

PHP - Klasės, objektai ir OOP



Code Academy

Programuok savo ateitį!

- » Klasė - tai programuotojo sukurta duomenų struktūra, kuri gali turėti savo savybes ir metodus (vidines funkcijas).
- » Tai šablonas pagal kurį yra kuriami tos klasės objektai. Angliškoje literatūroje objektai dažniau vadinami „**instance of a class**“.

Kas yra klasė?

```
class Student
{
    private $name;
    private $surname;
    private $course;
    private $fiedOfStudy;
    private $modules;

    public function getFullName()
    {
        return $this->name . ' ' . $this->surname;
    }
}
```

- » \$name, \$surname, ... \$modules yra klasės (studento šablono) savybės.
- » getFullName – metodas.
- » Funkcijos aprašytos klasės viduje yra vadinamos metodais. Metodai dažniausiai skirti manipuluoti toks klasės objekto savybėmis. Metodai kviečiami iš klasės objekto.
- » Raktažodis **\$this** traktuojamas, kaip objektas iškvietęs metodą.

O kas yra objektas?

- » Objektas tai konkretus egzempliorius sumodeliuotas pagal klasės struktūrą. Objektas tai klasės realizacija (***angl. instance of a class***).
- » Norint sukurti klasės objektą, reikalingas konstruktorius klasėje, kuriame aprašome veiksmų seką objektui sukurti. Dažnai konstruktavimo metu reikalaujami pradiniai parametrai.
- » Klasė su objektu susijusi taip pat kaip smuiko šablonas susijęs su konkrečiu smuiku.
- » Pagal šią analogiją, smuiko gaminimo procesas būtų atliekamas konstruktoriuje, o medžiagos paduodamos per konstruktoriaus parametrus.

```
class Student
{
    // ... savybės ...
    private $name;
    private $surname;
    private $course;
    private $fieldOfStudy;
    private $modules;

    // ... konstruktorius ...
    public function __construct($name, $surname, $fieldOfStudy)
    {
        $this->name = $name;
        $this->surname = $surname;
        $this->fieldOfStudy = $fieldOfStudy;
        $this->modules = [];
        $this->course = 1;
    }

    // ... metodai ...
    public function getFullName()
    {
        return $this->name . ' ' . $this->surname;
    }
}
```



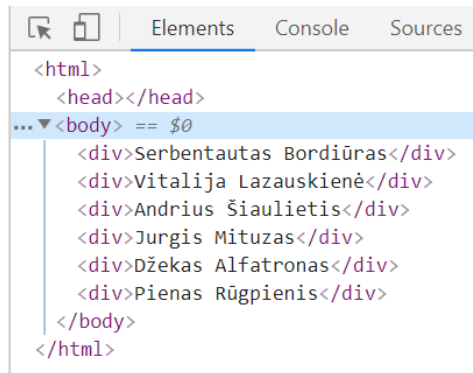
Code Academy

Programuok savo ateitį!

```
$studentoObjektas1 = new Student(  
    'Serbentautas',  
    'Bordiūras',  
    'Krūmų apkarpymo inžinerija');  
$studentoObjektas2 = new Student('Vitalija', 'Lazauskienė', 'Žurnalistika');  
$studentoObjektas3 = new Student('Andrius', 'Šiaulietis', 'Šonaslydis');  
  
$studentai = [  
    new Student('Jurgis', 'Mituzas', 'Teologija'),  
    new Student('Džekas', 'Alfatronas', 'Vadyba'),  
    new Student('Pienas', 'Rūgpienis', 'Pramoninė pienininkystė'),  
];
```

```
echo '<div>'.$studentoObjektas1->getFullName().'</div>';  
echo '<div>'.$studentoObjektas2->getFullName().'</div>';  
echo '<div>'.$studentoObjektas3->getFullName().'</div>';  
  
foreach ($studentai as $studentas) {  
    echo '<div>'.$studentas->getFullName().'</div>';  
}
```

Serbentautas Bordiūras
Vitalija Lazauskienė
Andrius Šiaulietis
Jurgis Mituzas
Džekas Alfatronas
Pienas Rūgpienis



4 pagrindinės Obejktiškai Orientuoto Programavimo savybės

- » **Paveldimumas** - tai ryšys tarp klasių, kuris papildo ar apibrėžia jau esamą klasės struktūrą. Šis ryšys turi pasižymėti logika: klasė **A** yra klasė **B**, pvz. : `class Pušis extends Medis.`
- » **Inkapsuliacija** – tai savybių ar metodų pasiekiamumo ribojimas.
- » **Abstrakcija** – tai principas kuomet tėvinėje klasėje aprašyti suvaržymai įpareigoja ją paveldinčias klases realizuoti tam tikrą funkcionalumą.
- » **Polimorfizmas** – tai daugiakūniškumas, daugialypiškumas. To pačio rezultato formavimas naudojant labai panašią, ar identišką išraišką.

