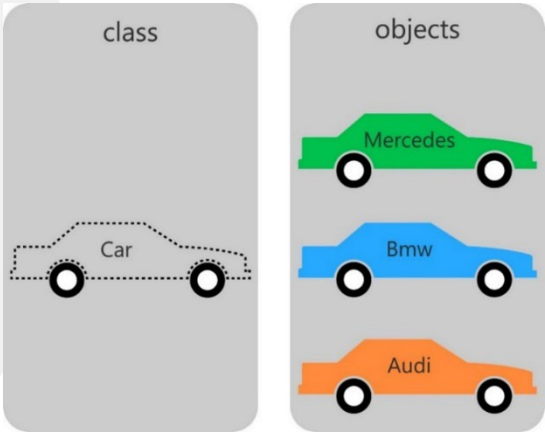
CAP. 9
INTRODUCERE ÎN OOP

OOP este abrevierea de la **Object Oriented Programming** (Programare Orientată pe Obiecte). OOP este un mod de programare în care atenția este mai mult concentrată asupra obiectelor (datelor) pe care dorim să le manipulăm decât pe logica necesară pentru manipularea lor. În schimb în programarea procedurală un program este văzut ca o succesiune logică de instrucțiuni care, pornind de un set de date de intrare, le prelucrează și furnizează rezultate (date de ieșire).

Important la acest concept este să fie înțeleasă bine diferența dintre **clasă (class)** și **obiect (object)**:

- Clasa este o structură de programare, care conține variabile (numite proprietăți) folosite pentru a stoca informații (în cadrul obiectelor) și funcții (numite metode) prin care se pot crea diferite funcționalități bazate pe informațiile stocate;
- Obiectele sunt structuri de date create cu ajutorul unei clase, și prin care se alocă date concrete pentru proprietăți și care particularizează astfel și răspunsul generat de metodele din acea clasă.

Pentru a înțelege mai bine conceptul, prin analogie, o clasă poate fi asemănată cu planul / schița unei case, iar obiectul este însăși casa ce conține valori concrete pentru proprietățile (culoare, ferestre, etc.) definite în schiță. Așa cum putem folosi același plan pentru a construi mai multe case, așa și în OOP, folosind aceeași clasă se pot crea unul sau mai multe obiecte. Conceptul poate fi aplicat pentru orice tip de obiect (vezi Figura alăturată).



Avantajele OOP

Programarea orientată pe obiecte are mai multe avantaje față de programarea procedurală:

- OOP este mai rapidă și mai ușor de executat;
- OOP oferă o structură clară pentru programe;
- Conceptul de clasă ne permite să definim orice structură de date de care avem nevoie și care nu există definită în limbaj;
- OOP ajută la menținerea codului PHP DRY „Don't Repeat Yourself" și face codul mai ușor de întreținut, modificat și depanat;
- OOP face posibilă crearea de aplicații complet reutilizabile cu mai puțin cod și într-un timp de dezvoltare mai scurt.

Observație: Principiul „nu te repeta" (DRY – Don't Repeat Yourself) se referă la reducerea repetiției codului. Ar trebui să extrageți secvențele de cod care se regăsesc în mai multe locații din aplicație și să le plasați într-un singur loc și să le refolosiți în loc să le repetați.

Observație: Un alt concept ce ar trebui respectat în cazul POO este principiul "păstrează-l simplu" sau KISS (Keep it Simple Stupid). Nu folosiți funcționalități OOP doar fiindcă puteți. Folosiți funcționalități moderne deoarece acestea aduc beneficii în încercarea de a rezolva probleme specifice.

