**Name:** Bonifacio, Princess Anne A.

**Year, Course & blk:** 2nd Year, BSIT-01

**Subject: Object-Oriented Programming**

**WEEK2 | Laboratory Activity 2: Comprehensive Application of Access Modifiers, Constructors, Destructors, and Method Overloading**

**CODE:**

<?php

// Class definition for Book

class Book {

public $title; // Public property accessible from outside the class

protected $author; // Protected property accessible within the class and its subclasses

private $price; // Private property accessible only within the class

// Constructor to initialize book properties

public function \_\_construct($title, $author, $price) {

$this->title = $title;

$this->author = $author;

$this->price = $price;

}

// Method to get the details of the book

public function getDetails() {

return "Title: {$this->title}, Author: {$this->author}, Price: \${$this->price}";

}

// Method to update the price of the book

public function setPrice($price) {

$this->price = $price;

}

// Magic method \_\_call to handle calls to non-existent methods

public function \_\_call($name, $arguments) {

echo "Stock updated for '{$this->title}' with arguments: " . implode(", ", $arguments) . "\n";

}

}

// Class definition for Library

class Library {

public $name; // Public property accessible from outside the class

private $books = []; // Private array to store Book objects

// Constructor to initialize the library name

public function \_\_construct($name) {

$this->name = $name;

}

// Method to add a book to the library

public function addBook(Book $book) {

$this->books[$book->title] = $book;

}

// Method to remove a book from the library by title

public function removeBook($title) {

if (isset($this->books[$title])) {

unset($this->books[$title]);

echo "Book '$title' removed from the library.\n";

} else {

echo "Book '$title' not found in the library.\n";

}

}

// Method to list all books in the library

public function listBooks() {

echo "Books in the library:\n";

foreach ($this->books as $book) {

echo $book->getDetails() . "\n";

}

}

// Destructor to clear the library when it's no longer needed

public function \_\_destruct() {

echo "The library '{$this->name}' is now closed.\n";

}

}

// Implementation

// Create instances of Book

$book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", 10.99);

$book2 = new Book("1984", "George Orwell", 8.99);

// Create an instance of Library

$library = new Library("City Library");

// Add books to the library

$library->addBook($book1);

$library->addBook($book2);

// Update the price of a book

$book1->setPrice(12.99);

// Call a non-existent method on the Book object to trigger \_\_call()

$book1->updateStock(50);

// List all books in the library

$library->listBooks();

// Remove a book from the library

$library->removeBook("1984");

// List books after removal

echo "\nBooks in the library after removal:\n";

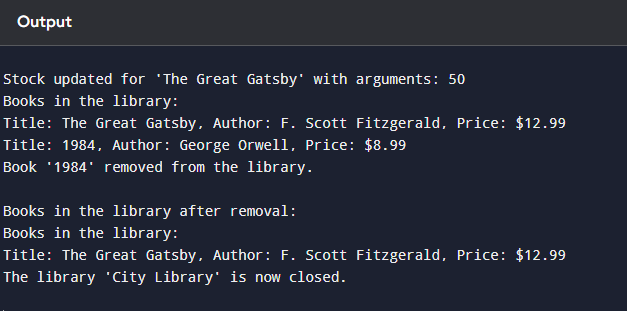
$library->listBooks();

// Destroy the Library object at the end to trigger the destructor

unset($library);

?>

**OUTPUT:**

****

**Brief Explanation of the Approach:**

In developing the library management system, we aimed to create a simple yet functional application using object-oriented programming (OOP) principles. Here's how each concept was applied in the provided code:

1. **Encapsulation and Access Modifiers**:
   * **Encapsulation**: Encapsulation was achieved by organizing the code into classes (Book and Library), each responsible for its own data and behavior.
   * **Access Modifiers**:
     + **Public**: Properties like title in the Book class and name in the Library class are public because they need to be accessible from outside the class. This allows us to directly access and modify these properties from any part of the program.
     + **Protected**: The author property in the Book class is protected, meaning it is accessible within the class and its subclasses. This restricts direct access from outside, providing controlled exposure.
     + **Private**: The price property in the Book class and the books array in the Library class are private. This ensures these properties can only be accessed or modified within the class itself, protecting the integrity of the data.
2. **Initialization and Cleanup**:
   * **Constructor (\_\_construct)**:
     + The constructor in the Book class initializes the title, author, and price properties when a new book object is created, ensuring each book has all its essential information set upon creation.
     + The constructor in the Library class sets the library's name and initializes an empty books array, preparing the library object to store books.
   * **Destructor (\_\_destruct)**:
     + The destructor in the Library class provides a cleanup mechanism by outputting a message when the library object is destroyed. This indicates that the library's resources are being released, which is crucial in managing the lifecycle of objects in PHP.
3. **Flexible Method Handling with Magic Methods**:
   * **\_\_call() Magic Method**:
     + The \_\_call() method in the Book class allows for handling calls to non-existent methods. In this scenario, it simulates method overloading by allowing us to call a method like updateStock() that doesn't actually exist in the class. This method outputs a message indicating that stock has been updated, providing a flexible way to extend the class's functionality.
4. **Library Operations**:
   * **addBook(Book $book)**: This method adds a Book object to the library's private books array. It associates each book with its title as the key, ensuring efficient access and management.
   * **removeBook($title)**: This method removes a book from the library by its title. It checks if the book exists in the library before attempting to remove it, providing feedback if the book is not found. This helps maintain the library's data integrity.
   * **listBooks()**: This method lists all the books currently in the library by iterating over the books array and calling the getDetails() method of each Book object. This showcases how the library interacts with the books while keeping the details encapsulated within the Book class.
5. **Implementation of the Solution**:
   * We created instances of Book and Library to represent individual books and the library itself.
   * Books were added to the library using the addBook() method, and the price of a book was updated using the setPrice() method.
   * We demonstrated the flexibility of the \_\_call() method by calling a non-existent method, updateStock(), which triggered the magic method.
   * The removeBook() method was used to remove a book from the library, and we listed the books before and after removal to show the changes.
   * Finally, the unset() function was called to destroy the Library object, which triggered the destructor and outputted the closing message.

**Summary:**

This approach effectively applies OOP principles to create a manageable and extendable library management system. Encapsulation ensures that data is protected and only accessible through defined interfaces. Access modifiers control the visibility of properties and methods, maintaining the integrity of the system. Constructors and destructors manage the initialization and cleanup of resources, while the \_\_call() magic method provides flexibility in handling method calls, simulating overloading where needed.