Napredno PHP Programiranje

Uvod u Objektno-Orijentirano Programiranje

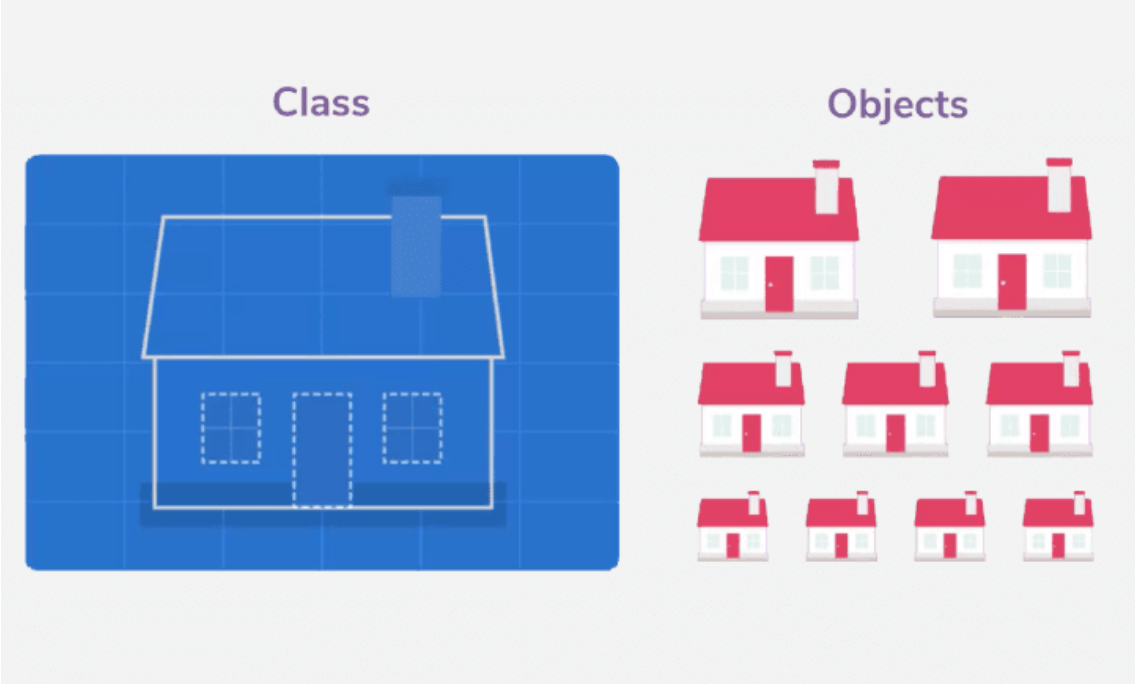
- **Definicija OOP:**
  - Programski pristup temeljen na objektima i klasama.
- **Prednosti OOP-a**:
  - ***Modularnost*:** Omogućava razvoj aplikacija kroz modularni pristup, gdje je lakše održavati i ažurirati dijelove aplikacije neovisno.
  - ***Ponovna upotreba koda*:** Nasljeđivanje i polimorfizam omogućavaju veću ponovnu upotrebu koda.
  - ***Skalabilnost*:** OOP pristup olakšava upravljanje većim projektima i njihovo skaliranje.
- **Nedostsci OOP-a**
  - ***Složenost*:** Objektno orijentirane aplikacije mogu biti složenije za razumijevanje i razvoj.
  - ***Performanse*:** Objekti i klasni mehanizmi mogu zahtijevati više resursa procesora i memorije.

Osnovne karakteristike

- **Apstrakcija:**
  - Izdvajanje esencijalnih karakteristika od sekundarnih.
- **Enkapsulacija:**
  - Podaci unutar objekta zaštićeni su od izravnog pristupa izvana. Objekti komuniciraju putem definiranih sučelja.
- **Nasljeđivanje:**
  - Omogućava jednoj klasi da naslijedi značajke (atribute i metode) druge klase.
- **Polimorfizam:**
  - Mogućnost da se isto ime funkcije koristi za različite tipove. To omogućava da se ista funkcija ponaša različito ovisno o kontekstu u kojem se koristi.

