Webbutveckling med PHP

OBJEKTORIENTERING

Utbildare: Mikael Olsson

mikael.olsson@emmio.se
076-174 90 43

NACKADEMIN

Objektorientering

- Inkapsling - svart box
- Oönskad påverkan mellan programmets olika delar minimeras

- Lättare att återanvända programdelar från ett program i ett annat, eftersom objektklasserna, ofta blir generella och användbara i olika sammanhang.

Adressbok

Försök 1: Excel

	A	B	D	F	G
1	**Namn**	**Gatuadress**	**Tel hem**	**Arbetsgivare**	**Bil1**
2	Elin Nilsson	Kalmarsundsgatan 5	0321-321 54	Ulricehamns kommun	Volvo KCX 123
3	Olle Andersson	Gatan 3	011-12 34 56		
4	Eva Ask	Vägen 5	013-98 65 32		Nissan PUK 456
5					
6					

Vad händer om vi vill lägga till ett nummer för en person?

Adressbok

	A	B	C	D	F	G
1	**Namn**	**Gatuadress**	**Tel arb**	**Tel hem**	**Arbetsgivare**	**Bil1**
2	Elin Nilsson	Kalmarsundsgatan 5	0321-123 45	0321-321 54	Ulricehamns kommun	Volvo KCX 123
3	Olle Andersson	Gatan 3		011-12 34 56		
4	Eva Ask	Vägen 5		013-98 65 32		Nissan PUK 456
5						
6						
7						

Adressbok

- Slöseri – flera fält är tomma
- Redundans - Samma värde förekommer flera gånger
- Oflexibelt – om vi behöver flera fält måste vi lägga till det för hela databasen
- Hur kan vi göra det bättre?

	A	B	C	D	E	F	G	H
1	**Namn**	**Gatuadress**	**Tel arb**	**Tel hem**	**Tel mobil**	**Arbetsgivare**	**Bil1**	**Bil2**
2	Elin Nilsson	Kalmarsundsgatan 5	0321-123 45	0321-321 54	070-123 456 78	Ulricehamns kommun	Volvo KCX 123	Hyundai PUF 321
3	Olle Andersson	Gatan 3		011-12 34 56				
4	Eva Ask	Vägen 5		013-98 65 32			Nissan PUK 456	
5								

ER-modellering (ERM)

- Entity-relationship model
  - Entities (enheter)
  - Relationships (relationer)
  - Attributes (egenskaper)

Entity

- Något med en egen existens, ett substantiv
- Kan vara ett fysiskt objekt eller en händelse

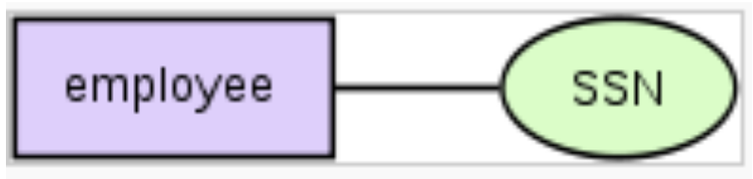
Två relaterade entities

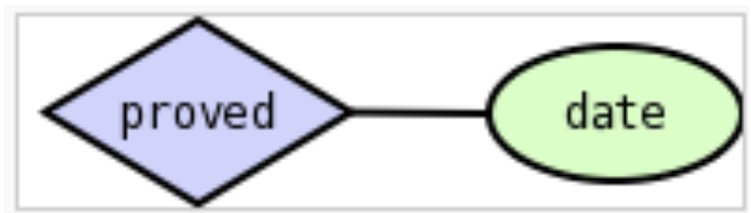
Relationships

- Beskriver hur två eller flera entities hör ihop.
- Tänk verb som länkar ihop två eller flera entities.
  - Ett ägandeförhållande mellan ett företag och en dator.
  - Ett tillhörandeförhållande mellan en anställd och en avdelning.
  - Ett utförandeförhållande mellan en artist och en sång.

Attributes

- Entities och relationer kan ha attribut (egenskaper).





