TeleSoftas

SOLID Principles in PHP

Nerijus Žutautas

TeleSoftas

Klaipėda PHP

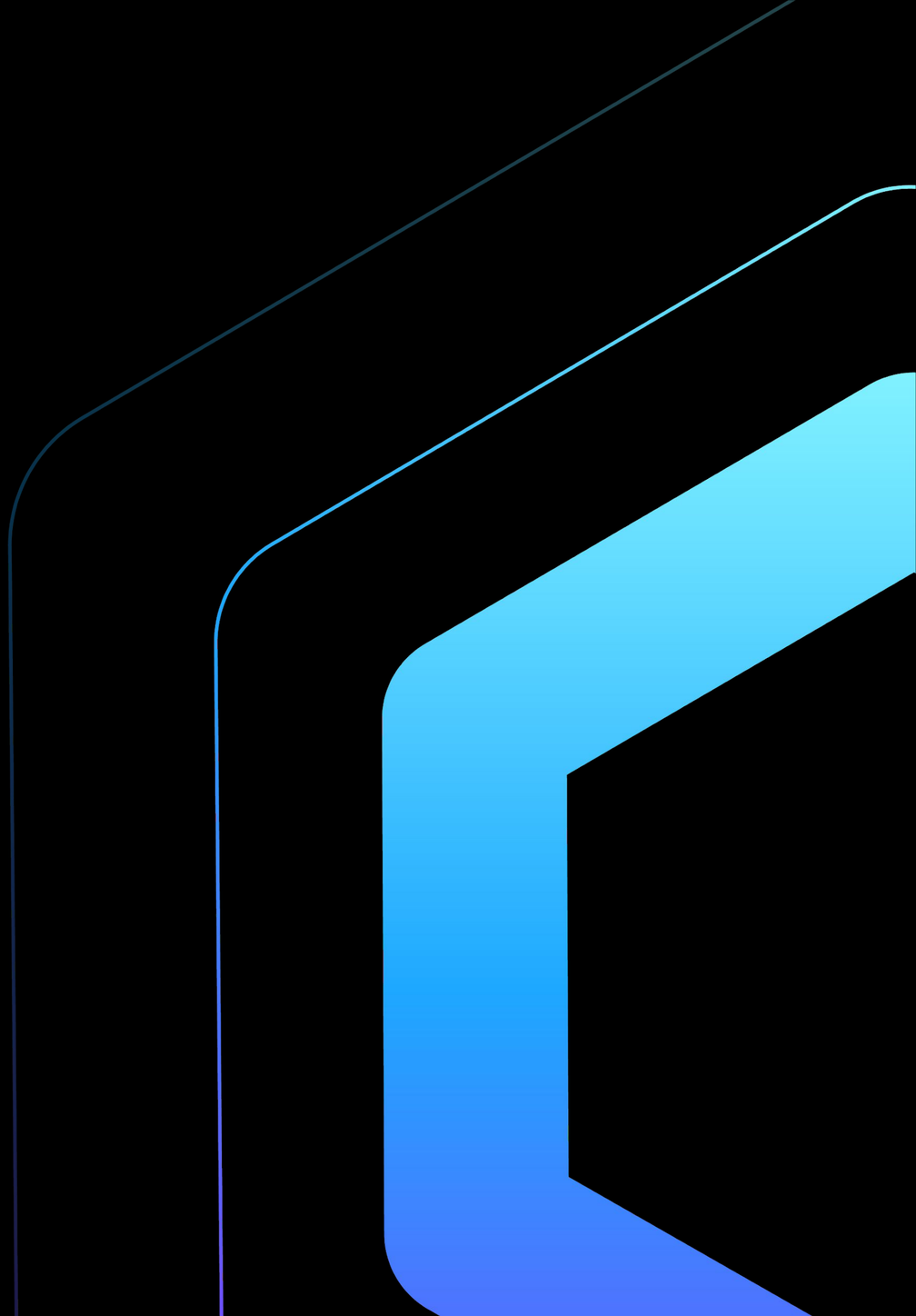
Nerijus Žutautas

Introduction

SOLID is an acronym for the first five object-oriented design (OOD) principles by Robert C. Martin (also known as Uncle Bob).

These principles establish practices that lend to developing software with considerations for maintaining and extending as the project grows. Adopting these practices can also contribute to avoiding code smells, refactoring code, and Agile or Adaptive software development.

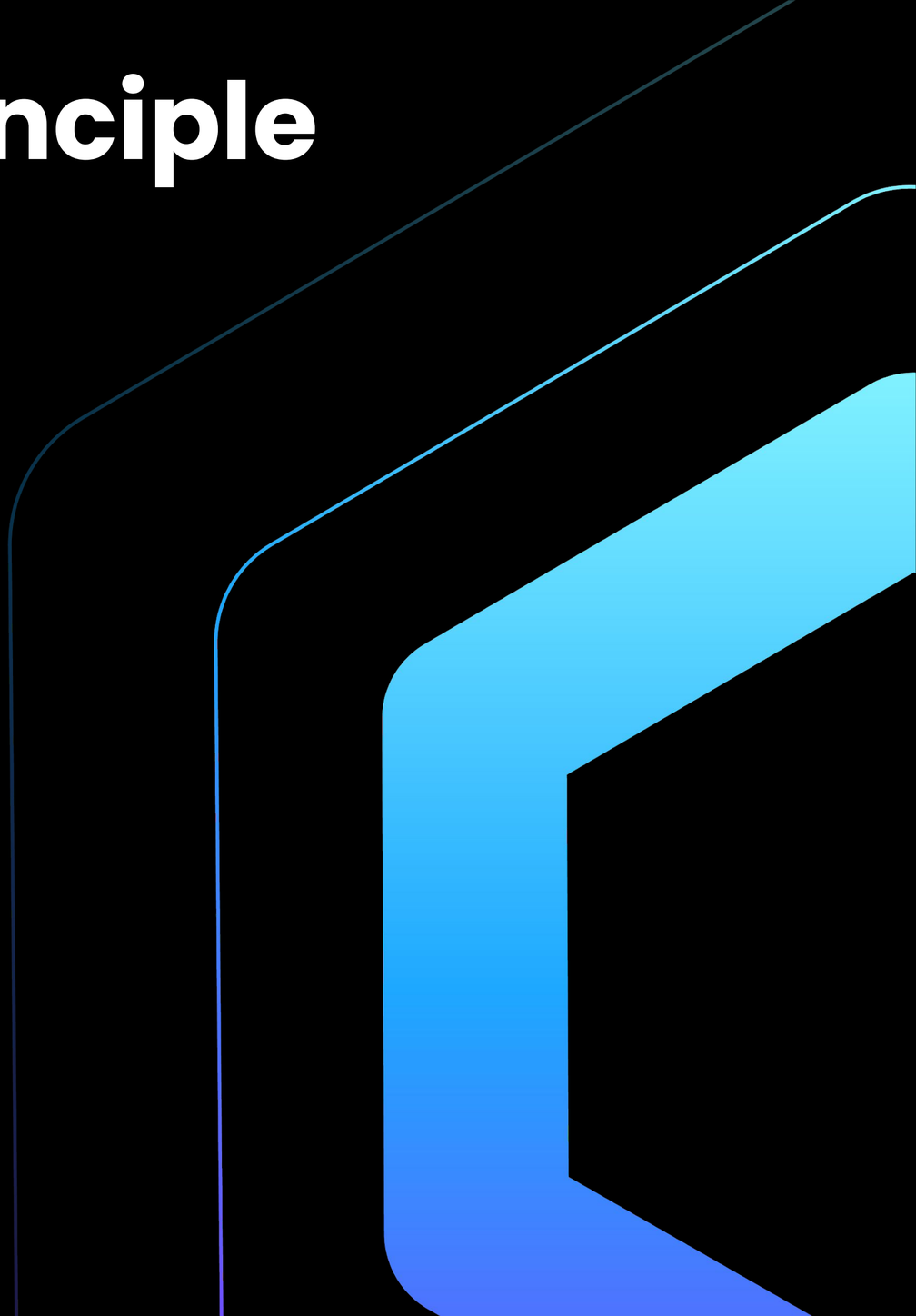
S - Single-responsiblity Principle
O - Open-closed Principle
L - Liskov Substitution Principle
I - Interface Segregation Principle
D - Dependency Inversion Principle



