Абстрактные классы

Пусть у вас есть класс **User**, а от него наследуют классы **Employee** и **Student**.

При этом предполагается, что вы будете создавать объекты классов **Employee** и **Student**, но объекты класса **User** - не будете, так как сам класс **User** используется только для группировки общих свойств и методов своих наследников.

В этом случае можно принудительно запретить создавать объекты класса **User**, чтобы вы или другой программист где-нибудь их случайно не создали.

Для этого существуют так называемые *абстрактные* классы.

Абстрактные классы представляют собой классы, предназначенные для наследования от них. При этом объекты таких классов нельзя создать.

Для того, чтобы объявить класс абстрактным, нужно при его объявлении написать ключевое слово **abstract**:

```php
<?php
  abstract class User
  {

  }
?>
```

Итак, давайте напишем реализацию абстрактного класса **User**. Пусть у него будет приватное свойство **name**, а также геттеры и сеттеры для него:

```php
<?php
  abstract class User
  {
    private $name;

    public function getName()
    {
      return $this->name;
    }

    public function setName($name)
    {
      $this->name = $name;
    }
  }
?>
```

Попытка создать объект класса **User** вызовет ошибку:

```php
<?php
  $user = new User; // выдаст ошибку
?>
```

А вот унаследовать от нашего класса будет можно. Сделаем класс **Employee**, который будет наследовать от нашего абстрактного класса **User**:

```php
<?php
  class Employee extends User
  {
    private $salary;

    public function getSalary()
```

```
    {
      return $this->salary;
    }

    public function setSalary($salary)
    {
      $this->salary = $salary;
    }

  }
?>
```

Создадим объект класса **Employee** - все будет работать:

```
<?php
  $employee = new Employee;
  $employee->setName('Коля'); // метод родителя, т.е
    . класса User
  $ employee->setSalary(1000); // свой мет
    од, т.е. класса Employee

  echo $employee->getName(); // выведет 'Ко
    ля'
  e cho $employee->getSalary(); // выведет 1000
?>
```

Аналогично можно создать объект класса **Student**, наследующий от **User**:

```
<?php
  class Student extends User
  {
    private $scholarship; // стипендия

    public function getScholarship()
    {
      return $this->scholarship;
    }

    public function setScholarship($scholarship)
    {
      $this->scholarship = $scholarship;
    }
  }
?>
```

Задача 35.1

Самостоятельно, не подсматривая в мой код, реализуйте такой же абстрактный класс **User**, а также классы **Employee** и **Student**, наследующие от него.

Абстрактные методы

Абстрактные классы также могут содержать **абстрактные методы**.

Такие методы не должны иметь реализации, а нужны для того, чтобы указать, что такие методы должны быть у потомков.

А собственно *реализация таких методов - уже задача потомков*.

Для того, чтобы объявить метод абстрактным, при его объявлении следует написать ключевое слово **abstract**.

Давайте попробуем на практике. Пусть предполагается, что все потомки класса **User** должны иметь метод **increaseRevenue** (увеличить доход).

Этот метод должен брать текущий доход пользователя и увеличивать его на некоторую величину, переданную параметром.

Сам класс **User** не знает, какой именно доход будет получать наследник - ведь у работника это зарплата, а у студента - стипендия.

Поэтому каждый потомок будет реализовывать этот метод по-своему.

Фишка тут в том, что абстрактный метод класса **User** заставляет программиста реализовывать этот метод в потомках, иначе PHP выдаст ошибку.

Таким образом вы, или другой программист, работающий с вашим кодом, никак не сможете забыть реализовать нужный метод в потомке.

Итак, давайте попробуем на практике. Добавим абстрактный метод **increaseRevenue** в класс **User**:

```php
<?php
  abstract class User
  {
    private $name;

    public function getName()
    {
      return $this->name;
    }

    public function setName($name)
    {
      $this->name = $name;
    }

    // Абстрактный метод без тела:
    abstract public function increaseRevenue($va
      lue);
  }
?>
```

Пусть наш класс **Employee** пока останется без изменений. В этом случае, даже если не создавать объект класса **Employee**, а просто запустить код, в котором определяются наши классы, - PHP выдаст ошибку.

Давайте проверим - запустите приведенный ниже код:

```php
<?php
  abstract class User
  {
    private $name;

    public function getName()
    {
      return $this->name;
    }

    public function setName($name)
    {
      $this->name = $name;
    }

    abstract public function increaseRevenue($va
      lue);
  }

  class Employee extends User
  {
    private $salary;

    public function getSalary()
    {
      return $this->salary;
    }

    public function setSalary($salary)
    {
      $this->salary = $salary;
    }

```

```
    }
33 ?>
```

Давайте теперь напишем реализацию метода **increaseRevenue** в классе **Employee**:

```php
<?php
  class Employee extends User
  {
    private $salary;

    public function getSalary()
    {
      return $this->salary;
    }

    public function setSalary($salary)
    {
      $this->salary = $salary;
    }

    // Напишем реализацию метода:
    public function increaseRevenue($value)
    {
      $this->salary = $this->salary + $value;
    }
  }
?>
```

Проверим работу нашего класса:

```php
<?php
  $employee = new Employee;
  $employee->setName('Коля'); // установим имя
  $employee->setSalary(1000); // установим зар
    плату
  $ employee->increaseRevenue(100); // увеличим зар
    плату

  echo $employee->getSalary(); // выведет 1100
?>
```

Реализуем метод **increaseRevenue** и в классе **Student**. Только теперь наш метод будет увеличивать уже стипендию:

```php
<?php
  class Student extends User
  {
    private $scholarship; // стипендия

    public function getScholarship()
    {
      return $this->scholarship;
    }

    public function setScholarship($scholarship)
    {
      $this->scholarship = $scholarship;
    }

    // Метод увеличивает стипендию:
    public function increaseRevenue($value)
    {
      $this->scholarship = $this->scholarship + $
        value;
     }
  }
?>
```

Задача 35.2

Добавьте в ваш класс **User** такой же абстрактный метод **increaseRevenue**. Напишите реализацию этого метода в классах **Employee** и **Student**.