Uppdelning

- Vilka entities har vi i vår adressbok?
  - Personer
  - Adresser/bostäder
  - Bilar

FirstName	LastName	Address	Rooms	Car 1	Car 2
Eva	Vik	Vägen 1	3	Volvo V70 – KXC122	Ford Ka – GRE479
Stina	Nilsson	Gatan 3	1	Ford Ka – ASD542	
Lars	Nilsson	Gatan 3	1		

Uppdelning

- Person

FirstName	LastName
Eva	Vik
Stina	Nilsson
Lars	Nilsson

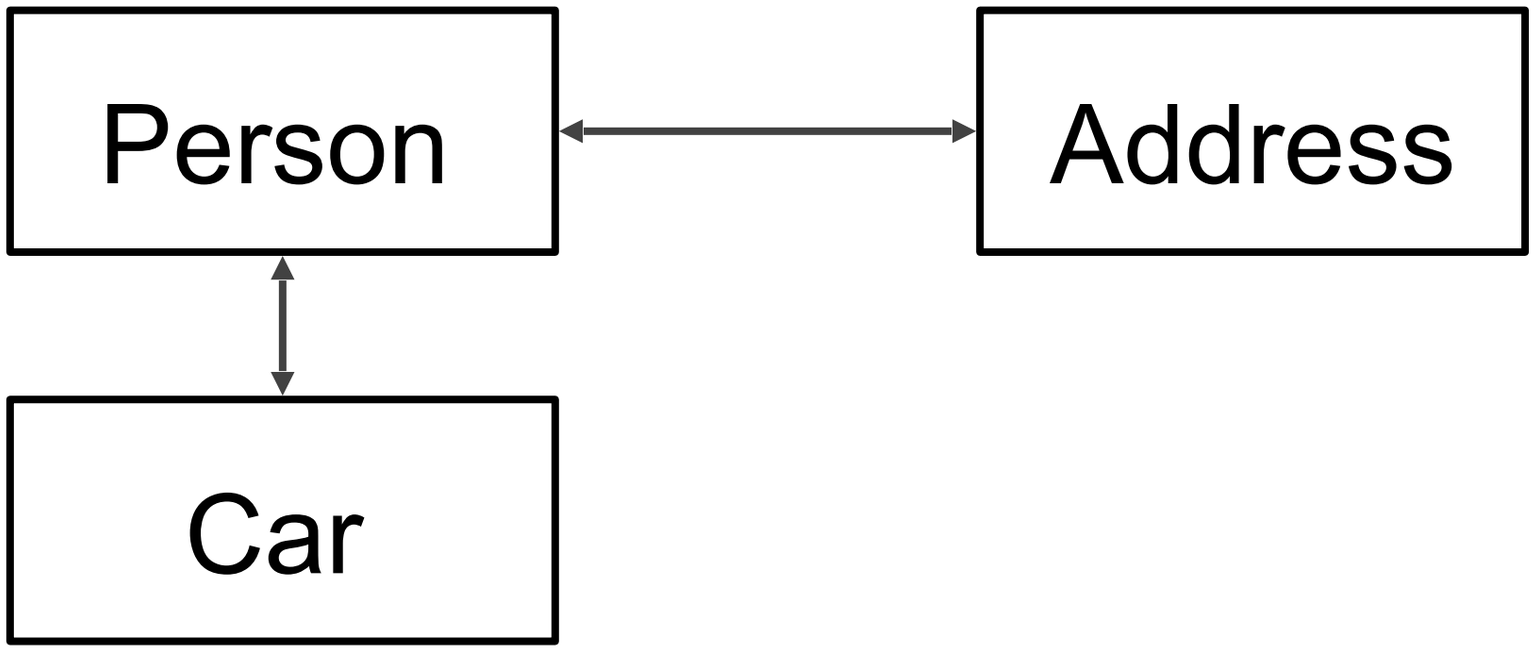
- House

Address	Rooms
Vägen 1	3
Gatan 3	1

- Car

Car	Registration
Volvo V70	KXC122
Ford Ka	GRE479
Ford Ka	ASD542

Relation



Gruppuppgift

- Gör en ER-modell över
- Utbildning / kurser
- TV-guide
- Boksamling

Objektorientering

- I objektorienterad programmering kan vi på liknande sätt sätta upp *klasser* för våra entiteter.

