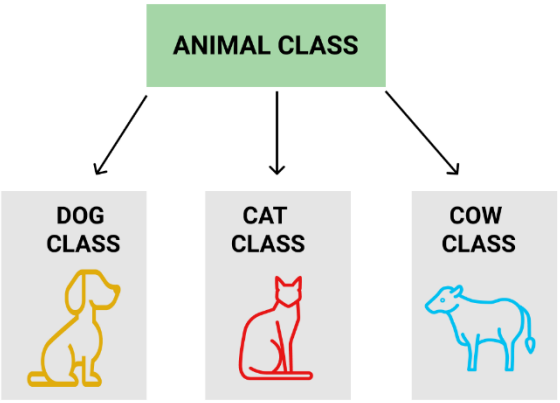
CAP. 10
MOȘTENIREA ȘI MODIFICAREA ACCESULUI

10.1 Moștenirea

Pentru a reduce redundanța codului, o clasă se poate construi pornind de la o clasă deja existentă preluând unele sau chiar toate caracteristicile clasei principale. Această proprietate se numește moștenire și, utilizată corect, poate reduce timpul necesar scrierii codului; codul devenind astfel și mai concis.



Moștenirea este unul din cele mai utile instrumente ale Programării Orientate pe Obiect - OOP. Prin moștenire se înțelege transmiterea proprietăților, constantelor și a funcțiilor de la o clasă la alta, într-o structura ierarhică. Prima clasă este clasa de bază, denumită "părinte", iar pornind de la aceasta se poate crea o sub-clasă, denumită "copil". Sub-clasa "copil" moștenește proprietățile și metodele clasei "părinte" (cele care sunt cu atribut "**public**" sau "**protected**" – mai multe detalii în cadrul acestui curs), pe care le poate folosi în instrucțiunile din propriul cod și le transmite când se creează o instanță de obiect din ea.

La definirea unei clase, aceasta poate, opțional, moșteni o clasă părinte definită anterior folosind cuvântul cheie **extends**. Sintaxa este următoarea:

```
class Copil extends Parinte {
    // corp clasa Copil
}
```

10.1.1 Exemplu de moștenire

În exemplul următor clasa **Fruit** este extinsă cu ajutorul clasei **Strawberry**. Noua clasă va conține atât proprietățile **name** și **color** cât și metodele **__construct** și **intro**. În plus aceasta implementează o nouă metodă numită **message**.

```
class Fruit
{
    public $name;
    public $color;

    public function __construct($name, $color)
    {
        $this->name = $name;
        $this->color = $color;
    }
    public function intro()
```

```
    {
        echo "Acest fruct este {$this->name} și are culoarea {$this-
>color}.";
    }
}

class Strawberry extends Fruit
{
    public function message()
    {
        echo "Sunt o căpșuna.";
    }
}

$strawberry = new Strawberry("Strawberry", "red");
$strawberry->message();
$strawberry->intro();
```

10.1.2 Suprascrierea metodelor

Metodele definite în clasa copil pot rescrie definirea metodelor cu același nume din clasa părinte. În următorul exemplu metodele **__construct** și **intro** din clasa părinte sunt suprascrise.

```
class Fruit
{
    public $name;
    public $color;

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

    public function intro()
    {
        echo "Acest fruct este {$this->name} și are culoarea {$this-
>color}.";
    }
}

class Strawberry extends Fruit
{
    public $weight;

    public function __construct($name, $color, $weight)
    {
        $this->name = $name;
        $this->color = $color;
```

```
        $this->weight = $weight;
    }

    public function intro()
    {
        echo "Acest fruct este {$this->name}, are culoarea {$this-
>color} și o greutate de {$this->weight} grame.";
    }
}

$strawberry = new Strawberry("Strawberry", "red", 50);
$strawberry->intro();
```

10.1.3 Apelarea constructorului părinte

În loc de a defini un constructor nou pentru clasele copil, se poate scrie unul prin apelarea constructorului părinte. Astfel se reduce redundanța codului și se crește lizibilitatea acestuia. Apelarea constructorului părinte se face cu ajutorul operatorului **parent**.

```
class Strawberry extends Fruit
{
    public $weight;

    public function __construct($name, $color, $weight)
    {
        // apelarea constructorului parinte
        parent::__construct($name, $color);
        $this->weight = $weight;
    }

    public function intro()
    {
        echo "Acest fruct este {$this->name}, are culoarea {$this-
>color} și o greutate de {$this->weight} grame.";
    }
}
```

