Лабораторна робота №4

Тема: Об'єктно-орієнтоване програмування в PHP

Мета роботи: навчитися працювати з класами

Хід роботи

Завдання 1. (Організація класів по каталогах в проєкті)

- Створіть пустий проєкт PHP.
- Створіть каталоги: "Models", "Controllers", "Views".
- У кожному каталозі створіть по одному класу, наприклад, "UserModel", "UserController", "UserView".
- В кожному класі реалізуйте просту функціональність, наприклад, виведення повідомлення чи повернення значень.

Завдання 2. (Автопідключення класів за допомогою spl_autoload_register. PHPDoc)

- Додайте PHPDoc коментарі до всіх класів, вказавши їх призначення та властивості.
- Створіть файл **autoload.php**, який буде містити функцію для автопідключення класів.
- Використайте **spl_autoload_register** для автоматичного підключення класів на основі їхніх імен та розташування.

Завдання 3. (Неймспейси)

- Додайте неймспейси до класів у попередньому завданні. Наприклад, "namespace Models;" для "UserModel".
- Змініть файл **autoload.php** так, щоб він також враховував неймспейси при підключенні класів

Завдання 4. (Автопідключення класів з неймспейсами)

- Використовуйте аналогічний підхід до підключення класів, але тепер з урахуванням неймспейсів.
- Переконайтеся, що класи виводять повідомлення чи результати виклику.

Завдання 5 (Створення класу. Методи GET і SET)

1) Створіть клас **Circle** з полями: координати центру і радіус кола
2) Створіть конструктор, що приймає значення для 3-х полів
3) Створіть метод **__toString()**, що повертає рядок в форматі: «Коло з центром в (x, y) і радіусом radius»
4) Створіть методи **GET** і **SET** для всіх 3-х полів
5) Створіть об'єкт та перевірте всі його методи

Завдання 6 (Модифікатори доступу)

1) В класі з попереднього завдання зробіть всі поля private.
2) Створіть метод, що приймає об'єкт коло, і повертає **true**, якщо дані кола перетинаються, і **false**, якщо вони не перетинаються.

Завдання 7 (Статичні властивості і методи)

1) Створіть директорію **text**, а в ній 3 текстових файла
2) Створіть клас зі статичним полем **dir="text"**
3) Створіть 2 статичних методи в класі: на читання та запис в файл:
   - Ім'я файлу передається як параметр метода.
   - В метод «**на запис в файл**» передається ще й рядок, який потрібно дописати в файл.
   - Директорія береться зі статичного поля
4) Створіть метод, що дозволяє стерти вміст файлу

Перевірте роботу всіх методів

Завдання 8 (Наслідування)

1) Створіть клас **Human** з властивостями, що характеризують людину (зріст, маса, вік…). Створіть методи **GET** і **SET** для кожної властивості
2) Створіть клас **Student**, який успадковуватиметься від класу **Human**:
   1. Додайте властивості, специфічні тільки для студента (назва ВНЗ, курс…)
   2. Додайте в клас методи **GET** і **SET** для всіх нових властивостей.
   3. Реалізуйте метод, який буде переводити студента на новий курс (тобто просто збільшувати значення поля «курс» на 1)
3) Створіть клас **Programmer**, який успадковуватиметься від класу **Human**:
   - Додайте властивості, специфічні тільки для програміста (масив з мовами програмування, які він знає, досвід роботи…).
   - Додайте в клас методи **GET** і **SET** для всіх нових властивостей.
   - Реалізуйте метод, який буде додавати в масив з мовами ще одну мову.

Перевірте роботу всіх класів і всіх методів. Не забудьте змінити зріст і масу у студентів і програмістів, скориставшись методами з батьківського класу **Human**

Завдання 9 (Абстрактні класи)

1) Зробіть клас **Human** абстрактним.
2) Напишіть метод «**Народження дитини**» в класі **Human**, що викликає метод «**Повідомлення при народженні дитини**» (не забудьте поставити модифікатор **protected**), який буде абстрактним
3) Реалізуйте «**Повідомлення при народженні дитини**» у класів **Student** та **Programmer**

Перевірте роботу методів «народження»

Завдання 10 (Інтерфейси)

1) Створіть інтерфейс «**Прибирання будинку**», в якому опишіть 2 методи: «**Прибирання кімнати**» і «**Прибирання кухні**»
2) Додайте створений інтерфейс в клас **Human**
3) Реалізуйте у кожному класі-спадкоємці (**Student** та **Programmer**) обидва методи
4) Реалізація повинна бути у вигляді одного з рядків: «**Студент прибирає кімнату**», «**Студент прибирає кухню**», «**Програміст прибирає кімнату**», «**Програміст прибирає кухню**»,
5) Перевірте роботу методів прибирання в обох класах

