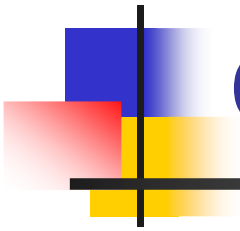
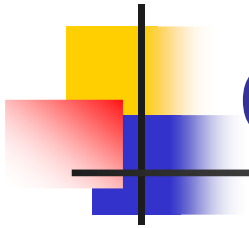


Objekty v PHP 5.x

This is an object-oriented system.
If we change anything, the users object.



Objektové PHP

- Objekty se poprvé objevili v PHP ve verzi 4.
  - Nepříliš zdařilý pokus.
  - Programátoři získali nedůvěru k OOP v PHP.

- Ve verzi 5 se konečně objevily použitelné objekty a třídy.
  - Syntax a sémantika je blízká Javě nebo C#.
    - Objekty jsou referenční typy.
  - Neexistuje mechanismus pozdní vazby (virtuální metody).
    - V PHP jsou všechny metody „virtuální" (interpretované).
  - V kombinaci s některými vlastnostmi PHP lze vytvářet komplikovanější návrhové vzory, než v jiných jazycích.
    - Např. dosazování hodnot proměnných do kódu:

```
$obj = new $className();
$obj->$methodName();
```

Základy syntaxe

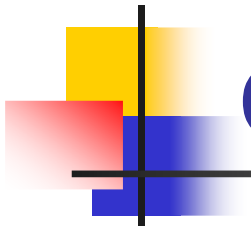
```
class Foo {
    public $var = 0;

    public function bar() {
        echo $this->var;
    }
}

$instance = new Foo();
$instance->var = 42;
$instance->bar();
$instance = null;
```

Deklarace členské proměnné

Třída viditelnosti

Deklarace metody

Operátor pro vytváření instancí tříd

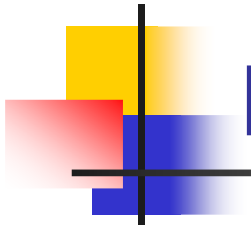


Členské proměnné

- Členské proměnné **musí** mít uvedenou třídu viditelnosti.
  - **`public`** – veřejně přístupné
  - **`protected`** – přístupné pouze z metod dané třídy a tříd odvozených
  - **`private`** – přístupné pouze z metod dané třídy

- Do proměnných jde přiřazovat, i když nebyly deklarovány.
  - Automaticky mají viditelnost **`public`**.

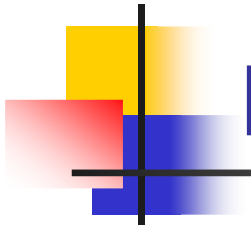
```
class Foo {
    private $bar;
}
$foo = new Foo();
$foo->bar = 1;      // Chyba! bar je private.
$foo->barbar = 42;  // Ok. Nastavi se nova polozka.
```



Metody

- Metody **mohou** mít definovaný typ viditelnosti.
  - Stejné typy jako u proměnných.
  - Pokud není typ uveden, použije se implicitně **`public`**.
- Uvnitř metody je hostující objekt přístupný přes **`$this`**.
  - **`$this`** se **musí** používat pro přístup k položkám objektu.
  - Jinak není možné rozlišit mezi členskými a lokálními proměnnými.
- U metod nefunguje klasický overloading.
  - Nelze definovat víc metod se stejným jménem.
  - Ale je možné používat stejné techniky jako u funkcí.
    - Implicitní hodnoty, lib. počet argumentů, …

Příklad 1



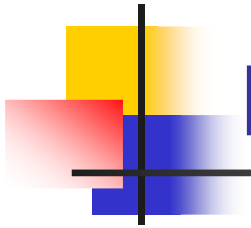
Dědičnost

- Třídy je možné odvozovat (dědit) od jiných tříd.
  - Dědit lze jen od jedné třídy (neexistuje vícenásobná dědičnost).
  - Vícenásobnou dědičnost lze nahradit **`interfaces`** (viz dále).

- Děděné metody mohou být předefinovány.
  - Metody označené klíčovým slovem **`final`** předefinovat nejdou.
  - Všechny metody se chovají jako virtuální.
    - Důsledek absence typovaných proměnných v PHP.
  - Metodu předka lze volat přes konstrukci **`parent::metoda()`**.
    - Případně volat konkrétního předka **`ParentClass::metoda()`**.