- » **Paveldimumas** – tai ryšys tarp klasių, kur viena klasė traktuojama vaicine, o kita tėvine. Vaikinė klasė preplečia tėvinę. Arba kitaip tariant, vaikinė klasė paveldi tėvinės klasės savybes bei metodus.
- » Klasė gali paveldėti tik vieną tėvinę klasę. Tačiau klasė gali turėti daug ją paveldinčių klasių (vaikų).
- » **Paveldėjimo tikslas** - aprašyti bendras vaikinių klasių savybes ir metodus tėvinėje klasėje. Taip supaprastinant struktūrą ir įgalinant kitas OOP savybes (polimorfizmas, abstrakcija). Tuomet kodas tampa švaresnis ir perpanaudojamas.

```
class LivingThing
{
    private $birthDate;
}

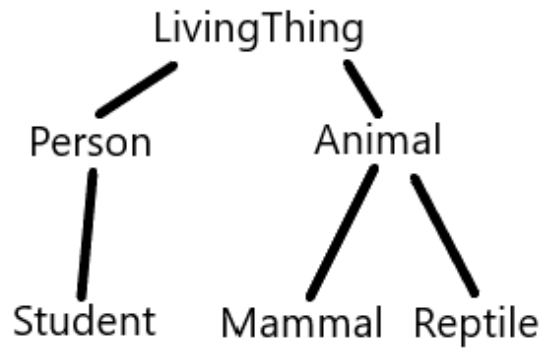
class Person extends LivingThing
{
    private $fullName;
}

class Student extends Person
{
    private $university;
}
```

```
class Animal extends LivingThing
{
    private $kind;
}

class Mammal extends Animal
{
    private $vegetarian;
}

class Reptile extends Animal
{
    private $hasLegs;
}
```



Paveldimumas - pastebėjimai

- » Svarbu pastebėti, jog tėvinė klasė neturi informacijos apie ją paveldinčias vaikinės klases.
- » Jeigu klasė praplečia aukštesnę klasę, o toji praplečia dar aukštesnę, tai pati „žemiausioji“ praplečia jas abi (paveikslukas praeitoje skaidrėje).
- » Prisiminkime, jog paveldimumas turi būti įgalinamas tik tuomet, kuomet galime teigti:
 - » Vaikinė klasė **YRA** tėvinė klasė
 - » Vaikinė klasė **PRAPLĖČIA** tėvinės klasės apibrėžimą.

Paveldimumas - paveldimumu susijusių objektų konstravimas

```
class LivingThing
{
    private $birthDate;
}
```

Kuriant naują objektą pagal klasę, kuri paveldi kitą klasę automatiškai yra priskiriamos ir tėvo savybės, bei metodai.

```
class Person extends LivingThing
{
    private $name;
    private $surname;
}
```

```
var_dump(new Person);
```

```
object(Person)[7]
  private 'name' => null
  private 'surname' => null
  private 'birthDate' (LivingThing) => null
```

Paveldimumas - paveldimumu susijusių objektų konstravimas

- » Dažnai (beveik visada) kuriant naują objektą norime įvykdyti pradinis objekto konstravimo veiksmus. Būtent tam ir yra reikalingas konstruktorius.
- » Įgalinant pradinis konstravimo veiksmus objektui, dažniausiai norime įgalinti ir tėvinės klasės konstruktoriaus veiksmus, jeigu yra aprašytas paveldimumas.
- » Kreiptis į betkurį tėvinės klasės metodą, įskaitant konstruktorių galime taip:

```
parent::__construct();
```

```
class Person extends LivingThing
{
    private $name;
    private $surname;

    public function __construct($name, $surname, $birthDate) {
        parent::__construct($birthDate);
        $this->name = $name;
        $this->surname = $surname;
    }
}
```

```
class LivingThing
{
    private $birthDate;

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

```
$person1 = new Person('Stalčius', 'Komodauskas', '1991-07-26');
var_dump($person1);
```

```
object(Person)[7]
  private 'name' => string 'Stalčius' (length=9)
  private 'surname' => string 'Komodauskas' (length=11)
  private 'birthDate' (LivingThing) => string '1991-07-26' (length=10)
```



Code Academy

Programuok savo ateitį!

- » **Inkapsuliacija** – tai objekto savybių tiesioginio prieinamumo apribojimas. Yra 3 pasiekiamumo tipai :
 - » **Private** - savybė pasiekama tik klasės viduje.
 - » **Protected** – savybė pasiekama klasės viduje ir iš paveldinčių klasių.
 - » **Public** – savybė pasiekama iš klasės viduje, paveldinčių klasių ir klasės išorėje.
- » **Inkapsuliacijos tikslai:**
 - » **Savybių korektiškumo valdymas:** metai: -12, vardas: 74.
 - » **Autorizacija:** paprastas vartotojas bando gauti “jautrius“ duomenis.
 - » **Lygiagrečių procesų prieinamumo valdymas:** banko sąskaitos naudojimas.
 - » **Papildomi veiksmai:** Keičiant trikampio kraštinę, turi keistis ir kiti geomet. duomenys.
 - » **Kodo švarinimui:** Kitas programuotojas turi matyti tik naudingas ir reikalingas savybes.