У цьому завданні я створила невеликий PHP-проект, який демонструє ключові аспекти мови, включаючи:

- **Автозавантаження класів:** Для зручної організації коду та уникнення ручного підключення кожного файлу з класом.
- **Модель-Вигляд-Контролер (MVC):** Для розділення логіки застосунку, відображення даних та обробки дій користувача.
- **Просторові імена (Namespaces):** Для організації класів у логічні групи та уникнення конфліктів імен.
- **Основи об'єктно-орієнтованого програмування (ООП):** Класи, об'єкти, властивості, методи, конструктори, успадкування (inheritance), абстрактні класи, інтерфейси.
- **Статичні методи:** Для створення утилітарних класів з методами, які не потребують створення об'єкта.

Основні виконані частини:

1. **Реалізація автозавантаження класів (autoload.php):** Я створила скрипт autoload.php, який використовує функцію spl_autoload_register() для автоматичного підключення файлів класів при їх першому використанні. Автозавантажувач шукає класи спочатку у відповідних директоріях просторів імен (Controllers, Models, Views), потім у директорії Classes та, нарешті, у кореневій директорії:

```
// autoload.php
```

```
spl_autoload_register(function ($className) {
    // ... логіка пошуку та підключення файлу класу ...
});
```

Це значно спрощує керування залежностями між файлами.

2. **Розробка базової структури MVC (Controllers, Models, Views):** Я створила три директорії (Controllers, Models, Views) для організації класів за їх роллю в архітектурі MVC. У кожній директорії є базовий клас (UserController, UserModel, UserView) з простором імен. Це демонструє розділення відповідальності:

```
// Controllers/UserController.php
namespace Controllers;
class UserController {
    public function processRequest() {
        return "Запит оброблено контролером";
    }
}

// Models/UserModel.php
namespace Models;
class UserModel {
    public function getMessage() {
        return "Повідомлення з моделі користувача";
    }
}

// Views/UserView.php
namespace Views;
class UserView {
    public function render() {
        return "Це відображення для користувача";
    }
}
```

3. **Створення утилітарного класу FileManager зі статичними методами:** Я розробила клас FileManager з статичними методами (writeToFile, readFromFile, clearFile) для виконання базових операцій з файлами. Статичні методи дозволяють використовувати ці функції без необхідності створювати екземпляр класу:

```
// FileManager.php
class FileManager {
    public static $dir = 'text';
    public static function writeToFile($filename, $text) { /* ... */ }
    public static function readFromFile($filename) { /* ... */ }
    public static function clearFile($filename) { /* ... */ }
}
```

4. **Реалізація класу Circle:** Я створила клас Circle для представлення кола з властивостями (координати центру та радіус) та методами для їх отримання та встановлення, а також методом intersects() для перевірки

перетину з іншим колом та магічним методом __toString() для рядкового представлення об'єкта:

```
// Circle.php
class Circle {
    private $x;
    private $y;
    private $radius;
    public function __construct($x, $y, $radius) { /* ... */ }
    public function getX() { /* ... */ }
    // ... інші гетери та сетери ...
    public function __toString() { /* ... */ }
    public function intersects(Circle $other) { /* ... */ }
}
```

5. **Демонстрація успадкування та інтерфейсів (Human, Student, Programmer, Cleaning):** Я визначила інтерфейс Cleaning з методами cleanRoom() та cleanKitchen(). Абстрактний клас Human реалізує цей інтерфейс та містить загальні властивості для людей (зріст, вага, вік) та абстрактний метод birthMessage(). Класи Student та Programmer успадковують від Human та реалізують специфічну поведінку, включаючи реалізацію birthMessage() та методів інтерфейсу Cleaning:

```
// Classes/Cleaning.php
interface Cleaning {
    public function cleanRoom();
    public function cleanKitchen();
}

// Classes/Human.php
abstract class Human implements Cleaning {
    // ... властивості та методи ...
    abstract protected function birthMessage();
}

// Classes/Student.php
class Student extends Human {
    // ... властивості та методи ...
    protected function birthMessage() { return "Студент народив дитину!"; }
    public function cleanRoom() { return "Студент прибирає кімнату"; }
    public function cleanKitchen() { return "Студент прибирає кухню"; }
}

// Classes/Programmer.php
class Programmer extends Human {
    // ... властивості та методи ...
    protected function birthMessage() { return "Програміст народив дитину!"; }
    public function cleanRoom() { return "Програміст прибирає кімнату"; }
    public function cleanKitchen() { return "Програміст прибирає кухню"; }
}
```

6. **Використання створених класів у головному скрипті (index.php):** У файлі index.php я демонструю роботу всіх створених класів,

включаючи використання класів MVC, статичних методів FileManager, класу Circle та класів, що демонструють успадкування та інтерфейси.

