# ЛАБОРАТОРНА РОБОТА № 4

## Об’єктно-орієнтоване програмування в PHP

***Мета:*** 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.

*Результат виконання:*

1. Створюємо пустий проєкт PHP, а в ньому каталоги "Models", "Controllers" та "Views".

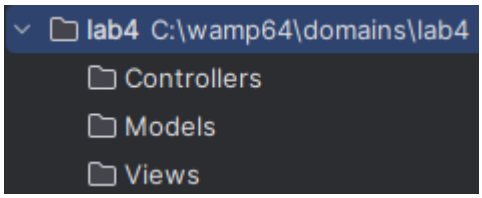


Рис.1.1 Створені пусті каталоги в проєкті

2. У кожному каталозі створюємо по одному класу.

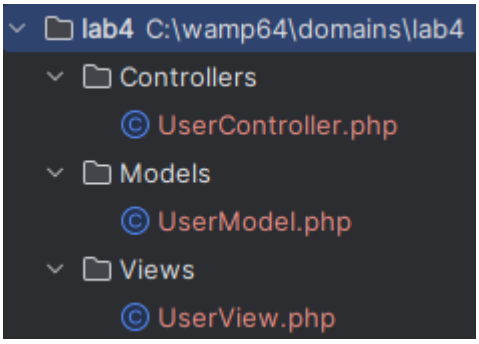


Рис.1.2 Створені класи у кожному з каталогів.

3. Реалізуємо просту функціональність.

*Лістинг коду:*

***UserModel.php***

```php
<?php
class UserModel
{
    public function getUserInfo() {
        return "Hello! I`m User:)";
    }
}
```

***UserController.php***

```php
<?php
class UserController
{
    public function showUserInfo() {
        $userModel = new UserModel();
        $userInfo = $userModel->getUserInfo();
        echo $userInfo;
    }
}
```

***UserView.php***

```php
<?php
class UserView
{
    public function renderUserInfo($userInfo) {
        echo "<h5>$userInfo</h5>";
    }
}
```

***task***

```php
<title>Task1[ClassOrganization]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
include 'templates/parts/menu.php';

require_once 'Models/UserModel.php';
require_once 'Controllers/UserController.php';
require_once 'Views/UserView.php';

$userController = new UserController();
$userController->showUserInfo();

$userView = new UserView();
$userView->renderUserInfo("↑Information about user↑");
```

*Результат виконання програми:*

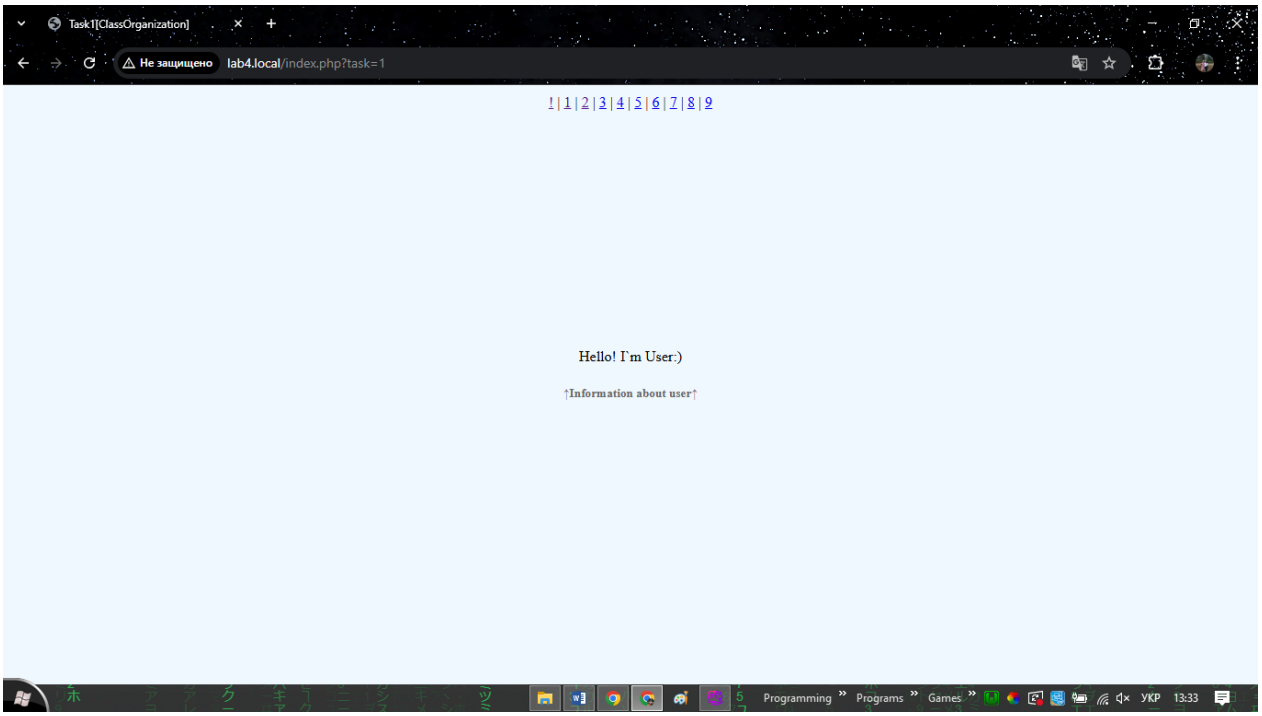


Рис.1.3 Перевірка роботи класів

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.

*Результат виконання:*

1. До всіх класів додаємо PHPDoc коментарі.

