Lecture 3

CST 226-2

Web Application Development

Sanjeewanie Senanayake

sanjeewanie@uwu.ac.lk

Department of Computer Science and Technology

Object-Oriented PHP

Lesson Learning Outcomes

» After successful completion of this lesson you will be able to,
- **Understand the core concepts** of object-oriented programming
- **Create** your own **classes** and **create objects** from those classes
- **Use the concept of encapsulation** to encapsulate data within objects and enforce data integrity
- **Implement inheritance in PHP** and use it effectively to reuse code and build class hierarchies
- **Define interfaces, implement them** in classes
- **Use abstract classes** as a foundation for concrete classes
- **Utilize polymorphism in PHP** and use it to have flexible and extensible code
- **Organize the PHP code** using namespaces
- **Apply OOP principles to** solve **real-world programming problems**

Lesson Outline

- » Object-Oriented Programming (OOP)
- » Core Principles of OOP
- » Creating and Using Classes in PHP
- » Data Hiding and Access Modifiers
- » Implementing Inheritance
- » Interfaces and Abstract Classes
- » Polymorphism in PHP
- » Code Organization and Dependency Management

Object-Oriented Programming

» Focuses on **organizing code around objects**

» Provides **a structured and modular approach** to software development

» Objects represent **real-world entities or concepts**

» **Encapsulate** both **data** (attributes or properties) and **behavior** (methods or functions)

» A **class serves as a blueprint or template** for creating objects

» It **defines the common attributes and methods** that objects of that class will have

Core Principles of OOP

» Classes:

- Classes are the **fundamental building blocks of OOP**
- **Define the structure and behavior** of objects
- A **blueprint that describes the properties and methods** that objects of that class will have

» Objects:

- Objects are **instances of classes**
- Represent **specific entities or instances of the class** and hold their own **unique state and behavior**
- Objects **interact with each other** by **invoking methods** and **exchanging messages**

Core Principles of OOP cont.

» Encapsulation:

- Encapsulation is the process of **bundling data and methods together within an object**
- It **hides the internal implementation details** and **exposes only the necessary interfaces to interact with the object**
- Encapsulation ensures **data security** and **code organization**

» Inheritance:

- Inheritance allows the **creation of new classes based on existing classes**
- It enables the **derived classes** (subclasses or child classes) **to inherit properties and methods from a base class** (superclass or parent class)
- Inheritance **promotes code reuse** and **allows the creation of specialized classes**

Core Principles of OOP cont.

» Polymorphism:

- Polymorphism means that **objects of different classes can be treated as objects of a common superclass**
- This allows **flexibility in writing code that can work with objects of different types**
- Also, this **promotes code modularity** and **simplifies adding new functionality**

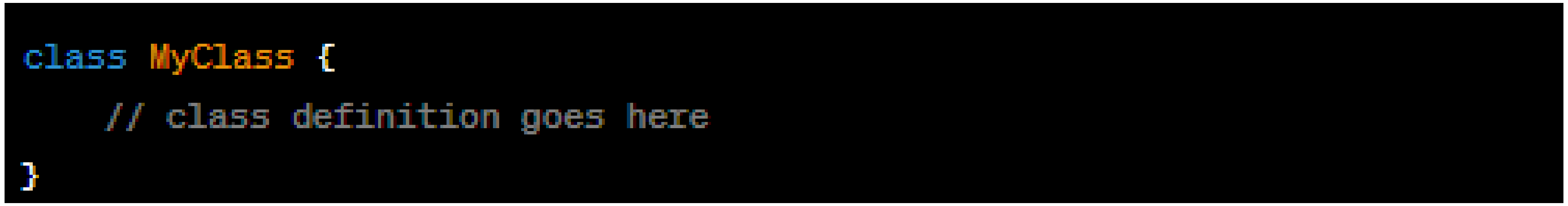
» Abstraction:

- Abstraction focuses on **defining the essential features of an object** and **hiding the unnecessary implementation details**
- Abstract classes and interfaces are **used to create common behaviors and characteristics** shared by multiple objects
- Abstraction allows programmers to **work with simplified models of complex systems**

Creating and Using Classes in PHP

» Class Declaration:

- Begin by **declaring a class** using the **class** keyword, followed by the name of the class
- Recommended to use **PascalCase for class names**

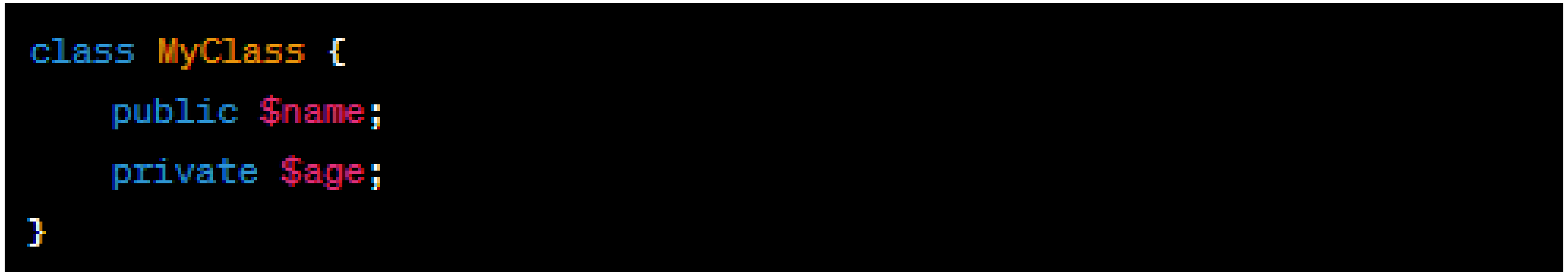


```
class MyClass {
    // class definition goes here
}
```

Creating and Using Classes in PHP cont.

» Properties:

- Define **the properties (also known as attributes or variables) that will hold the data** for each object of the class
- Properties are declared with the **public**, **protected**, or **private** visibility keywords, followed by the variable name
- Default is **public**

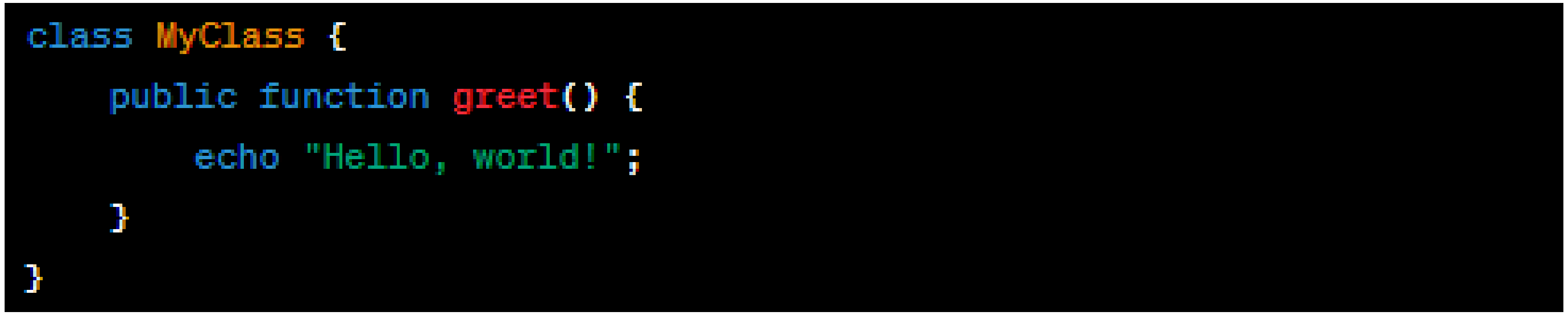


```
class MyClass {
    public $name;
    private $age;
}
```

Creating and Using Classes in PHP cont.

» Methods:

- Define **the methods (also known as functions) that will perform actions or provide behavior** for the objects of the class
- Methods are declared in a **similar way to regular functions**, but they are **written inside** the class

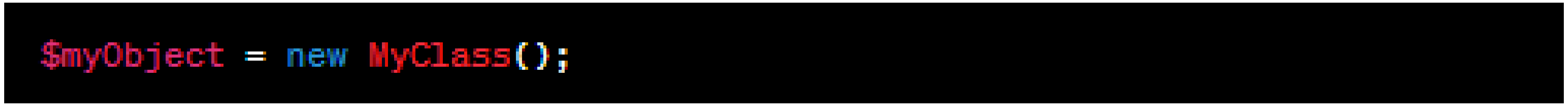


```
class MyClass {
    public function greet() {
        echo "Hello, world!";
    }
}
```

Creating and Using Classes in PHP cont.