Single-responsiblity Principle

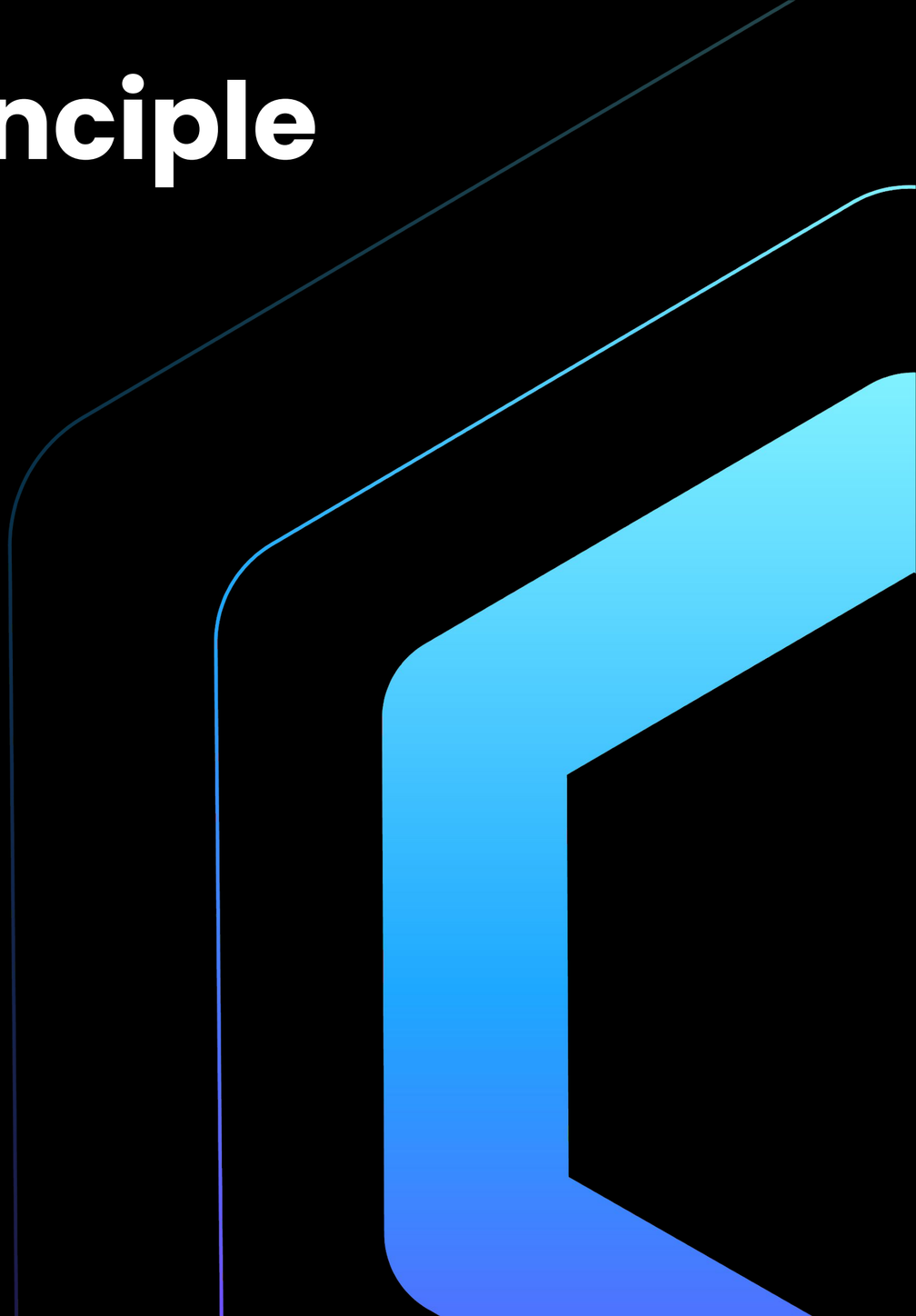
Single-responsiblity Principle

```php
class Book {
    protected string $title;

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

    public function getTitle(string $title): string {
        return $this->title;
    }

    public function formatJson(): string {
        return json_encode($this->getTitle());
    }
}

$book = new Book('Page title');

echo $book->formatJson();
```



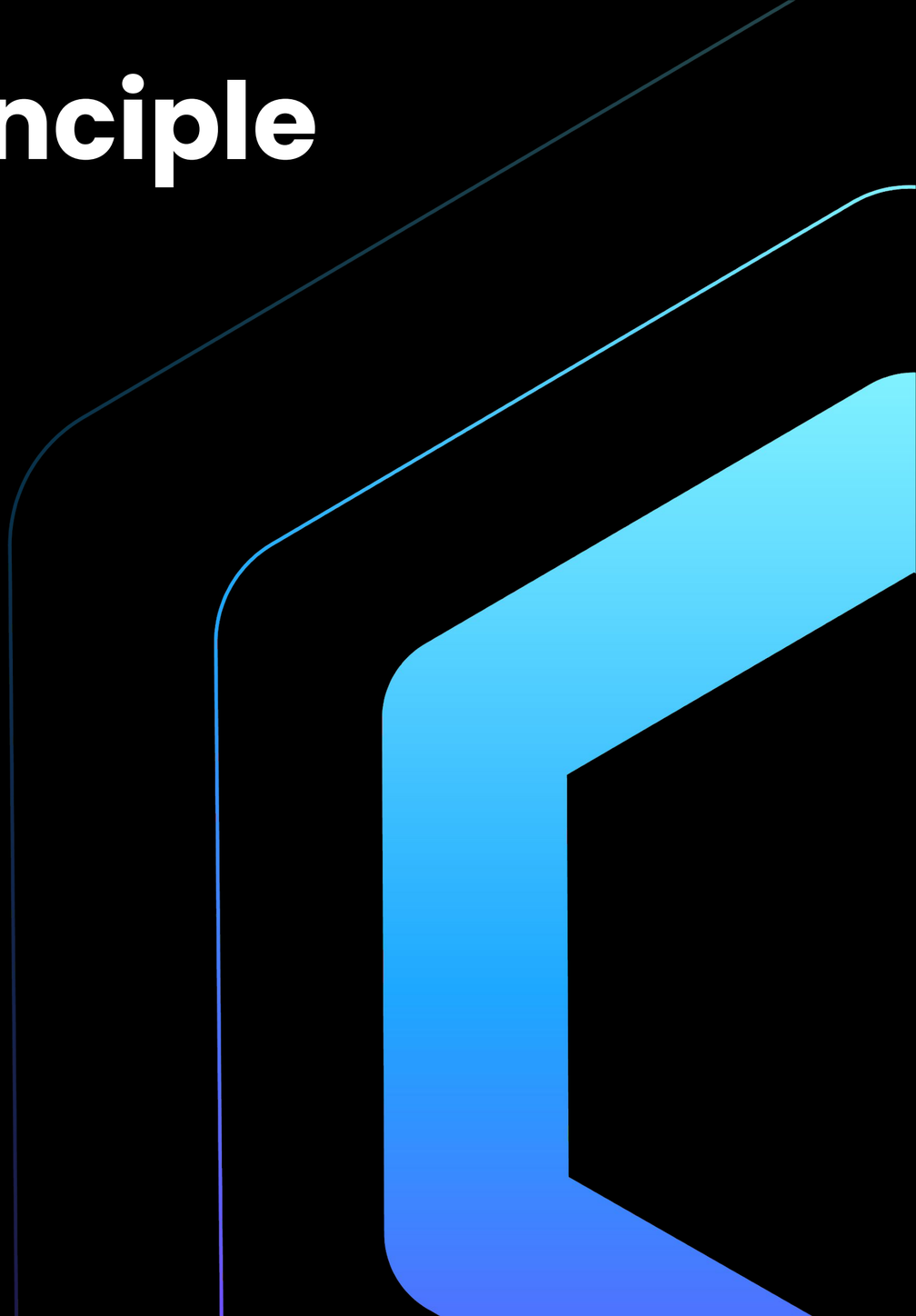
Single-responsiblity Principle

```php
class Book {
    protected string $title;

    /* */

    public function formatJson(): string {
        return json_encode($this->getTitle());
    }

    public function formatArray(): array {
        return [$this->getTitle()];
    }

    public function formatSerialize(): string {
        return serialize($this->getTitle());
    }
}

$book = new Book('Book title');
$book->formatJson();
$book->formatArray();
$book->formatSerialize();
```

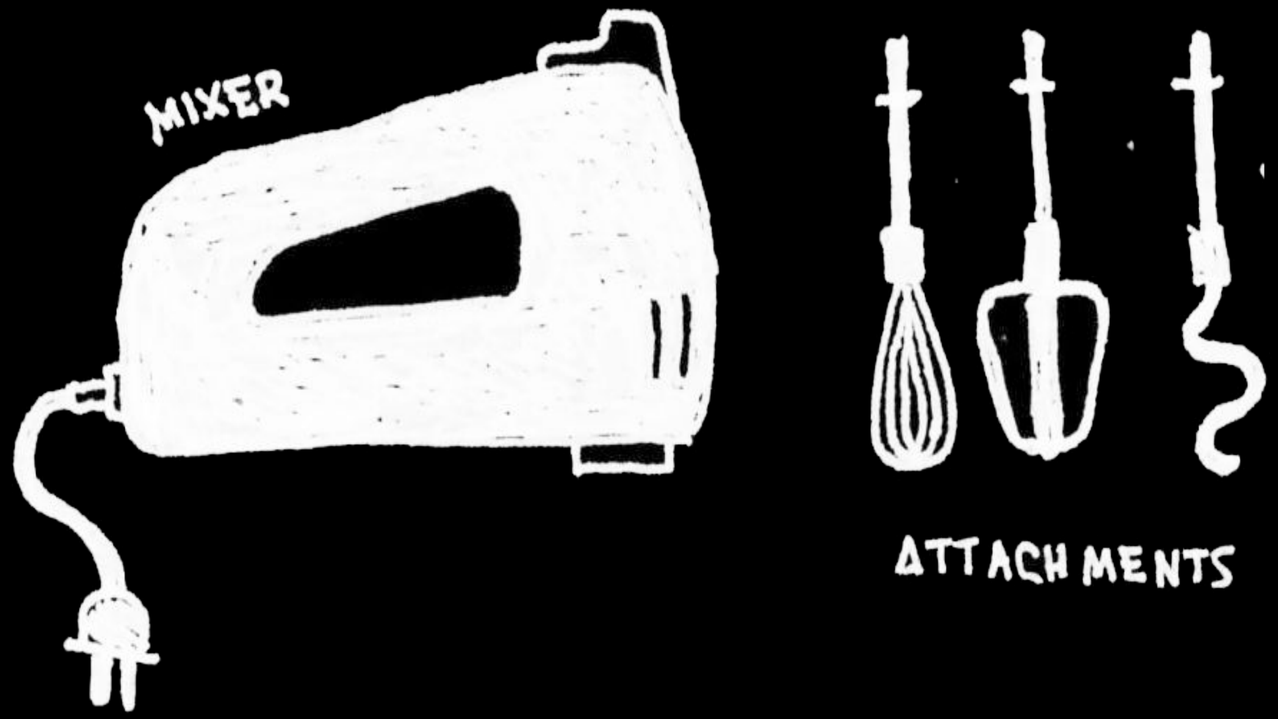


Single-responsiblity Principle

```php
class Book {
   protected string $title;

   public function __construct(string $title) {
      $this->title = $title;
   }
   public function getTitle(): string {
      return $this->title;
   }
}

class JsonBookFormatter {
    public function format(Book $book): string {
        return json_encode($book->getTitle());
    }
}

$book = new Book('Book title');

$jsonBookFormatter = new JsonBookFormatter();

echo $jsonBookFormatter->format($book);
```