- Klasser kan ha olika medlemsvariabler

- Klasser kan ha olika metoder

Vad är en klass?

- En klass är en ritning, en mall för en entitet som vi vill använda i ett program.
- Vi kan t ex skapa en klass av typen Bil.
    - Den här klassen kan innehålla egenskaper (färg, modell, årsmodell mm).
    - Den kan även innehålla metoder (funktioner) som specificerar vad bilen gör (kör, backar, stannar mm).

Skapa en klass

```
class Book {

}
```

Medlemsvariabler

- *Medlemsvariabler* kallas även *egenskaper* (*properties*).
- Variabler som hör till objektet.
- Kan ha olika *synlighet*.
    - Vi kommer att prata mer om synlighet senare. Just nu räcker det att veta att medlemsvariabler än så länge måste deklareras med nyckelordet `public`.

Klassvariabler

```
class Book {
  public $title;
  public $numPages;
}
```

Instansiera en klass

- Att skapa en klass gör ingenting i sig.
- Man kan skapa ett *objekt* från en klass.
- Objektet kallas en instans av klassen.
- Man skapar ett objekt med nyckelordet `new` och klassnamnet.

```
$yellowPages = new Book();
```

Instansiera en klass

```
class Book {
  public $title;
  public $numPages;
}

$yellowPages = new Book();
```

Använda klassvariabler

- Man använder medlemsvariabler hos ett objekt genom att använda deras namn.

```
$object->varName;

$yellowPages->numPages = 250;

echo $yellowPages->numPages;
```

Skapa klass

- *Uppgift*: Skapa klassen Student.
    - Ge klassen egenskaperna *name*, *age* och *course*.
    - Sätt lämpliga värden på medlemsvariablerna och skriv sedan ut dem.
- Om du får tid över, skapa fler medlemsvariabler.
    - Lösningsförslag: http://sandbox.onlinephpfunctions.com/code/2c531cc18e406d9b0fccaef60bc5589ac0c34513

Metoder

- Medlemsfunktioner, eller *metoder*, fungerar nästan som vanliga funktioner.

- Deklareras i en klass.

- Tar emot data, manipulerar data eller returnerar data.

Metoder

```
class Book {
  public $title;
  public $numPages;

  function setNumPages($numOfPages){
    $this->numPages = $numOfPages;
  }

  function getNumPages(){
    return $this->numPages;
  }

  function setTitle($Title){
    $this->title = $Title;
  }

  function getTitle(){
    return $this->title;
  }
}
```

Magiska metoder

- The function names `__construct()`, `__destruct()`, `__call()`, `__callStatic()`, `__get()`, `__set()`, `__isset()`, `__unset()`, `__sleep()`, `__wakeup()`, `__toString()`, `__invoke()`, `__set_state()`, `__clone()` and `__debugInfo()` are magical in PHP classes. You cannot have functions with these names in any of your classes unless you want the magic functionality associated with them.

- https://www.php.net/manual/en/language.oop5.magic.php

- https://dzone.com/articles/9-magic-methods-php-0

Vad är *$this*?

- Inuti klassen finns inte objektet. Om man vill använda sig av medlemsvariabler inuti klassen måste man hitta på en annan lösning.

- Istället använder man *psuedo-variabeln* `$this`.

```
class Book {
  public $title;
  public $numPages;

  function setNumPages($noOfPages) {
    $this->numPages = $noOfPages;
  }
}
```

Använda metoder

- Man använder metoder på ungefär samma sätt som medlemsvariabler.

```
$object->functionName();

$yellowPages->getNumPages();
```

Skapa klass

- *Uppgift*: I klassen Student:
    - Skapa metoden `print()` som skriver ut studentens alla egenskaper.
- Om du får tid över (tankenöt, lösning kommer):
    - Skapa medlemsvariabeln `$teacher`.
    - Skapa klassen `Teacher` med lämpliga medlemsvariabler.
    - Skapa ett objekt av klassen `Teacher` och sätt `$student->teacher` till objektet.
    - Ändra `print()`-metoden så att den skriver ut läraren också.

