Instrukcja 3

PROGRAMOWANIE OBIEKTOWE

Programowanie strukturalne jest szybkie i efektywne. Niestety gdy mamy do czynienia z naprawdę dużym projektem organizacja kodu poprzez struktury może stać się bardzo kłopotliwa, a skrypty nieczytelne. Inżynierowie programowania szukali rozwiązania, które pozwoliło by w łatwy sposób dzielić programy na moduły łatwo dające się włączać/wyłączać z aplikacji. Tak zrodziła się idea obiektowości. Programista tworzy klasę – twór mogący posiadać swoje własne zmienne (zwane polami) i funkcje (zwane metodami). Trzeba pamiętać by odróżniać KLASĘ od OBIEKTU; klasą jest definicja konkretnych składowych (pola i metody); obiektem jest natomiast zmienna typu własnego (typem jest klasa).
Przykład klasy:

```
class Klasa {
	private $var1 = 12;
	public $var2 = 13;

	function f() {
		echo $this->var1;
	}
}
$klasa = new Klasa();
echo $klasa->var2;
```

Jak widać definicja klasy zawsze rozpoczyna się od słowa zastrzeżonego class. Po nim podaje się jej nazwę (dowolną, jedynie nie można wybrać na nazwę słów zastrzeżonych przez język). Dobrym programistycznym zwyczajem jest rozpoczynanie nazwy klasy od dużej litery (chociaż nie jest to przymusowe). Wszystkie składowe klasy umieszcza się pomiędzy jej klamrami; Przyjrzyjmy się teraz zdefiniowanym składowym.
Każda ze składowych (zmienna lub funkcja) może posiadać określenie „widoczności”. Umożliwia to przykładowo ukrywanie konkretnej składowej przed dostępem do niej spoza samej klasy. Jako przykład niech posłuży wcześniejszy przykład. Proszę zmodyfikować go przy funkcji echo; zmienną var2 proszę zamienić na var1. Parser powinien zwrócić błąd. Nie zwróciłby błędu gdybyśmy wywołali funkcję f(). Tutaj klasa wywołuje „swoją”zmienną dzięki czemu mamy do niej dostęp. Tak właśnie zachowują się składowe prywatne. Przeciwieństwem ich są składowe publiczne – je można normalnie wywoływać spoza samej klasy (jak to miało miejsce w przypadku zmiennej $var2). Do czego taki podział jest potrzebny? Między innymi do tego, aby którykolwiek z modułów, bez pozwolenia twórcy klasy, nie został zmodyfikowany niejawnie przez inny element. Przeważnie dostęp do konkretnych pól klasy odbywa się właśnie poprzez funkcje (zwykło się je nazywać get_<nazwa_zmiennej>() do pobrania wartości, set_<nazwa_zmiennej>(wartosc) do ustawienia); uodparnia to program na występowanie nieoczekiwanych błędów. Domyślnie funkcje klasy są definiowane jako publiczne; im również można nadać właściwość prywatną (wtedy są widoczne tylko dla danej klasy). Innym zastosowaniem określania widoczności składowych ujawni się wraz z pojęciem dziedziczenia klas (chodzi o nienakładanie się składowych klasy bazowej i pochodnej). Pojęcie „widoczności” pól i metod w programowaniu nazywa się hermetyzacją (enkapsulacją).
Osobnego komentarza wymaga pseudo-zmienna $this. Zmienna ta tworzy się wraz z obiektem. Przechowuje ona odnośnik do niego samego. Dzięki temu możemy np. w metodzie klasy odwoływać się do poszczególnych jej pól. Uwydatnia to przykład funkcji drukującej wartość $var1, która to przy normalnym wywołaniu powoduje wyświetlenie informacji o próbie dostępu do składowej prywatnej. We wspomnianej funkcji, jeżeli nie użylibyśmy zmiennej $this, również nie

mielibyśmy dostępu do $var1; to z kolei wynika z właściwości funkcji (widzi tylko zmienne definiowane w jej ciele bądź parametrach wejściowych, nie poza nią).
Linia:

$klasa = new Klasa();

powoduje utworzenie zmiennej typu obiektowego $klasa. Tutaj pojawia się nam operator new. Dzięki niemu właśnie tworzona jest NOWA instancja (byt) obiektu (niezależna od żadnego innego aktualnego bytu). Drugim operatorem obiektów jest słowo clone. W przeciwieństwie do operatora new nie tworzy on nowego obiektu lecz klonuje aktualnie powołany do życia.

$klasa2 = clone $klasa;

$klasa2 przejmuje wszystkie wartości z $klasa. Od tego momentu są osobnymi bytami – zmiana wartości w jednym nie powoduje zmiany w drugim. Oczywiście możliwe jest także zreferowanie jednego obiektu do drugiego. Wtedy zachowuje się on jak każda zreferowana zmienna – w chwili zmiany parametru jednej zmiennej powoduje zmianę tego parametru w drugiej.

