T: Programowanie obiektowe w PHP (podstawy).

1. W kontekście **programowania obiektowego** obiektem może być prawie każda rzecz bądź koncepcja - obiekt fizyczny taki jak biurko lub klient, ale także **konceptualny** istniejący jedynie w oprogramowaniu, np. pole tekstowe lub plik.

2. Główną zaletą programowania obiektowego jest jego zdolność do rozpoznawania i wspomagania **kapsułkowania**, znanego również jako **ukrywanie danych**.

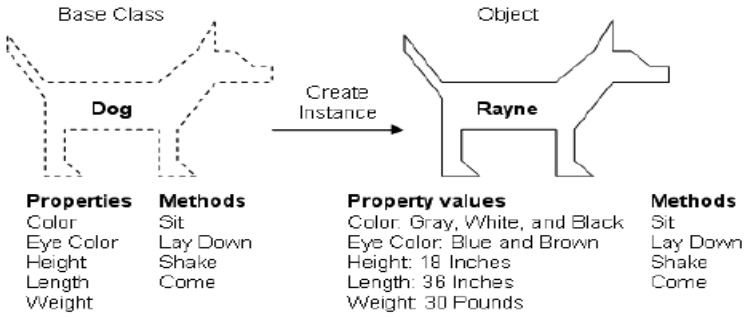
3. Programowanie obiektowe to sposób tworzenia programów komputerowych, który definiuje programy za pomocą obiektów - elementów łączących stan, właściwości (czyli dane) i zachowanie, zdarzenia (czyli funkcje zwane metodami).

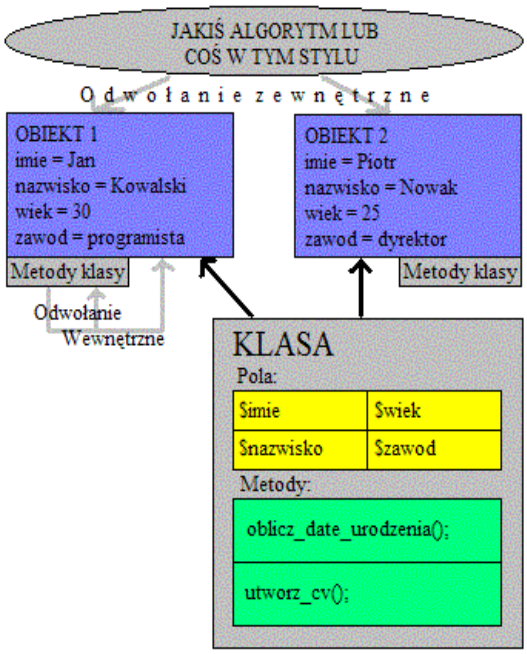
4. **Klasa** to opis obiektu – jego definicja, zawiera opis cech i zachowanie obiektów.

5. **Obiekt** jest instancją klasy – fizycznym tworem, który ma cechy zawarte w opisie klasy.

6. Klasa i obiekt to dwa kluczowe pojęcia towarzyszące programowaniu obiektowemu. Pisanie kodu obiektowego polega na definiowaniu klas, związków i relacji między nimi oraz manipulowaniu obiektami. **Klasa jest typem danych zaś obiekt — zmienną**.

7. Przykłady klas i obiektów:





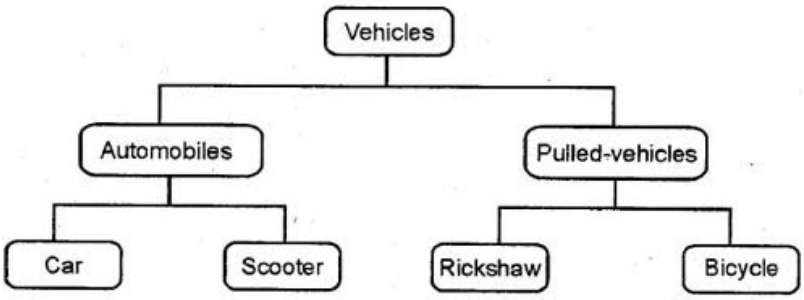
8. **Polimorfizm** oznacza wielopostaciowość, lub inaczej zdolność obiektów do różnych zachowań, w zależności od bieżącego kontekstu wykonania programu. W języku PHP wykonanie akcji polega na wywołaniu odpowiedniej funkcji. Polimorfizm w tym języku polega na tym, że wywoływane mogą być różne wersje tej samej funkcji.

9. Jedną z najważniejszych cech programowania obiektowego jest **dziedziczenie**. Dziedziczenie (ang. inheritance) to tworzenie nowej klasy na podstawie jednej lub kilku istniejących wcześniej klas bazowych. Dziedziczenie umożliwia bardzo ważną rzecz we współczesnym programowaniu: ponowne wykorzystywanie raz napisanego kodu, oraz rozszerzanie i dostosowywanie go do własnych potrzeb.

10. Klasa dodająca nowe właściwości do istniejącej już klasy jest **wyprowadzona** z klasy pierwotnej. Ta pierwotna klasa jest nazywana klasą **bazową**.

11. Klasy wyprowadzone, automatycznie otrzymują „cechy przodka" czyli składowe klasy bazowej, prócz tego klasa potomna (wyprowadzona) posiada swoje własne „cechy", czyli własne składowe dane i metody.