Lösningsförslag

- Lösningsförslag till extrauppgiften:

- http://sandbox.onlinephpfunctions.com/code/12d7d46da4be344825b8540fec81c0ecf391b700

Konstruktor

- Konstruktorer är särskilda metoder som används för att sätta default-värden på ett objekt när det skapas, eller automatiskt göra något med objektet.

```
class Book {
  public $title;
  public $numPages;

  function __construct() {
    $this->$numPages = 250;
  }
}
```

Konstruktor

- När du skapar ett nytt objekt av klassen nu kommer den att få de värden som sätts i konstruktorn.

```
class Book {
  public title;
  public $numPages;

  function __construct() {
    $this->$numPages = 250;
  }
}

$yellowPages = new Book();
echo $yellowPages->numPages; // 250
```

Övning

- *Uppgift*: Skapa en konstruktor för din klass Student.
    - Gör så att *course* får standardvärdet ”PHP”.
- Extra uppgift: En konstruktor är en vanlig funktion. Gör en konstruktor som låter dig sätta namnet på studenten när du skapar den genom att ta en parameter.
    - Sätt ett default-värde för parametern.

Lösningsförslag

- Lösningsförslag till extrauppgiften:

- http://sandbox.onlinephpfunctions.com/code/4abe207109d9720cf9101ab88903f8d4242d7d88

Objektorientering

```php
class Student {
    public $name;
    public $age;
    public $teacher;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Teacher {
    public $name;
    public $age;

    function __construct($name = '', $age = 35) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

- Finns det några likheter mellan klasserna?
- Vad gör vi om vi behöver lägga till telefonnr till båda?

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->print();
$teacher->print();
```

- Student och Teacher *ärver* från Person.

- En barnklass som ärver av en föräldraklass har samma klassvariabler och metoder som föräldraklassen.

Övning

- Skapa klassen `Vehicle` med medlemsvariablerna `color`, `model`, `speed` och `noOfWheels`.

    - Skapa metoderna `accelerate()` och `decelerate()` som båda tar en parameter och ökar resp minskar hastigheten med parametern. Kontrollera att parametern inte är orimlig, t ex hastigheten inte blir mindre än 0.

    - Skapa metoden `print()` som skriver ut egenskaperna för fordonet.

- Skapa klasserna `Motorcycle` och `Sportscar` som ärver av `Vehicle`.

    - Skapa objekt av båda klasserna och kör metoden `print()` på dem.

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
    public $teacher;
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->teacher = $teacher;

$student->print();
$teacher->print();
```

- Student har dock ingen medlemsvariabel för `$teacher` längre.

- Vi kan dock lägga till variabler till vår nya klass, sedan finns den tillgänglig för nya objekt.

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
    public $teacher;

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
        echo 'Teacher: ' . print_r($this->teacher, true) . PHP_EOL;
    }
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->teacher = $teacher;

$student->print();
$teacher->print();
```

- `print()` i Student skriver inte ut läraren.

- Vi kan ladda över (*override*) metoder från föräldrar.

Arv

```php
class Person {
    public $name;
    public $age;

    function __construct($name = '', $age = 25) {
        $this->name = $name;
        $this->age = $age;
    }

    function print() {
        echo 'Name: ' . $this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
    }
}
```

```php
class Student extends Person {
    public $teacher;

    function print() {
        parent::print();$this->name . PHP_EOL;
        echo 'Age: ' . $this->age . PHP_EOL;
        echo 'Teacher: ' . print_r($this->teacher, true) . PHP_EOL;
    }
}

class Teacher extends Person {
}

$student = new Student('kalle');
$teacher = new Teacher('Micke', 42);

$student->teacher = $teacher;

$student->print();
$teacher->print();
```

- Nu har vi dock redundans igen, vi gör ju redan samma sak i föräldrametoden.
- Vi kan dock anropa föräldrametoden från vår metod.