» Creating Objects:

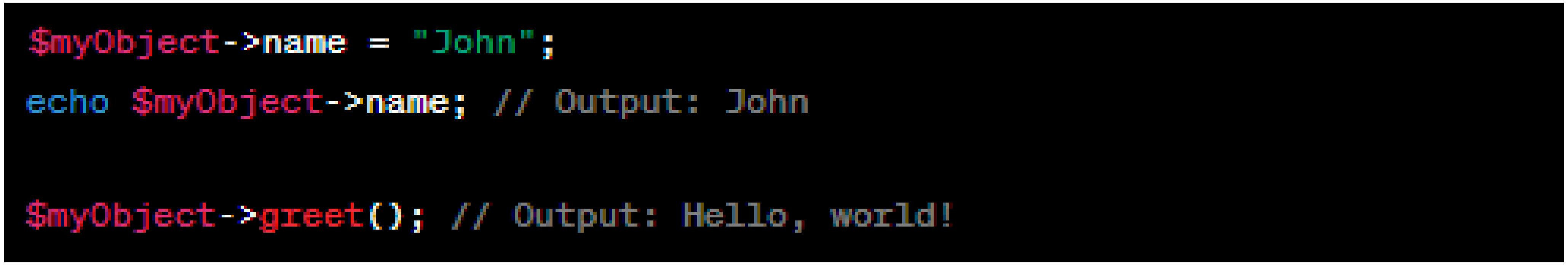
- To create an object of the class, **use the new keyword** followed by the class name and parentheses



```php
$myObject = new MyClass();
```

» Accessing Properties and Methods:

- **Use the object operator ->** to **access properties and methods** of the object

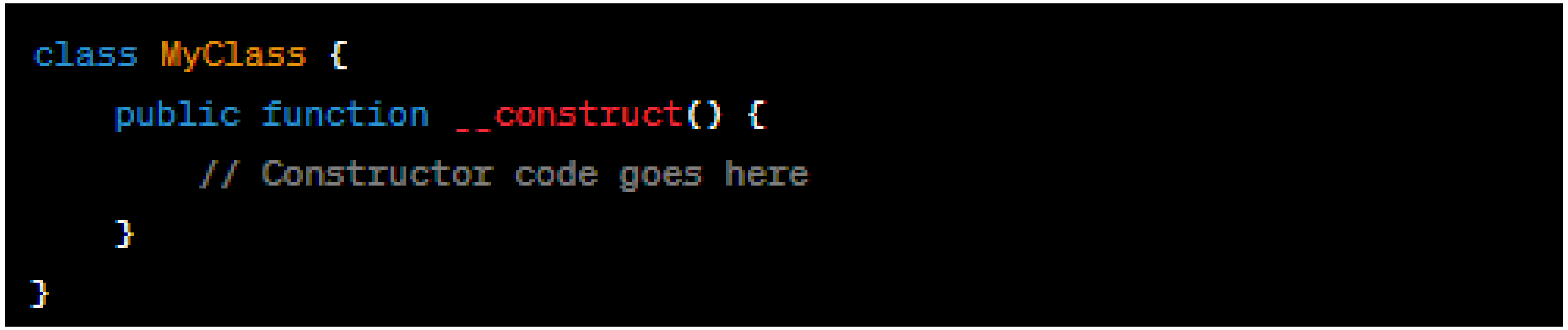


```php
$myObject->name = "John";
echo $myObject->name; // Output: John

$myObject->greet(); // Output: Hello, world!
```

Creating and Using Classes in PHP cont.

» Constructor:

- A constructor is **a special method that is automatically called when an object of a class is created**
- It is **used to initialize the object's state** and **perform any necessary setup operations**

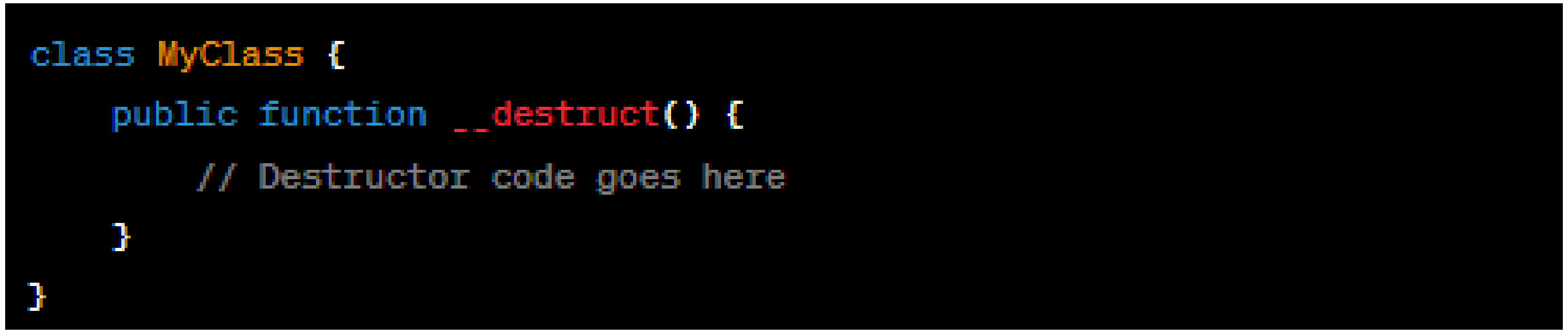


```
class MyClass {
    public function __construct() {
        // Constructor code goes here
    }
}
```

Creating and Using Classes in PHP cont.

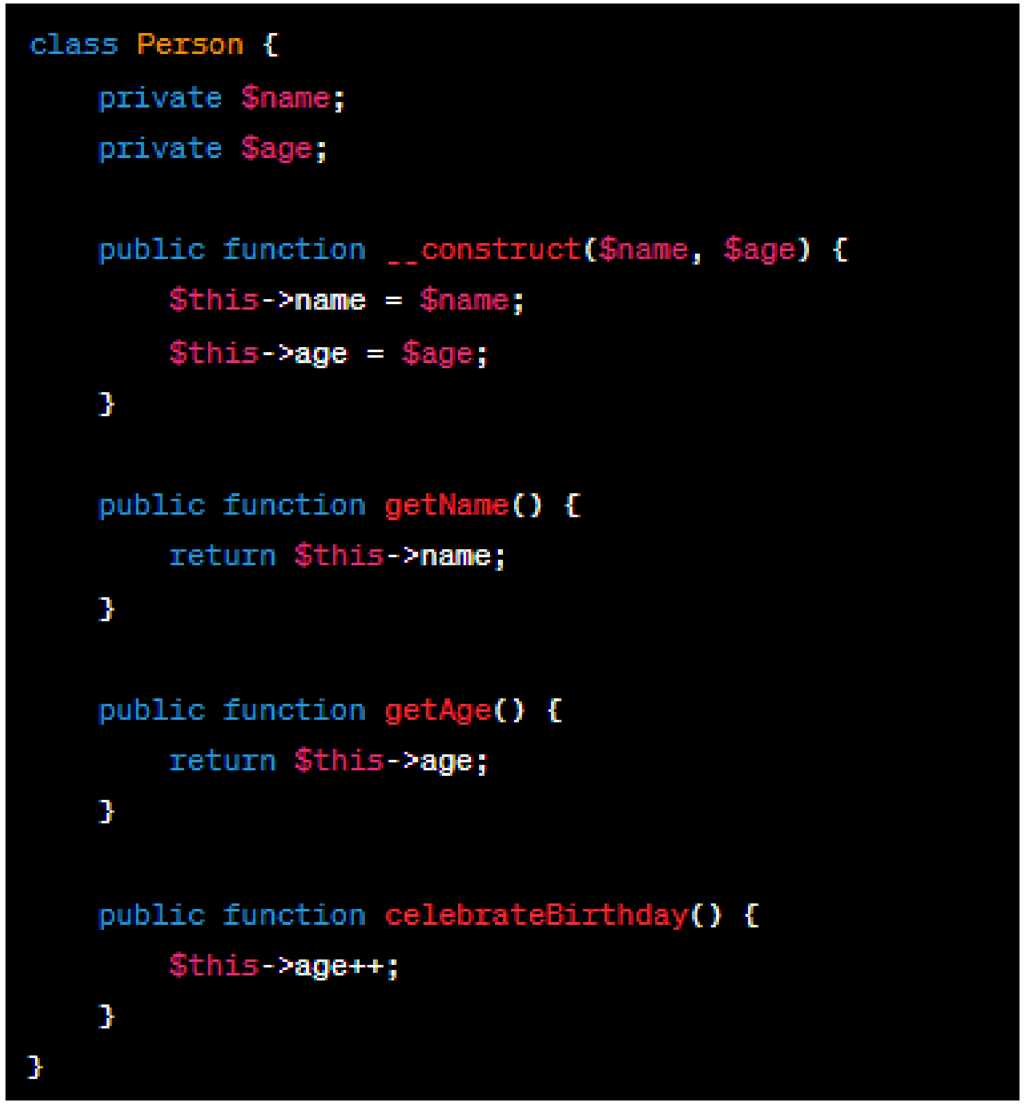
» Destructor:

- __destruct() is **automatically invoked when an object is no longer referenced** or **when the script execution ends**
- It is **used to perform cleanup tasks**, such as releasing resources (closing files, database connections, etc.) or freeing memory



```php
class MyClass {
    public function __destruct() {
        // Destructor code goes here
    }
}
```

Creating and Using Classes in PHP cont.



```php
class Person {
    private $name;
    private $age;

    public function __construct($name, $age) {
        $this->name = $name;
        $this->age = $age;
    }

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

    public function getAge() {
        return $this->age;
    }

    public function celebrateBirthday() {
        $this->age++;
    }
}
```

```php
$p = new Person(“Amali”, 22);
$name = $p->getName();
echo $name;
```