9.1 Concepte OOP

În cadrul programării orientate pe obiecte există o serie de termeni importanți pe care îi veți întâlni:

- **Clasă** - Acesta este un tip de date definit de programator, care include funcții locale, precum și date locale. Vă puteți gândi la o clasă ca la un șablon pentru realizarea mai multor instanțe de același tip.
- **Obiect** - O instanță individuală a structurii de date definită de o clasă. Definiți o clasă o dată și apoi faceți multe obiecte care îi aparțin. Obiectele sunt cunoscute și ca instanțe.
- **Proprietate** - Aceasta este o variabilă definită în interiorul unei clase. Poate fi invizibilă în exteriorul clasei și poate fi accesată prin intermediul metodelor. Acest tip de variabilă este numită proprietate a obiectului odată ce un obiect este creat.
- **Metodă** – Aceasta este o funcție definită în interiorul unei clase și este utilizată pentru a accesa datele obiectului.
- **Moștenire** - Când o clasă este definită prin referință către proprietățile și metodele existente într-o clasă părinte, acest concept se numește moștenire.
- **Clasa părinte** - O clasă care este moștenită de o altă clasă. Aceasta se mai numește și clasă de bază sau super clasă.
- **Clasă copil** - O clasă care moștenește de la o altă clasă. Aceasta se mai numește și o subclasă sau o clasă derivată.
- **Polimorfism** - Acesta este un concept orientat pe obiect în care aceeași funcție poate fi utilizată în scopuri diferite. De exemplu, numele funcției va rămâne același, dar necesită un număr diferit de argumente și poate îndeplini sarcini diferite.
- **Supraîncărcare** - Un tip de polimorfism în care unii sau toți operatori au implementări diferite în funcție de tipurile argumentelor lor. În mod similar, funcțiile pot fi supraîncărcate cu implementări diferite.
- **Abstracția datelor** - Orice reprezentare a datelor în care detaliile implementării sunt ascunse (abstrase).
- **Încapsulare** - Se referă la un concept în care utilizăm proprietățile și metodele clasei fără a fi necesară cunoașterea structurii ei.
- **Constructor** - Se referă la un tip special de funcție care va fi apelată automat ori de câte ori se creează un obiect dintr-o clasă.
- **Destructor** - Se referă la un tip special de funcție care va fi apelată automat ori de câte ori un obiect este șters sau iese din domeniul de aplicare.

9.2 Definirea unei clase

Definirea unei clase începe cu un cuvântul cheie "**class**", urmat de numele clasei și corpul ei, cuprins între acolade "**{ … }**". În corpul clasei se definesc proprietățile și metodele ei. Proprietățile sunt variabile definite în interiorul clasei, iar metodele sunt funcții create în interiorul clasei.

Structura generală a unei clase este următoarea:

```
class NumeClasa {
        // cod
}
```

Observație: Numele clasei poate conține orice combinație de litere și cifre, caracterul underscore "_" , dar nu poate începe cu o cifră. Modul în care este scris NumeClasa este cunoscut sub numele de **studly caps**, prima literă din fiecare cuvânt începe cu literă mare. De asemenea numele clasei este un substantiv ce descrie un obiect, de exemplu (Car, CarEngine, StudentDetails)

Un exemplu concret de definire a unei clase în PHP este prezentat în cele ce urmează.

```
class ClasaPhp {
    public $var1;
    public $var2 = "un string";

    function functiaMea () {
       return $this->var1 . $this->var2;
    }
}
```

Declararea proprietăților poate începe cu folosirea cuvântului cheie **var** sau a modificatorului de acces (**public** în acest caz). Mai multe informații privind modificatorii de acces vor fi prezentate în cursul următor.

Declararea metodei este similară cu o funcție standard, diferența constând în faptul că poate accesa proprietățile obiectului cu ajutorul referinței realizată cu ajutorul cuvântului cheie **$this**. Împreună cu **$this**, numele proprietății devine o singură variabilă.

Mai jos este dat un exemplu în care este creată o clasă **Fruit** cu 2 proprietăți **name** și **color**, respectiv 4 metode folosite pentru accesarea și setarea valorilor proprietăților.