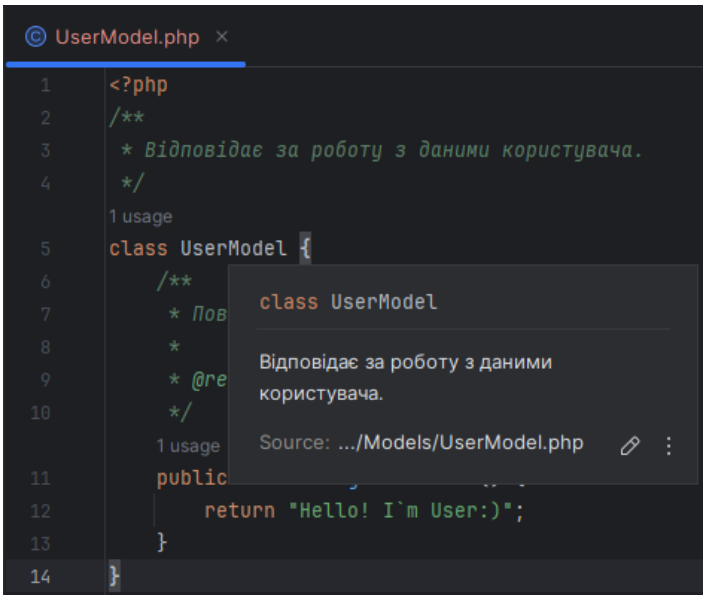


Рис.2.1 Відображення PHPDoc коментарю до класу UserModel

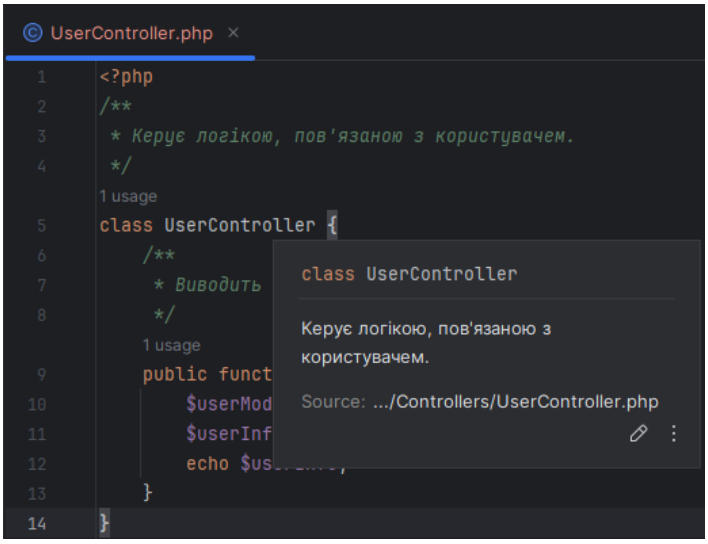


Рис.2.2 Відображення PHPDoc коментарю до класу UserController

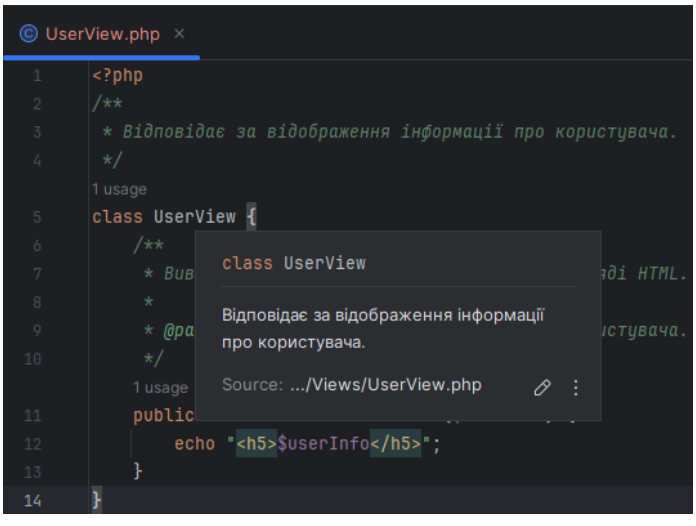


Рис.2.3 Відображення PHPDoc коментарю до класу UserView

2. Створюємо файл **autoload.php**.
3. Використовуємо **spl_autoload_register** для автоматичного підключення класів на основі їхніх імен та розташування.

*Лістинг коду:*

***UserModel.php***

```php
<?php
/**
 * Відповідає за роботу з даними користувача.
 */
class UserModel {
    /**
     * Повертає інформацію про користувача.
     *
     * @return string Інформація про користувача.
     */
    public function getUserInfo() {
        return "Hello! I`m User:)";
    }
}
```

***UserController.php***

```php
<?php
/**
 * Керує логікою, пов'язаною з користувачем.
 */
class UserController {
    /**
     * Виводить інформацію про користувача.
     */
    public function showUserInfo() {
        $userModel = new UserModel();
        $userInfo = $userModel->getUserInfo();
        echo $userInfo;
    }
}
```

***UserView.php***

```php
<?php
/**
 * Відповідає за відображення інформації про користувача.
 */
class UserView {
    /**
     * Виводить інформацію про користувача у вигляді HTML.
     *
     * @param string $userInfo Інформація про користувача.
     */
    public function renderUserInfo($userInfo) {
        echo "<h5>$userInfo</h5>";
    }
}
```

***autoload.php***

```php
<?php

spl_autoload_register(/**
 * @throws Exception
```

```
 */ function ($className) {
    $classMap = [
        'UserController' => 'Controllers/UserController.php',
        'UserModel' => 'Models/UserModel.php',
        'UserView' => 'Views/UserView.php',
    ];

    if (isset($classMap[$className])) {
        $filePath = __DIR__ . '/' . $classMap[$className];

        if (file_exists($filePath)) {
            require_once $filePath;
            return;
        }
    }
    throw new Exception("Помилка: клас '$className' не знайдено.");
});
```

***task***

```
<title>Task2[ClassAutoConnect]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
include 'templates/parts/menu.php';

require_once 'autoload.php';

$userController = new UserController();
$userController->showUserInfo();

$userView = new UserView();
$userView->renderUserInfo("↑USER MASSAGE↑");
```

*Результат виконання програми:*

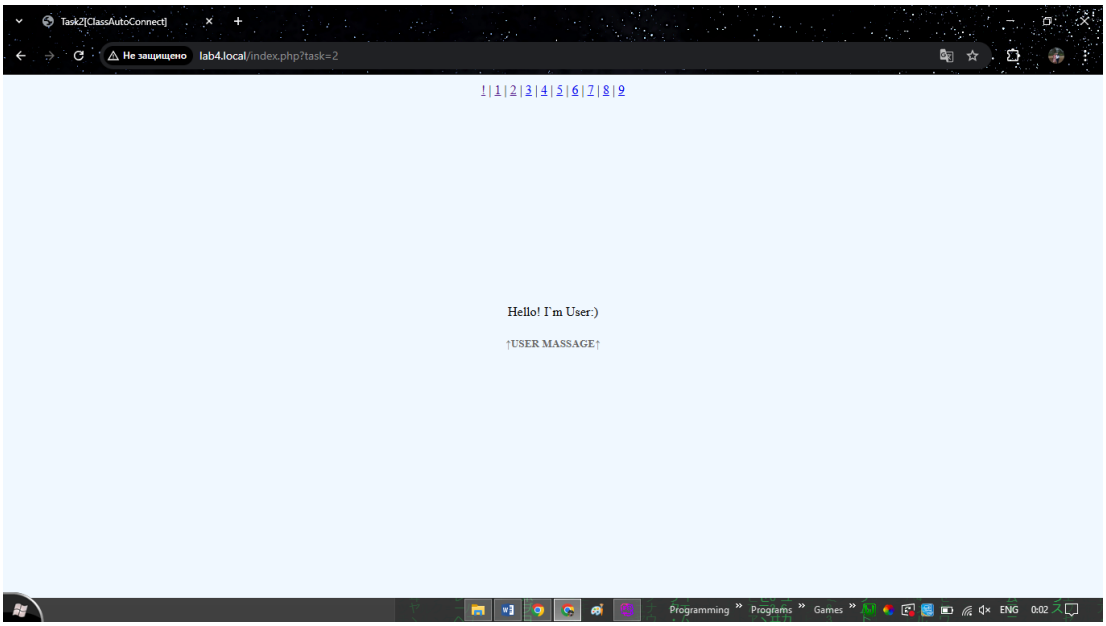


Рис.2.4 Перевірка роботи автопідключення класів

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.