Zadanie:
Proszę stworzyć skrypt z klasą o dowolnej nazwie (np. Pracownik, Osoba), która będzie zawierała następujące pola: id, imię, nazwisko, funkcja. Wszystkie pola powinny być prywatne. Proszę stworzyć metody dostępowe dla każdej ze zmiennej. Dodatkowo proszę stworzyć kolejną klasę o dowolnej nazwie (np. Funkcja, Stanowisko) zawierającą pola o nazwie id, nazwa, miejsce. One również powinny być prywatne i powinny posiadać swoje metody dostępowe. Proszę teraz przetestować swój program w następujący sposób: utworzyć kilka obiektów pierwszej klasy (np. 3). Każdej z nich ustawić odpowiednie wartości pól. Pole funkcja proszę uzupełnić inaczej – trzeba do niej przypisać wartość obiektu utworzonego z drugiej klasy. Proszę przynajmniej do dwóch obiektów pierwszej klasy przypisać ten sam obiekt utworzony z drugiej klasy. Na koniec do pierwszej klasy należy dołączyć metodę wyświetlającą wszystkie pola tej klasy (w tym pola drugiej) w czytelnej postaci dla odbiorcy. Ostatecznie proszę wywołać te metody.

1. Wbudowane metody klas.

Klasy mogą posiadać w zasadzie dowolne metody. Jednak istnieją też metody wbudowane, które wywołują się nawet bez jawnego ich wywołania przez programistę. Przeważnie funkcje są już domyślnie zdefiniowane; w związku z powyższym nie zachodzi potrzeba pisania ich kodu samemu; możliwe jest jednak ich „nadpisanie” - nazywa się to przeładowaniem funkcji. Oznacza to, że jeżeli w definicji naszej klasy pojawi się funkcja o takiej właśnie nazwie to jej domyślna wersja zostanie zastąpiona naszą. Funkcje, które można przeładować posiadają nazwę rozpoczynającą się od '__', np. __call(), __construct(), __destruct(), itp.

void __construct(mixed argument1, …, mixed argumentN) – wszystkie parametry tej metody są opcjonalne (czyli nie trzeba podawać żadnego). Jej wywołanie jest automatyczne. Wywołuje się w chwili użycia operatora new (stąd jego nazwa - konstruktor). Można oczywiście samemu wywoływać tę metodę; tę możliwość omówimy jednak przy innej okazji. Dzięki konstruktorowi da się przypisać pożądane wartości dla poszczególnych pól klasy (bądź domyślne w przypadku braku argumentów). Jeżeli klasa nie posiada zdefiniowanego własnego konstruktora stosowany jest domyślny konstruktor (z domyślnej, podstawowej klasy)

void __destruct(void) – metoda ta wykonuje się gdy obiekt wskazanej klasy jest niszczony. Własne definiowanie destruktora przydaje się przeważnie wtedy, kiedy dany obiekt ma coś wykonać przy zakończeniu swojego istnienia (modyfikacja zmiennych, wyświetlenie jakiegoś komunikatu itp.).

Tak jak konstruktor, destruktor wywoływany jest automatycznie np. przy zakończeniu działania skryptu; może też zostać wywołany podczas działania skryptu w wypadku gdy nie ma już w kodzie więcej odniesień do danego obiektu.

public void __set(string $nazwa, mixed $value) – metoda ta jest wywoływana gdy następuje wywołanie zapisu do zastrzeżonej (prywatnej) zmiennej ($nazwa określa jej nazwę, $value wartość jaką ma przyjąć)

public mixed __get(string $nazwa) – metoda wykona się przy wywołaniu odczytu wartości prywatnej zmiennej ($nazwa określa jej nazwę)

public bool __isset(string $nazwa)/void __unset(string $nazwa) – funkcje te wywołują się w chwili gdy wywoływane są funkcje isset()/unset() dla zmiennych prywatnych.

public string __toString(void) – funkcja decyduje co zostanie zwrócone, gdy obiekt zostanie potraktowany jako ciąg znakowy.

void __clone(void) – metoda pozwala określić jak ma zachować się obiekt po sklonowaniu; chodzi głównie o ewentualną zmianę konkretnych pól klasy dla nowo tworzonej (np. zamiana imienia na pusty ciąg znakowy).

Zadanie.