Задача 35.3

Добавьте также в ваш класс **User** абстрактный метод **decreaseRevenue** (уменьшить зарплату). Напишите реализацию этого метода в классах **Employee** и **Student**.

Некоторые замечания

При наследовании от абстрактного класса, **все** методы, помеченные абстрактными в родительском классе, должны быть определены в дочернем классе.

При этом область видимости этих методов должна совпадать или быть менее строгой. Что значит менее строгой: например, если абстрактный метод объявлен как **protected**, то реализация этого метода должна быть **protected** или **public**, но не **private**.

Объявления методов также должны совпадать: количество обязательных параметром должно быть одинаковым. Однако класс-потомок может добавлять необязательные параметры, которые не были указаны при объявлении метода в родителе.

Практика

Пусть нам необходимо работать с геометрическими фигурами, например, с квадратами, прямоугольниками, треугольниками и так далее.

Пусть каждая фигура будет описываться своим классом, при этом мы хотим сделать так, чтобы каждый класс имел метод для вычисления площади и метод для вычисления периметра фигуры.

Давайте сделаем для этого абстрактный класс **Figure** с двумя абстрактными методами **getSquare** и **getPerimeter**.

Почему класс **Figure** абстрактный: потому что он не описывает реально существующую геометрическую фигуру и, соответственно, объект этого класса мы не будем создавать.

Почему методы **getSquare** и **getPerimeter** абстрактные: потому что каждая фигура имеет свой алгоритм вычисления площади и периметра и, соответственно, класс **Figure** не может написать реализацию этих методов.

Зачем нам вообще нужен класс **Figure**: чтобы наследовать от него и таким образом заставить всех наследников реализовать указанные методы.

Итак, напишем реализацию класса **Figure**:

```php
<?php
  abstract class Figure
  {
    abstract public function getSquare();
    abstract public function getPerimeter();
  }
?>
```

Пусть теперь мы хотим создать класс **Quadrate** для описания геометрической фигуры квадрат.

Как известно, у квадрата все стороны равны, поэтому для описания квадрата нам нужно задать только его ширину.

Давайте для этого сделаем приватное свойство **$a**, значение которого будет задаваться в конструкторе класса:

```php
<?php
  class Quadrate
  {
    private $a;

    public function __construct($a)
```

```
    {
      $this->a = $a;
    }
  }
?>
```

Давайте теперь унаследуем наш класс **Quadrate** от класса **Figure**:

```
<?php
  class Quadrate extends Figure
  {
    private $a;

    public function __construct($a)
    {
      $this->a = $a;
    }
  }

  /*
    Код класса не рабочий
    и будет выдавать ошибку,
    так как мы не написали
    реализацию методов родителя.
  */
?>
```

Сейчас наша реализация класса **Quadrate** не рабочая, так как мы не написали реализацию абстрактных методов родителя.

Давайте сделаем это:

```
<?php
  class Quadrate extends Figure
  {
    private $a;

    public function __construct($a)
    {
      $this->a = $a;
    }

    public function getSquare()
    {
      return $this->a * $this->a;
    }

    public function getPerimeter()
    {
      return 4 * $this->a;
    }
  }
?>
```

Давайте создадим квадрат со стороной 2 и найдем его площадь и периметр:

```
<?php
  $quadrate = new Quadrate(2);
  echo $quadrate->getSquare(); // выведет 4
  echo $quadrate->getPerimeter(); // выведет 8
?>
```

Задача 35.4

Сделайте аналогичный класс **Rectangle** (прямоугольник), у которого будет два приватных свойства: **$a** для ширины и **$b** для длины.

Данный класс также должен наследовать от класса **Figure** и реализовывать его методы.

Усложним

Сейчас все методы класса **Figure** - абстрактные. Это, конечно же, не обязательно. Пусть наш класс имеет еще и метод **getRatio**, который будет находить отношение площади к периметру (то есть одно делить на второе).

Этот метод уже будет не абстрактный, а иметь реализацию, и все потомки смогут воспользоваться этим методом.

Почему мы можем написать реализацию этого метода прямо в классе **Figure**: потому что этот метод будет одинаковым для всех потомков.

Итак, добавим наш метод:

```php
<?php
  abstract class Figure
  {
    abstract public function getSquare();
    abstract public function getPerimeter();

    // Метод для вычисления отношения площади к п
      ериметру:
     public function getRatio()
    {
      return $this->getSquare() / $this->getPerimeter();
    }
  }
?>
```

Обратите внимание на следующее: хотя методы **getSquare** и **getPerimeter** абстрактные и не имеют реализации, мы их все равно можем использовать в своем методе **getRatio**, хотя реализация этих методов появится только в потомках.

Применим наш метод:

```php
<?php
  $quadrate = new Quadrate(2);
  echo $quadrate->getSquare(); // выведет 4
  echo $quadrate->getPerimeter(); // выведет 8

  echo $quadrate->getRatio(); // выведет 0.5
?>
```

Задача 35.5

Добавьте в класс **Figure** метод **getSquarePerimeterSum**, который будет находить сумму площади и периметра.