*Лістинг коду:*

***UserModel.php***

```php
<?php
namespace Models;
/**
 * Відповідає за роботу з даними користувача.
 */
class UserModel {
    /**
     * Повертає інформацію про користувача.
     *
     * @return string Інформація про користувача.
     */
    public function getUserInfo() {
        return "Hello! I`m User:)";
    }
}
```

***UserController.php***

```php
<?php
namespace Controllers;
use Models\UserModel;
/**
 * Керує логікою, пов'язаною з користувачем.
 */
class UserController {
    /**
     * Виводить інформацію про користувача.
     */
    public function showUserInfo() {
        $userModel = new UserModel();
        $userInfo = $userModel->getUserInfo();
        echo $userInfo;
    }
}
```

***UserView.php***

```php
<?php
namespace Views;
/**
 * Відповідає за відображення інформації про користувача.
 */
class UserView {
    /**
     * Виводить інформацію про користувача у вигляді HTML.
     *
     * @param string $userInfo Інформація про користувача.
     */
    public function renderUserInfo($userInfo) {
        echo "<h5>$userInfo</h5>";
    }
}
```

***autoload.php***

```php
<?php
spl_autoload_register(function ($className) {
```

```
    $className = str_replace('\\', '/', $className);

    $filePath = __DIR__ . '/' . $className . '.php';

    if (file_exists($filePath)) {
        require_once $filePath;
    } else {
        throw new Exception("Помилка: клас '$className' не знайдено за шляхом:
$filePath");
    }
});
```

***task***

```
<title>Task3[Namespaces]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
include 'templates/parts/menu.php';

use Controllers\UserController;
use Views\UserView;

require_once 'autoload.php';

$userController = new UserController();
$userController->showUserInfo();

$userView = new UserView();
$userView->renderUserInfo("↑USER SAY↑");
```

*Результат виконання програми:*

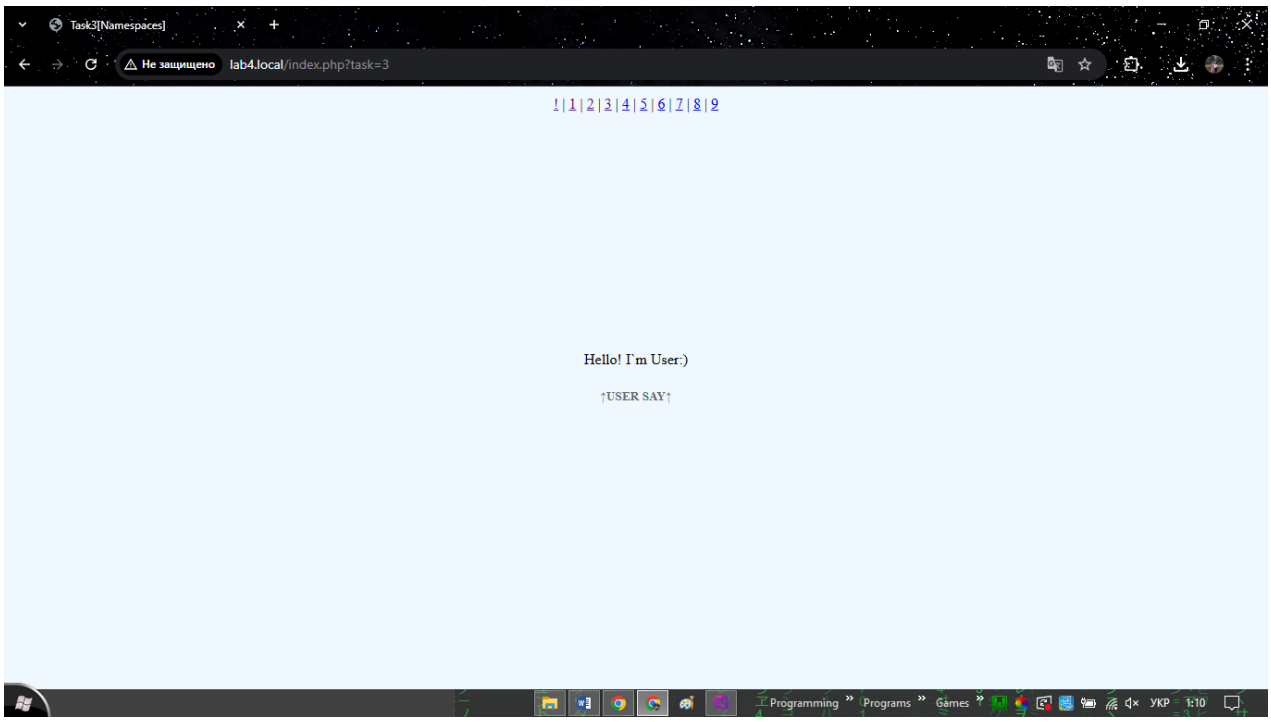


Рис.3.1 Перевірка роботи автопідключення класів

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.

*Лістинг коду:*

***UserModel.php***

```php
<?php
namespace Models;
/**
 * Відповідає за роботу з даними користувача.
 */
class UserModel {
    /**
     * Повертає інформацію про користувача.
     *
     * @return string Інформація про користувача.
     */
    public function getUserInfo() {
        return "Hello! I`m User:)";
    }
    public function displayMessage() {
        echo "<br>UserModel_Done✓<br>";
    }
}
```

***UserController.php***

```php
<?php
namespace Controllers;
use Models\UserModel;
/**
 * Керує логікою, пов'язаною з користувачем.
 */
class UserController {
    /**
     * Виводить інформацію про користувача.
     */
    public function showUserInfo() {
        $userModel = new UserModel();
        $userInfo = $userModel->getUserInfo();
        echo $userInfo . "<br>";
    }
    public function displayMessage() {
        echo "<br>UserController_Done✓<br>";
    }
}
```

## *UserView.php*

```php
<?php
namespace Views;
/**
 * Відповідає за відображення інформації про користувача.
 */
class UserView {
    /**
     * Виводить інформацію про користувача у вигляді HTML.
     *
     * @param string $userInfo Інформація про користувача.
     */
    public function renderUserInfo($userInfo) {
        echo "<h5>$userInfo</h5>";
    }
    public function displayMessage() {
        echo "UserView_Done✓<br>";
    }
}
```

## *autoload.php*

```php
<?php
spl_autoload_register(function ($className) {
    $className = str_replace('\\', '/', $className);

    $filePath = __DIR__ . '/' . $className . '.php';

    if (file_exists($filePath)) {
        require_once $filePath;
    } else {
        throw new Exception("Помилка: клас '$className' не знайдено за шляхом:
$filePath");
    }
});
```

## *task*

```php
<title>Task4[AutoConnectClassesWithNamespaces]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
include 'templates/parts/menu.php';

require_once 'autoload.php';

use Controllers\UserController;
use Models\UserModel;
use Views\UserView;

$userController = new UserController();
$userController->showUserInfo();
$userController->displayMessage();

$userModel = new UserModel();
$userModel->displayMessage();

$userView = new UserView();
$userView->renderUserInfo("★USER INFORMATION★");
$userView->displayMessage();
```