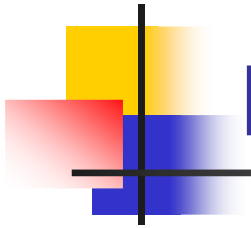
```
class MyFoo extends Foo {
    public function Bar() {
        parent::Bar();
... } ... }
```

Příklad 2



Konstruktor

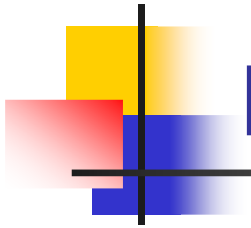
- Speciální metoda **`__construct()`**.
  - Volá se automaticky při vytváření objektu (operátor **new**).
    - Používá se pro inicializaci objektů.
  - Může mít také parametry.
    - Ale nelze jej přetěžovat (chová se stejně jako běžná metoda).
  - Konstruktor nemusí být definován.
    - Použije se konstruktor předka, nebo implicitní (prázdný) konstruktor.
  - Pokud je definován, konstruktor předka se implicitně nevolá.
    - Ale můžeme jej zavolat explicitně (viz dědičnost).
  - Konstruktor by měl být veřejný.
    - Pokud je privátní, instance dané třídy je možné vytvářet pouze ze členských metod.
      - Typicky se pak používají statické metody pro vytváření objektů.



Destruktor

- Speciální metoda **`__destruct()`**.
  - Volá se, když zaniknou všechny reference na daný objekt.
    - Na konci skriptu se automaticky ničí všechny objekty.
  - Nemusí být definován.
    - Automaticky se použije destruktor otce, nebo implicitní (prázdný destruktor).
  - Destruktor nesmí házet výjimky.
    - Při likvidaci objektů na konci skriptu by výjimku neměl kdo chytit.
  - Pokud je definován, destruktor předka se implicitně nevolá.
    - Ale můžeme jej zavolat explicitně (viz dědičnost).
  - Destruktor by měl být veřejný.
    - Pokud veřejný není, je potřeba zajistit, aby se vždy poslední reference na objekt zničila ve členské metodě.
      - Navíc musí být všechny takové objekty zničeny před koncem skriptu.

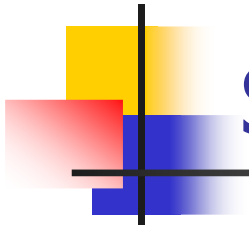
Příklad 3



Konstanty tříd

- Třídy mohou mít konstanty.
  - Uvnitř třídy se deklarují konstrukcí `const name = value;`
  - Nedefinují viditelnost (automaticky jsou public).
  - Přistupuje se k nim přes název třídy a operátor `::`
    - Uvnitř třídy lze použít místo názvu hodnotu **`self`**.

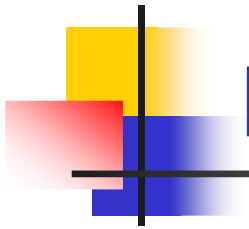
```
class Foo {
    const bar = 42;
    function echoBar() { echo self::bar; }
}

echo Foo::bar;
```



Statické proměnné a metody

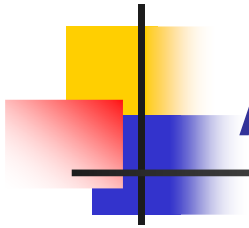
- Statické položky tříd
  - Deklarují se klíčovým slovem **`static`** před proměnnou/metodou.
  - Přistupuje se k nim přes operátor `::` (obdobně jako ke konst.).
    - Např. **`MyClass::$statVar`** nebo **`MyClass::myFunc()`**
  - Mají typ viditelnosti stejně jako nestatické položky.
    - Pozor na dědičnost. Re-deklarace způsobí vytvoření nové proměnné.
  - Statické proměnné existují pouze v 1 instanci pro danou třídu.
    - Bez ohledu na počet objektů.
  - Statické metody se volají na třídě (ne na objektu).
    - Uvnitř nemají k dispozici **`$this`** a pracují jen se statickými daty.
    - Klasické metody lze také volat staticky (pokud nepoužívají **`$this`**).
      - PHP může generovat warning.
      - Je lepší nepoužívat.

