BRAINSTER

Workshop 19: PHP - OOP

Brainster Web Development Academy

Exercise 1

Create an abstract class Card and place it inside namespace Banking.
It has 3 protected properties:

- $pan (primary account number),
- $balance and
- $discount that is 5 by default.

Write the constructor where you will set these 3 properties.

Make setter methods for the discount and the balance, so you can modify them after creating objects. This class should also contain an abstract method payment() that expects one parameter: $expenses, and a public method info() that returns "{$this->pan} {$this->balance}";

After you have written the classes that extend this class, think if there is any common functionality between the children classes that can be extracted in this class.
Hint: the payment method doesn't necessarily need to be abstract. Since the payment() method works same for both of the children classes (checks if there is enough money and withdraws it if so), maybe it would be better to implement this method here, instead of making it abstract. The children classes would then need to only keep logic about applying the proper discount before making the payment.

Exercise 1

Create class DebitCard and place it inside namespace Master.
Create another class DebitCard and place it inside namespace Visa.
Both classes should extend the class Card.

The payment() method in the *Master class* first calculates the expense considering the discount like this:
$discountedExpense = ($expense - ($expense * $this->discount / 100));
Tha payment() method in the *Visa class* calculates the expense the same way, with one difference: if the amount is larger than 6000, the discount is increased to 10 (meaning 10%).

If there is not enough money on the card to make the payment, the method should print:
"Not enough money. To pay: X, balance: Y".

If there is enough money, this method should make the payment, update the balance on the card and return the following message:
“{cardname} payment: $balance - $discounted = $newBalance”,
where {cardname} is either Master or Visa depending on which card we’re paying from.

Exercise 1

Create an index.php file where you will create 2 cards (1 Master and 1 Visa).

```
$master = new Master('123456789', 10000);
$visa = new Visa('123412341234', 20000);
```

Then make the following payments. The comments are the expected results:

```
$master->payment(1000); //Master payment: 10000 - 950 = 9050
$master->payment(1000); //Master payment: 9050 - 950 = 8100
$master->payment(7000); //Master payment: 8100 - 6650 = 1450
$master->payment(5000); //Not enough money. To pay: 4750, balance: 1450

$visa->payment(1000); //Visa payment: 20000 - 970 = 19030
$visa->payment(10000); //Visa payment: 19030 - 9000 = 10030
$visa->payment(500); //Visa payment: 10030 - 485 = 9545
$visa->payment(50000); //Not enough money. To pay: 45000, balance: 9545
```

* make sure to reset the Visa's discount to 3% after payment larger than 6000 is made. Common error is to increase it to 10% once a payment larger than 6000 is made and never reset it back to 3%. If that was the case, in the example code above when we call `$visa->payment(500);` it would have withdrawn 450 instead 485 like it does.

Exercise 2

Create a class Team that has only 1 property: $name;
Write the constructor that will set the name and a method getTeam() that returns the team's name.

Create a class Match.
This class has the following private properties: $host, $guest, $hostGoals, $guestGoals.
Create the constructor and the getters for these properties.
The $host and $guest properties must be objects from the class Team.

In the index.php create couple objects of the Team class and simulate couple of matches between them.

```
$team1 = new Team("Warriors");
$team2 = new Team("Expendables");
$team3 = new Team("Avengers");

$match1 = new Match($team1, $team2, 2, 1);
$match2 = new Match($team2, $team1, 3, 0);
$match3 = new Match($team1, $team3, 0, 0);
```

* The $match1 and $match2 are a rematch between the Warriors and Expendables teams. The $match3 object is just one match that is not correlated with any of the other two matches.

Exercise 2

Then write 2 functions inside index.php (just ordinary functions, not methods, not within a class).
The first function is checkIfRematch() that takes 2 takes two objects of the class Match as arguments.
It checks if the 2 matches are part of a rematch between the teams. The logic behind this is the following: If the name of the host in the first match is equal to the name of the guest in the second match, and if the name of the host in the second match is equal to the name of the guest in the first match, then the function should return true. If not, then return false.

The second function is duel(). It also takes two objects of the class Match as arguments. This function first calls the checkIfRematch function. If the matches are not a part of a rematch this function prints: "It is not a rematch". Else, this function calculates which team scored more goals and prints the winner. If they scored same amount of goals, it prints "It is a tie".

Using the matches from the previous slide, the duel function should work like this:

```
duel($match1, $match3); //It is not a rematch
duel($match1, $match2); //Expendables is a winner by total score: Warriors 2 : 4 Expendables
```

If we change the $match2 in the previous slide like:

```
$match2 = new Match($team2, $team1, 2, 1);
```

Then the duel function should print:

```
duel($match1, $match2); //It is a tie with score: Warriors 3 : 3 Expendables
```

Exercise 3

We need to write all necessary classes to be able to keep information about people and their wardrobe.
For every person we keep the name and surname.
For every person we also keep information about the person's clothes in his wardrobe.
Hint: $wardrobe should be an array in which we will push objects. We add new clothes using the setter for this property.

All clothes have size and color. There are three types of clothes: shirts, skirts and trousers.
Hint: make a WardrobeItem class to store size and color and classes Shirt, Skirt and Trousers that extend the WardrobeItem class.

For the shirts we know their sleeve type. Available options are 'long' and 'short'. If anything other is specified when the print function is called on the shirt object it should print 'type not specified'.

```
$shirt = new Shirt("M", "black", "short");
$shirt->print(); //Shirt, size: M, color: black, type: short sleeves
$shirt = new Shirt("M", "black", "helloworld");
$shirt->print(); //Shirt, size: M, color: black, type: Not specified sleeves
```

For the skirts we keep information about the length.

```
new Skirt("S", "red", 40);
```

For the trousers we know their type: skinny, flare, capri or ripped. Anything other prints 'not specified' too.

```
new Trousers("L", "blue", "capri");
```

Exercise 3

In the index.php file create one person object and couple of WardrobeItem objects. When we create a person his wardrobe is initially empty. If we call the print method at this time it should work like this:

```
$person1 = new Person("Jane", "Doe");
$person1->print(); //Jane Doe, wears: No clothes yet
```

Add couple of wardrobe items to the person's wardrobe. Call the print method again. It should now print:

```
$person1->print();
//Jane Doe, wears:
//Shirt, size: M, color: black, type: short sleeves
//Trousers, size: L, color: blue, type: capri
//Skirt, size: S, color: red, length: 40cm
```

Inside the person class write a method that **will order all the items in the wardrobe by size**.
Then call this method to order the wardrobe and call the print method. It should now print:

```
$person1->orderWardrobe();
$person1->print();
//Jane Doe, wears:
//Skirt, size: S, color: red, length: 40cm
//Shirt, size: M, color: black, type: short sleeves
//Trousers, size: L, color: purple, type: capri
```


Break a leg

If you haven't finished all exercises, please try to finish them at home.

Each class you create during this workshop should be in a separate file. When creating objects make sure to include/require all of the necessary classes.

Functions & operators that can help you with the exercises in these workshop (google their usage):

get_class($object)

- returns the classname of the given object. It will include the namespace if the class belongs to one. Use the explode('\\', $classname) if you need to access exact part of the full class name.

instanceof

- type operator, used to determine whether a PHP variable is an instantiated object of a certain class. (more info)