*Результат виконання програми:*

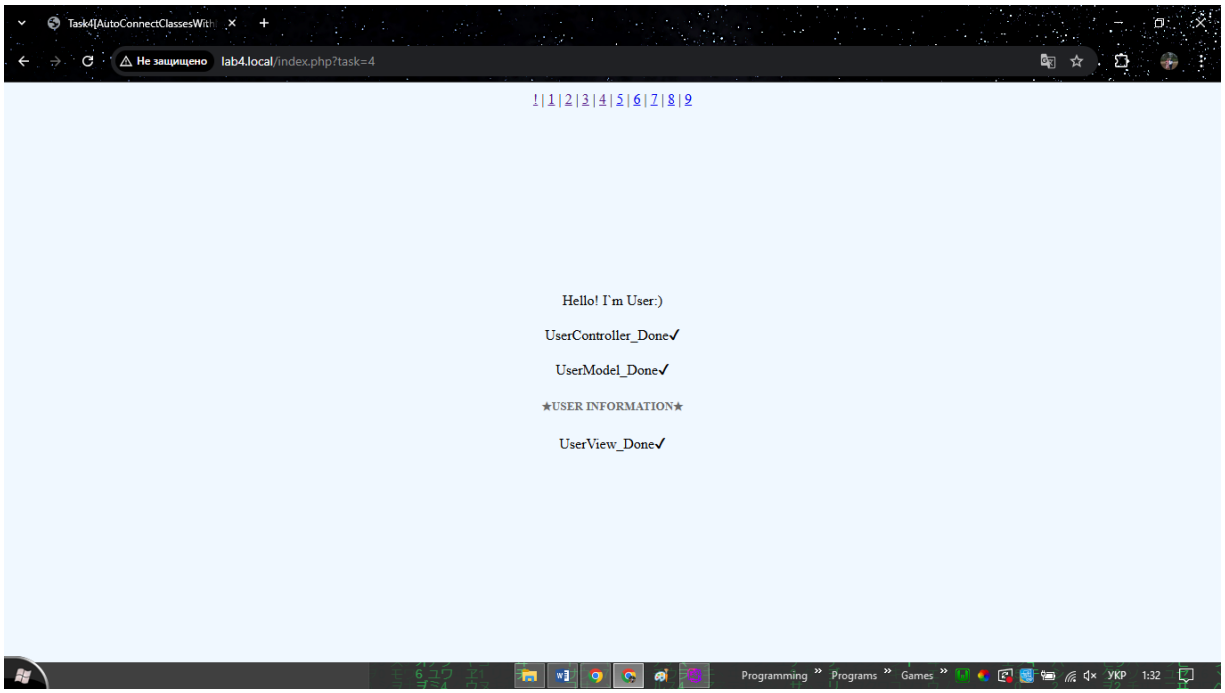


Рис.4.1 Перевірка роботи автопідключення класів

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.

*Лістинг коду:*

***Circle.php***

```php
<?php
class Circle {
    private $x;
    private $y;
    private $radius;

    public function __construct($x, $y, $radius) {
        $this->x = $x;
        $this->y = $y;
        $this->radius = $radius;
    }

    public function __toString() {
        return "о з центром в ($this->x, $this->y) і радіусом $this->radius";
    }
```

```
    //GET
    public function getX() {
        return $this->x;
    }
    public function getY() {
        return $this->y;
    }
    public function getRadius() {
        return $this->radius;
    }
    //SET
    public function setX($x) {
        $this->x = $x;
    }
    public function setY($y) {
        $this->y = $y;
    }
    public function setRadius($radius) {
        $this->radius = $radius;
    }
}
```

***task***

```
<title>Task5[ClassCreate.GET/SET]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php

use Classes\Circle;

include 'templates/parts/menu.php';

require_once 'Classes/Circle.php';

$circle = new Circle(3, 4, 5);
// __toString()
echo $circle . "<br><br>";
//GET
echo "Координата X: " . $circle->getX() . "<br>";
echo "Координата Y: " . $circle->getY() . "<br>";
echo "Радіус: " . $circle->getRadius() . "<br>";
//SET
$circle->setX(10);
$circle->setY(20);
$circle->setRadius(15);

echo "<br>" . $circle;
```

*Результат виконання програми:*

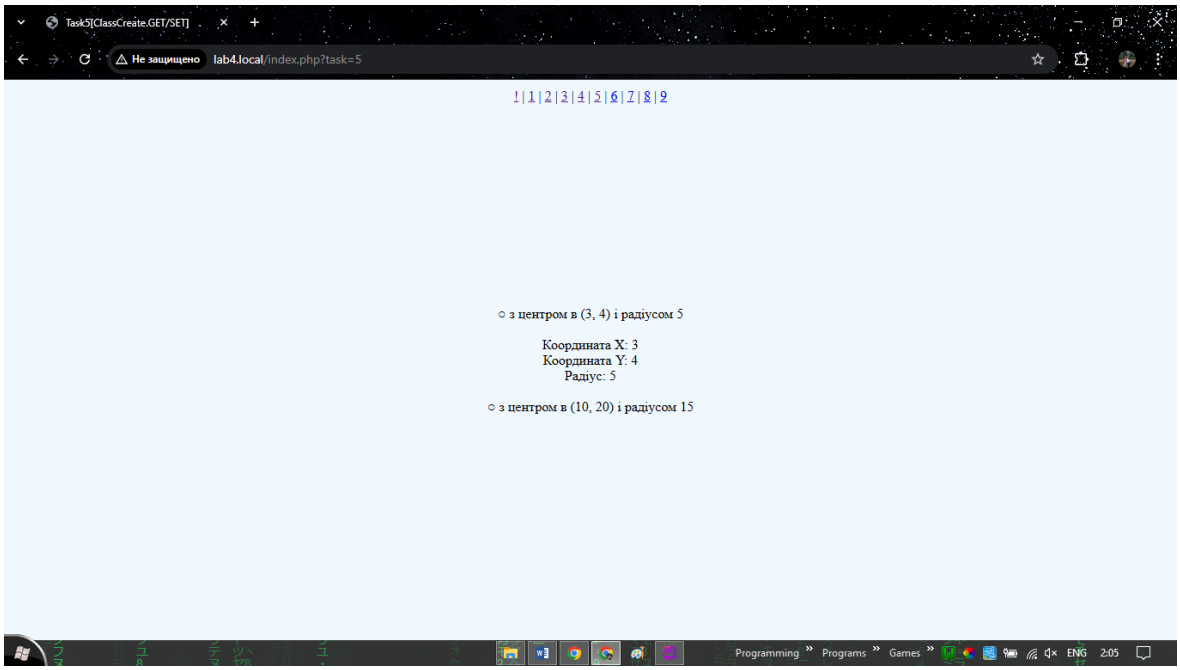


Рис.5.1 Перевірка роботи всіх методів

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.