Scope Resolution Operator (::)

- Scope Resolution Operator, dubbelkolon ger tillgång till statiska, konstanta och överskrivna egenskaper eller metoder i en klass.

- När man refererar till dessa items utanför en klass använder man namnet på klassen.

```
class MyClass {
    const CONST_VALUE = 'A constant value';
}

echo MyClass::CONST_VALUE;
```

Scope Resolution Operator (::)

- Tre speciella nyckelord, `self`, `parent` och `static`, används för att få tillgång till egenskaper eller metoder inifrån en klass.

```php
class OtherClass extends MyClass
{
    public static $my_static = 'static var';

    public static function doubleColon() {
        echo parent::CONST_VALUE . "\n";
        echo self::$my_static . "\n";
    }
}

OtherClass::doubleColon();
```

Static

- Klassvariabler är *statiska* variabler, som oftast används för sånt som är unikt för klassen men gemensamt för alla objekt av klassen.

- http://sandbox.onlinephpfunctions.com/code/d91fd0287d600a3eb55c43fddcc4a8c00693d15d

- https://www.geeksforgeeks.org/when-to-use-static-vs-instantiated-classes-in-php/

Övning

- `Sportscar` behöver medlemsvariabeln `noOfDoors`.
    - Anpassa `print()`-metoden för `Sportscar`.
- Skapa klassen `Truck` som ärver från `Vehicle` och har variabeln `loads` (hur många kubikmeter den kan lasta).
    - Hur håller vi reda på hur mycket som är lastat just nu?
    - Skapa metoden `unload()` som tar en parameter och lastar av så mycket från lasten. Kontrollera att man inte försöker lasta av mer än vad som finns.
        - Om parametern är för stor, returnera `false`.
        - Annars, returnera så mycket som har lastats av.
        - Om parametern är tom, lasta av allt och returnera hur mycket som har lastats av.

Traits

- Traits kan användas för att ge tillgång till samma metod i flera klasser.

- Om man använder en trait i en klass har den tillgång till alla variabler och metoder och tvärtom.

```php
trait MyTrait1
{
   function Hello() {
      echo 'Hello';
   }
}

trait MyTrait2
{
   function World() {
      echo 'World';
   }
}

class MyClasse1
{
   use MyTrait1;
   use MyTrait2;

   function __construct() {
      $this->Hello();
      $this->World();
   }
}

$Test = new MyClasse1;
```

Övning

- Skapa två stycken traits.
    - Den ena ska implementera metoden `booking()` som tar ett datum och ett objekt som parameter och skriva ut ett meddelande om att en bokning är skapad för besiktning för fordonet.
    - Den andra ska implementera metoden `paint()` som tar en parameter och byter färg på fordonet.
- Använd trait:sen i `Vehicle`-klassen och anropa dem från dina objekt.

Synlighet

- Synlighet styr när man kan få tillgång till variabler och metoder från utanför klassen.
  - En *public* variabel/metod är alltid tillgänglig.
  - En *private* variabel/metod är enbart tillgänglig inom klassen.
  - En *protected* variabel/metod är enbart tillgänglig inom klassen eller från en klass som ärvt ifrån den.

Synlighet

```php
class MyClass
{
    public $public = 'Public';
    protected $protected = 'Protected';
    private $private = 'Private';

    function printHello()
    {
        echo $this->public;
        echo $this->protected;
        echo $this->private;
    }
}

$obj = new MyClass();
echo $obj->public; // Works
echo $obj->protected; // Fatal Error
echo $obj->private; // Fatal Error
$obj->printHello(); // Shows Public, Protected and Private
```