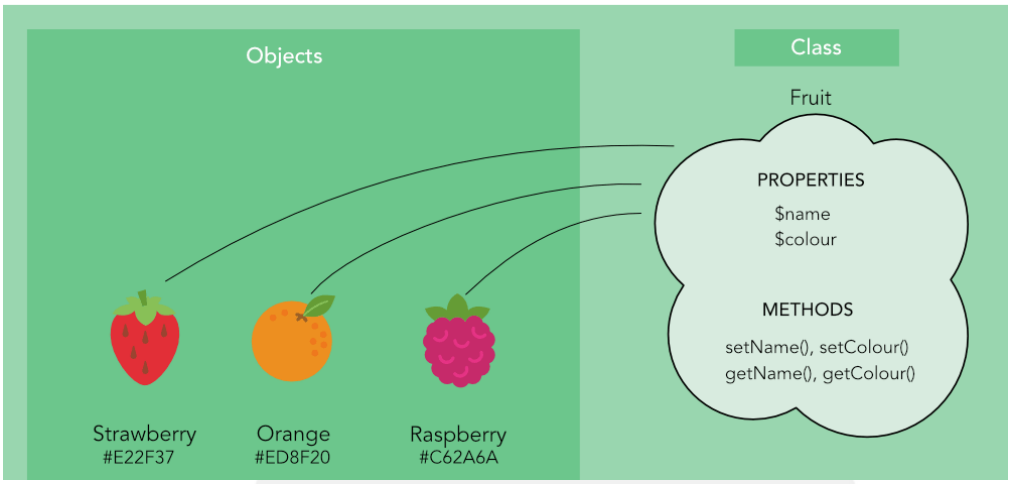
```
class Fruit
{
    // Proprietăți
    public $name;
    public $color;

    // Metode
    function setName($name)
    {
        $this->name = $name;
    }
    function getName()
    {
        return $this->name;
    }
    function setColor($color)
    {
        $this->color = $color;
    }
    function getColor()
    {
        return $this->color;
    }
}
```

9.2.1 Crearea unui obiect

După definirea unei clase, un nou obiect se poate instanția (crea) și stoca într-o variabilă folosind cuvântul cheie "**new**". Odată definită clasa, putem crea câte obiecte dorim plecând de la aceasta.

În Figura 9.2 avem un exemplu în care sunt create trei obiecte. Aceste obiecte sunt independente și au propriile seturi de proprietăți.



Codul PHP necesar creării celor trei obiecte din figura de mai sus este următorul:

```
$strawberry = new Fruit();
$strawberry->setName('Strawberry');
$strawberry->setColor('#E22F37');

$orange = new Fruit();
$orange->setName('Orange');
$orange->setColor('#ED8F20');

$raspberry = new Fruit();
$raspberry->setName('Raspberry');
$raspberry->setColor('#C62A6A');

echo $strawberry->getName() . $strawberry->getColor() . "<br>";
echo $orange->getName() . $orange->getColor() . "<br>";
echo $raspberry->getName() . $raspberry->getColor() . "<br>";
```

Operatorul folosit pentru obiecte ce oferă acces la proprietățile și metodele unei clase este "**->**". Acesta reprezintă o săgeată compusă din simbolurile "-" , respectiv "**>**". Accesarea proprietăților și metodelor se face după modelul următor**: $obiect->proprietate**, **respectiv $obiect->metoda()**.

Observație: Numele proprietății sau metodei nu începe cu simbolul $, doar numele obiectului începe cu "$".