*Лістинг коду::*

***Circle.php***

```php
<?php
namespace Classes;
class Circle {
    private $x;
    private $y;
    private $radius;

    public function __construct($x, $y, $radius) {
        $this->x = $x;
        $this->y = $y;
        $this->radius = $radius;
    }
    // __toString()
    public function __toString() {
        return "центр ($this->x, $this->y); радіус $this->radius";
    }
    //GET
    public function getX() {
        return $this->x;
    }
    public function getY() {
        return $this->y;
    }
    public function getRadius() {
```

```
        return $this->radius;
    }
    //SET
    public function setX($x) {
        $this->x = $x;
    }
    public function setY($y) {
        $this->y = $y;
    }
    public function setRadius($radius) {
        $this->radius = $radius;
    }

    public function intersects(Circle $otherCircle) {
        $dx = $this->x - $otherCircle->getX();
        $dy = $this->y - $otherCircle->getY();
        $distance = sqrt($dx * $dx + $dy * $dy);

        $radiusSum = $this->radius + $otherCircle->getRadius();
        $radiusDiff = abs($this->radius - $otherCircle->getRadius());

        return ($distance <= $radiusSum) && ($distance >= $radiusDiff);
    }
}
```

***task***

```
<title>Task6[AccessModifiers]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
use Classes\Circle;

include 'templates/parts/menu.php';

require_once 'Classes\Circle.php';

$circle1 = new Circle(0, 0, 5);
$circle2 = new Circle(4, 0, 3);

echo "o1: $circle1<br><br>";
echo "o2: $circle2<br><br>";

if ($circle1->intersects($circle2)) {
    echo "Кола перетинаються.<br><br>";
} else {
    echo "Кола не перетинаються.<br><br>";
}

$circle2->setX(10);
$circle2->setY(0);

echo "o2 (після зміни): $circle2<br><br>";

if ($circle1->intersects($circle2)) {
    echo "Кола перетинаються.<br>";
} else {
    echo "Кола не перетинаються.<br>";
}
```

*Результат виконання програми:*

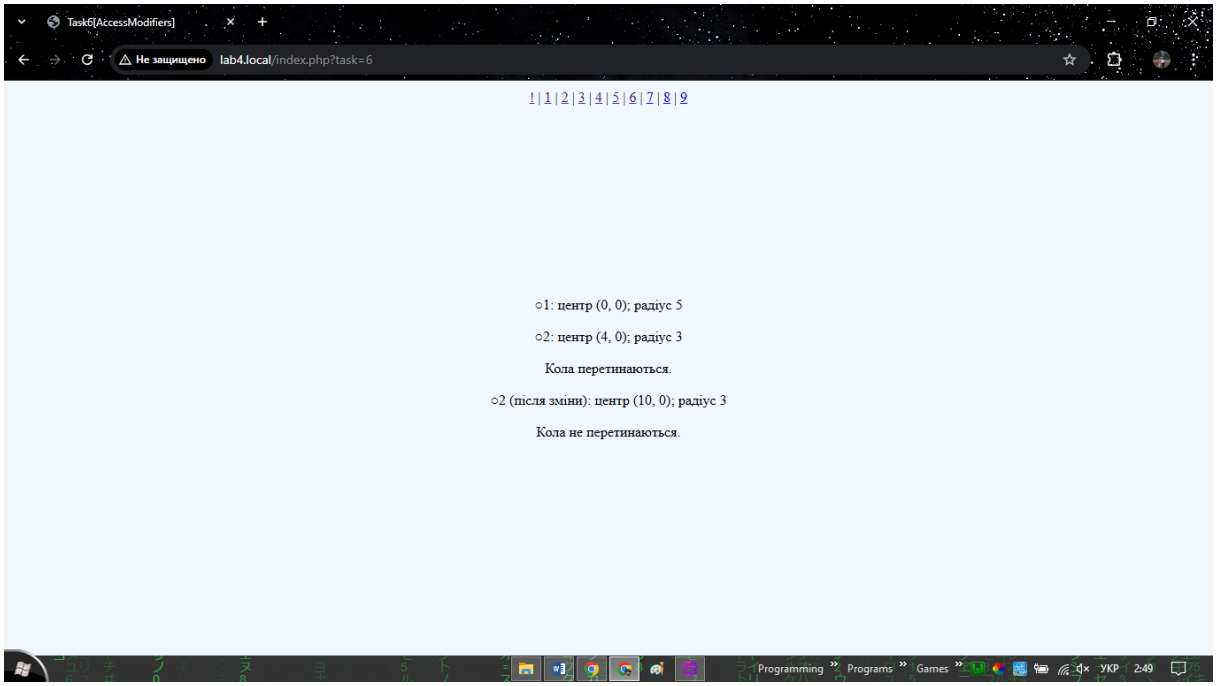


Рис.6.1 Перевірка роботи всіх методів

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.

*Результат виконання:*

1. Створюємо директорію text, а в ній 3 текстових файла.

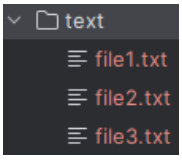


Рис.7.1 Директорія з новостворенимим текстовими файлами

2. Створюємо клас зі статичним полем dir=”text”.
3. Створюємо 2 статичних методи в класі: на читання та запис в файл.

4. Створюємо метод, що дозволяє стерти вміст файлу.

*Лістинг коду:*

***FileHandler.php***

```php
<?php
namespace Classes;
class FileHandler {
    public static $dir = "text";
    //ReadFile
    public static function readFile($filename) {
        $filePath = self::$dir . "/" . $filename;
        if (file_exists($filePath)) {
            return file_get_contents($filePath);
        } else {
            return "Файл $filename не знайдено.<br>";
        }
    }
    //RecFile
    public static function writeFile($filename, $content) {
        $filePath = self::$dir . "/" . $filename;
        file_put_contents($filePath, $content, FILE_APPEND);
        return "Дані успішно додані до файлу $filename.<br>";
    }
    //DelFile
    public static function clearFile($filename) {
        $filePath = self::$dir . "/" . $filename;
        if (file_exists($filePath)) {
            file_put_contents($filePath, "");
            return "Вміст файлу $filename успішно очищено.<br>";
        } else {
            return "Файл $filename не знайдено.<br>";
        }
    }
}
```

***task***

```php
<title>Task7[StaticPropertiesAndMethods]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
include 'templates/parts/menu.php';

use Classes\FileHandler;

require_once 'Classes/FileHandler.php';

echo "Читання файлу file1.txt:";
echo FileHandler::readFile("file1.txt") . "<br><br>";

echo "Запис у файл file1.txt:<br>";
echo FileHandler::writeFile("file1.txt", "Новий рядок для file1.txt\n") . "<br>";

echo "Читання файлу file1.txt після запису:<br>";
echo FileHandler::readFile("file1.txt") . "<br><br>";

echo "Очищення файлу file1.txt:<br>";
echo FileHandler::clearFile("file1.txt") . "<br>";

echo "Читання файлу file1.txt після очищення:<br>";
echo FileHandler::readFile("file1.txt") . "<br><br>";
```

Результат виконання програми:

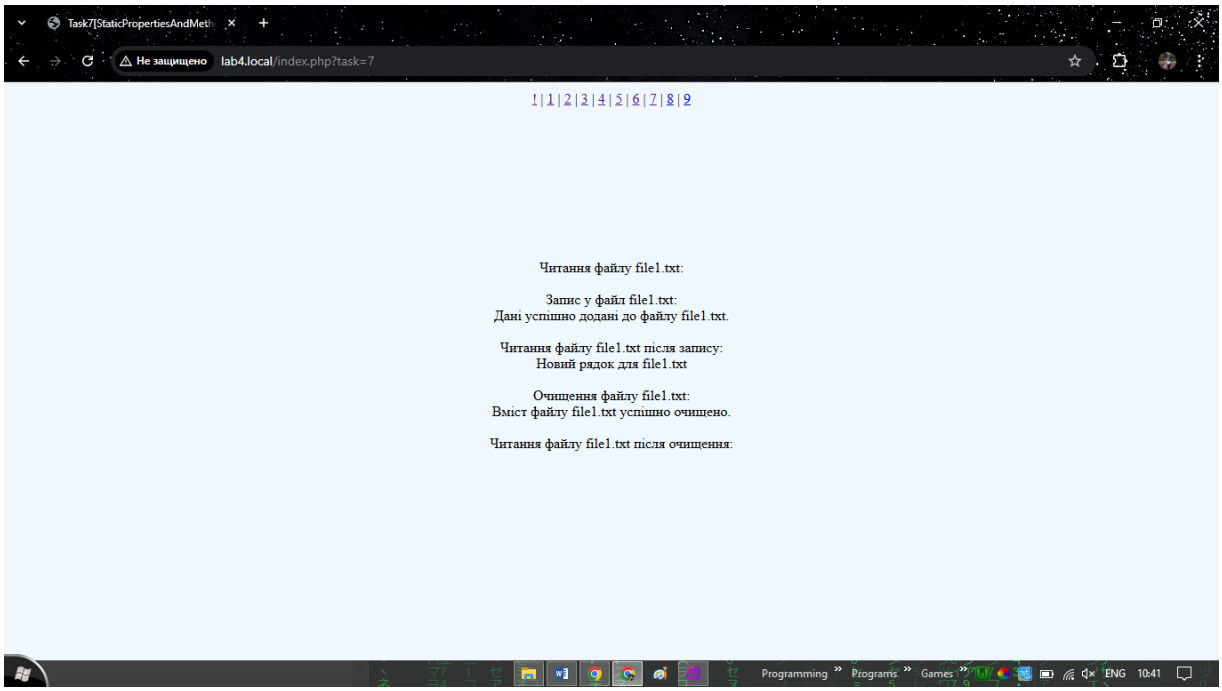


Рис.7,2 Перевірка роботи всіх методів

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**:
   - Додайте властивості, специфічні тільки для студента (назва ВНЗ, курс…).
   - Додайте в клас методи **GET** і **SET** для всіх нових властивостей.
   - 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**.

*Лістинг коду:*

***Human.php***

```php
<?php
namespace Classes;
class Human {
    private $height;
    private $weight;
    private $age;

    public function __construct($height, $weight, $age) {
        $this->height = $height;
        $this->weight = $weight;
        $this->age = $age;
    }
    //GET
    public function getHeight() {
        return $this->height;
    }
    public function getWeight() {
        return $this->weight;
    }
    public function getAge() {
        return $this->age;
    }
    //SET
    public function setHeight($height) {
        $this->height = $height;
    }
    public function setWeight($weight) {
        $this->weight = $weight;
    }
    public function setAge($age) {
        $this->age = $age;
    }
}
```

***Student.php***

```php
<?php
namespace Classes;
require_once 'Human.php';
class Student extends Human {
    private $university;
    private $course;

    public function __construct($height, $weight, $age, $university, $course) {
        parent::__construct($height, $weight, $age);
        $this->university = $university;
        $this->course = $course;
    }
    //GET
    public function getUniversity() {
        return $this->university;
    }
    public function getCourse() {
```

```
        return $this->course;
    }
    //SET
    public function setUniversity($university) {
        $this->university = $university;
    }
    public function setCourse($course) {
        $this->course = $course;
    }
    //NewCourse
    public function promoteToNextCourse() {
        $this->course++;
    }
}
```

***Programmer.php***

```
<?php
namespace Classes;
require_once 'Human.php';
class Programmer extends Human {
    private $languages = [];
    private $experience;

    public function __construct($height, $weight, $age, $languages, $experience) {
        parent::__construct($height, $weight, $age);
        $this->languages = $languages;
        $this->experience = $experience;
    }
    //GET
    public function getLanguages() {
        return $this->languages;
    }
    public function getExperience() {
        return $this->experience;
    }
    //SET
    public function setLanguages($languages) {
        $this->languages = $languages;
    }
    public function setExperience($experience) {
        $this->experience = $experience;
    }
    //NewProgrammingLanguage
    public function addLanguage($language) {
        $this->languages[] = $language;
    }
}
```

***task***

```
<title>Task8[Imitation]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
include 'templates/parts/menu.php';

use Classes\Programmer;
use Classes\Student;

require_once 'Classes/Student.php';
require_once 'Classes/Programmer.php';

$student = new Student(180, 70, 20, "Київський університет", 2);

echo "<b>Студент:</b>";
```

```php
echo "Зріст: " . $student->getHeight() . " см<br>";
echo "Маса: " . $student->getWeight() . " кг<br>";
echo "Вік: " . $student->getAge() . " років<br>";
echo "ВНЗ: " . $student->getUniversity() . "<br>";
echo "Курс: " . $student->getCourse() . "<br>";

$student->promoteToNextCourse();
echo "Після переходу на новий курс: " . $student->getCourse() . "<br><br>";

$programmer = new Programmer(175, 65, 25, ["PHP", "JavaScript"], 3);

echo "<b>Програміст:</b>";
echo "Зріст: " . $programmer->getHeight() . " см<br>";
echo "Маса: " . $programmer->getWeight() . " кг<br>";
echo "Вік: " . $programmer->getAge() . " років<br>";
echo "Мови програмування: " . implode(", ", $programmer->getLanguages()) . "<br>";
echo "Досвід роботи: " . $programmer->getExperience() . " роки<br>";

$programmer->addLanguage("Python");
echo "Після додавання нової мови: " . implode(", ", $programmer->getLanguages()) .
"<br>";
```