Activity 1

Creating a Book Class

» *Create a class named Book that represents a book.*

» *The Book class should have the following private properties:*
  - *title (string): to store the title of the book.*
  - *author (string): to store the author of the book.*
  - *year (int): to store the publication year of the book.*

» *Implement a constructor in the Book class that accepts the title, author, and year as parameters and initializes the corresponding properties.*

» *Implement getter methods for each property (getTitle(), getAuthor(), getYear()).*

» *Implement a setter method setYear($year) to update the publication year of the book.*

» *Create an instance of the Book class with the title “Hath Pana", author “Kamarathunga Munidasa", and year 1960.*

» *Display the details of the book by calling the getter methods (getTitle(), getAuthor(), getYear()).*

» *Update the publication year of the book to 1962 using the setter method setYear($year).*

» *Display the updated details of the book.*

Activity 2

Creating a Student Class

- *Create a class named Student that represents a student with the following properties:*
  - *name (string): to store the name of the student.*
  - *age (int): to store the age of the student.*
  - *grade (string): to store the grade level of the student.*
  - *subjects (array): to store an array of subjects the student is enrolled in.*
- *Implement the following methods in the Student class:*
  - *addSubject($subject): This method should add a subject to the student's list of subjects.*
  - *getSubjects(): This method should return the array of subjects the student is enrolled in.*
  - *getGrade(): This method should return the grade level of the student.*
  - *setGrade($grade): This method should set the grade level of the student.*
- *Create an instance of the Student class.*
- *Prompt the user to enter the name, age, and grade of the student.*
- *Use the setGrade() method to set the grade level of the student.*
- *Prompt the user to enter three subjects the student is enrolled in and use the addSubject() method to add them to the student's list of subjects.*
- *Display the student's name, age, grade, and subjects using the appropriate getter methods (getName(), getAge(), getGrade(), getSubjects()).*

Data Hiding and Access Modifiers

» In PHP, there are **three access modifiers**:

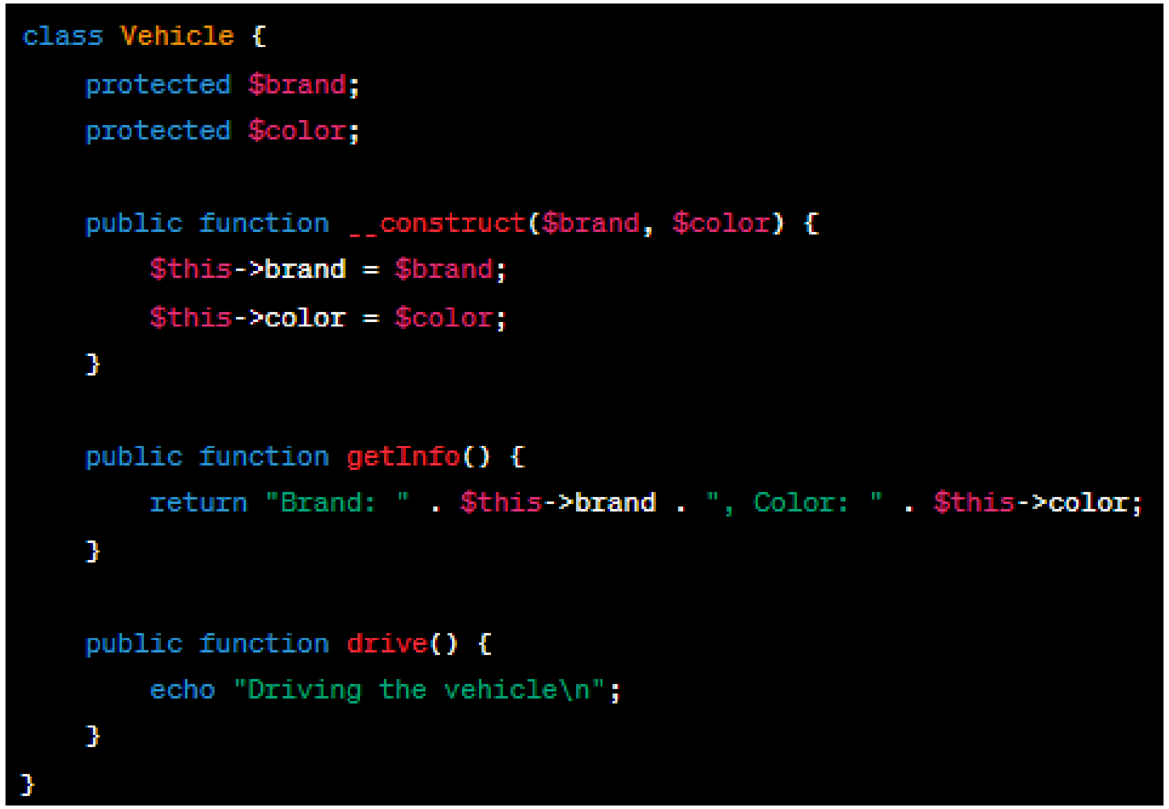
- **public**: When a property or method is declared as public, **it can be accessed from anywhere**
- **protected**: When a property or method is declared as protected, **it can only be accessed within the class itself and its subclasses (derived classes)**
- **private**: When a property or method is declared as private, **it can only be accessed within the class itself**

» By default, **if no access modifier is specified**, the **property or method is considered to be public**

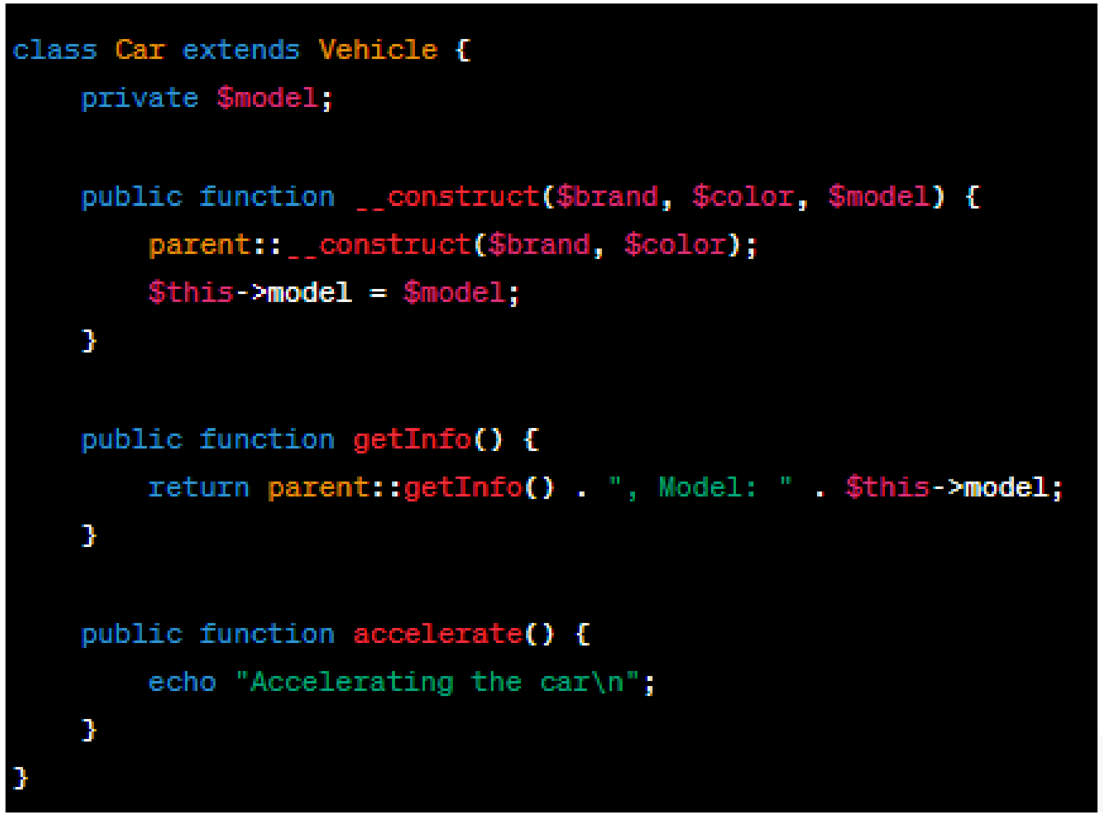
Implementing Inheritance

- » Inheritance is **a fundamental concept in object-oriented programming**
- » Allows classes to **inherit properties and methods from a parent** class
- » This **enables code reuse** and promoting **a hierarchical structure**
- » In PHP, you can **implement inheritance using the extends keyword**
- » **parent::** is **a keyword used to refer to the parent class or its members within a subclass**

Implementing Inheritance cont.



```php
class Vehicle {
    protected $brand;
    protected $color;

    public function __construct($brand, $color) {
        $this->brand = $brand;
        $this->color = $color;
    }

    public function getInfo() {
        return "Brand: " . $this->brand . ", Color: " . $this->color;
    }

    public function drive() {
        echo "Driving the vehicle\n";
    }
}
```



```php
class Car extends Vehicle {
    private $model;

    public function __construct($brand, $color, $model) {
        parent::__construct($brand, $color);
        $this->model = $model;
    }

    public function getInfo() {
        return parent::getInfo() . ", Model: " . $this->model;
    }

    public function accelerate() {
        echo "Accelerating the car\n";
    }
}
```

