PHP OOP

PROČ OOP

* umožňuje tvořit architekturu webových aplikací, která je velice důležitá
* používání komponent 3. strany
* větší projekty jsou přehlednější, další implementace, kvalitnější
* jedná se o modernější metodiku vývoje softwaru
* klademe důraz na znovupoužitelnost
* pokud je někde chyba, stačí opravit chybu na jednom místě
* píšeme spíš z pohledu člověka ne jak to vidí PC -> přehlednější, přirozenější
* PHP není objektový jazyk .. pouze objektově orientovaný
  + vnitřní funkce nejsou objektové

PROBLEMY SE STRUKTUROVÝM

* často se opakují funkce a nelze je plně využít
* při tvorbě GLOBAL nelze později uhlídat (funkce je můžou přepisovat)
* neobjevovanému kódu se říká „spaghetti code“

Základní jednotka -> **objekt** (člověk, pes, databáze) == **instance třídy**

* **atribut** (jméno, věk, u databáze -> heslo), náleží objektu
* **metoda** (schopnosti, které objekt umí -> mluvit, štěkat, VyhledejZaznam)
* **interface** = rozhraní

TŘÍDA

* vzor, podle kterého se objekty vytváří
* definuje jejich vlastnosti a schopnosti
* objekt, který se vytvoří podle třídy == instance
  + třída *Člověk()* -> **instance** $karel, $josef
  + mají stejné **rozhraní** ($jméno, $věk, JdiDoPrace(), Pozdrav() ), ale každý má jiné jméno, věk, …
* třídy píšeme do samostatných souborů a je dobré je pojmenovávat stejně, je dobrý udělat složku Třídy a tam vkládat všechny třídy

Magické metody

* magické, protože se spouští sami (automaticky) -> reagují na určitou událost
* umožňují nám provést tzv. zapouzdření
* Konstruktor
  + \_\_construct(libovolný počet $atributů)
  + k nastavení vnitřních stavu objektu
  + ušetří řádky
* Destruktor
  + \_\_destruct()
  + volá se, když se objekt odstraňuje o paměti
  + nemají moc velký význam (skripty běží chvilku a sami se mažou)
  + **Využití:**
    - pro uložení dat do databáze před zrušením objektu
    - ukončit databázové spojení
  + hlášku vypíše před zničením instance ($karel, $jan)
  + odstranění dojde až po ukončení aplikace / když již nemá cenu je uchovávat
* ToString
  + spustí se ve chvilky, kdy chceme instanci (obekt) převést na text. řetězec
  + když chceme vypsat objekt

**====================PŘÍKLAD======================**

<?php

// ../Tridy/Clovek.php

class Clovek

{

// code here (attributes, methods)

public function pozdrav() // metoda

{

echo („Ahoj“);

}

}

?>

<?php

// ../index.php

require\_once (‚Tridy/Clovek.php‘)

$karel = new Clovek();

$karel -> pozdrav(); // vypíše Ahoj

?>

* public = volání instance z venčí
* require\_once = musíme odkázat (once -> načte jen pokud ještě není načtená)

**====================PŘÍKLAD 2====================**

<?php

// ../Tridy/Clovek.php

class Clovek

{

// code here (attributes, methods)

public $jmeno;

public $prijmeni;

public $vek;

public function pozdrav() // metoda

{

echo („Ahoj, ja jsem “ . $this->jmeno); // **jmeno BEZ $**

}

. . .

}

?>

<?php

// ../index.php

require\_once (‚Tridy/Clovek.php‘)

$karel = new Clovek();

$karel -> jmeno = ‚Karel‘ ;

$karel -> prijmeni = ‚Novák‘ ;

$karel -> vek = ‚20‘ ;

$karel -> pozdrav() ; // vypíše Ahoj, ja jsem Karel

$jan = new Clovek();

$jan -> jmeno = ‚Honza‘ ;

$jan -> prijmeni = ‚Dvořák‘ ;

$jan -> vek = ‚22‘ ;

$jan -> pozdrav() ; // vypíše Ahoj, ja jsem Honza

?>

$this = instance, se kterou zrovna pracujeme

**====================KONSTRUKTOR======================**

public function \_\_construct($jmeno, $prijmeni, $vek)