*Результат виконання програми:*

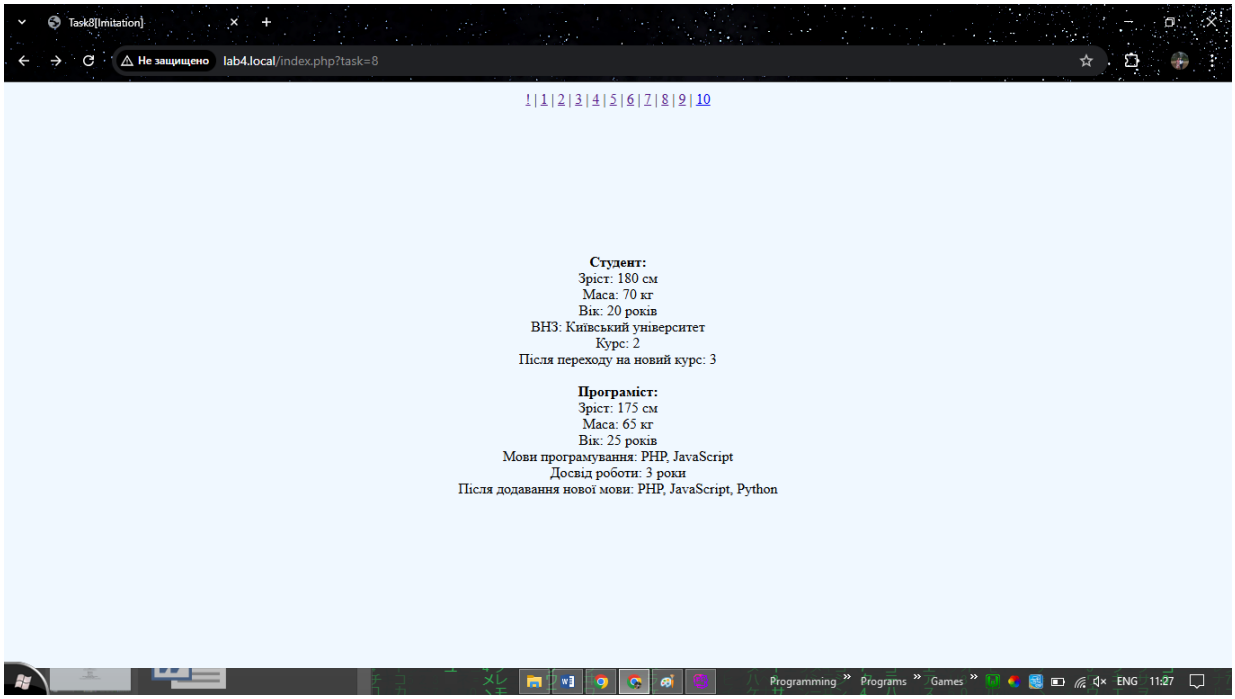


Рис.8.1 Перевірка роботи класів

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».

*Лістинг коду:*

***Human.php***

```php
<?php
namespace Classes;
abstract class Human {
    private $height;
    private $weight;
    private $age;

    public function __construct($height, $weight, $age) {
        $this->height = $height;
        $this->weight = $weight;
        $this->age = $age;
    }
    //GET
    public function getHeight() {
        return $this->height;
    }
    public function getWeight() {
        return $this->weight;
    }
    public function getAge() {
        return $this->age;
    }
    //SET
    public function setHeight($height) {
        $this->height = $height;
    }
    public function setWeight($weight) {
        $this->weight = $weight;
    }
    public function setAge($age) {
        $this->age = $age;
    }
    //ChildBirth
    public function giveBirth() {
        echo $this->onChildBirth() . "<br>";
    }

    abstract protected function onChildBirth();
}
```

***Student.php***

```php
<?php
namespace Classes;
require_once 'Classes/Human.php';
class Student extends Human {
    private $university;
    private $course;

    public function __construct($height, $weight, $age, $university, $course) {
        parent::__construct($height, $weight, $age);
        $this->university = $university;
        $this->course = $course;
    }
    //GET
    public function getUniversity() {
        return $this->university;
    }
```

```
    public function getCourse() {
        return $this->course;
    }
    //SET
    public function setUniversity($university) {
        $this->university = $university;
    }
    public function setCourse($course) {
        $this->course = $course;
    }
    //NewCourse
    public function promoteToNextCourse() {
        $this->course++;
    }
    //AbstractMethod
    protected function onChildBirth() {
        return "Студент народив дитину! Він/вона тепер має більше відповідаль-
ності.";
    }
}
```

***Programmer.php***

```
<?php
namespace Classes;
require_once 'Classes/Human.php';
class Programmer extends Human {
    private $languages = [];
    private $experience;

    public function __construct($height, $weight, $age, $languages, $experience) {
        parent::__construct($height, $weight, $age);
        $this->languages = $languages;
        $this->experience = $experience;
    }
    //GET
    public function getLanguages() {
        return $this->languages;
    }
    public function getExperience() {
        return $this->experience;
    }
    //SET
    public function setLanguages($languages) {
        $this->languages = $languages;
    }
    public function setExperience($experience) {
        $this->experience = $experience;
    }
    //NewProgrammingLanguage
    public function addLanguage($language) {
        $this->languages[] = $language;
    }
    //AbstractMethod
    protected function onChildBirth() {
        return "Програміст народив дитину! Тепер він/вона має менше часу на коду-
вання.";
    }
}
```

***task***

```
<title>Task9[AbstractClasses]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php
```

```php
include 'templates/parts/menu.php';

use Classes\Programmer;
use Classes\Student;

require_once 'Classes/Student.php';
require_once 'Classes/Programmer.php';

$student = new Student(180, 70, 20, "Київський університет", 2);

echo "<b>Студент:</b>";
$student->giveBirth();

$programmer = new Programmer(175, 65, 25, ["PHP", "JavaScript"], 3);

echo "<br><b>Програміст:</b>";
$programmer->giveBirth();
```

*Результат виконання програми:*

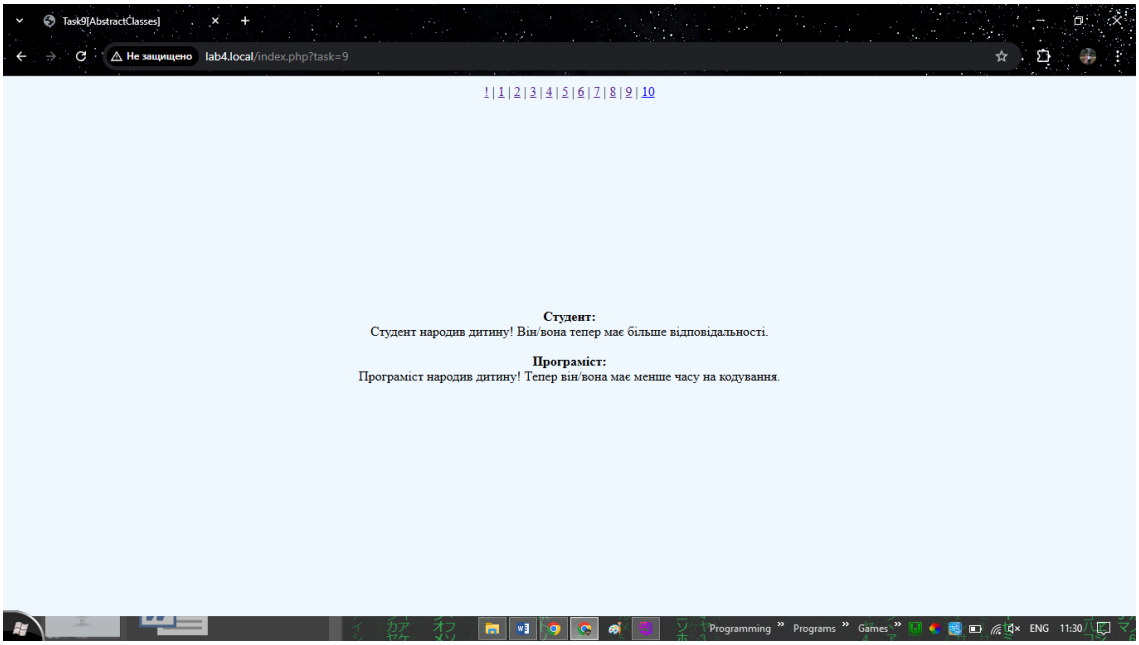


Рис.9.1 Перевірка роботи методів

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.