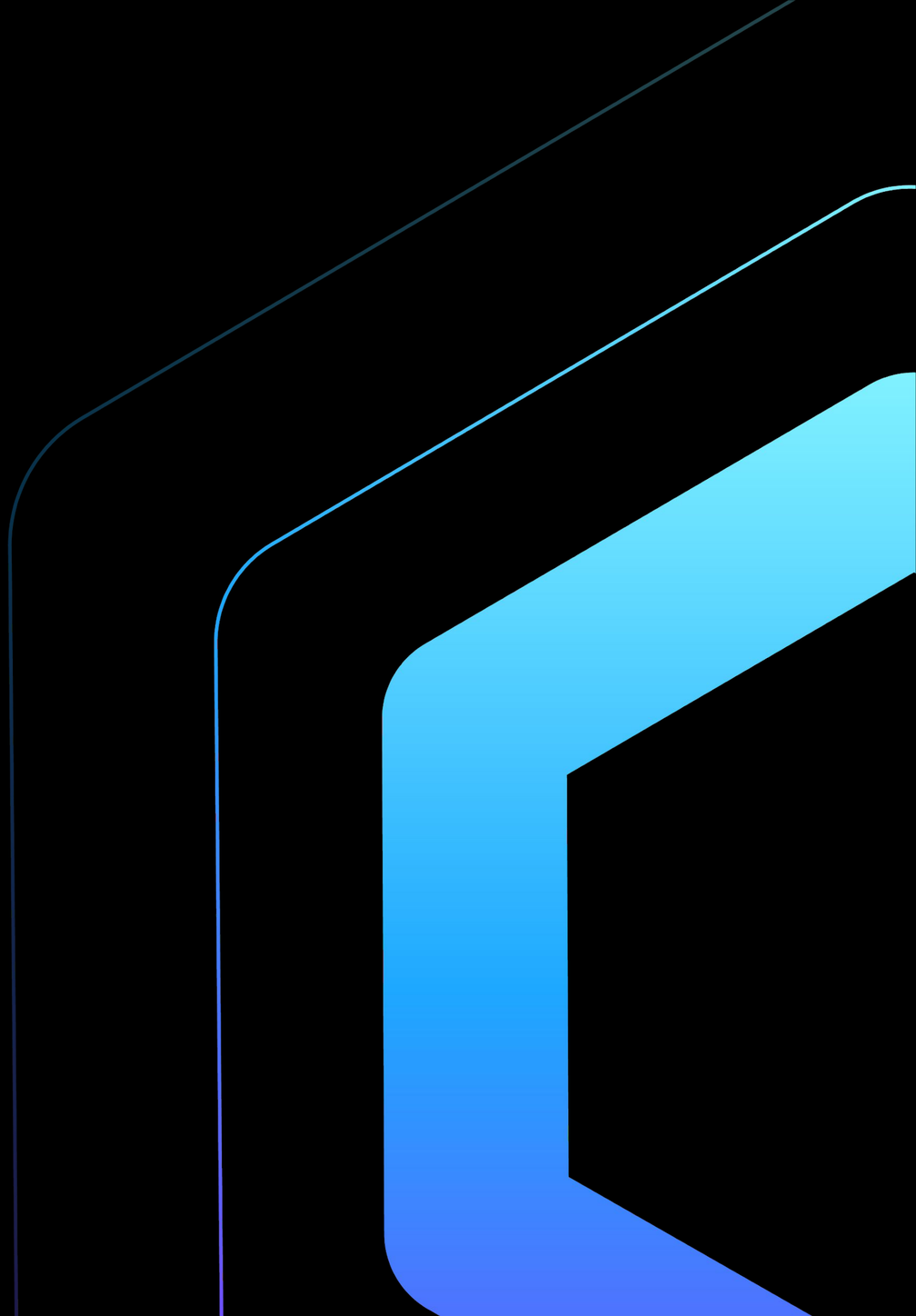


Open-closed Principle



Open-closed Principle

```php
class Rectangle {
    public int $width;
    public int $height;
}

.....................................................

class AreaCalculator {
    public array $rectangles = [];

    public function calculate(): int {
        $area = 0;
        foreach ($this->rectangles as $rectangle) {
            $area += $rectangle->width * $rectangle->height;
        }
        return $area;
    }
}
```

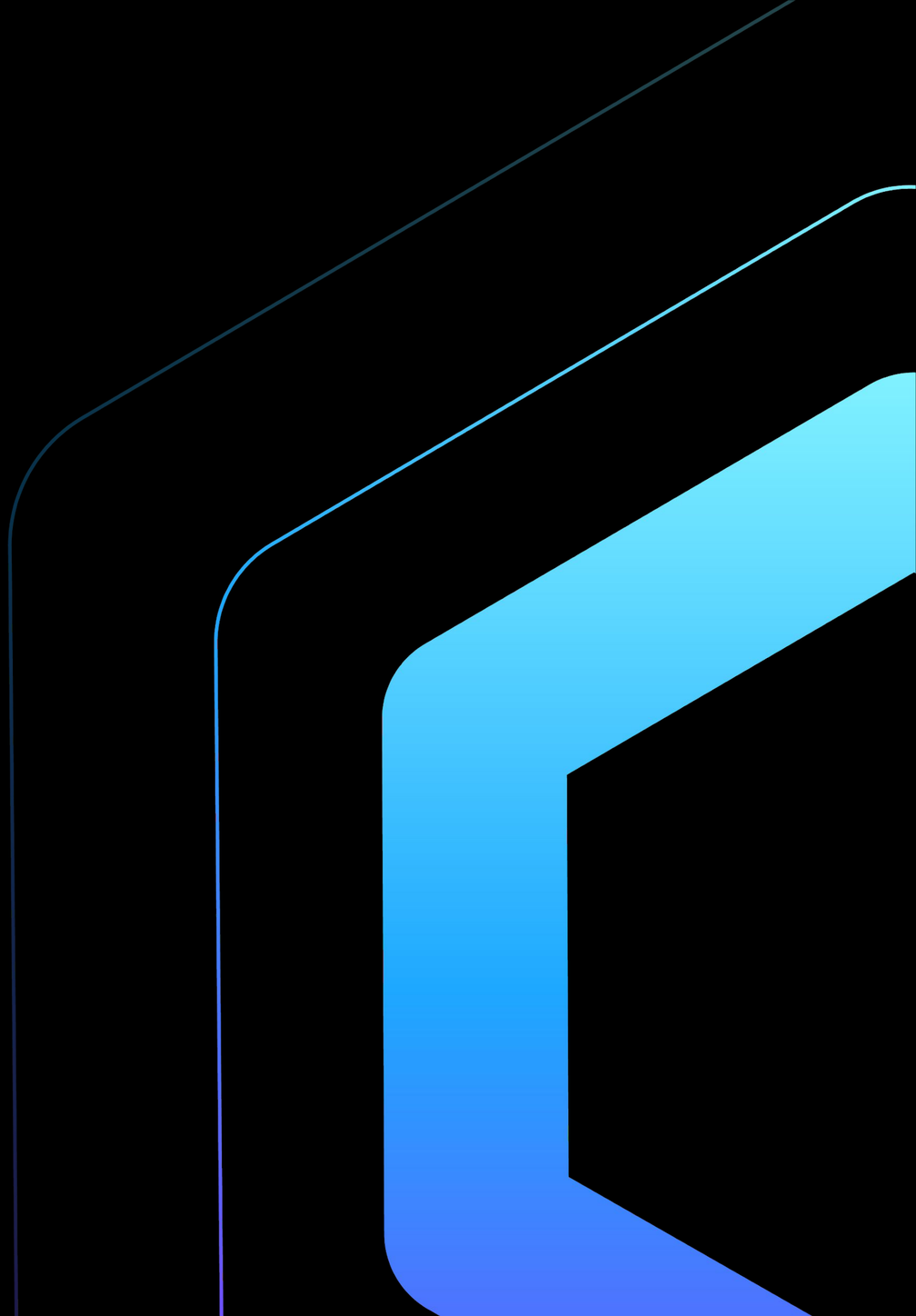


Open-closed Principle

```php
class Rectangle {
   public int $width;
   public int $height;
}

class Circle {
   public int $radius;
}
```

.....

```php
class AreaCalculator {
   public array $shapes = [];

   public function calculate(): int {
      $area = 0;
      foreach ($this->shapes as $shape) {
         if($shape instanceof Rectangle){
            $area += $shape->width * $shape->height;
         }else{
            $area += $shape->radius * pi() * 2;
         }
      }
      return $area;
   }
}
```