```
class LivingThing
{
    private $birthDate;

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

    public function setBirthDate($birthDate){
        // ... Apsaugų realizavimas ...
        $this->birthDate = $birthDate;
    }

    public function getBirthDate(){
        // ... Apsaugų realizavimas ...
        return $this->birthDate;
    }
}
```

Visų savybių apsaugos realizuojamos kuriant set'erus ir get'erus.



Code Academy

Programuok savo ateitį!


```
$livingThing = new LivingThing('2001-05-24');  
var_dump($livingThing);  
$forLaterUsage = $livingThing->getBirthDate();  
$livingThing->setBirthDate('2003-05-24');  
var_dump($livingThing);
```

C:\xampp\htdocs\code_ex\index.php:7:

```
object(LivingThing)[1]  
  private 'birthDate' => string '2001-05-24' (length=10)
```

C:\xampp\htdocs\code_ex\index.php:10:

```
object(LivingThing)[1]  
  private 'birthDate' => string '2003-05-24' (length=10)
```



Code Academy

Programuok savo ateitį!

Abstrakcija

- » Abstrakčios klasės yra skirtos įpareigoti savo vaikinės klases. Įpareigojimas reikalingas norint užtikrinti funkcionalumą.
- » Tai daroma abstrakčioje klasėje parašius metodo antraštę.
- » Tuomet abstrakčią klasę paveldinčios vaikinės klasės privalo:
 - » arba aprašyti (*angl.* implementuoti) metodą.
 - » arba perdeklaruoti, taip įpareigojimą perkeliant jau savo vaikinėms klasėms.
- » Tokiu būdu užtikrinama, jog visos abstrakčios klasės vaikinių klasių medis turės aprašytą metodą.

- » **Polimorfizmas** – tai daugiakūniškumas, daugialypiškumas. To pačio rezultato formavimas naudojant labai panašią, ar identišką išraišką.
- » Polimorfizmas pasireiškia kuomet perrašome (override) ar perkrauname (overload) funkcijas/metodus.
- » **Override** – perrašyti funkcijos/metodo logiką su identiškais įeinamaisiais parametrais.
- » **Overload** – papildyti funkcijos/metodo logiką, kuomet skiriasi parametų kiekis ir/arba jų tipai.

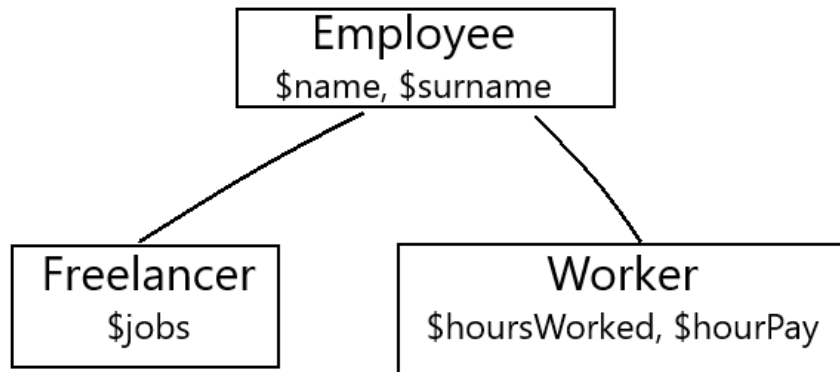
Polimorfizmo ir Abstrakcijos pavyzdys

```
abstract class Employee
{
    protected $name;
    protected $surname;

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

    public function getFullName()
    {
        return $this->name . ' ' . $this->surname;
    }

    abstract public function withdrawSalary();
}
```





```
class Freelancer extends Employee
{
    private $jobs;

    public function __construct($name, $surname)
    {
        parent::__construct($name, $surname);
        $this->jobs = [];
    }

    public function assignJob($job)
    {
        if ($job instanceof Job) array_push($this->jobs, $job);
        else echo '<h3 style="color: red">This is not a real job!</h3>';
    }

    public function finishJob($jobId)
    {
        foreach ($this->jobs as $job) {
            if($job->getId() == $jobId) $job->finishJob();
        }
    }

    public function withdrawSalary()
    {
        $sum = 0.0;
        foreach ($this->jobs as $index => $job) {
            if ($job->done()) {
                $sum += $job->getAmount();
                array_splice($this->jobs, $index, 1);
            }
        }
        return $sum;
    }
}
```