Příklad 4



Malá záludnost

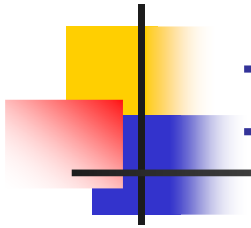
```
class A {
    function foo() {
        echo (isset($this)) ? 'dynamic' : 'static';
    }
}

class B {
    function bar() {
        A::foo();   // staticke volani
    }
}

A::foo();           // vypise 'static'
$obj = new B;
$obj->bar();        // vypise 'dynamic'
```



Abstraktní třídy a metody

- Abstraktní třídy a metody.
  - Deklarují se klíčovým slovem **`abstract`**.
  - Abstraktní třídy.
    - Nelze vytvářet jejich instance.
  - Abstraktní metody.
    - Nemají tělo.
      - Očekává se, že tělo bude definováno v odvozené třídě.
    - Třída s alespoň jednou abstraktní metodou musí být také abstraktní.

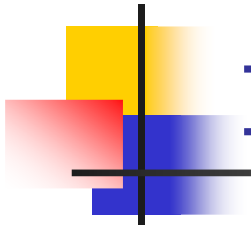
```
abstract class AbstractClass {
    abstract function foo();
}
class ConcreteClass extends AbstractClass {
    function foo() { … foo body … }
}
$obj = new ConcreteClass();
```



Interfaces

- Rozhraní (Interface)
  - Seznam veřejných metod, které musí třída implementovat.
  - Třída může implementovat víc rozhraní, pokud nekolidují názvy metod.

```
interface IFoo {
    public function bar($goo);
}

class Foo implements IFoo {
    public function bar($goo) {
        ...
    }
}
```

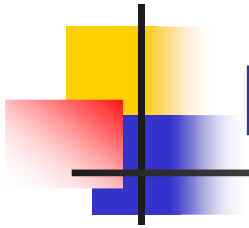


Iterace objektů

- Objekty se dají procházet podobně jako pole.
  - Např. konstrukcí **`foreach`**.
    - Klíčem je řetězec s název položky.
  - Iteruje se přes všechny viditelné položky.
  - Lze předefinovat způsob iterace implementováním interface **`Iterator`**.

```
class MyClass {
    public $var1 = 1;
    public $var2 = 2;
    private $var3 = 3;
}
$obj = new MyClass();
foreach ($obj as $key => $value) { ... }
```

Příklad 5



Kopírování (klonování) objektů

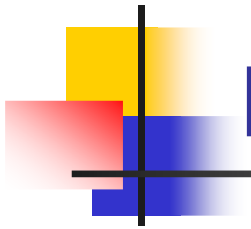
- Objekty se předávají referencí.
  - Přiřazení nekopíruje objekt, ale referenci.
  - Kopírování objektu je nutné vyvolat explicitně – klonováním.

```
$foo = new Foo();  $foo2 = clone $foo;
```

- Klonování provede "mělkou" kopii objektu.
  - Každá položka se zkopíruje operátorem přiřazení.
  - Operace, které se mají provést po klonování lze zapsat do speciální metody **`__clone()`**.
    - Uvnitř je k dispozici kopie objektu v proměnné **`$this`**.

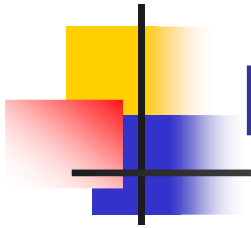
```
public function __clone() {
    $this->innerObj = clone $this->innerObj;
}
```

Příklad 6



Přetěžování přístupu k proměnným

- Přístup k nedefinovaným proměnným lze kontrolovat.
- Existují speciální privátní metody, které lze přetížit.
  - **`__set()`** – nastavení hodnoty členské proměnné
  - **`__get()`** – získání hodnoty členské proměnné
  - **`__isset()`** – přetížení funkce **`isset()`** pro členské proměnné
  - **`__unset()`** – přetížení funkce **`unset()`** pro členské proměnné
- Kontroluje se přístup pouze k členským proměnným, které nejsou viditelné.
  - Buď nejsou deklarovány, nebo jsou privátní.
  - K deklarovaným proměnným se přistupuje přímo.

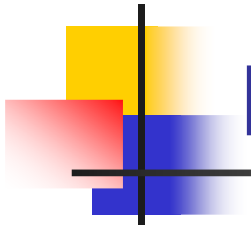


Přetěžování metod

- Obdobně jako u proměnných lze ošetřit přístup k nedefinovaným metodám.

- Přístup je ošetřen privátní metodou **`__call()`**.
  - Metodu je možné předefinovat a ošetřit tak volání metod, které nejsou definovány, nebo nejsou z volaného místa vidět.
  - Pokud je taková metoda zavolána, předá se řízení přímo do **`__call()`**, která dostane název volané metody a předané argumenty.