Proszę przebudować program z poprzedniego ćwiczenia. Modyfikacja polegać powinna na przeciążeniu metod konstruktora i destruktora (dopisać domyślne uzupełnianie pól); dodać do konstruktora możliwość dodawania parametrów (zamiast kolejnych modyfikacji poszczególnych parametrów). Wykorzystać funkcję __clone() do modyfikowania zmiennej id; zamiast każdorazowo tworzyć nowy obiekt i ustawiać id spróbować operatora clone (klonować pierwszy obiekt do drugiego, drugi do trzeciego itd.). Na koniec spróbować dodać metodę __toString() - ma wyświetlać np. imię i nazwisko wywoływanego obiektu z osobą.

2. Składowe statyczne.

Składowe klas mogą być opatrzone słowem zastrzeżonym static. Niezależnie czy użyjemy tego słowa przy deklaracji pola lub metody efekt będzie ten sam – konkretne pole będzie posiadało status statycznej (działa to tak jak użycie słowa static przy zmiennej lokalnej w danej funkcji); ponadto jeżeli używamy tego słowa w klasie dane pole lub metoda mogą być wywoływane nawet gdy obiekt danej klasy NIE ISTNIEJE. Przykładowo jeżeli w klasie dodamy metodę, która miała by zwracać jedynie wynik mnożenia podanego parametru to czasem nie warto jest tworzyć całego obiektu w danej części kodu – wtedy wystarczy wywołać tą metodę poprzez taką instrukcję:

$zmienna = <nazwa_klasy>::<nazwa_metody>(<parametry>);

Tutaj pojawia się kolejny klasowy operator - '::'. Jest on niejako spójnikiem - łączy nazwę klasy z jej statycznym polem/metodą. Dzięki niemu możemy się do tego typu składowej odwołać. Z kolei jeżeli chcemy w ciele klasy zmienić wartość pola statycznego to robimy to poprzez słowo self, np.

```
static private $var1 = 2;

public function increaseVar() {
        ++self::$var1;
}
```

Zadanie

Poprzednie zadanie z automatycznym nadawaniem id działało jednak wymagało od programisty klonowania obiektu najmłodszego. Proszę tak zmodyfikować kod aby klonowanie następowało po pierwszym, domyślnym obiekcie. Do tego celu należy stworzyć w programie obiekt-rodzica. Następnie każdy kolejny będzie tworzony poprzez poprzez klonowanie właśnie jego. Aby id się zwiększało należy jego deklarację poprzedzić słowem static.

Zadanie dodatkowe:
Proszę dorobić do aktualnego kodu formularz dodawania nowych osób i/lub nowych stanowisk. W formularzu dodawanie stanowiska dla konkretnej osoby proszę stworzyć jako combobox. Poszczególne dane proszę przekazywać pomiędzy skryptem jako zmienne sesyjne; dodawanie nowych z formularza osób jako post; formularz dodawania nowych stanowisk proszę przesyłać jako get. Proszę stworzyć odsyłacz, po którego aktywacji wyświetli nam się lista albo wszystkich dodanych osób, albo stanowisk. Po ewentualnym kliknięciu na daną osobę i/lub stanowisko proszę spowodować kasowanie tejże pozycji. Zarówno osoby jak i stanowiska należy przechowywać jako typ tablicowy (jeden element tablicy = jeden obiekt).

3. Dziedziczenie.

Programowanie obiektowe udostępnia mechanizm tzw. dziedziczenia. Weźmy pod uwagę nasz rozwijany przykład - posiadamy klasę Osoba z polami id, imię, nazwisko, funkcja. Sama w sobie jest uniwersalna – możemy pod nią podłączyć dowolną osobę z dowolnej instytucji, fabryki, szkoły, forum internetowego itd. Czasem jednak chcielibyśmy większej „personalizacji” pod konkretny projekt – przykładowo dostosowując klasę pod instytucję szkolną. Mamy w niej następujący podział: porządkowi (godziny dyżuru, telefon), nauczyciele (specjalność, e-mail, telefon, wychowawstwo klasy, opieka_sali, id_plan), administracja (numer_pokoju, godziny_urzedowania), uczniowie (id_klasy, funkcja_klasowa, id_rodzic, id_dziennika), rodzice (dane_kontaktowe, funkcja_w_szkole). Jak widać mamy kilka typów osób, które posiadają pewne unikalne dane prywatne. Deklarowanie w każdej klasie tych samych pól byłoby męczące. Tutaj pojawia się istota dziedziczenia klasowego – nasza klasa Osoba będzie klasą bazową; klasy Porzadkowy, Nauczyciel, Administrator, Uczen oraz Rodzic będą pochodnymi klasy Osoba. Deklaracja klasy pochodnej wygląda następująco:

```
class KlasaBaza {
//cialo klasy
}

class KlasaPchodna extends KlasaBaza {
//kod klasy bazowej
}
```