{

$this -> jmeno = $jmeno;

$this -> prijmeni = $prijmeni;

$this -> vek = $vek;

}

<?php

// ../index.php

require\_once (‚Tridy/Clovek.php‘)

$karel = new Clovek(‚Karel‘‚Novák‘,18‘);

$karel -> pozdrav() ; // vypíše Ahoj, ja jsem Karel

$jan = new Clovek(‚Jan‘‚Dvořák‘,22‘);

$jan -> pozdrav() ; // vypíše Ahoj, ja jsem Honza

?>

**====================DESTRUKTOR======================**

public function \_\_destuct()

{

echo („Byl jsem odstraněn z paměti“);

}

index.php

Ahoj, ja jsem Karel

Ahoj, ja jsem Honza

Byl jsem odstraněn z paměti

**====================ToString======================**

public function \_\_toString()

{

return $this->jmeno;

}

index.php

echo ($karel)

**==============================================================**

**==================== Další učivo (4 - 8 lekce) ======================**

**==============================================================**

**3. hlavní pilíře (zapouzdření, dědičnost, polymorfismus)**

**Příklad galerie**

**Referenční a primitivní datový typ**

TŘI ZÁKLADNÍ PILÍŘE

Zapouzdření (**černá skříňka**)

* umožňuje ukrýt některé metody a atributy, tak aby zůstali použitelné jen pro třídu zevnitř (pro vnitřní potřebu třídy)
* slovo = **private** (atributy i metody)
* rozdělení rozhraní (**interface**) třídy na veřejně přístupné (**public**) a vnitřní strukturu (**private**)
* *ukázka: MAMP / OOP / Škola / zapozdreni*

Dědičnost(inheritance)

* slouží k vytvoření nových datových struktur na základě starých
* **důležité** => class*nova\_třída* **extends** *existujici\_trida* (potomek)(mateřská třída / předek)
* nemusíme opisovat metody, které už existují pouze přidáme
* dědí stejné atributy + metody
* nedochází k duplikování kódu
* při změně v mateřské třídě se změní všude
* při vytváření konstruktoru musíme celý znovu vytvořit
  + $this->**pohavi** = $pohlavi;
  + **parent::*\_\_construct***($jmeno, $prijmeni, $vek);
* nejsou přístupné **PRIVATE** atributy pouze **PUBLIC**

Polymorfismus

* **metoda / metody, které mají všichni potomci definované se stejnou hlavičkou, ale jiným tělem**
* umožňuje používat jednotné rozhraní pro práci s různými typy objektů
* umožňuje přepsat metodu u podtřídy tak, aby dělala co chceme, rozhraní (**interface**) tak zůstane zachovalí a nemusíme přemýšlet

jak se u onoho objektu volá

* př.: metoda *speak()*, ale každý mluví jinak *(Haf, Mňau, Bů, .. )*
* díky polymorfismu se můžeme vyhnout větvení (if, else if, .. )

**==============================================================**

Protected

* *private* je pouze součástí jedné třídy, ne potomka
* slouží k modifikování pro potomky a **zvenku** je pořád **nepřístupný** (atribut, metoda)

Final

* pokud **nechceme** **přepsat** danou metodu, nebo třídu (nelze poté dědit)
* doporučuje se nepoužívat

Automatické volání tříd

* automaticky zavolá třídy, které vytvoříme (při vytvořeni instance)
* vytvoříme funkci: function NactiTridu($trida)
* v těle bude: require („Tridy/$trida.php“);
* volání: spl\_autoload\_register(„NactiTridu“);

**==============================================================**

PRIMITIVNÍ DATOVÉ TYPY:

* všechny datové typy kromě objektů (číslo, znak, pole, textový řetězec)
* pokud někam předáme primitivní typ **(hodnota se vždy kopíruje)**

REFERENČNÍ DATOVÝ TYPY:

* objekty
* **nelze kopírovat** (atribut není proměnná !)
  + **$a** = **new** Clovek('Karel', 'Novák', 30);
  + **$b** = **$a**;
  + **$b**->vek = 50;
  + print\_r ($b) <— prázdné

VYNUCENÍ REFERENCE

* předání primitivního typu —> referenci (objekt)
* **spíše se vyvarovat této možnosti**