OBJECT ORIENTED PHP


PHP CLASSES

```php
class Elephant
{
  public $numberOflegs = 4;
}
```

```php
$elephant = new Elephant;
echo $elephant->numberOflegs;
```

Använder pilen för att komma åt variabler och funktioner inuti en klass

variabler direkt inuti en klass kallas `properties`

funktioner inuti en klass kallas `methods`

```php
class Elephant
{
  public $numberOflegs = 4;
  public function sayHowManyLegs()
  {
    echo "I have " . $this->numberOflegs . "legs!";
  }
}
```

```php
$elephant = new Elephant;
$elephant->sayHowManyLegs();
```

Använd klasser när du:

- Ska skapa flera objekt med likadant beteende
- När du vill gruppera funktionalitet som hör ihop
- När du upptäcker att du upprepar dig mycket
- För att "säkra upp" din kod, mindre risk för att skriva fel

```php
class Elephant
{
  public $name;

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

}
```

`$this->` refererar till det skapade objektets egenskaper

`$this->` varierar beroende på vilket objekt vi pratar om

```
$first = new Elephant("Hathi")
```

```
$second = new Elephant("Mowgli")
```

Två olika elefanter med två olika `$this->` samt två olika namn

Statiska egenskaper kan inte använda `$this`

```
class Elephant
{
  public static speak()
  {
    echo 'ERUUUEEH!';
  }
}
```

```
Elephant::speak();
```

INHERITANCE

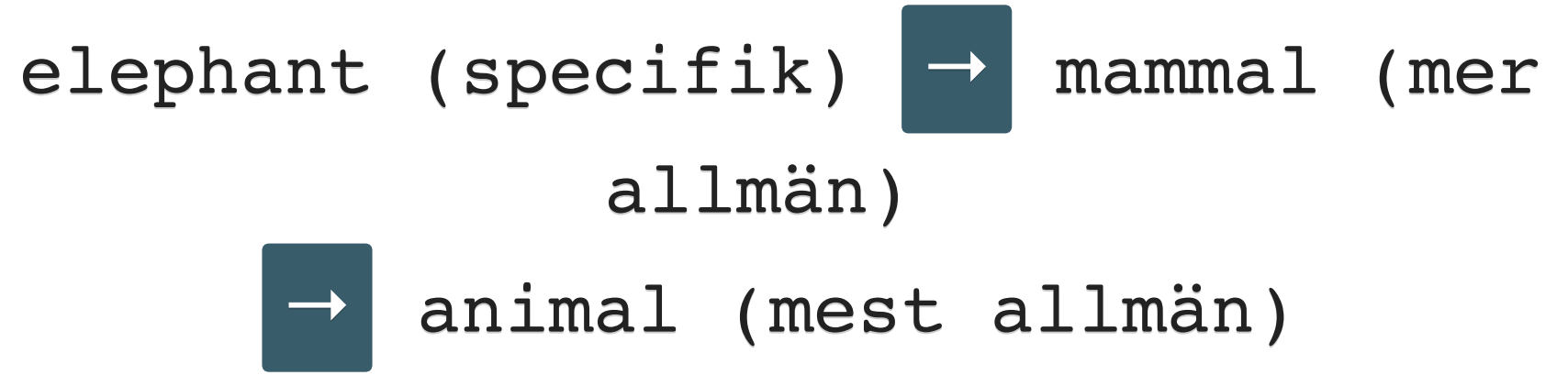
Is a thing

Arv är när en klass utgår från en annan. Om den ena klassen är ett specialfall av den andra, så är det naturligt att låta dem ärva från varandra.

`elephant (specifik)` 

`mammal (mer allmän)`

`animal (mest allmän)`



elephant ärver av mammal som ärver av animal

Allt som mammal kan kan även elephant

```
class Animal
{
  public function speak()
  {
    echo "I change shapes just to hide in this place"
  }
}
class Elephant extends Animal{}
```

Nyckelord: `extends`

OVERRIDE

```php
class Animal {
  public function makeNoise() {
    echo '...';
  }
}
class Pig extends Animal {
  public function makeNoise() {
    echo 'oink';
  }
}
class Cat extends Animal {
  public function makeNoise() {
    echo 'meow';
  }
}
```

Vi skriver över basklassens metoder

```php
class Vehicle {
  public $numberOfWheels;
}
class Car extends Vehicle {
  public function __construct() {
    $this->numberOfWheels = 4;
  }
}
```

`Vehicle` har redan numberOfWheels men denna variabel är olika beroende på vilken typ av Vehicle

Klassen Car ärver från Vehicle. Man säger ibland att Car **utökar** Vehicle.

Klassen Vehicle kallas för en **basklass** till Car.

Klassen Car kallas för en **subklass** till Vehicle.

Engelska: inherit, extend, base class, subclass.

OVERRIDE PARENT

```php
class Vehicle {
  public function goTo($destination) {
    echo "Åker fordon till $destination <br>";
  }
}
class Car extends Vehicle {
  public function goTo($destination) {
    echo "Åker bil till $destination <br>";
    parent::goTo($destination);
  }
}
```

`parent::` kallar på basklassen

PROTECTED

Vi kan även använda `protected`

Egenskapen/metoden kan kommas åt av alla subklasser

Men inte av andra klasser eller utanför klasserna

```php
class A {
  public $p1;
  protected $p2;
  private $p3;
}
class B extends A {
  public function whatWillHappen() {
    echo $this->p1;  // 1
    echo $this->p2;  // 2
    echo $this->p3;  // 3
  }
}
```

```php
class A {
  public $p1;
  protected $p2;
  private $p3;
}
class B extends A {
  public function whatWillHappen() {
    echo $this->p1;  // public! open for everyone!
    echo $this->p2;  // protected! Still open for B
    echo $this->p3;  // private! FATAL ERROR
  }
}
```


ABSTRACT
INTERFACE

ABSTRACT

Ibland är ingenting logiskt

Klassen `Animal` behöver ju egentligen aldrig implementeras

Det finns inget random `Animal`

Ibland vill man bara ha klasser som referens eller som ett kontrakt: `Abstract`

```php
abstract class Animal {
  abstract public function makeNoise();
}
class Cat extends Animal {
  public function makeNoise() {
    echo 'meow';
  }
}
class Dog extends Animal { }  //FATAL ERROR!
```

Det är upp till subklassen att implementera funktionen

INTERFACE

Liknande med `interface`: ett kontrakt

En klass behöver inte vara en subklass men ska ändå uppnå vissa kriterer och ärva vissa drag

Vi kan då implementera ett `Interface`

```
interface iCanJump {
  public function jump();
}
```

```
class Cat implements iCanJump {
  public function jump() { /**/ }  // ok
}
```

```
class Human implements iCanJump {
  public function jump() { /**/ }  // ok
}
```

ABSTRACT / INTERFACE

Det jobbiga är att vi skriver inte någon faktiskt kod

Vi säger bara att vissa klasser MÅSTE ha viss kod

Vi underlättar för andra och för oss själva: genom att skapa interfaces och abstrakta klasser definierar vi vad en viss klass kan eller ska göragöra.

Underlättar för andra att utöka och redigera koden om vi jobbar med andra