Open-closed Principle

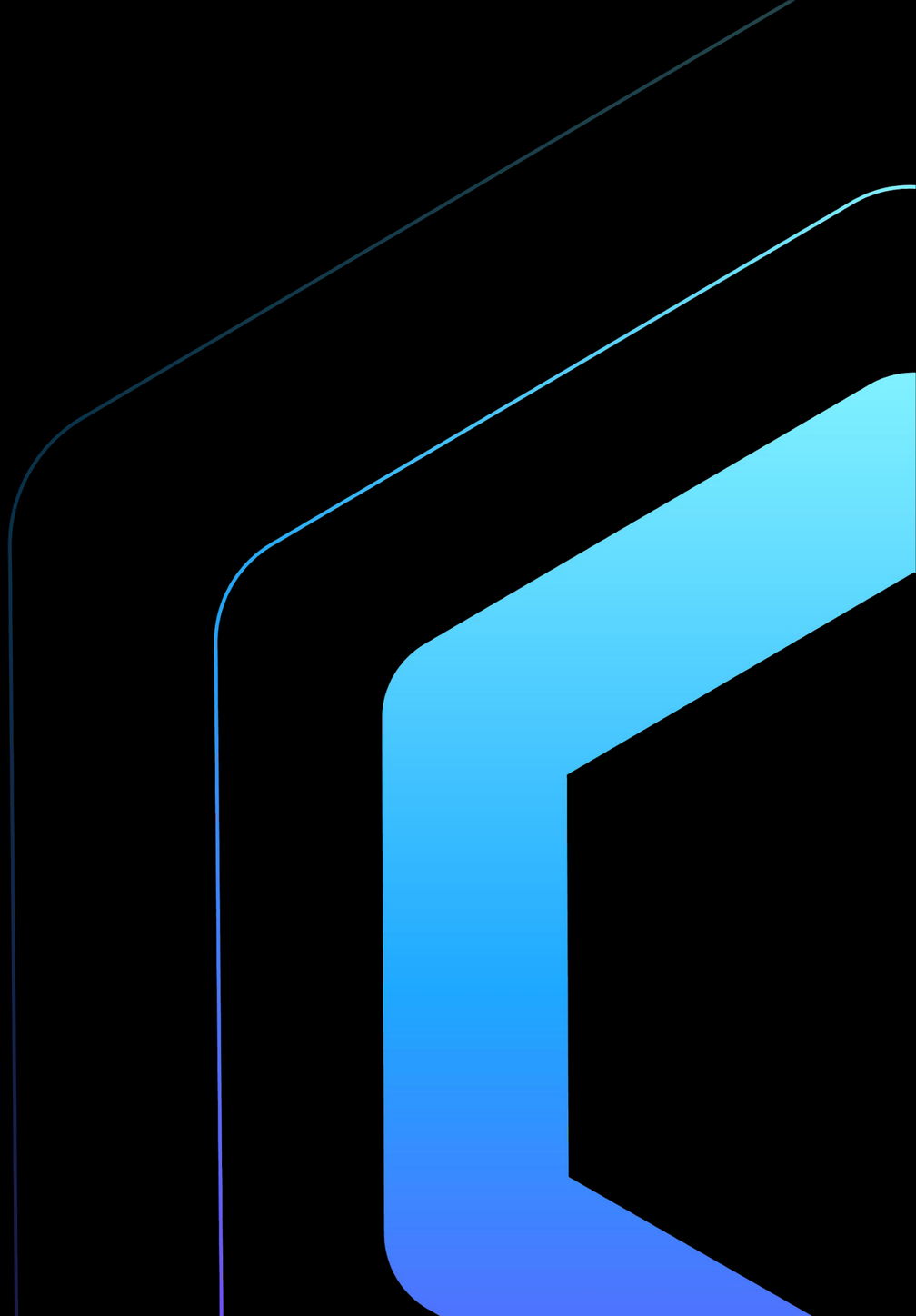
```php
class Rectangle {
   public int $width;
   public int $height;
}
```

```php
class Circle {
   public int $radius;
}
```

```php
class Square {
   public int $width;
}
```

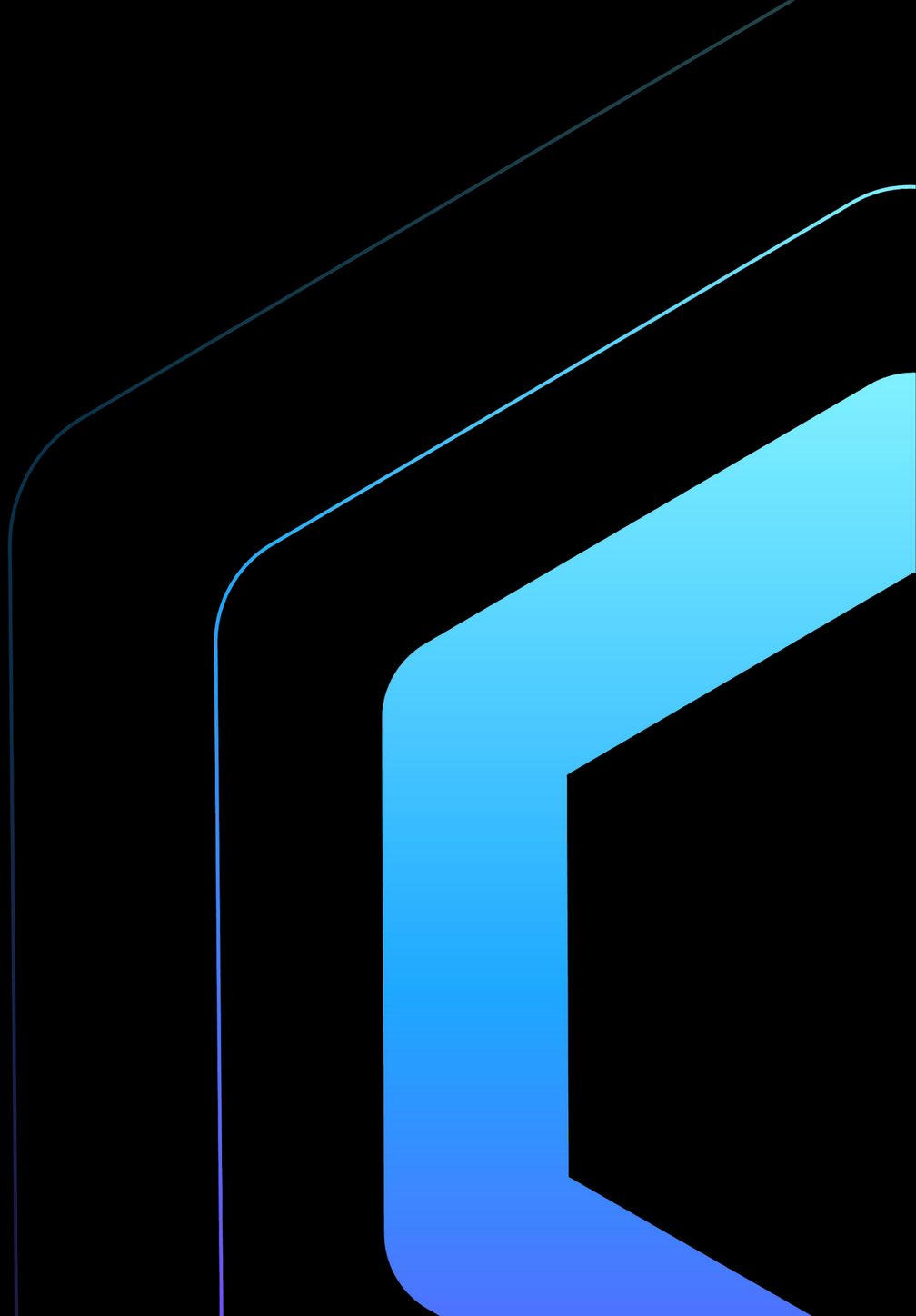
.....

```php
class AreaCalculator {
   public array $shapes = [];

   public function calculate(): int {
      $area = 0;
      foreach ($this->shapes as $shape) {
         switch get_class($shape){
            case 'Rectangle':
               $area += $shape->width * $shape->height;
            break;
            case 'Circle':
            /**/
         }
      }
      return $area;
   }
}
```



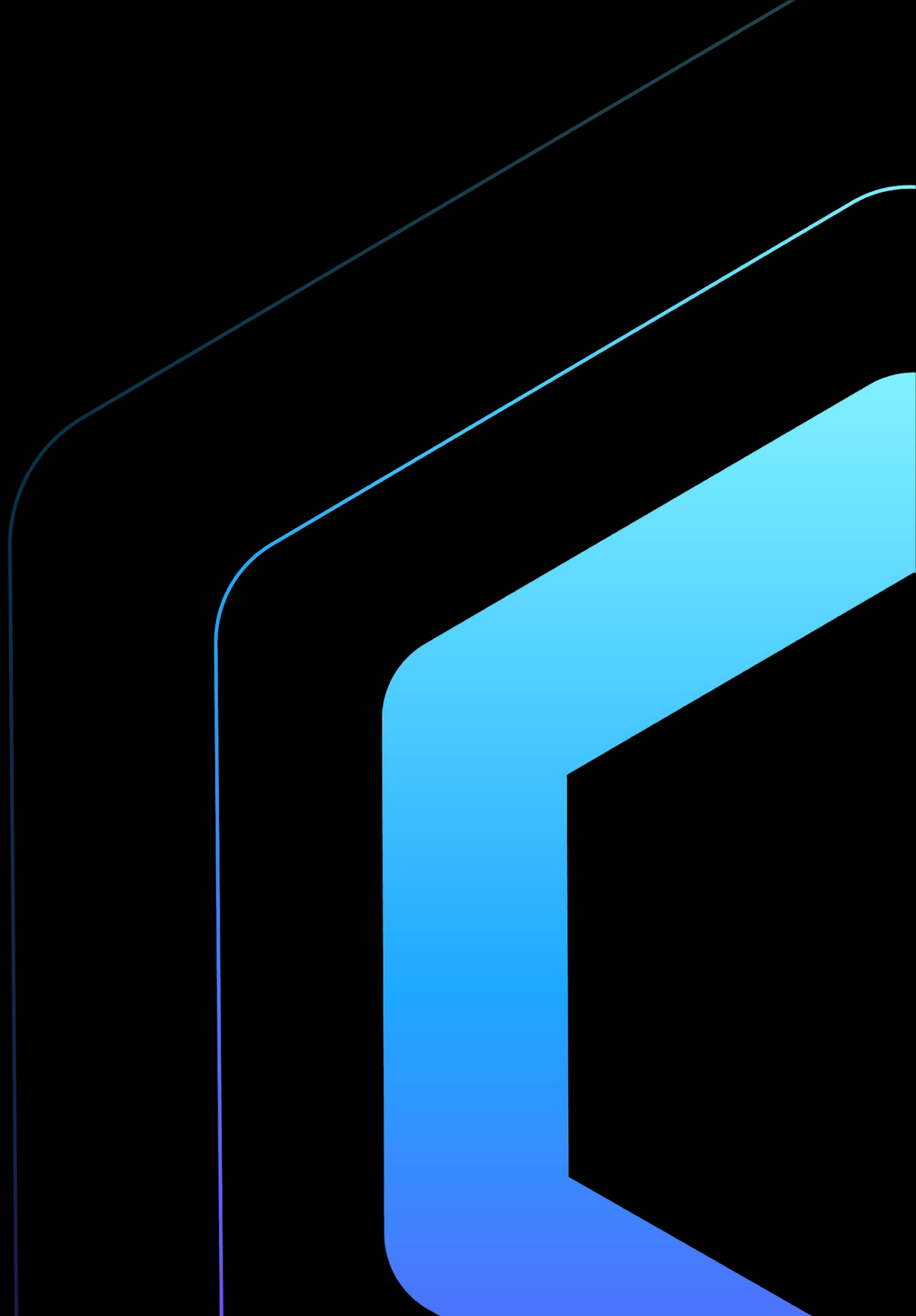
Open-closed Principle

```php
interface Shape {
    public function area(): int;
}

class AreaCalculator {
    private array $shapes;

    public function addShape(Shape $shape): void {
        $this->shapes[] = $shape;
    }

    public function calculate(): int {
        $area = 0;
        foreach ($this->shapes as $shape) {
            $area += $shape->area();
        }
        return $area;
    }
}
```