*Лістинг коду:*

***HouseCleaning.php***

```php
<?php
interface HouseCleaning {
    public function cleanRoom();
    public function cleanKitchen();
}
```

***Human.php***

```php
<?php
namespace Classes;
use HouseCleaning;

require_once 'interfaces/HouseCleaning.php';
abstract class Human implements HouseCleaning {
    private $height;
    private $weight;
    private $age;

    public function __construct($height, $weight, $age) {
        $this->height = $height;
        $this->weight = $weight;
        $this->age = $age;
    }
    //GET
    public function getHeight() {
        return $this->height;
    }
    public function getWeight() {
        return $this->weight;
    }
    public function getAge() {
        return $this->age;
    }
    //SET
    public function setHeight($height) {
        $this->height = $height;
    }
    public function setWeight($weight) {
        $this->weight = $weight;
    }
    public function setAge($age) {
        $this->age = $age;
    }
    //ChildBirth
    public function giveBirth() {
        echo $this->onChildBirth() . "<br>";
    }

    abstract protected function onChildBirth();
}
```

***Student.php***

```php
<?php
namespace Classes;
require_once 'Classes/Human.php';
class Student extends Human {
    private $university;
    private $course;

    public function __construct($height, $weight, $age, $university, $course) {
```

```
        parent::__construct($height, $weight, $age);
        $this->university = $university;
        $this->course = $course;
    }
    //GET
    public function getUniversity() {
        return $this->university;
    }
    public function getCourse() {
        return $this->course;
    }
    //SET
    public function setUniversity($university) {
        $this->university = $university;
    }
    public function setCourse($course) {
        $this->course = $course;
    }
    //NewCourse
    public function promoteToNextCourse() {
        $this->course++;
    }
    //AbstractMethod
    protected function onChildBirth() {
        return "Студент народив дитину! Він/вона тепер має більше
відповідальності.";
    }
    //InterfaceMethods
    public function cleanRoom() {
        return "Студент прибирає кімнату.";
    }
    public function cleanKitchen() {
        return "Студент прибирає кухню.";
    }
}
```

***Programmer.php***

```
<?php
namespace Classes;
require_once 'Classes/Human.php';
class Programmer extends Human {
    private $languages = [];
    private $experience;

    public function __construct($height, $weight, $age, $languages, $experience) {
        parent::__construct($height, $weight, $age);
        $this->languages = $languages;
        $this->experience = $experience;
    }
    //GET
    public function getLanguages() {
        return $this->languages;
    }
    public function getExperience() {
        return $this->experience;
    }
    //SET
    public function setLanguages($languages) {
        $this->languages = $languages;
    }
    public function setExperience($experience) {
        $this->experience = $experience;
    }
```

```
    //NewProgrammingLanguage
    public function addLanguage($language) {
        $this->languages[] = $language;
    }
    //AbstractMethod
    protected function onChildBirth() {
        return "Програміст народив дитину! Тепер він/вона має менше часу на коду-
вання.";
    }
    //InterfaceMethods
    public function cleanRoom() {
        return "Програміст прибирає кімнату.";
    }
    public function cleanKitchen() {
        return "Програміст прибирає кухню.";
    }
}
```

***task***

```
<title>Task10[Interfaces]</title>
<link rel="stylesheet" href="/templates/styles/style.css">
<?php

use Classes\Programmer;
use Classes\Student;

include 'templates/parts/menu.php';

require_once 'Classes/Student.php';
require_once 'Classes/Programmer.php';

$student = new Student(180, 70, 20, "Київський університет", 2);

echo "<b>Студент:</b>";
echo $student->cleanRoom() . "<br>";
echo $student->cleanKitchen() . "<br>";

$programmer = new Programmer(175, 65, 25, ["PHP", "JavaScript"], 3);

echo "<br><b>Програміст:</b>";
echo $programmer->cleanRoom() . "<br>";
echo $programmer->cleanKitchen() . "<br>";
```

*Результат виконання програми:*

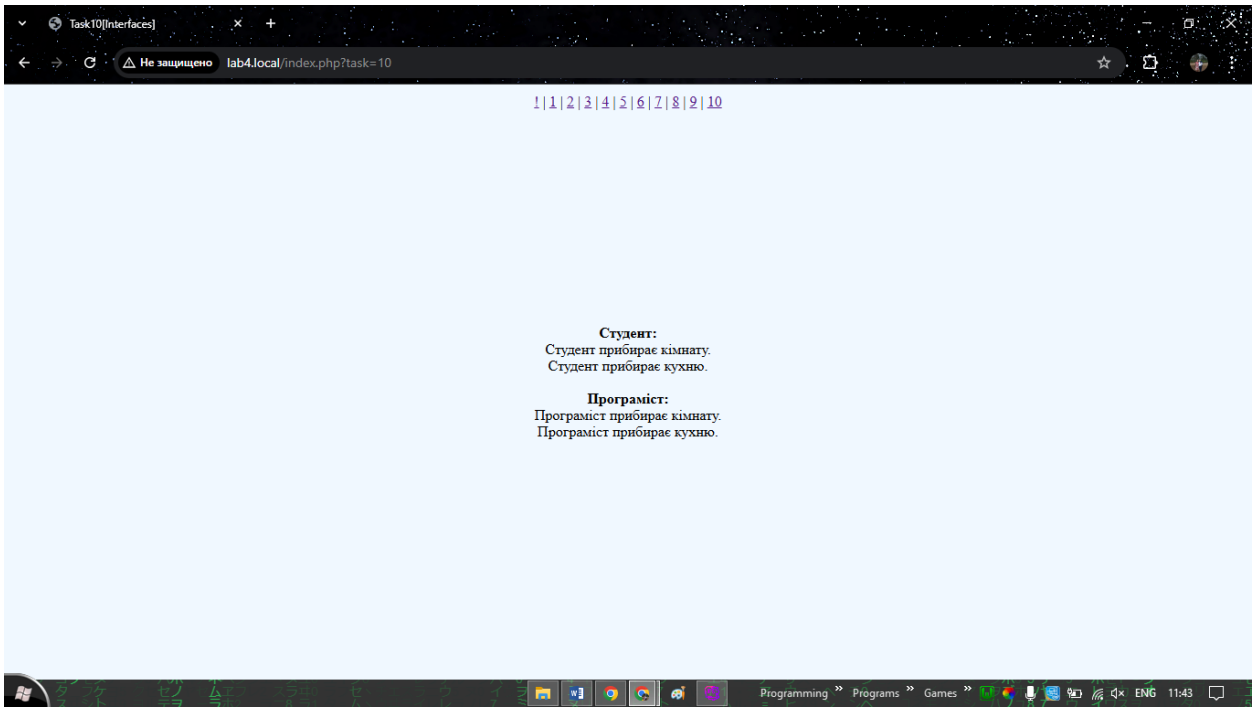


Рис.10.1 Перевірка роботи методів в класах

***Висновки:*** навчилися працювати з класами.