

Type hinting



Type hinting

- If a function expects a parameter of a certain type (class), the class name can be added before the parameter name.
- If an argument of the wrong type is used when calling the function, it will cause an error.



```
class Club {  
    private $players = [];  
  
    public function addPlayer(Player $player)  
    {  
        $this->players[] = $player;  
    }  
}  
  
class Player {  
}  
  
$udarnik = new Club();  
$kostolomac = new Player();  
  
$udarnik->addPlayer($kostolomac); // OK!  
$udarnik->addPlayer("Balerina"); // Error!
```



Type hinting

- In PHP 7, the built-in types can be used, and return type declaration is also supported



```
class Club {  
    private $name;  
  
    public function __construct(string $name)  
    {  
        $this->name = $name;  
    }  
  
    public function getName(): string  
    {  
        return $this->name;  
    }  
}
```

```
$udarnik = new Club("Udarnik");  
echo $udarnik->getName();
```



Type hinting

- If an object of the superclass is expected, an object of the subclass can also be used.



```
class Club {  
    private $players = [];  
  
    public function addPlayer(Player $player)  
    {  
        $this->players[] = $player;  
    }  
}
```

```
class Player {  
}
```

```
class Goalkeeper extends Player {  
}
```

```
$sudarnik = new Club();  
$kostolomac = new Goalkeeper();
```

```
$sudarnik->addPlayer($kostolomac); // OK!  
$sudarnik->addPlayer("Balerina"); // Error!
```



Exercise I

1. U prvom zadatku od cetvrtka (fajlovi i folder) postaviti Type hinting u funkcije dodavanja folder i dodavanja fajlova
2. U drugom zadatku od cetvrtka (geometrijski objekti) postaviti tip povratne vrednosti za svaku metodu koja sadrzi povratnu vrednost.



Static members



Static members

- Belong to the class, not to any object
- Both variables and function can be static
- Same access modifiers apply



```
class Product {  
    private static $count = 0;  
    public $name;  
  
    public function __construct() {  
        self::$count++;  
        echo "Produced so far: " . self::$count . "<br>";  
    }  
  
    public static function getCount() {  
        return self::$count;  
    }  
}  
  
$product1 = new Product();  
$product2 = new Product();
```



Static members

- Can be used by any object, but changes to static data are seen in all objects
- Accessed using the double colon (::) notation
- Like the current object is **\$this**, the current class is **self**, and the parent class is designated **parent**.



```
class Product {  
    public static $count = 0;  
    public $name;  
  
    public function __construct($productName) {  
        self::$count++;  
        $this->name = $productName;  
        echo "Produced so far: " . self::$count . "<br>";  
    }  
}
```

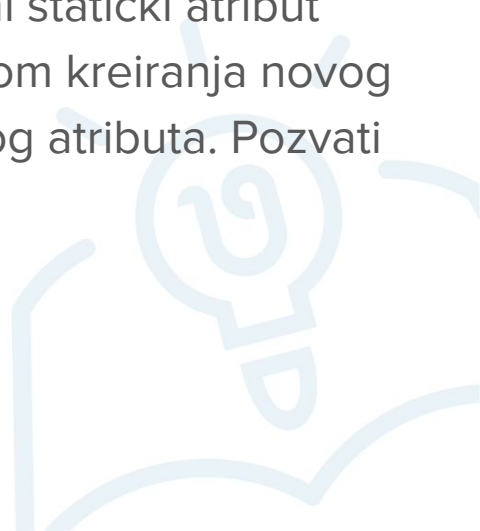
```
$milk = new Product("Milk");  
$bread = new Product("Bread");
```

```
echo $milk->name . "<br>";  
echo $bread->name . "<br>";  
echo "Total number of created products: " . Product::$count . "<br>";
```



Exercise II

1. U prvom zadatku od četvrtka (fajlovi i folder) postaviti Type hinting u konstruktore. Dodati i privatni statički atribut **\$brojac** u **Fajl** klasu i staticku metodu koja će vratiti vrednost tog atributa;
2. U drugom zadatku od četvrtka (geometrijski objekti) dodati brojač ukupno kreiranih geometrijskih objekata. Brojač postaviti kao privatni statički atribut odgovarajuće klase. Inkrementirati brojač unutar klase prilikom kreiranja novog objekta. Dodati i statičku metodu koja će vratiti vrednost ovog atributa. Pozvati ovu metodu na kraju programa i ispisati dobijenu vrednost.