12. Przykład dziedziczenia cech:



13. **Definicja klasy w PHP**:

```
class nazwa {
}
np.:

<?php
class Osoba
{
  public $imie;
  public $nazwisko;
  function show()
  {
    echo "Test metody show...";
  }
}
?>
```

14. **Utworzenie obiektu** – instancji klasy oraz ustawienie danych składowych obiektu:

```
$osoba1 = new Osoba();
$osoba1->imie = "Jan";
$osoba1->nazwisko = "Kowalski";
```

15. **Wyświetlenie danych składowych obiektu** oraz wywołanie metody klasy dla utworzonego obiektu:

```
echo "Wartość pola imie = $osoba1->imie <br />";
echo "Wartość pola nazwisko = $osoba1->nazwisko <br />";
$osoba1->show();
```

16. Dołączenie definicji klasy umieszczonej w zewnętrznym pliku:
require "osoba.php";
Zawartość pliku osoba.php jest podana w pkt 16.

17. Każda funkcja składowa posiada ukryty parametr, jest nim wskaźnik **this**.
    Wskaźnik ten **wskazuje na egzemplarz obiektu klasy, dla którego wywołana została dana funkcja składowa (metoda klasy)**. Dzięki wskaźnikowi this również wewnątrz definicji metod klasy możemy uzyskiwać dostęp do jej składowych.

18. Przykład użycia **this** wewnątrz klasy:
```
function show()
{
   echo "Imię: ".$this->imie."<br />";
   echo "Nazwisko: ".$this->nazwisko."<br />";
}
```

19. **Specjalna metoda klasy - konstruktor** jest wywoływany przy tworzeniu obiektu, może więc wykonywać przydatną inicjalizację zadań, takich jak nadawanie atrybutom sensownych wartości początkowych czy też tworzenie innych obiektów wymaganych przez ten obiekt.
    Konstruktor jest deklarowany w identyczny sposób jak inne operacje, ale posiada specjalną nazwę:
    __construct()

    Przykład użycia konstruktora ustawiającego dane składowe dla obiektu:
```
function __construct()
{
   $this->imie = "Nieznany";
   $this->nazwisko = "Nieznajomy";
}
```

    Podczas tworzenia obiektu powyższa specjalna metoda klasy zostanie automatycznie wywołana i ustawi domyślne dane składowe dla nowego obiektu.

20. Przykład **konstruktora z parametrami**:
```
function __construct($imie, $nazwisko)
{
   $this->imie = $imie;
   $this->nazwisko = $nazwisko;
}
```

    Podczas tworzenia obiektów należy dla powyższego konstruktora podać dwie dane składowe:
```
$osoba1 = new Osoba("Jan", "Kowalski");
$osoba2 = new Osoba("Anna", "Malinowska");
```

21. Przeciwieństwem konstruktora jest **destruktor**. Destruktorem jest metoda o nazwie **__destruct()** i jest ona wywoływana automatycznie, gdy obiekt przestaje istnieć.

22. **Przykład dziedziczenia** klasy B z klasy A:

```
<?php
class A
{
  var $poleA;
  function showA()
  {
    echo "Funkcja showA klasy A... <br />";
  }
}
```

```
class B extends A
{
  var $poleB;
  function showB()
  {
    echo "Funkcja showB klasy B... <br />";
  }
}
$obj = new B();
$obj->poleA = 100;
$obj->poleB = 200;
echo "Wartość pola poleA obiektu obj = $obj->poleA <br />";
echo "Wartość pola poleB obiektu obj = $obj->poleB <br />";
$obj->showA();
$obj->showB();
?>
```

Dziedziczenie działa tylko w jedną stronę. Podklasa, czyli dziecko, dziedziczy własności po swoim rodzicu, czyli nadklasie, lecz rodzic nie posiada własności dziecka.

23. W PHP wykorzystuje się **modyfikatory dostępu** (widoczne w pkt 13). Sterują one widocznością atrybutów i metod; umieszcza się je przed deklaracjami atrybutów i metod. PHP obsługuje następujące trzy modyfikatory dostępu:
    1. Modyfikatorem domyślnym jest **public**, co oznacza, że jeśli modyfikator dostępu dla atrybutu lub metody nie zostanie wskazany, to będzie nim właśnie public. Elementy publiczne są dostępne zewsząd - spoza klasy i z jej wnętrza.
    2. Modyfikator dostępu **private** oznacza, że dany element jest dostępny jedynie z wnętrza klasy. Elementy prywatne **nie zostaną odziedziczone**.
    3. Modyfikator dostępu **protected** oznacza, że dany element jest dostępny z wnętrza klasy **jak i jej podklasach** (klasach które dziedziczą z klasy bazowej). Protected jest rozwiązaniem pośrednim między private i public.

24. Przykład z dziedziczeniem, konstruktorami, wskaźnikiem this, modyfikatorami: (poniżej pliki: osoba.php, uzytkownik.php, index.php)

```
<?php
class Osoba
{
  public $imie;
  public $nazwisko;
  function __construct($imie,
$nazwisko)
  {
    $this->imie = $imie;
    $this->nazwisko = $nazwisko;
  }
  function show()
  {
    echo "Imię: ".$this->imie."<br />";
    echo "Nazwisko: ".$this-
>nazwisko."<br />";
  }
}
?>
```

```
<?php
require_once "osoba.php";

class Uzytkownik extends Osoba
{
  public $id;
  function __construct($imie,
$nazwisko, $id)
  {
    $this->imie = $imie;
    $this->nazwisko = $nazwisko;
    $this->id = $id;
  }
  function showId()
  {
    echo "Id: ".$this->id."<br />";
  }
}
?>
```

```
<!DOCTYPE html>
<html>
<head>
<meta http-equiv="Content-Type"
content="text/html; charset=utf-8">
<title>Dziedziczenie</title>
</head>
<body>
<?php
require "uzytkownik.php";

$user = new Uzytkownik("Jan", "Kowalski", 1);
$user->show();
$user->showId();
?>
</body>
</html>
```

require_once - dołącza plik do kodu PHP w miejscu wystąpienia, czyli działa identycznie jak **require**. Jednak w razie próby ponownego załączenia tych samych danych polecenie to zostanie zignorowane.