<?php

class Book {

public $title;

protected $author;

private $price;

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

$this->title = $title;

$this->author = $author;

$this->price = $price;

}

public function getDetails() {

return "Title: $this->title, Author: $this->author, Price: $" . number\_format($this->price, 2);

}

public function setPrice($price) {

$this->price = $price;

}

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

if ($name === 'updateStock') {

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

}

}

}

class Library {

private $books = [];

public $name;

public function \_\_construct($name) {

$this->name = $name;

}

public function addBook(Book $book) {

$this->books[] = $book;

}

public function removeBook($title) {

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

if ($book->title === $title) {

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

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

return;

}

}

}

public function listBooks() {

if ($this->books) {

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

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

}

} else {

echo "No books available in the Library.\n";

}

}

public function \_\_destruct() {

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

}

}

$book1 = new Book("The Great Gatsby", "F. Scott Fitzgerald", 12.99);

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

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

$library->addBook($book1);

$library->addBook($book2);

$book1->updateStock(50);

echo "Books in the Library:\n";

$library->listBooks();

$library->removeBook("1984");

echo "Books in the Library after removal:\n";

$library->listBooks();

unset($library);

?>

**Brief explanation of how you approached the problem, focusing on how each concept was used in your solution?:**

I tackled the problem by creating two main classes, `Book` and `Library`, each designed to manage specific data and functionality. I carefully used access modifiers to control how properties are accessed, ensuring data is kept safe and only modified in the right ways. I also included destructors to handle cleanup when objects are no longer in use, and I used the `\_\_call()` magic method to add flexibility by simulating method overloading.