Activity 3

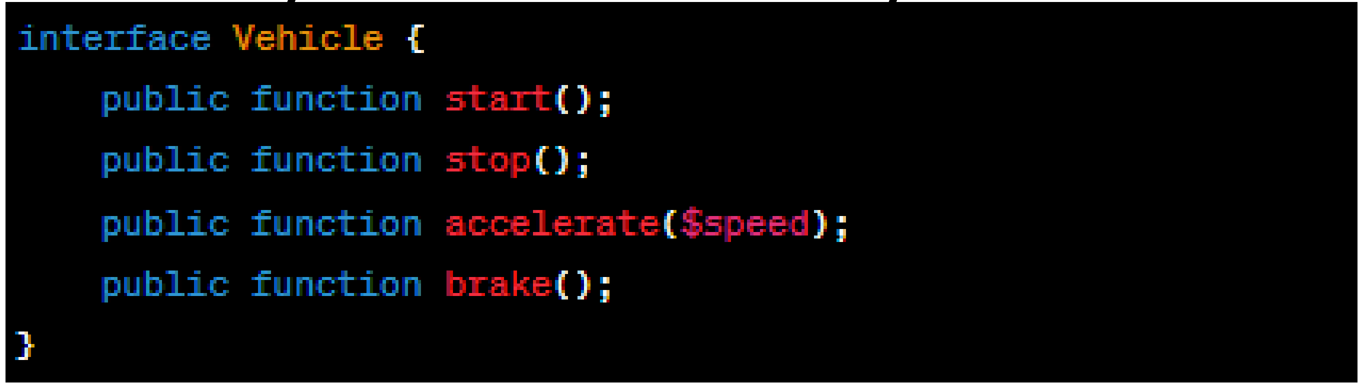
1. Create a base class called User.
   - The User class should have the following properties:
     - name (string): to store the name of the user.
     - email (string): to store the email address of the user.
   - Implement the following methods in the User class:
     - __construct($name, $email): This method should initialize the name and email properties.
     - getInfo(): This method should return a string that represents the user's name and email.
2. Create a class called Customer that inherits from the User class.
   - The Customer class should have an additional property:
     - customerId (string): to store the unique ID of the customer.
   - Implement the following methods in the Customer class:
     - __construct($name, $email, $customerId): This method should call the parent class's constructor and initialize the customerId property.
     - getInfo(): This method should return a string that represents the customer's name, email, and customer ID.

Activity 3 cont.

3. Create a class called Admin that also inherits from the User class.
    - The Admin class should have an additional property:
        - adminId (string): to store the unique ID of the admin.
    - Implement the following methods in the Admin class:
        - __construct($name, $email, $adminId): This method should call the parent class's constructor and initialize the adminId property.
        - getInfo(): This method should return a string that represents the admin's name, email, and admin ID.

Interfaces in PHP

- **A set of rules that classes must adhere to** or **a set of methods that a class must implement**
- **A blueprint for the methods that a class should provide**, without specifying the implementation details
- Interfaces are **useful for achieving abstraction**, **defining common behavior**, and **promoting code reusability**
- Use the **interface** keyword followed by **the interface name**



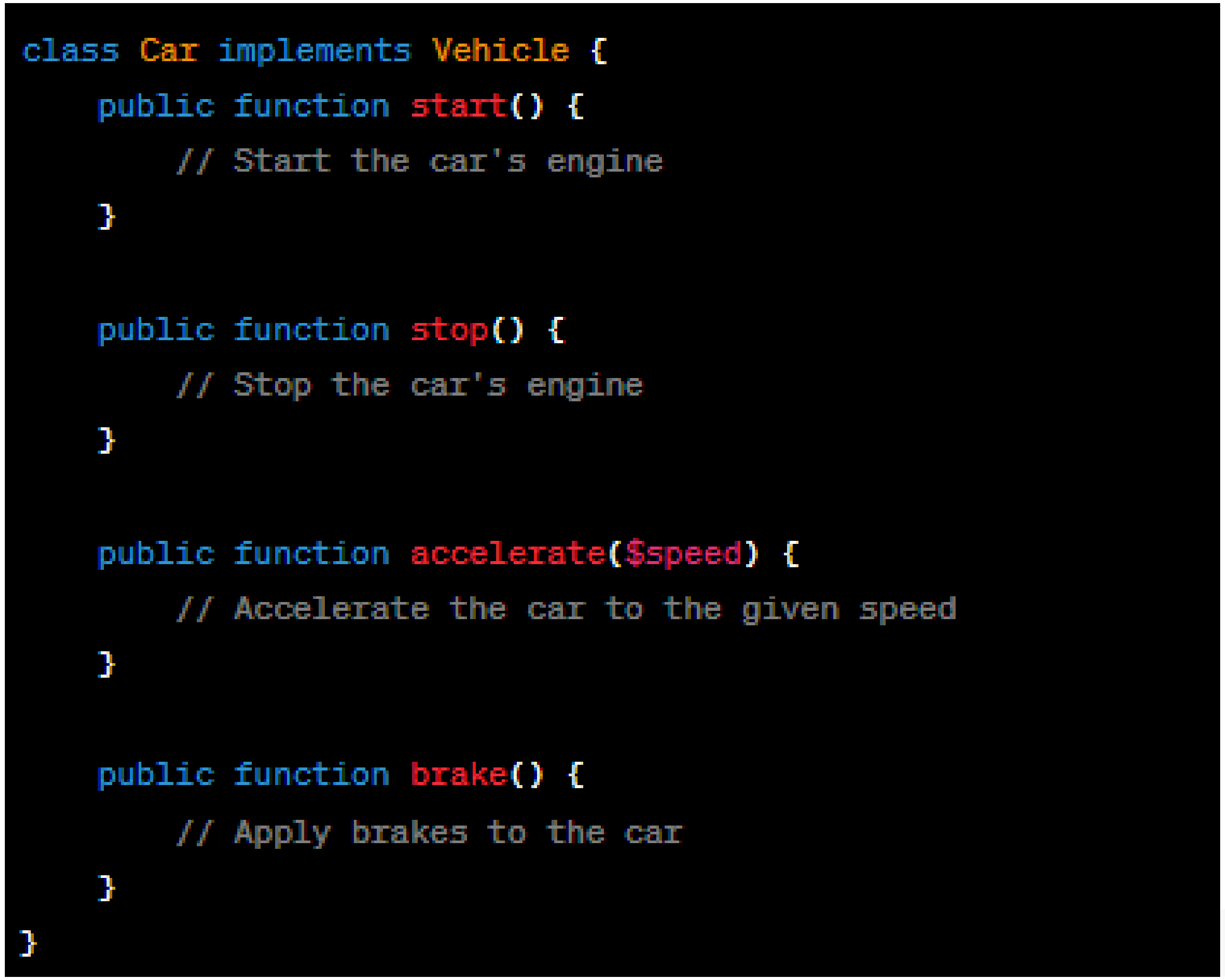
```php
interface Vehicle {
    public function start();
    public function stop();
    public function accelerate($speed);
    public function brake();
}
```

Interfaces in PHP cont.

» Use **implements** keyword followed by **the interface name**

» Interfaces are used to,

- Establish **a common set of behaviors**
- **Achieve polymorphism**
- **Decouple components**
- Provide **clear guidelines for implementation**



```php
class Car implements Vehicle {
    public function start() {
        // Start the car's engine
    }

    public function stop() {
        // Stop the car's engine
    }

    public function accelerate($speed) {
        // Accelerate the car to the given speed
    }

    public function brake() {
        // Apply brakes to the car
    }
}
```

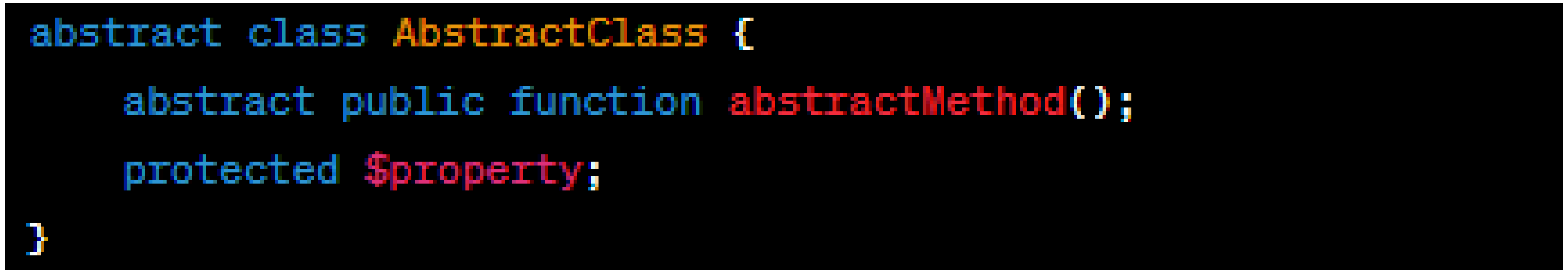
Abstract Classes

» A class that **serves as a blueprint for other classes**

» It is meant to be **extended by other classes**, which can **provide implementations for its abstract methods** and **inherit its properties** and **non-abstract methods**

» Abstract classes are **used to define common behavior** and **characteristics that can be shared** among multiple related classes

