování

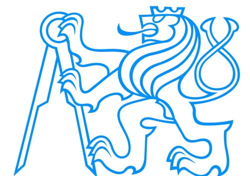
```php
<?php
class A {
	var $x = 1;
	var $is_clone = false;

	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";

?>
```

kuk cloning2.php



Nepřímé reference

V PHP4 se nedala přímo referencovat metoda nebo proměnná objektu, který byl výsledkem nějakého volání. V PHP5 to jde.



Referencing

```
class foo {
    public $bar = 3;
    function baz() {
        echo $this->bar;
    }
}
class bar {
    public $o;
    function __construct() {
        $this->o = new foo();
    }
}

$a = new bar();
$a->o->baz(); // 3
echo $a->o->bar; // 3

/* takto postupne musime v PHP4 */
$tmp =& $a->o;
$tmp->baz();
echo $tmp->bar;
```

kuk referencing.php