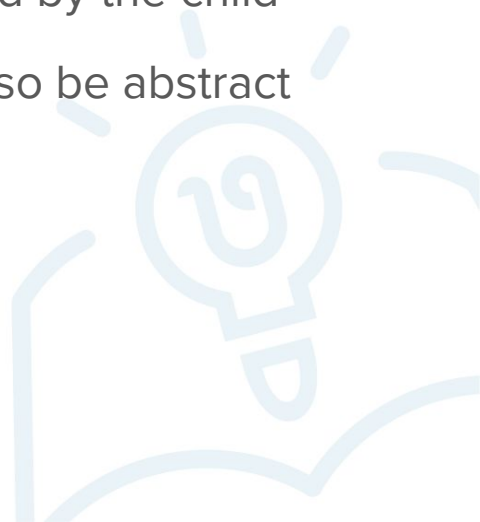


Abstract classes



Abstract classes

- Classes meant to be extended and not instantiated directly
- Abstract methods have no implementation and must be overridden
- Methods marked abstract in the parent class must be defined by the child
- Any class that contains at least one abstract method must also be abstract




```
abstract class Animal {  
    private $name;  
  
    public abstract function speak();  
    public function getName() { return $this->name; }  
}
```

```
class Dog extends Animal {  
    public function speak() { echo 'Woof!'; }  
}
```

```
$dog1 = new Dog();  
$dog1->speak();
```



Exercise III

U prvom zadatku od četvrtka (fajlovi i folder) definisati odgovarajuću klasu kao apstraktnu. U istu klasu dodati apstraktnu metodu **getTip()**. Implementirati ovu metodu u klasama naslednicima na način da vraćaju jednostavan string.



Interfaces



Interfaces

- Represent a “contract” that a class has to fulfill
- All methods declared in an interface must be public
- Unlike abstract classes, can contain no data or method implementation



```
interface Paintable {  
    public function setColor($color);  
    public function getColor();  
}
```



Interfaces

- Classes declare that they implement an interface with the **implements** keyword.
- All interface methods must be implemented in the class
- A class can implement multiple interfaces



```
interface Paintable {  
    public function setColor($color);  
    public function getColor();  
}
```

```
class Car implements Paintable {  
    public $color;  
  
    public function setColor($color) {  
        $this->color = $color;  
    }  
  
    public function getColor() {  
        return $this->color;  
    }  
}
```

```
class Wall implements Paintable {  
    public $wallColor;  
  
    public function setColor($color) {  
        $this->wallColor = $color;  
    }  
  
    public function getColor() {  
        return $this->wallColor;  
    }  
}
```

Interfaces

- Used as a type with type hinting, similar to classes




```
interface Paintable {  
    public function setColor($color);  
    public function getColor();  
}
```

```
class Car implements Paintable {  
    ....  
}
```

```
class Wall implements Paintable {  
    ...  
}
```

```
class Painter {  
    public function paint(Paintable $paintable, $color)  
    {  
        $paintable->setColor($color);  
    }  
}
```



Exercise IV

1. Napraviti interfejs **Prenosivo** koji deklarise tri metode: *spakuj*, *prenesi* i *raspakuj*. Metoda *prenesi* prima argument **\$pozicija**. Nakon toga kreirati klase **Racunar** i **Krevet** koje implementiraju ovaj interfejs. Napraviti program koji ce simulirati prenošenje dva objekta klase **Racunar** i **Krevet** iz jednog stana u drugi.