» The **abstract** keyword is used to **define an abstract class**



```
abstract class AbstractClass {
    abstract public function abstractMethod();
    protected $property;
}
```

Activity 4

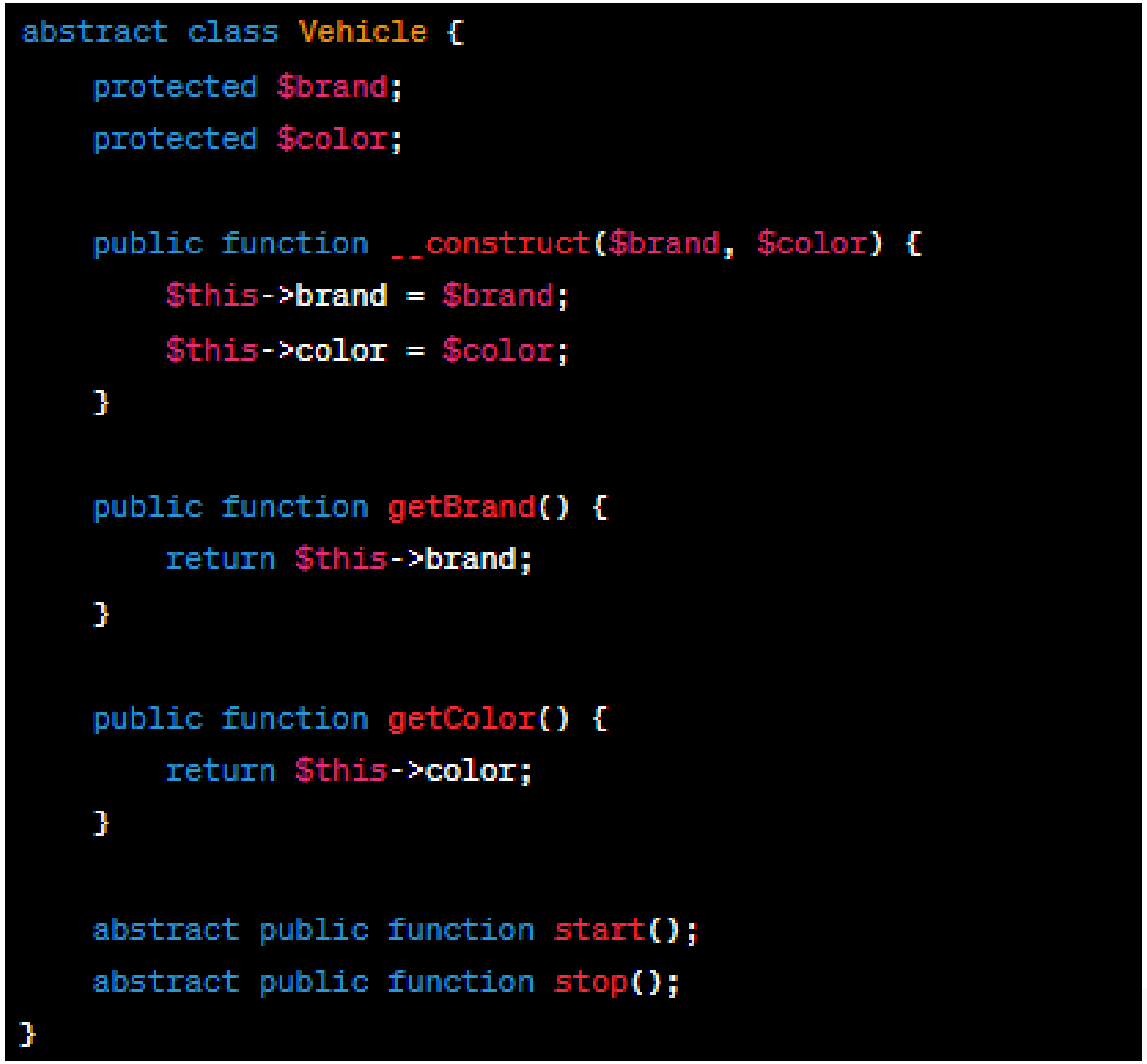
» *Discuss the following statement.*

» ***"Interface is a specific type of an abstract class"***

- *Do you agree with it?*
- *Explain your answer.*

Abstract Classes cont.

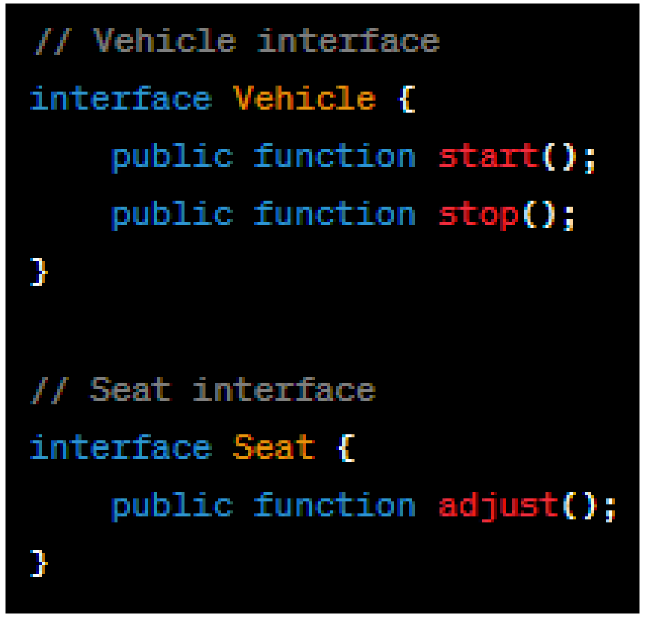
- » **Cannot be instantiated**
- » Can **have** both **abstract** and **non-abstract** methods
- » Can provide **default implementations for non-abstract methods**
- » Useful for **sharing code** and **implementing common functionality** among related classes



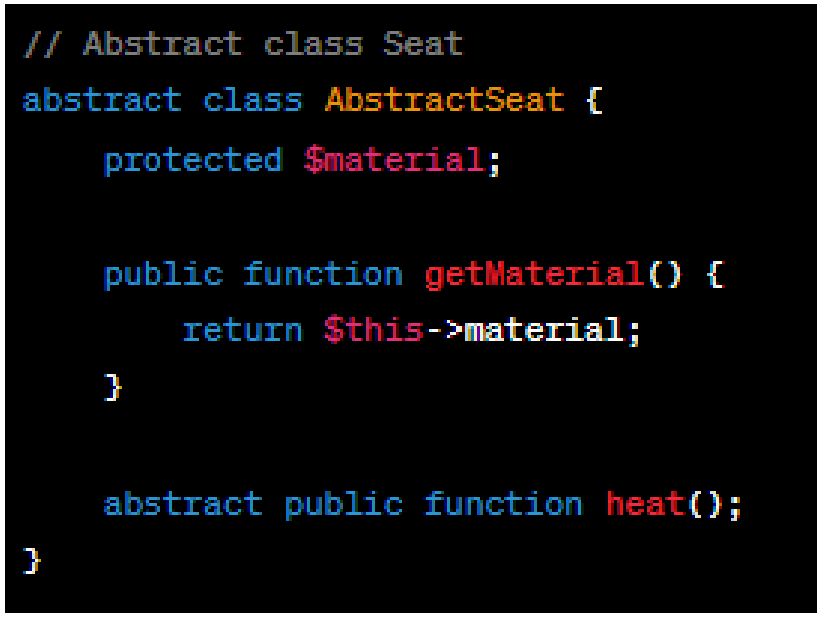
```
abstract class Vehicle {
    protected $brand;
    protected $color;

    public function __construct($brand, $color) {
        $this->brand = $brand;
        $this->color = $color;
    }

    public function getBrand() {
        return $this->brand;
    }

    public function getColor() {
        return $this->color;
    }

    abstract public function start();
    abstract public function stop();
}
```

Abstract Classes cont.

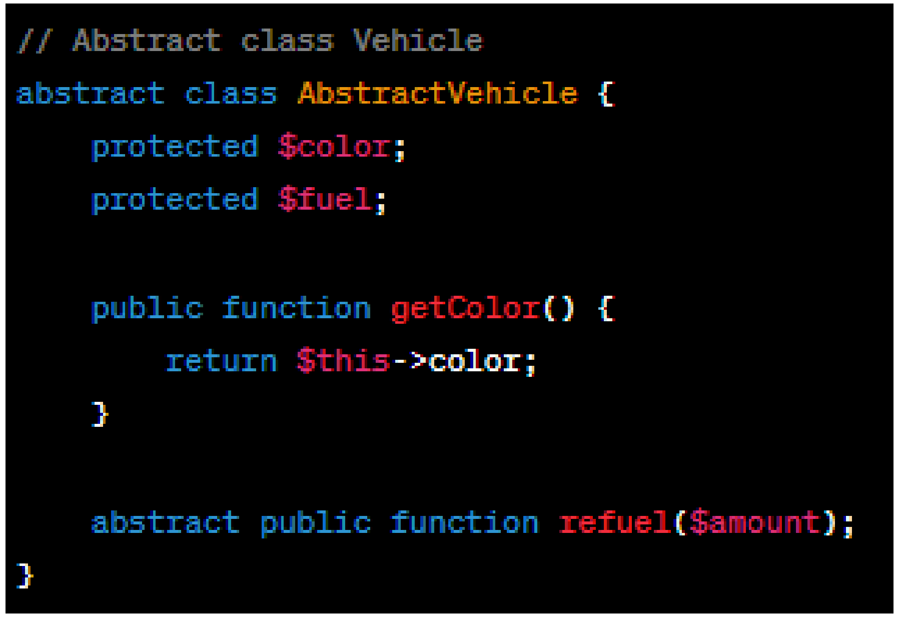
» A class can **only extend one abstract class** in PHP, unlike interfaces where multiple interfaces can be implemented



```php
// Vehicle interface
interface Vehicle {
    public function start();
    public function stop();
}

// Seat interface
interface Seat {
    public function adjust();
}
```