Klase i objekti

- **Definicija klase:**
  - *‘Blueprint’* za stvaranje objekata koji sadrži svojstva i metode.
- **Definicija objekta:**
  - Instanca klase.



Klase i objekti

```php
class Car {
    public $color;
    public $model;
    public function __construct($color, $model) {
        $this->color = $color;
        $this->model = $model;
    }
    public function message() {
        return "My car is a " . $this->color . " " . $this->model . "!";
    }
}
$myCar = new Car("black", "Volvo");
echo $myCar->message();
```

Kontrola pristupa svojstava i metoda

- **Public:**
  - Svojstva i metode dostupne unutar i izvan klase.
- **Protected:**
  - Svojstva i metode dostupne unutar klase i njenih nasljednika.
- **Private:**
  - Svojstva i metode dostupne samo unutar klase.

```php
class Vehicle {
    public $make;
    protected $engine;
    private $vin;

    public function __construct($make, $engine,
$vin) {
        $this->make = $make;
        $this->engine = $engine;
        $this->vin = $vin;
    }
}
```

Konstruktori i destruktori

- **Konstruktor:**
  - Metoda koja se automatski poziva prilikom stvaranja objekta.
- **Destruktor:**
  - Metoda koja se poziva kada objekt više nije potreban.

```php
class Book {
    public function __construct() {
        echo "A new book was created.";
    }
    public function __destruct() {
        echo "A book is being deleted.";
    }
}
$book = new Book();
```

Setteri i Getteri

- **Zašto Setteri i Getteri?**
  - Omogućuju kontroliran pristup svojstvima klase.
- **Setteri:**
  - Metode koje postavljaju vrijednost svojstva.
- **Getteri:**
  - Metode koje vraćaju vrijednost svojstva.

```php
class Account {
    private $balance;

    public function setBalance($amount) {
        if ($amount >= 0) {
            $this->balance = $amount;
        } else {
            echo "Balance cannot be negative.";
        }
    }

    public function getBalance() {
        return $this->balance;
    }
}
$acc = new Account();
$acc->setBalance(100);
echo "The balance is: " . $acc->getBalance();
```

Nasljeđivanje

- Korištenje koda iz bazne klase u izvedenoj klasi.

```php
class Vehicle {
    public $make;
    protected $engine;
    private $vin;

    public function __construct($make, $engine, $vin) {
        $this->make = $make;
        $this->engine = $engine;
        $this->vin = $vin;
    }
}

class Car extends Vehicle {
    public function getEngine() {
        return $this->engine; // Accessible due to protected
    }
}

$myCar = new Car("Toyota", "V6", "123ABC");
echo $myCar->getEngine();
```

Polimorfizam i sučelja

- **Polimorfizam:**
  - Primjer kako različite klase mogu implementirati iste metode na različite načine.
- **Sučelje:**
  - Obveza za klase da implementiraju određene metode.

```php
interface Shape {
    public function calcArea();
}

class Circle implements Shape {
    private $radius;

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

    public function calcArea() {
        return pi() * $this->radius * $this->radius;
    }
}

class Rectangle implements Shape {
    private $width;
    private $height;

    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }

    public function calcArea() {
        return $this->width * $this->height;
    }
}
```

Apstraktne klase

- **Definicija:**
  - Klase koje ne mogu biti instancirane i koriste se kao osnova za druge klase.
- **Svrha:**
  - Definirati temeljnu skicu za niz izvedenih klasa, prisiljavajući izvedene klase da implementiraju određene metode.

```php
abstract class Animal {
    protected $name;

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

    abstract public function makeSound();

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

class Dog extends Animal {
    public function makeSound() {
        return "Bark";
    }
}

$myDog = new Dog("Rover");
echo $myDog->getName() . " says " . $myDog->makeSound();
```

Statičke Metode

- **Definicija:**
  - Metode koje se mogu pozivati na razini klase, a ne na razini instance.
- **Svrha:**
  - Omogućavanje pristupa funkcijama koje ne zahtijevaju podatke o specifičnoj instanci klase.

```php
class MathHelper {
    public static function add($a, $b) {
        return $a + $b;
    }
}

echo MathHelper::add(5, 3);
```

Hvala na pažnji!