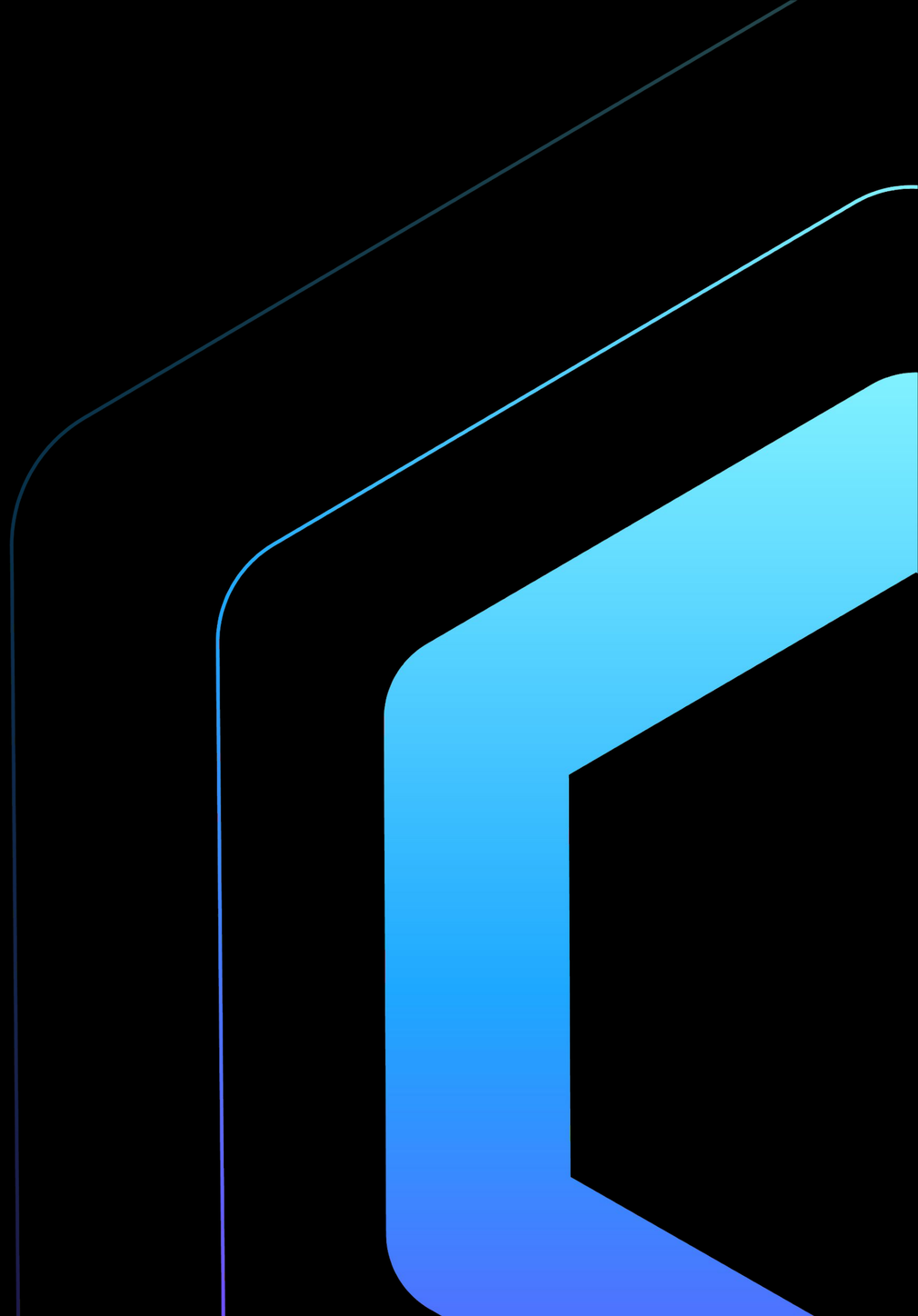
Open-closed Principle

```php
interface Shape {
   public function area(): int;
}

class Rectangle implements Shape {
   public function area(): int {
      return $this->width * $this->height;
   }
}

class Circle implements Shape {
   public function area(): int {
      return $this->radius * $this->radius * pi();
   }
}

class Square implements Shape {
   public function area(): int {
      return $this->width * $this->width;
   }
}
```



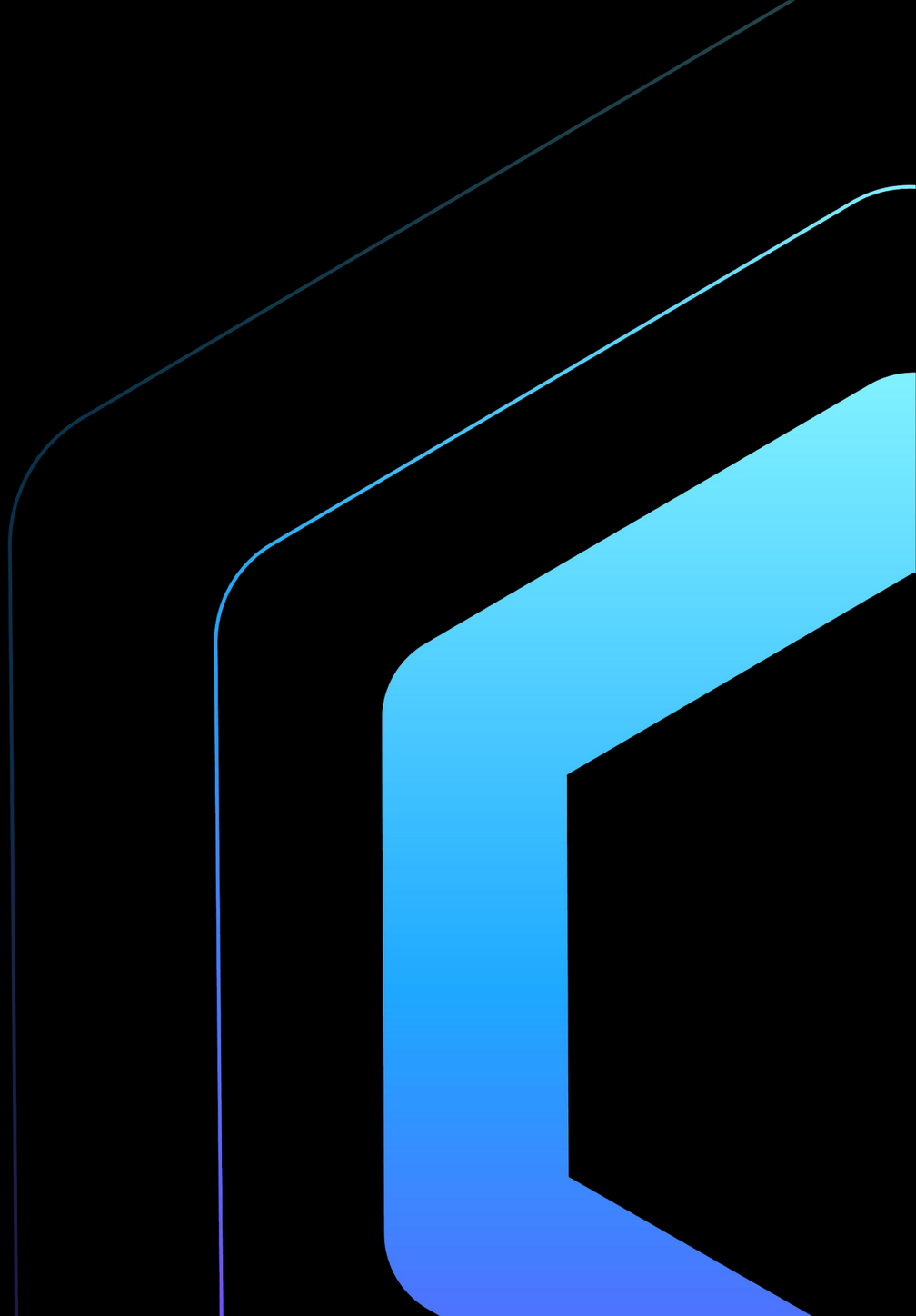
Open-closed Principle

```php
$areaCalculator = new Board;

$areaCalculator->addShape(new Square(5));
$areaCalculator->addShape(new Circle(10));
$areaCalculator->addShape(new Circle(3));
$areaCalculator->addShape(new Rectangle(3, 5));

echo $areaCalculator->calculate();
```

