PHP Access Modifiers & Inheritance Handbook

---

1) What Are Access Modifiers?

Access modifiers control **who can see and use** a property or method.

Three keywords:

- **public** → accessible from anywhere.
- **protected** → accessible in this class and subclasses only.
- **private** → accessible only in this class.

---

2) Visibility Rules

Public

```
class Example {
    public string $name = "Ada";
    public function sayHello(): void {
        echo "Hello, {$this->name}";
    }
}

$e = new Example();
echo $e->name;    // ✅ allowed
$e->sayHello();   // ✅ allowed
```

✅ Accessible:

- Inside the class

- In child classes
- Outside the class

Protected

```
class Example {
    protected string $secret = "42";
    protected function reveal(): void {
        echo $this->secret;
    }
}

class Child extends Example {
    public function show(): void {
        $this->reveal(); // ✅ allowed (protected works in subclass)
    }
}

$c = new Child();
$c->show();       // ✅ prints "42"
// $c->reveal();  // ❌ Error (protected)
```

✅ Accessible:

- Inside the class
- In child classes
  ❌ Not accessible from outside

Private

```
class Example {
    private string $pin = "1234";
    private function unlock(): void {
        echo "Unlocked!";
```

```
    }
    public function tryUnlock(): void {
        $this->unlock(); // ✅ allowed inside class
    }
}

class Child extends Example {
    public function attempt(): void {
        // $this->unlock(); ❌ Error (private not inherited)
    }
}

$e = new Example();
$e->tryUnlock(); // ✅ works
// $e->unlock(); // ❌ Error
```

✅ Accessible:

- Only inside **that exact class**
  ❌ Not in subclasses
  ❌ Not outside

---

3) Access Modifiers in Inheritance

Key Rules

- **public** stays public in child classes.
- **protected** stays protected (or can be made public).
- **private** is not inherited; child cannot access it directly.
- You **cannot** make a method less visible in a subclass.

Example:

```
class ParentClass {
```

```
    protected function greet(): void {
        echo "Hello";
    }
}

class ChildClass extends ParentClass {
    public function greet(): void { // Allowed: protected → public
        echo "Hi";
    }
}
```

❌ Not allowed:

```
class BadChild extends ParentClass {
    private function greet(): void {} // Error: visibility must be >=
parent
}
```

---

4) Access Modifiers on Properties

Same rules as methods:

- `public $x` → accessible everywhere.
- `protected $x` → accessible in same class & subclasses.
- `private $x` → only in same class.

---

5) Access Modifiers with Abstract Methods

- Abstract methods **must** keep the same visibility in subclasses or be made *more public*.
- You cannot make them more restrictive.

Example:

```
abstract class Shape {
    abstract protected function area(): float;
}

class Circle extends Shape {
    public function area(): float { // Allowed: protected → public
        return 0;
    }
}
```

---

6) Access Modifiers with Interfaces

- Interface methods are **always public**.
- Implementation **must** be public.

```
interface Resizable {
    public function resize(float $factor): void; // always public
}

class Box implements Resizable {
    public function resize(float $factor): void { /* ... */ }
}
```

---

7) Quick-Reference Table

Modifier	Same Class	Subclass	Outside
public	✅	✅	✅
protected	✅	✅	❌

private	✅	❌	❌

---

8) Example Showing All in Action

```
class Base {
    public string $pub = "Public";
    protected string $pro = "Protected";
    private string $pri = "Private";

    public function showBase(): void {
        echo "$this->pub, $this->pro, $this->pri\n";
    }
}

class Child extends Base {
    public function showChild(): void {
        echo "$this->pub, $this->pro\n"; // ✅
        // echo $this->pri; // ❌ private not accessible
    }
}

$b = new Base();
$b->showBase();  // ✅ Public method can access all
echo $b->pub;    // ✅
// echo $b->pro; // ❌
// echo $b->pri; // ❌

$c = new Child();
$c->showChild(); // ✅ Public method accessing public & protected
```

---

9) Best Practices

- Use **private** for strict encapsulation.
- Use **protected** when designing extensible classes for inheritance.

- Use **public** for your class's official API.
- Avoid making everything public — it reduces flexibility to change internals later.
- If you change visibility from protected to public, document why (it changes your API).