```
class Job{
    private $id;
    private $title;
    private $amount;
    private $done;

    public function __construct($title, $amount)
    {
        $this->title = $title;
        $this->amount = $amount;
        $this->done = false;
    }

    public function getId() {
        return $this->id;
    }

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

    public function getAmount() {
        return $this->amount;
    }

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

    public function finishJob() {
        $this->done = true;
    }
}
```

```

class Worker extends Employee{
    private $hoursWorked;
    private $hourPay;

    public function __construct($name, $surname, $hourPay)
    {
        parent::__construct($name, $surname);
        $this->hoursWorked = 0;
        $this->hourPay = $hourPay;
    }

    public function work($hours)
    {
        $this->hoursWorked += $hours;
    }

    public function withdrawSalary()
    {
        $amount = $this->hoursWorked * $this->hourPay;
        $this->hoursWorked = 0;
        return $amount;
    }
}

```

```

include 'Employee.php';
include 'Worker.php';
include 'Job.php';
include 'Freelancer.php';

// darbai freeLanceriams
$jobs = [
    new Job('Footer layout', 200), // 0
    new Job('Contacts Ajax CRUD', 600), // 1
    new Job('Authorization for client page', 700), // 2
    new Job('Authentication for moderator controllers', 400), // 3
    new Job('Payment form', 100), // 4
    new Job('.NET API for Payment actions', 1200), // 5
    new Job('Cyber security for register page', 400) // 6
];

// visi darbuotojai
$employees = [
    new Worker('Šriftas', 'Raidenis', 10), // 0
    new Worker('Stilija', 'Cė Esesauskaitė', 8), // 1
    new Freelancer('Eventas', 'Klikauskas'), // 2
    new Freelancer('Apas', 'Rekvestenis'), // 3
    new Worker('Lentėja', 'Bazienė', 10) // 4
];

```

Darbų priskyrimas, atlikimas ir spausdinimas

```
// freelanciam priskiriami darbai
$employees[2]->assignJob($jobs[0]);
$employees[2]->assignJob($jobs[1]);
$employees[2]->assignJob($jobs[4]);

$employees[3]->assignJob($jobs[2]);
$employees[3]->assignJob($jobs[3]);
$employees[3]->assignJob($jobs[5]);
$employees[3]->assignJob($jobs[6]);

// freelanceriai baigia darbus
$employees[2]->finishJob(0);
$employees[2]->finishJob(1);
$employees[2]->finishJob(4);

$employees[3]->finishJob(2);
$employees[3]->finishJob(3);
$employees[3]->finishJob(6);
```

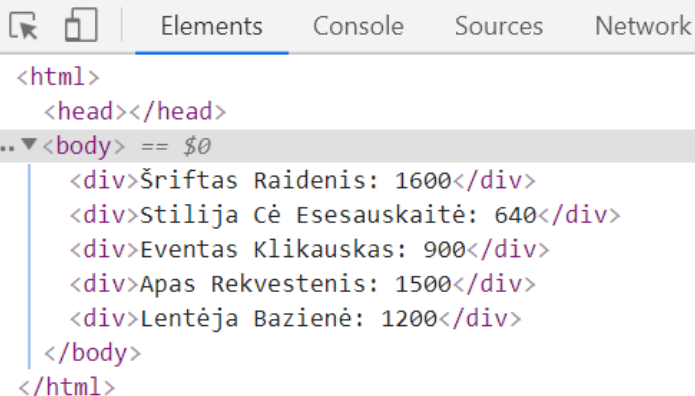
```
// darbuotojai išdirba valandas
```

```
$employees[0]->work(160);
$employees[1]->work(80);
$employees[4]->work(120);
```

```
// spausdinami visi darbuotojų atlyginimai
foreach ($employees as $employee) {
    $fullName = $employee->getFullName();
    $salary = $employee->withdrawSalary();
    echo "<div>$fullName: $salary</div>";
}
```

Rezultatai

Šriftas Raidenis: 1600
Stilija Cė Esesauskaitė: 640
Eventas Klikauskas: 900
Apas Rekvestenis: 1500
Lentėja Bazienė: 1200



The screenshot shows the 'Elements' tab of a browser's developer tools. The HTML structure is as follows:

```
<html>
  <head></head>
  <body> == $0
    <div>Šriftas Raidenis: 1600</div>
    <div>Stilija Cė Esesauskaitė: 640</div>
    <div>Eventas Klikauskas: 900</div>
    <div>Apas Rekvestenis: 1500</div>
    <div>Lentėja Bazienė: 1200</div>
  </body>
</html>
```