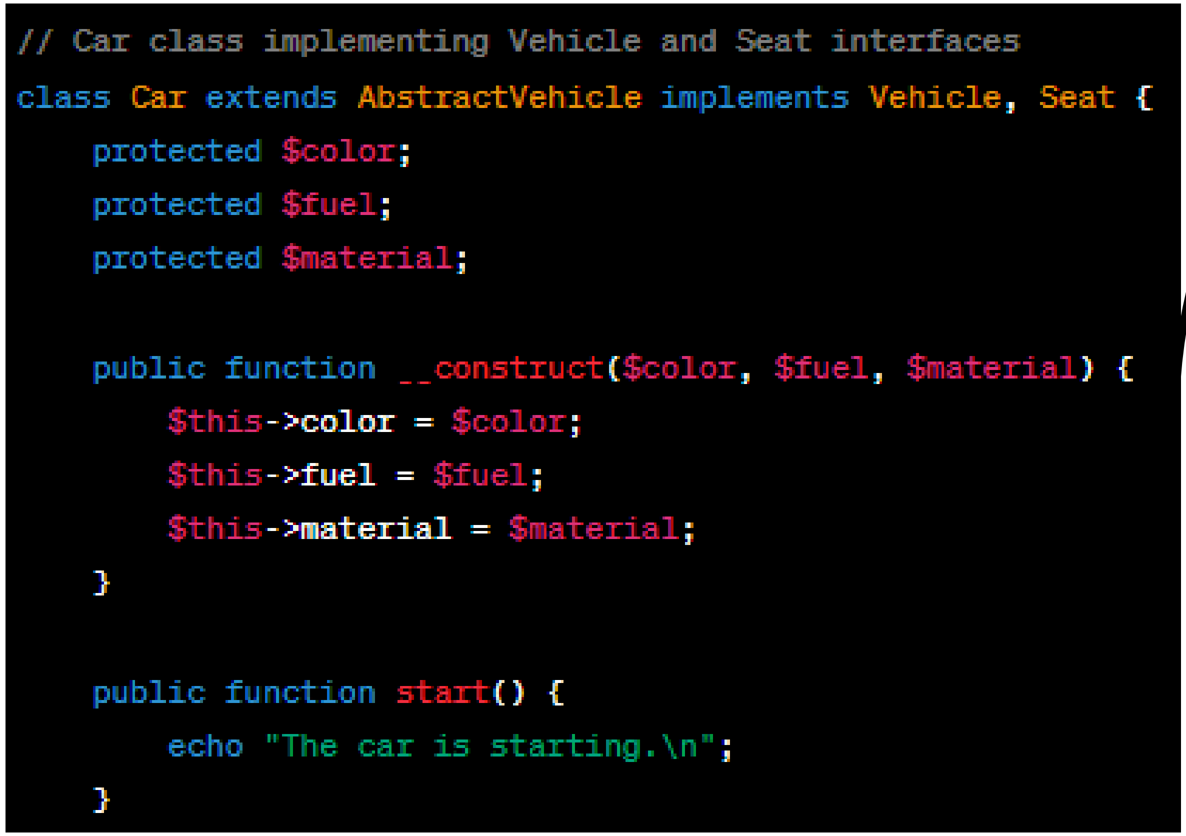


```php
// Abstract class Seat
abstract class AbstractSeat {
    protected $material;

    public function getMaterial() {
        return $this->material;
    }

    abstract public function heat();
}
```

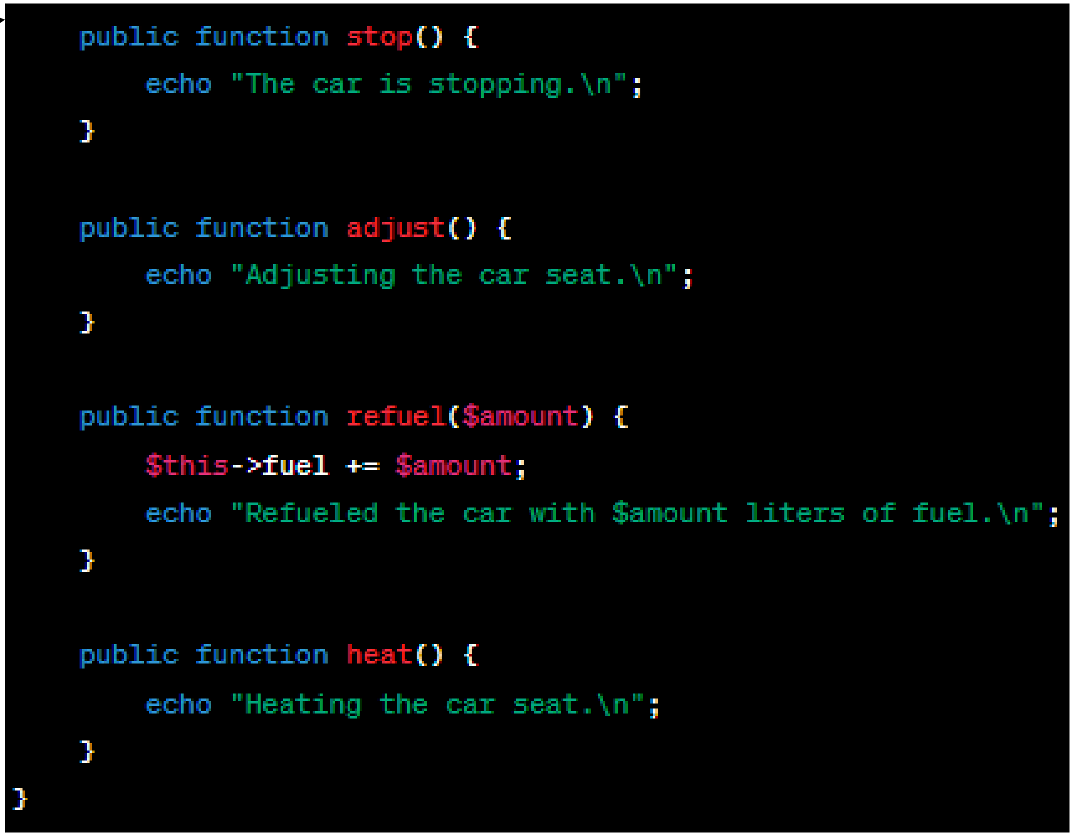


```php
// Abstract class Vehicle
abstract class AbstractVehicle {
    protected $color;
    protected $fuel;

    public function getColor() {
        return $this->color;
    }

    abstract public function refuel($amount);
}
```

Abstract Classes cont.



```php
// Car class implementing Vehicle and Seat interfaces
class Car extends AbstractVehicle implements Vehicle, Seat {
    protected $color;
    protected $fuel;
    protected $material;

    public function __construct($color, $fuel, $material) {
        $this->color = $color;
        $this->fuel = $fuel;
        $this->material = $material;
    }

    public function start() {
        echo "The car is starting.\n";
    }
```



```php
    public function stop() {
        echo "The car is stopping.\n";
    }

    public function adjust() {
        echo "Adjusting the car seat.\n";
    }

    public function refuel($amount) {
        $this->fuel += $amount;
        echo "Refueled the car with $amount liters of fuel.\n";
    }

    public function heat() {
        echo "Heating the car seat.\n";
    }
}
```

Polymorphism in PHP

» In general, polymorphism refers to **the ability of an object to take on different forms**

» In PHP, polymorphism **allows you to write code that operates on objects of different classes** but **treats them as objects of a common parent class or interface**

» Polymorphism can be achieved in **two ways**:

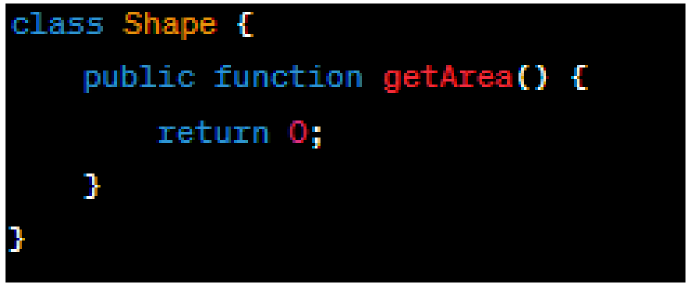
- **Method overriding** - A child class can provide its own implementation of a method defined in the parent class
- **Interface implementation**

» PHP **won’t support method overloading**

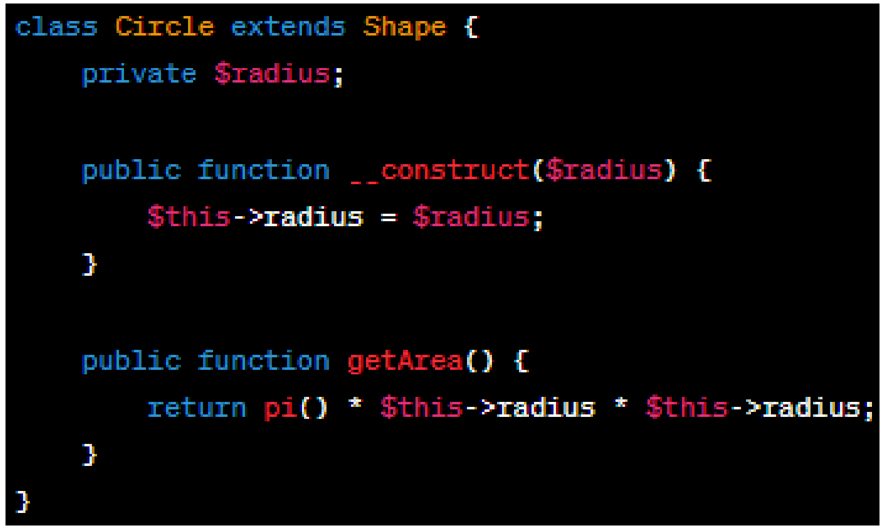
Polymorphism in PHP cont.