W naszym wypadku:

```
class Osoba {
private $id;
private $imie;
private $nazwisko;
private $funkcja;

public function get_imie() {
```

```
	return $this->imie;
}

public function set_imie($imie) {
	$this->imie = $imie;
}
//pozostale set i get;
}

class Porzadkowy extends Osoba {
private $dyzur;
private $numer_kontakt;
//set i get dla parametrow
}

$porzadkowy = new Porzadkowy();
$porzadkowy->set_imie(„Marian”);

echo $porzadkowy->get_imie();
```

Zadanie:
Proszę utworzyć klasy pochodne do klasy Osoba. Proszę oprogramować dodawanie nowych osób do szkoły oraz ich funkcje (np. nauczyciel, uczeń, absolwent itd.) Dodatkowo proszę się zastanowić nad utworzeniem nowych klas – Dziennik oraz Plan. Jakie pola powinny zawierać?

Wraz z dziedziczeniem klasy pojawia się jeszcze jeden typ hermetyzacji danych – protected (obok public oraz private). Jeżeli pole lub metoda są poprzedzone słowem protected to wskazana składowa nie będzie widziana poza klasą (traktowana jest jako prywatna); jednak dla klasy pochodnej będzie ona widziana jako publiczna (należy pamiętać, że składowa private klasy bazowej NIE JEST widoczna w klasie bazowej).
Proszę przetestować jak wcześniejszy przykład zachowa się, gdy składowe klasy bazowej zostaną zmieniona na protected.

4. Polimorfizm

Polimorfizm (wielopostać) w ujęciu obiektowości nazywa się też podtypowaniem. Polega on na zamienianiu/poszerzaniu możliwości wcześniej zdefiniowanej metody klasy bazowej. Dzięki temu np. funkcja wyświetl() w klasie pochodnej wyświetli nam dane z klasy pochodnej nawet gdyby w klasie bazowej istniała identyczna funkcja wyświetlająca dane charakterystyczne dla klasy bazowej.

Przykład.

```
class Baza {
public $dane_baza = „To jest tekst klasy bazowej”;

public function wyswietl() {
	echo $this->dane_baza;
}
}
```

```
class Pochodna extends Baza {
public $dane_pochodne = „To jest tekst klasy pochodnej”;

public function wyswietl() {
        echo $this->dane_pochodne;
}
}

$zm = new Pochodna();
$zm1 = new Baza();
$zm->wyswietl();
$zm1->wyswietl();
```

Przykład oddaje idee polimorfizmu (podtypowania).

Zadanie.
Do poprzedniego przykładu (ze szkołą) proszę podtypować w klasach pochodnych funkcję z klasy bazowej odpowiadającą za wyświetlanie zapisanych danych osobowych (na tą chwilę po wywołaniu funkcji drukującej wyświetlą się tylko dane z klasy bazowej).

5.Klasy abstrakcyjne.

Klasami abstrakcyjnymi nazywamy takie, które posiadają jedynie DEKLARACJE odpowiednich metod (bez definiowania co mają konkretnie robić). Tego typu klasy można wykorzystywać jako bazowe dla kolejnych klas; Dzięki nim można nadpisywać działanie odpowiednich metod w kolejnych klasach. Parser wymusza zdefiniowanie abstrakcyjnych metod w klasie pochodnej (wywoła błąd krytyczny w przypadku ich braku). Stosowanie klas abstrakcyjnych ujednolica kod klas i czyni go czytelniejszym.

Przykład.

```
abstract class Sterowanie {
        abstract public function wyswietl();
        abstract public function dodaj($wartosc);
        abstract public function usun($watosc);
}

class Osoba extends Sterowanie {
private $id;
private $name;

public function wyswietl() {
        echo $id . „ „ . $name;
}

public function dodaj($wartosc) {
        $this->id = $this->id + 1;
        $this->name[$this->id] = $wartosc;
}

public function usun($wartosc) {
        $this->id = $this->id – 1;
```

```
        unset($this->name[$wartosc]);
}
}

class Funkcja extends Sterowanie {
private $id;
private $name;

public function wyswietl() {
        echo $id . „ „ . $name;
}

public function dodaj($wartosc) {
        $this->id = $this->id + 1;
        $this->name[$this->id] = $wartosc;
}

public function usun($wartosc) {
        $this->id = $this->id – 1;
        unset($this->name[$wartosc]);
}
}
```

Zadanie.
Proszę zmodyfikować poprzedni przykład o klasę abstrakcyjną odpowiadającą za funkcję dodawanie, modyfikowanie i usuwanie odpowiednich pól klas Osoby oraz Funkcji. Ponadto proszę dodać do tej abstrakcyjnej klasy metodę wyświetlającą pola poszczególnych klas. Proszę zobaczyć jak funkcjonuje tak zmodyfikowany skrypt.