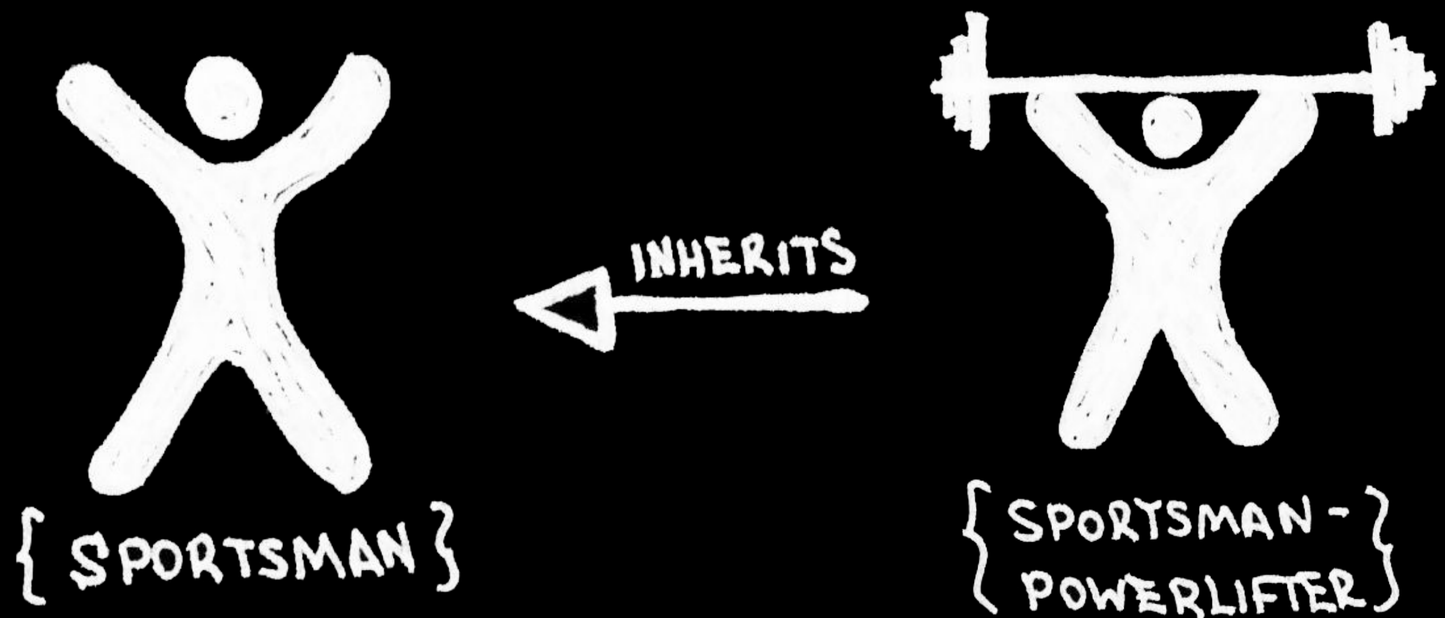


Open-closed Principle

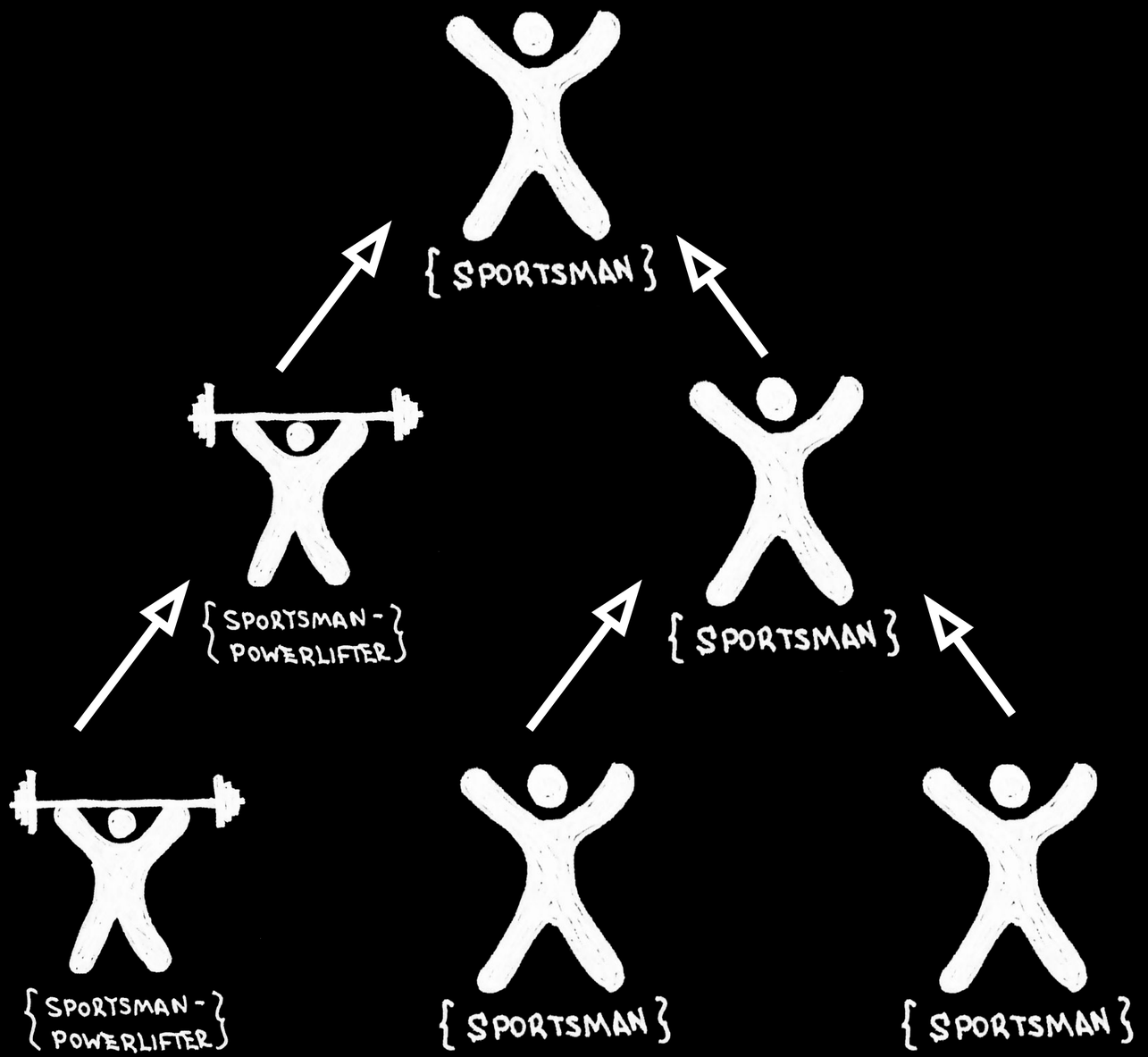
```
class Triangle implements Shape {
   public function area(): int {
      return $this->width * $this->height / 2;
   }
}

class Romb implements Shape {
   public function area(): int {
      return $this->width * $this->height;
   }
}

class Ellipse implements Shape {
   public function area(): int {
      return $this->radiusA * $this->radiusB * pi();
   }
}

$areaCalculator = new Board;
$areaCalculator->addShape(new Romb(5, 3));
$areaCalculator->addShape(new Circle(3));
$areaCalculator->addShape(new Ellipse(3, 5));
echo $board->calculateArea();
```



