## 🔹 1. Static Variables, Member Functions and Inheritance

### Definition:

A static variable retains its value even after the function exits and is shared across all instances of the class.

### Use Cases:

* To keep track of data that must persist across function calls.
* To share values among all instances of a class.

### Syntax & Example:

| class Counter {  public static $count = 0;   public function increment() {  self::$count++;  }   public function getCount() {  return self::$count;  } }  $obj1 = new Counter(); $obj1->increment();  $obj2 = new Counter(); $obj2->increment();  echo $obj2->getCount(); // Output: 2 (shared value) |
| --- |

### 

### Key Points:

* Accessed using self::$varName inside the class.
* Accessed using ClassName::$varName outside the class.
* Static variables belong to the class, not to any object.

## 🔹 2. Static Member Functions (Static Methods)

### Definition:

A static method belongs to the class rather than any specific object instance.

### Use Cases:

* Utility/helper functions.
* When object state is not needed.

### Syntax & Example:

| class MathHelper {  public static function square($num) {  return $num \* $num;  } }  // Call without creating an object echo MathHelper::square(5); // Output: 25 |
| --- |

### 

### 

### Key Points:

* Cannot access $this inside a static method.
* Can only access static properties and other static methods.
* Useful for utility classes (e.g., Logger::log(), DB::connect()).

## 

## 🔹 3. Inheritance in PHP

### Definition:

Inheritance allows a class (child) to inherit properties and methods from another class (parent).

### Benefits:

* Reusability of code.
* Extending existing functionality.

### Syntax:

| class ParentClass {  public $name = "Parent";   public function greet() {  return "Hello from Parent!";  } }  class ChildClass extends ParentClass {  public function greetChild() {  return "Hello from Child!";  } }  $child = new ChildClass(); echo $child->greet(); // Inherited method echo $child->greetChild(); // Own method |
| --- |

### 

### 

### 

### 

### 

### 

### 

### 

### 

### 

### ✅ Real-Life Version of Your Code:

#### 💡 Scenario:

We have a Vehicle class that defines general behavior for all vehicles. Then we create a Car class that extends it with more specific functionality.

### 🔧 PHP Code:

| <?php // Parent class class Vehicle {  public $brand = "Generic Vehicle";   public function startEngine() {  return "Engine started for: {$this->brand}<br>";  }  public function autoControl(){  return "Engine started for: {$this->brand}<br>";  } }  // Child class class Car extends Vehicle{  public $brand = "Toyota Corolla";  public function playMusic() {  return "Playing music in the {$this->brand}<br>";  } }  // Create an object of the Car class $myCar = new Car();  // Call inherited method from Vehicle echo $myCar->startEngine(); // Inherited from Vehicle  // Call method from Car echo $myCar->playMusic(); // Defined in Car ?> |
| --- |

### 

### 🧾 Output:

Engine started for: Toyota Corolla

Playing music in the Toyota Corolla

### ✅ Explanation:

| Code Element | Real-Life Meaning |
| --- | --- |
| Vehicle class | Base concept (e.g., all vehicles can start engines) |
| Car class | A specific type of vehicle with more capabilities |
| startEngine() | Shared behavior across all vehicle types |
| playMusic() | Specific to the Car, not all vehicles might have this |

### Access Modifiers in Inheritance:

| Modifier | Accessible in Class | Inherited Class | Outside Class |
| --- | --- | --- | --- |
| public | ✅ | ✅ | ✅ |
| protected | ✅ | ✅ | ❌ |
| private | ✅ | ❌ | ❌ |

### 

### 

### Method Overriding:

You can override a parent method in the child class.

| class A {  public function show() {  return "A";  } }  class B extends A {  public function show() {  return "B";  } }  $obj = new B(); echo $obj->show(); // Output: B |
| --- |

### Calling Parent Methods:

Use parent:: to call parent methods when overridden.

| class A {  public function greet() {  return "Hello from A";  } }  class B extends A {  public function greet() {  return parent::greet() . " and B";  } }  $obj = new B(); echo $obj->greet(); // Output: Hello from A and B |
| --- |

## ✅ Summary:

| Concept | Description |
| --- | --- |
| Static Variable | Belongs to class, not object; retains value globally. |
| Static Method | Called without object; can't use $this. |
| Inheritance | Reuse code by extending parent class. |
| Access Modifiers | Control visibility (public, protected, private) |
| Method Overriding | Child method replaces parent version. |

### **🔧 Exercise 1: Basic Inheritance**

**Instruction:** Create a Person class with a name property and a sayHello() method. Then create a Teacher class that inherits from Person and adds a subject property. Instantiate the Teacher and display both name and subject using the methods.

### **🔧 Exercise 2: Method Overriding**

**Instruction:** Create a Device class with a method powerOn() that echoes a generic message. Then create a Laptop class that extends Device and overrides the powerOn() method with a more specific message. Call the method from both classes and observe the difference.

### **🔧 Exercise 3: Parent Method Access**

**Instruction:** Create a User class with a method getRole() that returns "User". Create an Admin class that extends User and overrides the getRole() method but still calls the parent method using parent::getRole() and appends " & Admin".

### **🔧 Exercise 4: Add Child Method**

**Instruction:** Make a Vehicle class with a property type. Then create a Truck class that extends Vehicle and adds a method loadGoods() that echoes "Goods loaded in Truck". Instantiate the Truck class and call both the inherited and own method.

### **🔧 Exercise 5: Multiple Child Classes**

**Instruction:** Create a Shape class with a method getArea() that returns 0. Then create two child classes: Rectangle and Circle. Each should override getArea() and compute the area based on their respective formulas.