Synlighet

```php
class MyClass
{
    // Declare a public constructor
    public function __construct() { }

    // Declare a public method
    public function MyPublic() { }

    // Declare a protected method
    protected function MyProtected() { }

    // Declare a private method
    private function MyPrivate() { }

    // This is public
    function Foo()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate();
    }
}

$myclass = new MyClass;
$myclass->MyPublic(); // Works
$myclass->MyProtected(); // Fatal Error
$myclass->MyPrivate(); // Fatal Error
$myclass->Foo(); // Public, Protected and Private work
```

Synlighet

```
class MyClass2 extends MyClass
{
    // This is public
    function Foo2()
    {
        $this->MyPublic();
        $this->MyProtected();
        $this->MyPrivate(); // Fatal Error
    }
}

$myclass2 = new MyClass2;
$myclass2->MyPublic(); // Works
$myclass2->Foo2(); // Public and Protected work, not Private
```

Övning

- Gör variablerna `model` och `color` privata och anpassa koden så allt funkar.

- Gör metoden `print()` protected och gör den tillgänglig.

Hur använder vi OOP?

- Lista produkter, t ex

```
class Product {
    public add($name, $description, $price) {
        // Control parameters and add product to db.
    }

    public getProducts($noOfProducts) {
        // Get the number of products as an array
    }
}

// Page:
$product_obj = new Product();
$products = $product_obj->getProducts(9);

foreach($products) {
    // Print out product.
}
```

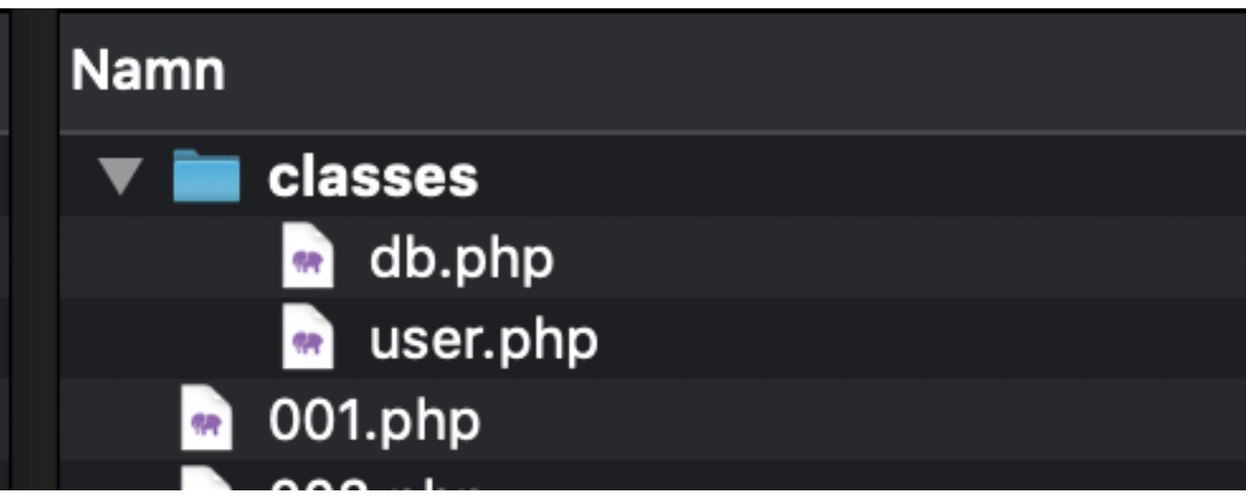
Hur använder vi OOP?

- Lista produkter, t ex

```php
<?php

class User {
    private $firstName;

    public function register($username, $password) {
        // Control parameters and register user.
    }

    public function login($username, $password) {
        // Check if user exists
    }

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

// Page:
if ($userSentForm) {
    $user = new User();

    // Psuedo...
    $user->login(filter_input($username), filter_input($password));

}
```

Hur använder vi OOP?

- Skapa en fil för varje klass och inkludera den när du behöver.




```php
// Page:
include_once "classes/user.php";

if ($userSentForm) {
    $user = new User();
```

Utvärdering

- Prata i grupper om 2-3 personer i två minuter.

- Vad har varit bra idag?

- Vad skulle kunna förbättras?