Liskov Substitution Principle

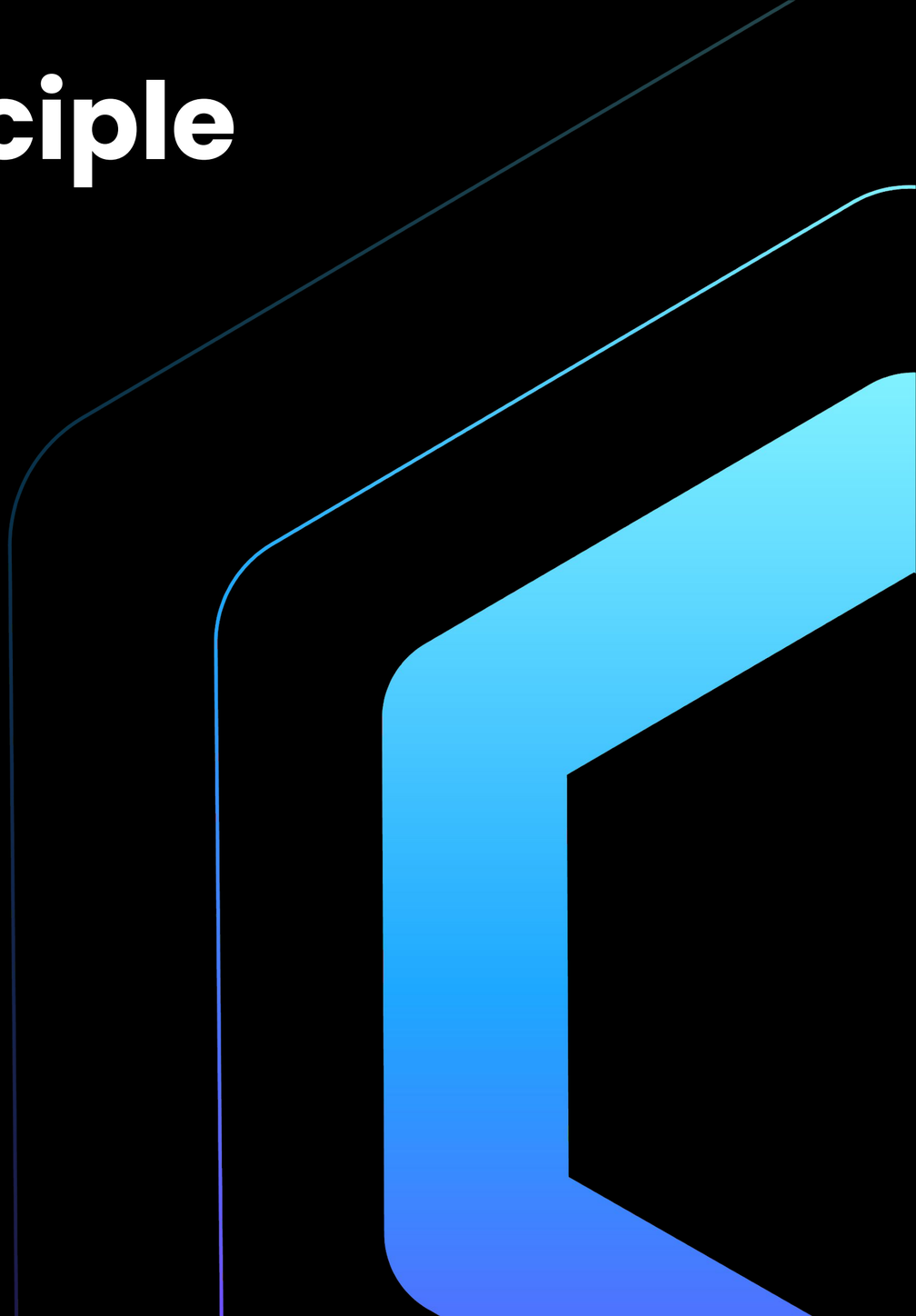


Liskov Substitution Principle



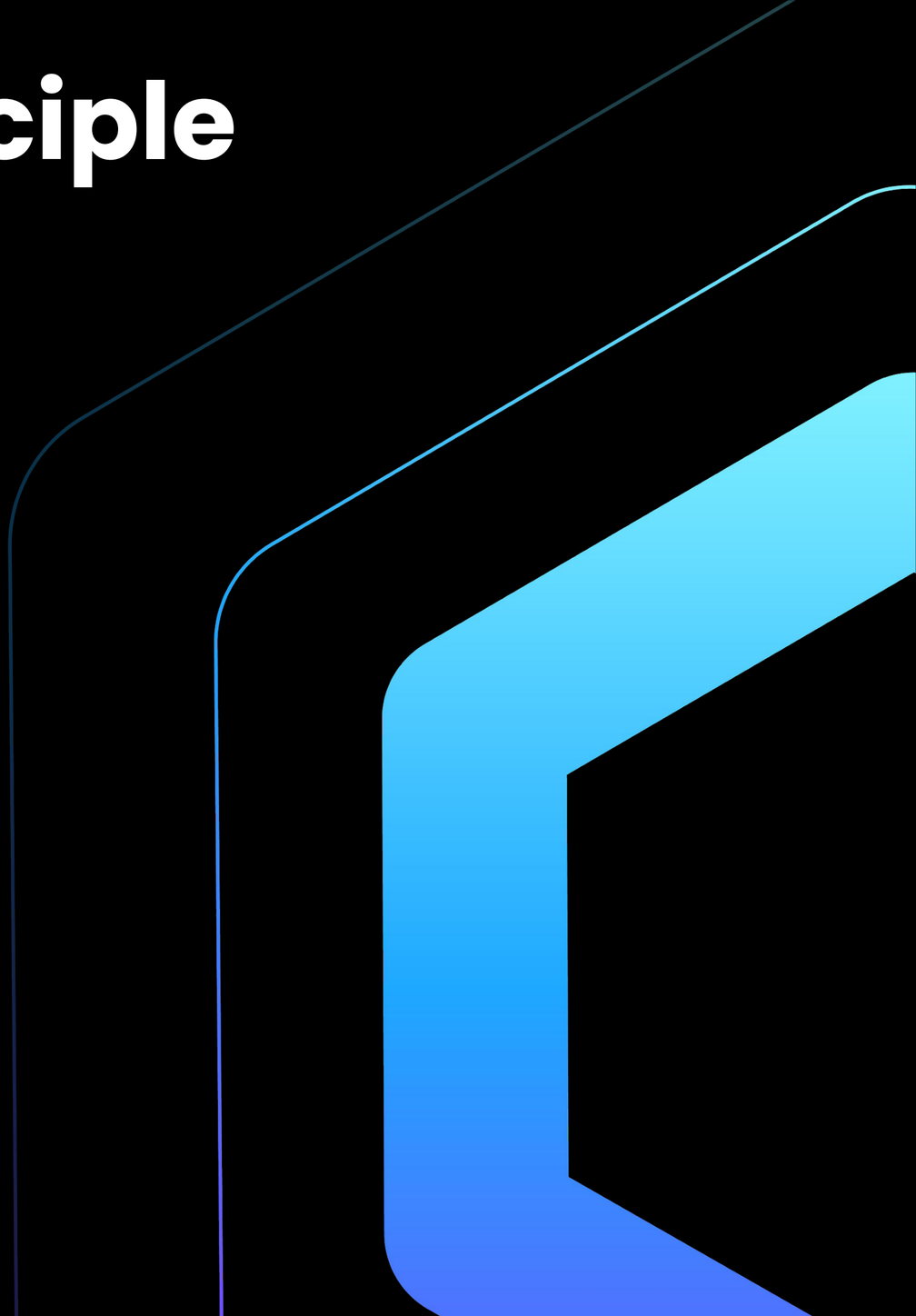
Liskov Substitution Principle

```php
class Shipping {
    public function total(int $weight, string $location): int {
        /**/
        if($price <= 0){
            throw new ZeroPriceException('Price can’t be lower than zero');
        }
        return $price;
    }

    private function secretFormula(): int {
        /**/
    }
}

class AirShipping extends Shipping {
    public function total(int $weight, string $location): int {
        /**/
        if($location === 'Spain'){
            return 0;
        }
        return $price
    }
}
```

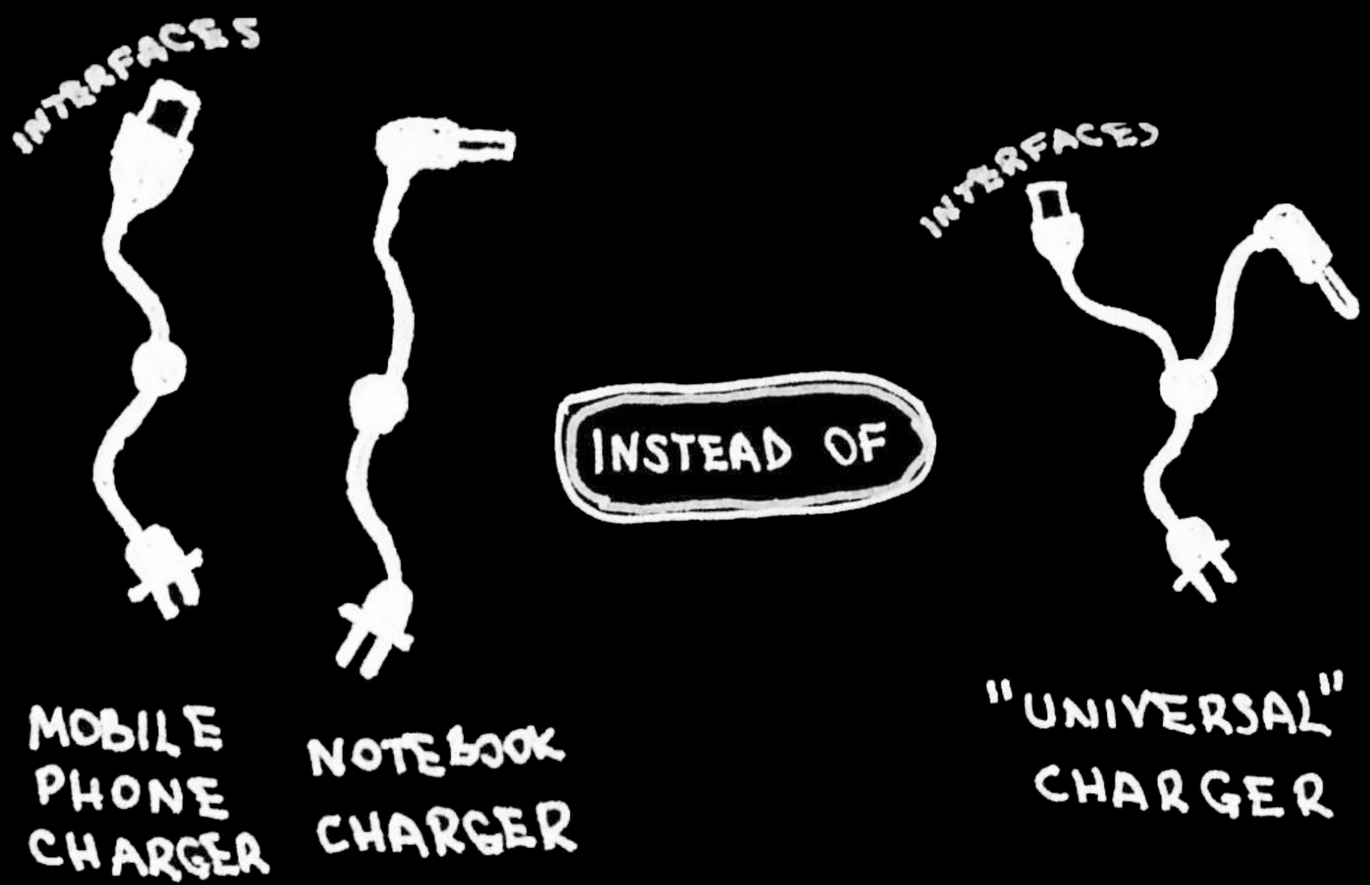


Liskov Substitution Principle

```php
abstract class BaseShipping {
    abstract public function calculate();

    private function calculator(): int {
        /**/
    }
}

class Shipping extends BaseShipping {
    public function total(int $weight, string $location): int {
        /**/
    }
}

class AirShipping extends BaseShipping {
    public function total(int $weight, string $location): int {
        /**/
    }
}
```



Interface Segregation Principle



Interface Segregation Principle

```php
interface Workable {
    public function canCode(): bool;
    public function code(): string;
    public function test(): string;
}
```

```php
class Developer implements Workable {
    public function canCode(): bool {
        return true;
    }
    public function code(): string {
        return 'coding';
    }
    public function test(): sring {
        return 'testing in localhost';
    }
}
```

```php
class Tester implements Workable {
    public function canCode(): bool {
        return false;
    }
    public function code(): string {
        throw new Exception('Opps! I can not code');
    }
    public function test(): sring {
        return 'testing in test server';
    }
}
```

```php
class ProjectManagement {
    public function processCode(Workable $member): void {
        if ($member->canCode()) {
            $member->code();
        }
    }
}
```

Interface Segregation Principle

```php
interface Testable
{
    public function test(): string;
}
```