» Method Overriding

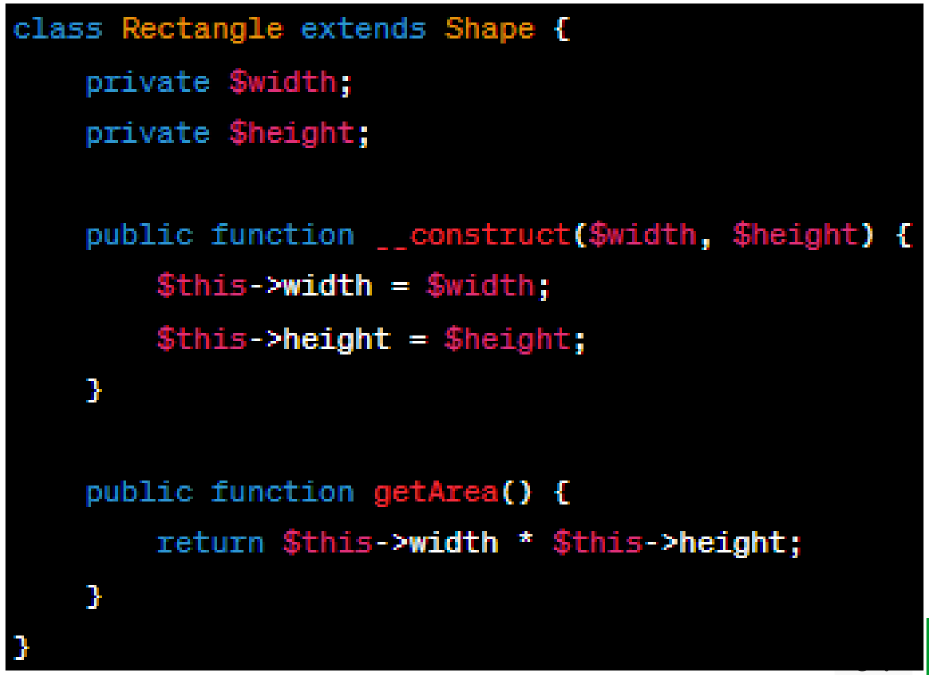
- The **child class provides its own implementation of the method**, which is used instead of the parent class's implementation when the method is called on an object of the child class



```php
class Shape {
    public function getArea() {
        return 0;
    }
}
```



```php
class Circle extends Shape {
    private $radius;

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

    public function getArea() {
        return pi() * $this->radius * $this->radius;
    }
}
```

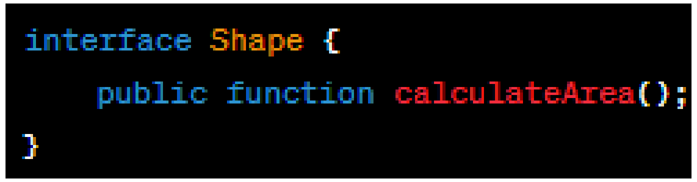


```php
class Rectangle extends Shape {
    private $width;
    private $height;

    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }

    public function getArea() {
        return $this->width * $this->height;
    }
}
```

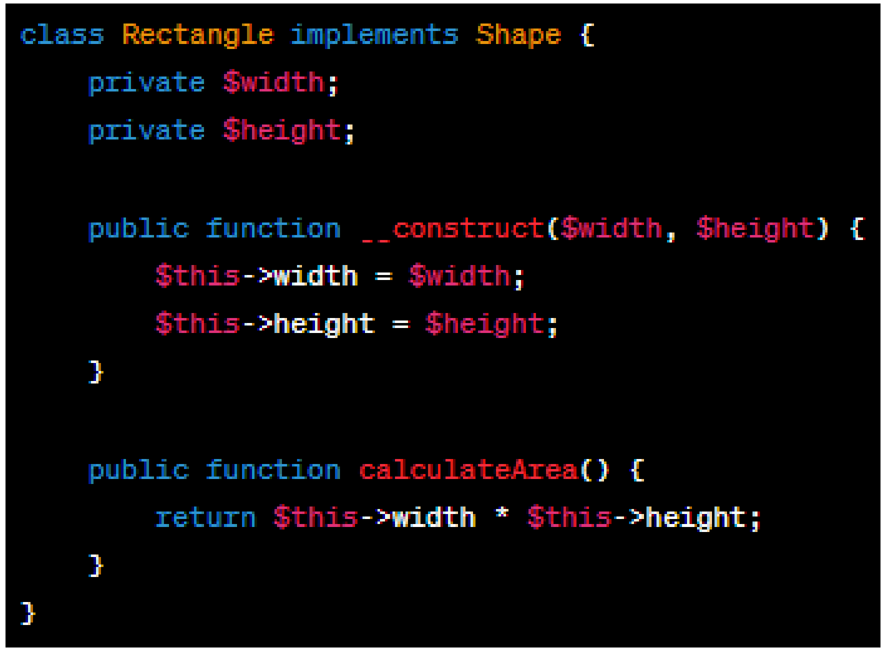
Polymorphism in PHP cont.

» Interface Implementation

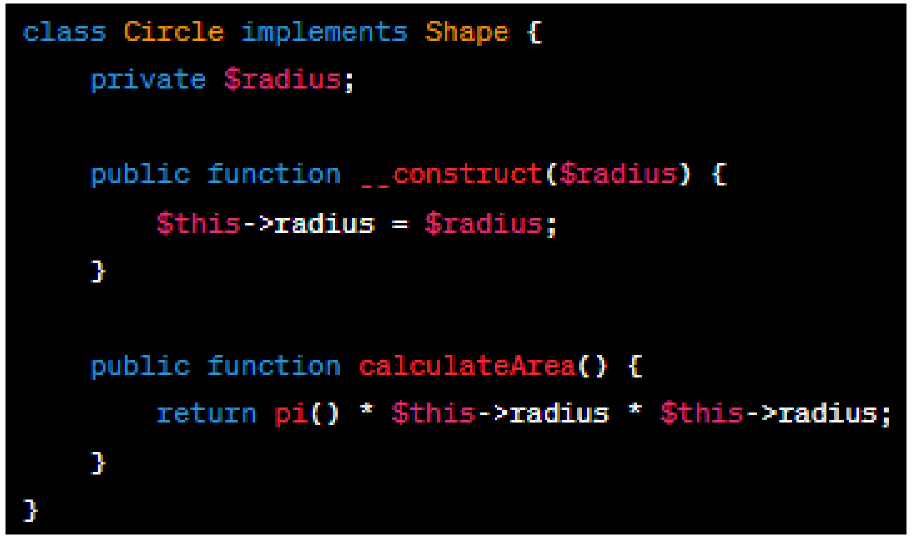
- The process of defining a class that fulfills the contract specified by an interface



```php
interface Shape {
    public function calculateArea();
}
```



```php
class Rectangle implements Shape {
    private $width;
    private $height;

    public function __construct($width, $height) {
        $this->width = $width;
        $this->height = $height;
    }

    public function calculateArea() {
        return $this->width * $this->height;
    }
}
```



```php
class Circle implements Shape {
    private $radius;

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

    public function calculateArea() {
        return pi() * $this->radius * $this->radius;
    }
}
```

‘require’ and ‘include’ in PHP

» These **two are used to include external PHP files** into a script

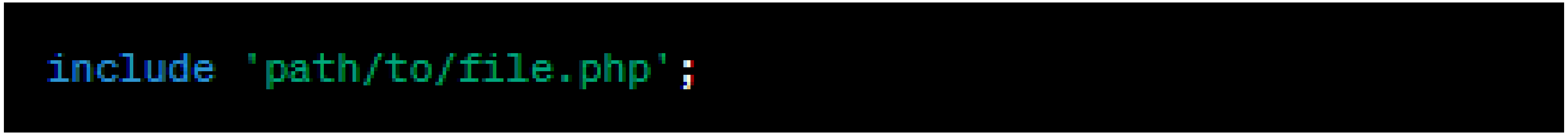
» require:

- If the **file cannot be found or there is an error** during the inclusion process
- **Halt the script execution**
- Commonly **used for essential files or dependencies** that are required for the script to function correctly

```php
require 'path/to/file.php';
```

‘require’ and ‘include’ in PHP cont.