Observație: În cazul suprascrierii metodelor din clasa părinte, variantele metodelor din clasa părinte pot fi accesate în cadrul clasei copil cu ajutorul expresiei:

***parent*::nume_metoda(**argumente**)**.

10.1.4 Cuvântul cheie final

Începând cu PHP 5 s-a introdus în limbaj cuvântul cheie **final**, ce împiedică clasele copil să suprascrie metodele din clasa părinte precedate de acest cuvânt.

```php
class Fruit
{
    final public function intro()
    {
        // cod
    }
}

class Strawberry extends Fruit
{
    // generează eroare
    public function intro()
    {
        // cod
    }
}
```

Observație: Dacă clasa însăși este definită ca și **final** atunci aceasta nu poate fi extinsă.

```php
final class Fruit
{
    // cod
}

// generează eroare
class Strawberry extends Fruit
{
    // cod
}
```

10.2 Modificatorii de acces

Modificatorii de acces ne permit să controlăm accesul sau vizibilitate pentru proprietăți și metode.

Toate exemplele de până acum au folosit proprietăți și metode cu modificatorul de acces public, dar în PHP există trei modificatori de acces:

- **public** – proprietatea sau metoda poate fi accesată de oriunde;
- **protected** – proprietatea sau metoda poate fi accesată în interiorul clasei sau în clasele care moștenesc direct respectiva clasă;
- **private** – proprietatea sau metoda poate fi accesată doar în interiorul clasei.

În exemplul următor se poate observa efectul modificatorilor de acces asupra proprietăților clasei **Fruit**. Doar proprietatea cu modificatorul public poate fi accesată din exteriorul clasei (după instanțiere).

```
class Fruit
{
    public $name;
    protected $color;
    private $weight;
}

$mango = new Fruit();
$mango->name = 'Mango';   // OK
$mango->color = 'Yellow'; // ERROR
$mango->weight = '300';   // ERROR
```

Observație: Până în momentul în care se specifică explicit, proprietățile și metodele unei clase, sunt publice.

10.2.1 Modificatorul private

De cele mai multe ori utilizarea modificatorului public este calea cea mai rapidă de a implementa o clasă ce conține proprietăți și metode, reducând de cele mai multe ori numărul de linii de cod necesare. Cu toate acestea există situații în care dorim să prevenim accesul la o proprietate din afara clasei. Acest lucru poate fi realizat cu ajutorul cuvântului cheie **private**. În plus, metodele getter / setter (prezentate în detaliu în următorul capitol) reprezintă singura modalitate de a accesa proprietatea. Și metodele getter / setter pot oferi o logică personalizată pentru a manipula valoarea proprietății.

De exemplu, dacă doriți ca valoarea proprietății **$name** să nu fie goală, puteți adăuga logica de validare în metoda **setName()**, după cum urmează:

```
class Customer
{
    private $name;

    public function setName($name)
    {
        $name = trim($name);

        if ($name == '') {
            return false;
        }
        $this->name = $name;

        return true;
    }

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

```
}

$customer = new Customer();

$customer->setName(' Bob ');
echo $customer->getName();
```

Când clasa **Customer** este moștenită de altă clasă folosind **extends**, funcțiile **getName** și **setName** vor fi vizibile, în schimb proprietatea **$name** nu va putea fi modificată direct.

10.2.2 Modificatorul protected

La fel ca modificatorul de acces **private**, modificatorul de acces **protected** restricționează accesul la variabilele și metodele unei clase în afara acesteia. Dar variabilele și funcțiile clasei protejate pot fi accesate în interiorul clasei și în interiorul subclasei (o clasă care moștenește clasa).

```
class Customer
{
    protected $name;

    public function setName($name)
    {
        $name = trim($name);

        if ($name == '') {
            return false;
        }
        $this->name = $name;

        return true;
    }

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

class Subscriber extends Customer
{
    public function toString()
    {
        $presentation = "Numele abonatului este:" . $this->name;
        return $presentation;
    }
}
$subscriber = new Subscriber();