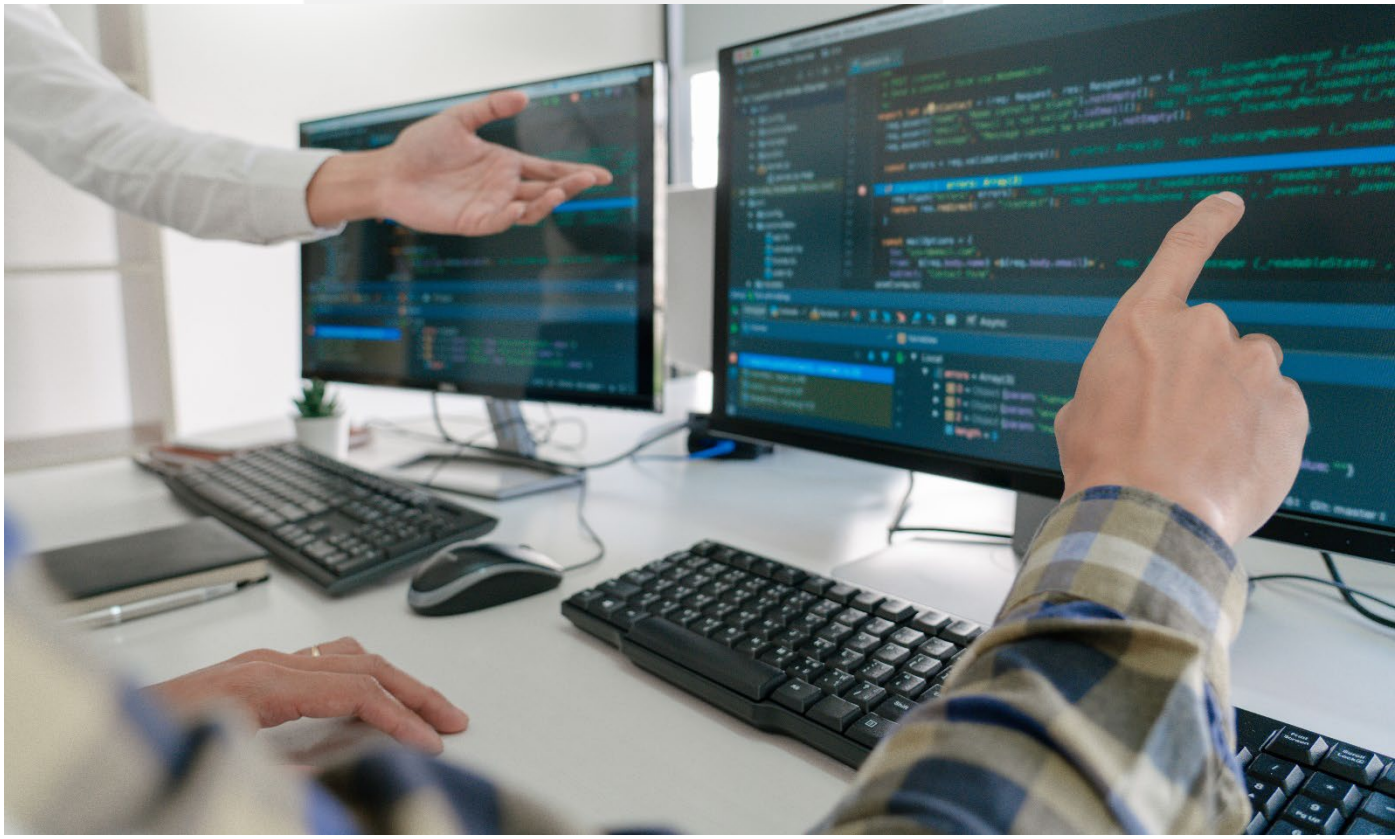
9.2.2 Proprietățile unui obiect

În corpul unei clase, poți declara variabile speciale, numite proprietăți. În PHP 4 acestea trebuiau precedate de cuvântul cheie **var** în momentul declarării. Începând cu PHP 5 acest lucru nu mai este necesar, dar o variabilă poate fi precedată de unul dintre modificatorii de acces: **public**, **protected** sau **private**.

```
class Reteta
{
    public $titluRetetă;
    public $ingrediente = array();
    public $instructiuni = array();
    public $rezultat;
    public $cuvinteCheie = array();
    public $autor = "Alena Holligan";
}

$reteta = new Reteta();
var_dump($reteta);
```

Observație: Convenția pentru numirea proprietăților este formatul **camel case** (cocoașă de cămilă). Precum "cocoașele" din spate, numele începe cu litera mică și următorul cuvânt începe cu literă mare. De exemplu, **$ingrediente** sau **$listaIngrediente**.



9.2.3 Metodele unui obiect

Într-o formă simplă, metodele sunt funcții declarate în interiorul unei clase. Sunt de obicei - dar nu întotdeauna - apelate prin intermediul unei instanțe de obiect folosind operatorul "**->**".

Declararea unei funcții în cadrul unei clase poate fi precedată, de asemenea, de un modificator de acces. Dacă acesta lipsește, se subînțelege că modificatorul setat implicit este **public**.