» include:
- **If the file cannot be found or there is an error** during inclusion
- It will **generate a warning message** and **continue script execution**
- **Used for non-essential files or dependencies** that can be missing without causing critical issues



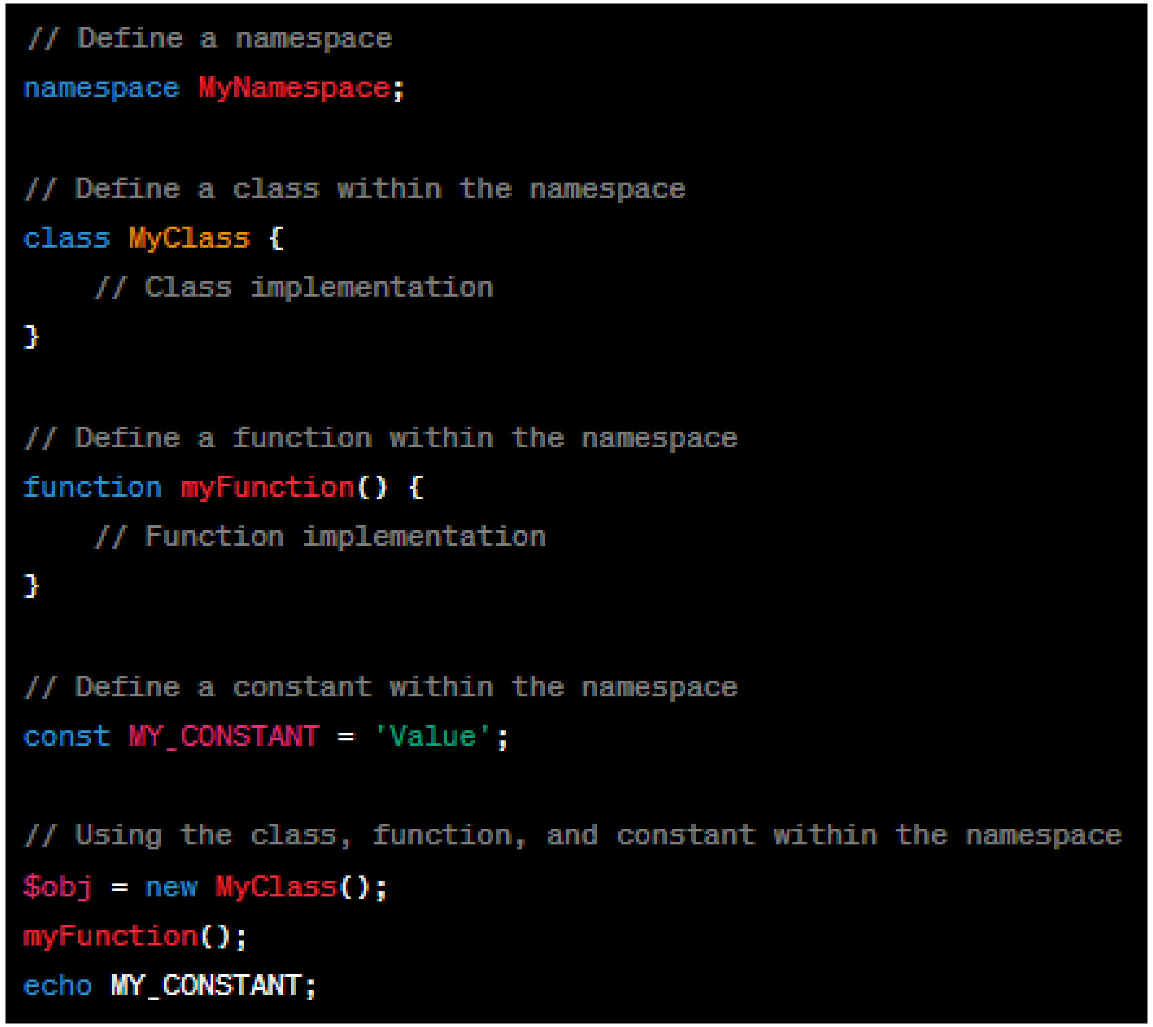
```php
include 'path/to/file.php';
```

» PHP provides **require_once** and **include_once** as well
- Work similarly
- Ensure that a **file is included only once**, even if multiple inclusion attempts are made

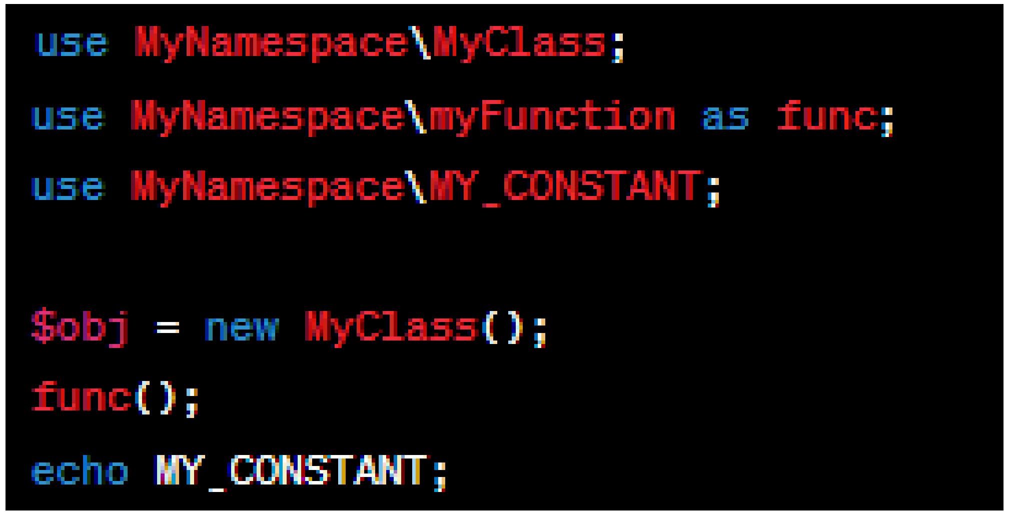
Use of Namespaces

- » **Similar to packaging** concept in Java
- » A way to **organize your code avoid naming conflicts** and **improve code readability** and **maintainability**
- » Namespaces are particularly **useful in larger PHP projects** or when working **with third-party libraries**
- » Use **namespace keyword to define** a namespace
- » You can use the **use keyword to import a namespace** or specific elements from a namespace
- » If you have **multiple classes under one namespace**, you should **add one by one** using use keyword

Use of Namespaces cont.



```php
// Define a namespace
namespace MyNamespace;

// Define a class within the namespace
class MyClass {
    // Class implementation
}

// Define a function within the namespace
function myFunction() {
    // Function implementation
}

// Define a constant within the namespace
const MY_CONSTANT = 'Value';

// Using the class, function, and constant within the namespace
$obj = new MyClass();
myFunction();
echo MY_CONSTANT;
```



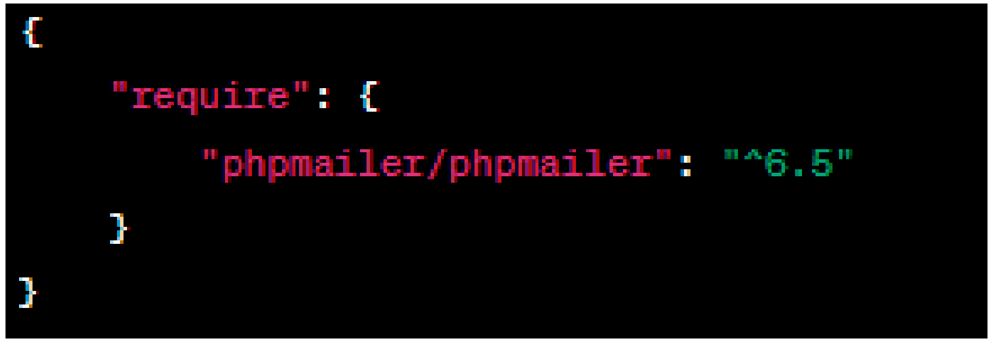
```php
use MyNamespace\MyClass;
use MyNamespace\myFunction as func;
use MyNamespace\MY_CONSTANT;

$obj = new MyClass();
func();
echo MY_CONSTANT;
```

Dependency Management in PHP

» The **process of managing external libraries**, **packages**, and **dependencies** that your PHP project relies on

» It involves **handling the installation**, **versioning**, and **updating of these dependencies** to ensure smooth integration and functionality within your project

» You can use **Composer** as **a tool to manage your dependencies**



```
{
    "require": {
        "phpmailer/phpmailer": "^6.5"
    }
}
```

Can you remember?

- » Object-Oriented Programming (OOP)
- » Core Principles of OOP
- » Creating and Using Classes in PHP
- » Data Hiding and Access Modifiers
- » Implementing Inheritance
- » Interfaces and Abstract Classes
- » Polymorphism in PHP
- » Code Organization and Dependency Management