$subscriber->setName('Tom');
echo $subscriber->getName();
echo "<br>";
echo $subscriber->toString();
```

10.3 Metode importante pentru clasele PHP (magic methods)

Metodele magice PHP sunt metode speciale ce sunt disponibile în orice clasă. Metodele magice suprascriu acțiunile implicite atunci când obiectul efectuează acțiunile.

Prin convenție, numele metodelor magice încep cu o linie de subliniere dublă "__" (underscore). PHP rezervă numele care încep cu o linie de subliniere dublă "__" pentru metodele magice.

Până acum, ați întâlnit două metode magice **constructorul** și **destructorul**, ce folosesc metodele **__construct()** și **__destruct()**. Metoda **__construct()** este invocată automat când un obiect este creat iar **__destruct()** este apelată când obiectul este șters.

Pe lângă metodele __contruct() și __destruct(), PHP are și următoarele metode magice:

- __get() – folosit când se citește o proprietate inexistentă sau inaccesibilă;
- __set() – folosit când se scrie o proprietate inexistentă sau inaccesibilă;
- __toString() – folosit când un obiect este tratat ca string.

10.3.1 Metodele de tip getter și setter

Când încercați să scrieți într-o proprietate inexistentă sau inaccesibilă, PHP apelează automat metoda **__set()**. În cazul în care doriți să suprascrieți metoda __set() sintaxa acesteia este următoarea:

public function __set (*string* **$nume**, *mixed* **$valoare**): *void*

Metoda __set() acceptă numele și valoarea proprietății în care scrieți.

Când încercați să accesați o proprietate care nu există sau o proprietate care este inaccesibilă, de exemplu, proprietate privată sau protejată, PHP apelează automat metoda **__get()**. Metoda __get() acceptă un argument care este numele proprietății pe care doriți să o accesați:

public function __get (*string* **$nume**): *mixt*

Observație: În cazul în care folosiți proprietăți împreună cu modificatorii de acces **private**, respectiv **protected**, și doriți să oferiți dezvoltatorului acces controlat asupra acestora, cu un nivel de precizie mult mai ridicat, se recomandă utilizarea metodelor de tip getter și setter specifice fiecărei proprietăți în parte. Numele acestora va fi precedat mereu de cuvintele **get**, respectiv **set** sub forma: **getNumeProprietate()**, respectiv **setNumePropritate**($valoare).

Următorul exemplu ilustrează cum trebuie să utilizați metodele **__set()** și **__get()**:

```
class HtmlElement
{
    private $attributes = [];
    private $tag;

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

    public function __set($name, $value)
```

```
        {
            $this->attributes[$name] = $value;
        }

        public function __get($name)
        {
            if (array_key_exists($name, $this->attributes)) {
                return $this->attributes[$name];
            }
        }
    }
```

10.3.2 Metoda toString

O altă metodă magică din PHP este __toString(). Sintaxa utilizată în cazul metodei este următoarea:

public function __toString ( ): *string*

Metoda __toString() nu acceptă niciun parametru și returnează un șir de caractere.

Când utilizați un obiect ca un șir, PHP va apela automat metoda magică __toString(). Dacă metoda nu există, PHP generează o eroare.

Următorul exemplu definește clasa BankAccount, creează o nouă instanță a BankAccount și o afișează:

```
class BankAccount
{
    private $accountNumber;
    private $balance;

    public function __construct(
        $accountNumber,
        $balance
    ) {
        $this->accountNumber = $accountNumber;
        $this->balance = $balance;
    }

    public function __toString()
    {
        return "Cont: $this->accountNumber. Sold: $$this->balance";
    }
}

$account = new BankAccount('123456789', 100);
echo $account;
```

Pentru a utiliza obiectul $account ca șir, trebuie să implementați metoda __toString() care returnează reprezentarea sub formă de șir de caractere a obiectului de tip BankAccount. În cazul în care această metodă lipsește ultima linie de cod din exemplul anterior va genera o eroare (experimentați acest lucru – ștergeți metoda din definirea clasei BankAccount și observați efectul).

Teme

1. Creați o clasă **Produs** cu două proprietăți: **numeProdus** și **pret**. Folosiți conceptul de încapsulare. Creați o funcție **toString()** ce afișează informații pentru obiectele **Produs** sub forma: "Produs: **$numeProdus**, pret: **$pret**.".
2. Creați o clasă **Telefon** ce moștenește clasa **Produs**. Aceasta va avea suplimentar alte două proprietăți: **descriere** și **disponibilitate** (cu valorile: disponibil sau indisponibil). Folosiți conceptul de încapsulare pentru clasa nou creată. Creați o funcție **toString** care suprascrie funcția moștenită din **Produs** și afișează următoarea informație: "Produsul: **$numeProdus**, cu următoarea descriere: **$descriere**, având prețul: **$pret**, este: disponibil/indisponibil.".
3. Creați 2 obiecte **Produs** și 2 obiecte **Telefon**. Afișați informațiile fiecărui produs folosind funcțiile **toString**().