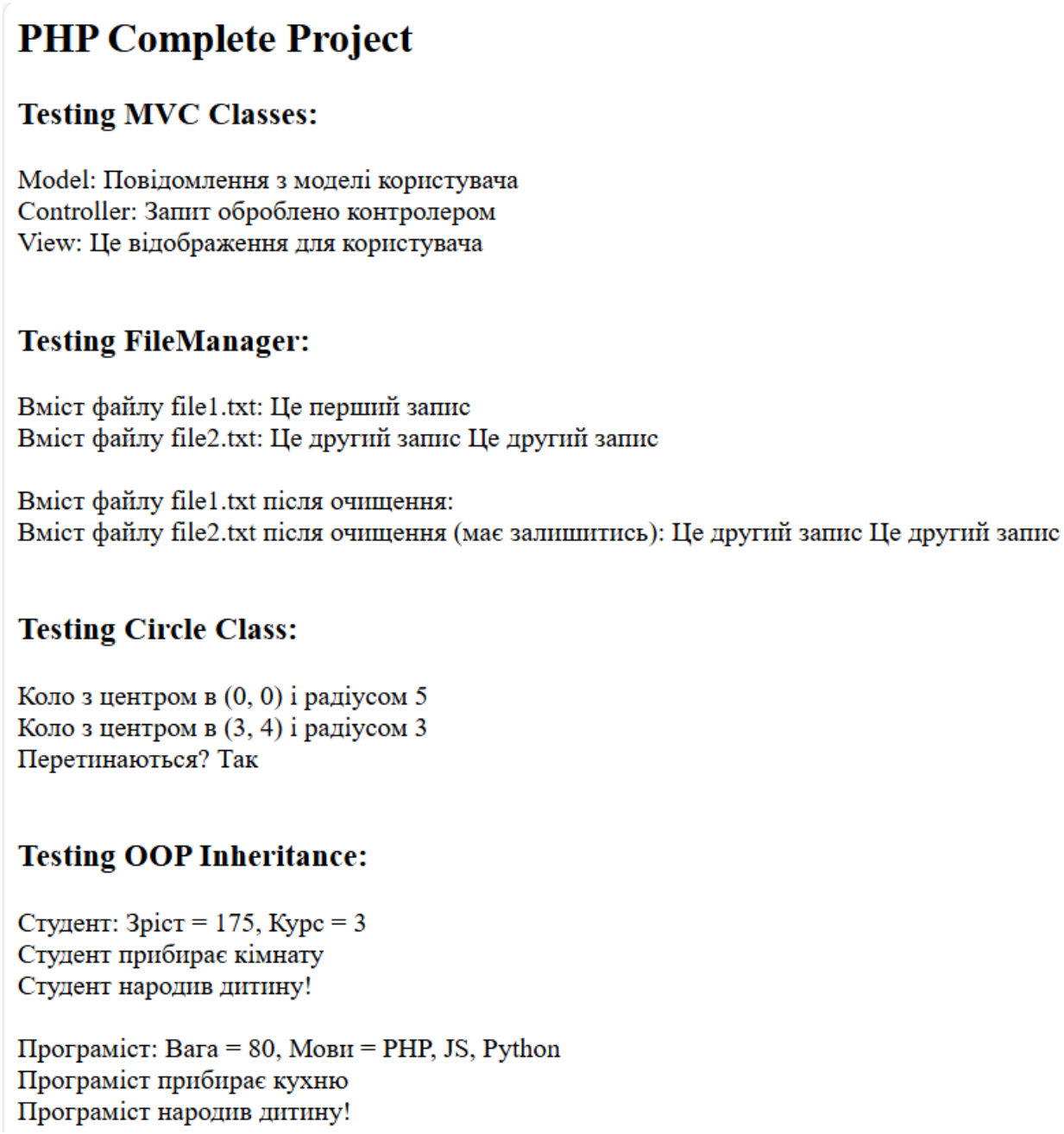

Рис. 1 Результат виконання роботи

Висновок: На сьогоднішньому зайнятті я навчилася працювати з класами