Abstract classes and methods

We use **abstract classes** when we want to commit the programmer (either oneself or someone else) to write a certain class method, but we are only sure about the name of the method, and not the details of how it should be written. To take an example, circles, rectangles, octagons, etc. may all look different, but are all 2D shapes nonetheless, and thus all possess the traits of area and circumference. So, it makes perfect sense to group the code that they have in common into one parent class. In this parent class, we would have the two properties of area and circumference, and we might even consider adding a method that calculates the area (which might be problematic since different shapes require different calculations). In these kinds of cases, when we need to commit the child classes to certain methods that they inherit from the parent class but we cannot commit about the code that should be used in the methods, we use abstract classes.

> *We use abstract classes and methods when we need to commit the child classes to certain methods that they inherit from the parent class but we cannot commit about the code that should be written inside the methods.*

An **abstract class** is a class that has at least one **abstract method**. Abstract methods can only have names and arguments, and no other code. Thus, we cannot create objects out of abstract classes. Instead, we need to create child classes that add the code into the bodies of the methods, and use these child classes to create objects.

How to declare classes and methods as abstract?

See the following example:

```
abstract class Car { }
```

We put the abstract methods that are also declared with the `abstract` keyword within the abstract class. Abstract methods inside an abstract class don't have a body, only a name and parameters inside parentheses.

In the example given below, we create a public abstract method, `calcNumMilesOnFullTank()`, that is the skeleton for methods that we will create in the child classes. Once created, these methods will return the number of miles a car can be driven on a tank of gas.

```
// Abstract classes are declared with the abstract keyword, and conta
abstract class Car {
  abstract public function calcNumMilesOnFullTank();
}
```

It is important to know that once we have an abstract method in a class, the class must also be abstract.

Can we have non abstract methods inside an abstract class?

An abstract class can have non abstract methods. In fact, it can even have properties, and properties couldn't be abstract.

```php
abstract class Car {
  // Abstract classes can have properties
  protected $tankVolume;

  // Abstract classes can have non abstract methods
  public function setTankVolume($volume)
  {
    $this -> tankVolume = $volume;
  }

  // Abstract method
  abstract public function calcNumMilesOnFullTank();
}
```

How to create child classes from an abstract class?

Since we cannot create objects from abstract classes, we need to create child classes that inherit the abstract class code. Child classes of abstract classes are formed with the help of the `extends` keyword, like any other child class. They are different in that they have to add the bodies to the abstract methods.

> *The child classes that inherit from abstract classes must add bodies to the abstract methods.*

Let's create a child class with the name of Honda, and define in it the abstract method that it inherited from the parent, `calcNumMilesOnFullTank()`.

```php
class Honda extends Car {
  // Since we inherited abstract method, we need to define it in the
    // by adding code to the method's body.
  public function calcNumMilesOnFullTank()
}
```

We can create another child class from the `Car` abstract class and call it `Toyota`, and here again define the abstract method `calcNumMilesOnFullTank()` with a slight change in the calculation. We will also add to the child class its own method with the name of `getColor()` that returns the string "beige".

```
class Toyota extends Car {
  // Since we inherited abstract method, we need to define it in the 
    // by adding code to the method's body.
  public function calcNumMilesOnFullTank()
  {
    return $miles = $this -> tankVolume*33;
  }

  public function getColor()
  {
    return "beige";
  }
}
```

Let's create a new object, `$toyota1`, with volume of 10 Gallons, and let it return the number of miles on full tank and the car's color.

```
$toyota1 = new Toyota();
$toyota1 -> setTankVolume(10);
echo $toyota1 -> calcNumMilesOnFullTank();//330
echo $toyota1 -> getColor();//beige
```

Conclusion

revisit the concept of abstraction, but this time through the use of **interface**.

<< PREVIOUS TUTORIAL

< ALL THE TUTORIALS >

NEXT TUTORIAL >>

Fx Mahbub
5 years ago

Usually I do not give feedback if the tutorial/learning isn't really well. I found your teaching way/tutorial most efficient and useful one! good luck mate. go ahead

22 0 • Reply • Share ›

Only the latest tutorial-No spam

Enter your email

SUBSCRIBE