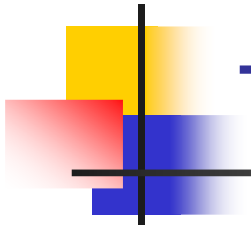
Příklad 7



Další vlastnosti objektů

- Existují další speciální metody, které lze předefinovat:
  - **`__sleep()`** – metoda je volná při serializaci (uložení na disk)
  - **`__wakeup()`** – metoda volaná při deserializaci (načtení z disku)
  - **`__toString()`** – metoda definující chování při přetypování objektu na řetězec

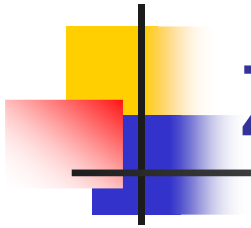
- Chování porovnávacích operátorů.
  - **`($o1 == $o2)`** má hodnotu **`true`**, pokud jsou oba objekty stejné třídy a všechny jejich položky se rovnají.
  - **`($o1 === $o2)`** má hodnotu **`true`**, pokud jsou obě proměnné obsahují referenci na jeden objekt.
  - Operátory **`!=`** a **`!==`** se chovají jako přesné negace svých protějšků.



Type Hinting

- Typová kontrola v PHP (Type Hinting)
  - Od PHP verze 5.1
  - Před parametry funkcí (a metod) je možné uvádět typ a vynutit si tak předání hodnoty konkrétního typu.
  - Typují se pouze objekty a pole.
    - Je-li před parametrem název třídy, musí být jeho hodnota objekt dané (nebo odvozené) třídy, nebo mít hodnotu **`null`**.
    - Obdobně klíčové slovo **`array`** vynutí, že hodnota parametru bude pole (ovšem s libovolným obsahem).
    - Jiné typy nejsou podporovány.
  - Nedodržení typu při volání má za následek PHP fatal error.

```
function foo(MyClass $obj, array $params) { ... }
```

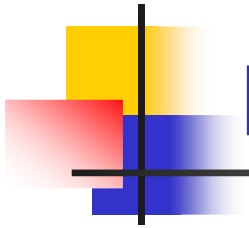


Zjišťování typu objektu

- Operátor **`instanceof`**
  - Ověřuje, zda je objekt instancí dané třídy, nebo třídy odvozené.
  - Také umí ověřit, zda daný objekt implementuje nějaký interface.

```
if ($foo instanceof FooClass) ...
```

- Další související funkce pro testování typů:
  - **`get_class()`** – vrací název třídy daného objektu jako řetězec
  - **`get_parent_class()`** – vrací název rodičovské třídy
  - **`is_a()`** – zjišťuje, zda je objekt dané třídy
  - **`is_subclass_of()`** – zjišťuje, zda je objekt potomkem dané třídy
    - Funkce **`is_a()`** a **`is_subclass_of()`** jsou zastaralé – nahrazuje je operátor **`instanceof`**.

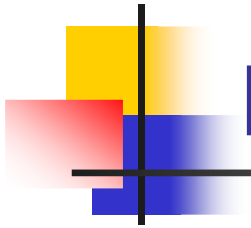


Funkce pro práci s třídami a objekty

- Další funkce pro práci s třídami a objekty
  - Zjišťování existence
    - **`class_exists()`** – zjišťuje, zda daná třída existuje
    - **`interface_exists()`** – zjišťuje, zda byl interface definován
    - **`method_exists()`** – ověří, zda má objekt určitou metodu
  - Hromadné výpisy
    - **`get_declared_classes()`** – vrací pole definovaných tříd
    - **`get_declared_interfaces()`** – vrací definovaná rozhraní
    - **`get_class_methods()`** – vrací seznam metod třídy
    - **`get_object_vars()`** – vrací pole členských proměnných objektu
    - **`get_class_vars()`** – vrací pole deklarovaných proměnných třídy
  - Nepřímé volání metod

```
call_user_func_array(array($obj, 'methodName'),
    $params);
```

Příklad 8



Použití objektů a tříd v PHP

- Třídy a objekty zapouzdřují logické části aplikace.
  - Lepší sémantika, přehlednost, bezpečnost, …
  - Snazší spolupráce více lidí na jednom projektu.
  - Rychlejší předávání dat (objekty se předávají referencí).
    - I když to např. pole také s použitím copy-on-write.
- Používání návrhových vzorů.
  - Osvědčené metody programování známé z jiných jazyků.
    - Singleton, Factory, Adapter, Front-Controller, Model-View-Control, …
- Nevýhody OOP.
  - Většinou delší kód.
  - O trochu pomalejší zpracování.