Încă un exemplu de declarare și utilizare a unei clase ce stochează informații pentru o rețetă este prezentat în continuare.

Observație: Convenția pentru numirea metodelor este de asemenea **camel case** (cocoașă de cămila). De exemplu, **$afiseazaReteta()**.

```
class Reteta
{
    public $titluReteta;
    public $listăIngrediente = array();
    public $instructiuni = array();
    public $rezultat;
    public $cuvinteCheie = array();
    public $autor = "Alena Holligan";

    public function afiseazaReteta()
    {
        return $this->titluReteta . " de " . $this->autor;
    }
}

$reteta1 = new Reteta();
$reteta1->autor = "Grandma Hooligan";
$reteta1->titluReteta = "My First Recipe";

$reteta2 = new Reteta();
$reteta2->autor = "Betty Crocker";
$reteta2->titluReteta = "My Second Recipe";

echo $reteta1->afiseazaReteta();
echo "<br>";
echo $reteta2->afiseazaReteta();
```

9.2.4 Constructorul

Constructorul este un tip special de funcție care este apelată automat când un obiect este creat. Acesta permite control asupra acțiunilor ce trebuie să aibă loc de fiecare dată la instanțierea obiectelor.

PHP oferă o funcție speciala numita **__construct()** pentru definirea constructorilor. Un constructor poate avea oricâte argumente sunt necesare.

Observație: Numele constructorului începe cu două simboluri underscore "__".

După definirea unui constructor ce primește argumente, la instanțierea obiectelor, numele clasei va trebui însoțit de numărul de argumente specificat în cadrul funcției **__construct()**. În exemplul următor, orice obiect nou creat, va trebui să primească la instanțiere două argumente, un nume și o culoare.

Utilizarea unui constructor cu argumente, simplifică setarea parametrilor prin reducerea instrucțiunilor necesare a fi executate de fiecare dată, aplicând astfel principiul DRY.

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

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

    // Metode
    function setName($name)
    {
        $this->name = $name;
    }
    function getName()
    {
        return $this->name;
    }
    function setColor($color)
    {
        $this->color = $color;
    }
    function getColor()
    {
        return $this->color;
    }
}

$apple = new Fruit("Apple", "red");
echo $apple->getName();
echo "<br>";
echo $apple->getColor();
```

9.2.5 Destructorul

Un destructor este apelat atunci când obiectul este distrus sau scriptul și-a încheiat execuția. Dacă creați o funcție **__destruct()**, PHP va apela automat această funcție la sfârșitul scriptului.

Observație: Funcția de distrugere începe, de asemenea cu două simboluri underscore (__).

Exemplul de mai jos are o funcție __construct() care este apelată automat când creați un obiect dintr-o clasă și o funcție __destruct() care este apelată automat la sfârșitul scriptului:

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

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

    // Destructor
    function __destruct() {
        echo "The fruit is {$this->name}.";
    }
}

$apple = new Fruit("Apple", "red");
```

Teme

1. Creați o funcție ce alege aleatoriu o valoare dintr-un vector primit ca argument (folosiți funcția array_rand() - verificați documentația).
2. Construiți o clasă "Carte" cu proprietățile: titlu, autor, an, pagini. Creați o funcție ce afișează proprietățile obiectului în formatul "Nume Autor, 'Titlu carte', nr. pagini: 123, an apariție: 2005."
3. Realizați un script ce generează un vector ce conține 10 obiecte Carte. Valorile pentru câmpurile titlu, autor, an, pagini vor fi luate la întâmplare cu ajutorul funcției create la punctul 1 din câte un vector cu valori predefinite (creați un vector cu 10 titluri de carte, un vector cu 8 autori, un vector cu 5 valori pentru ani și un vector cu 10 valori pentru numărul de pagini).
4. Creați o funcție ce alege la întâmplare un element din vectorul cu 10 cărți și afișează proprietățile folosind funcția creată la subiectul 2.
5. Creați o funcție ce afișează toate titlurile de carte din vectorul creat anterior.
6. Creați o funcție ce afișează numărul de cărți apărute în fiecare an pentru anii găsiți în vectorul de la subiectul numărul 3.