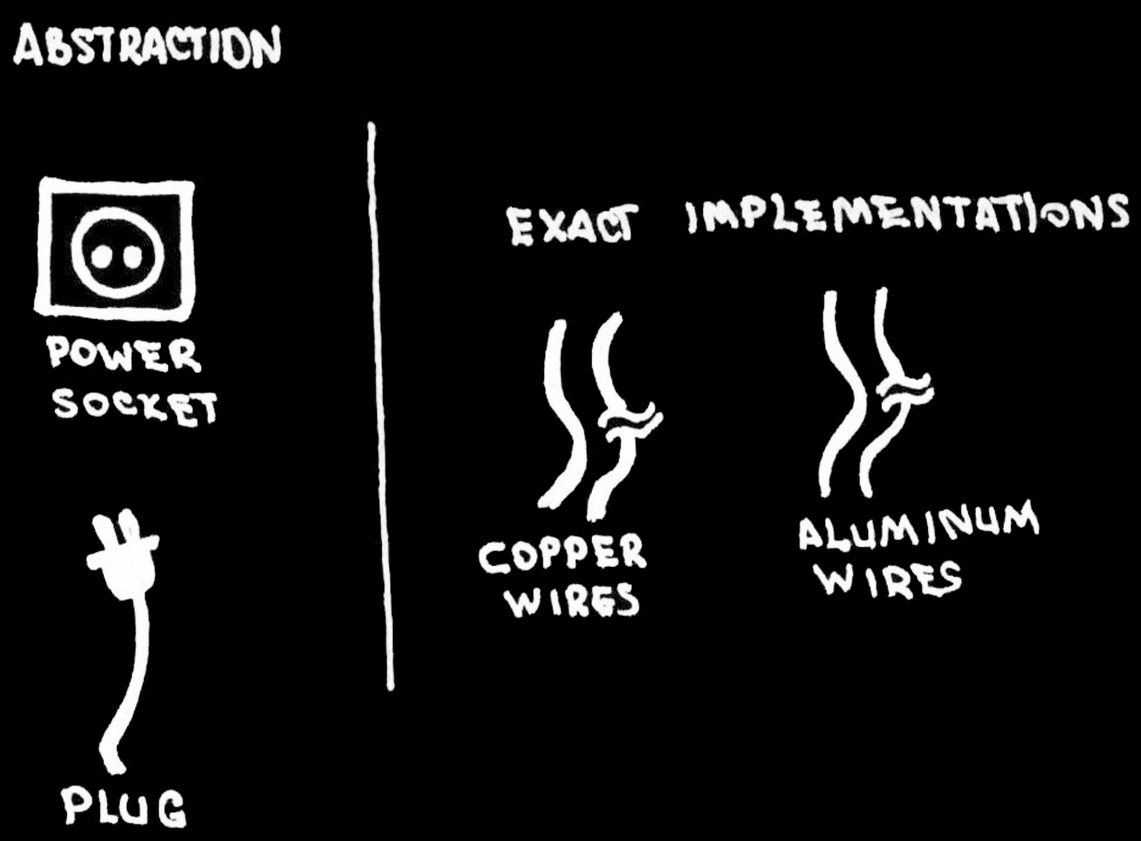
```php
interface Codeable
{
    public function code(): string;
}
```

```php
class Tester implements Testable {
    public function test(): string
    {
        return 'testing in test server';
    }
}
```

```php
class Programmer implements Codeable, Testable {
    public function code(): string
    {
        return 'coding';
    }
    public function test(): string
    {
        return 'testing in localhost';
    }
}
```

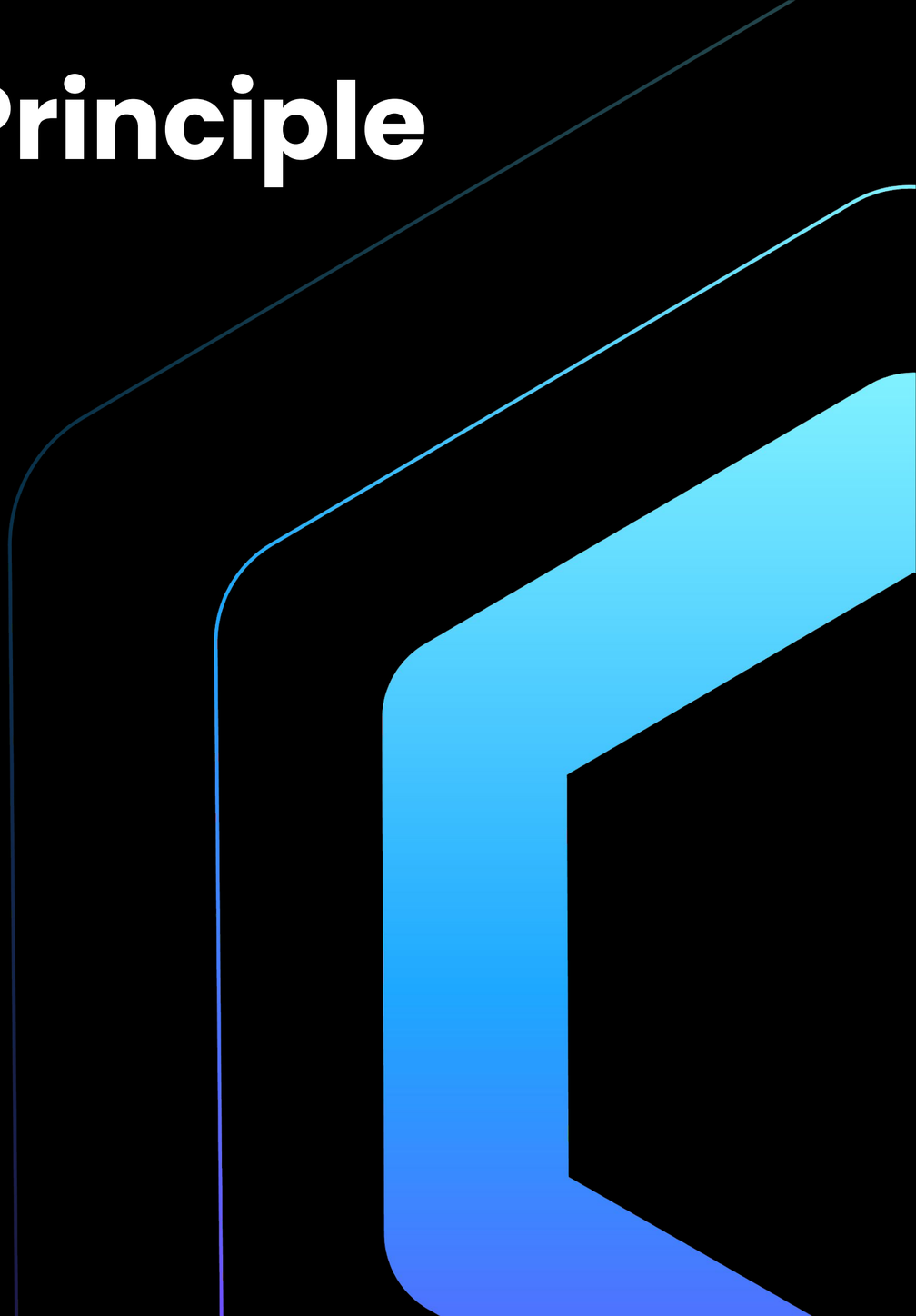
```php
class ProjectManagement {
    public function processCode(Codeable $member){
        $member->code();
    }
}
```

Dependency Inversion Principle



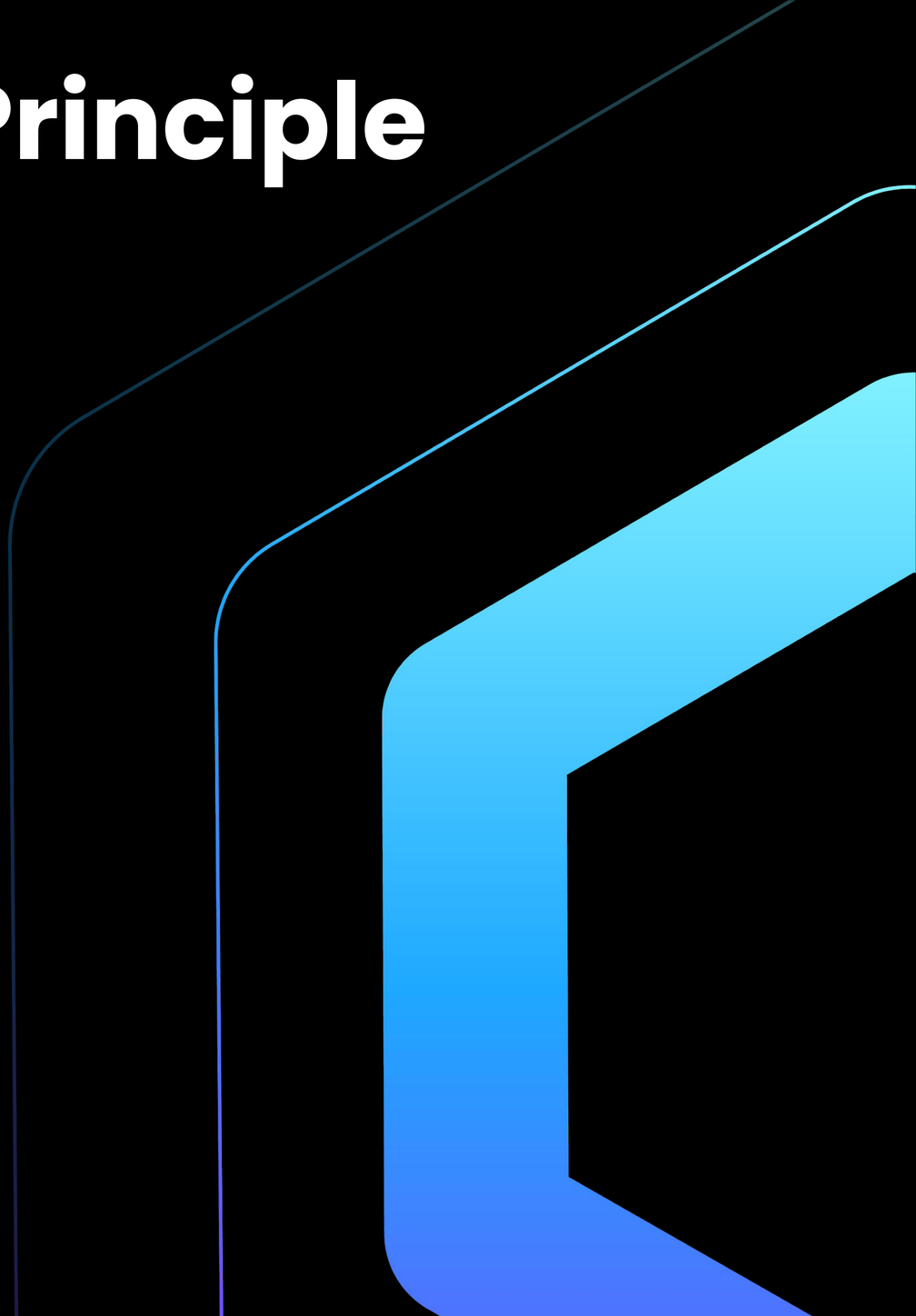
Dependency Inversion Principle

```php
class MySqlConnection {
    public function connect(): void {}
}

class Post {
    private MySqlConnection $dbConnection;
    public function __construct() {
        $this->dbConnection = new MySqlConnection;
        $this->dbConnection->connect();
    }
}
```



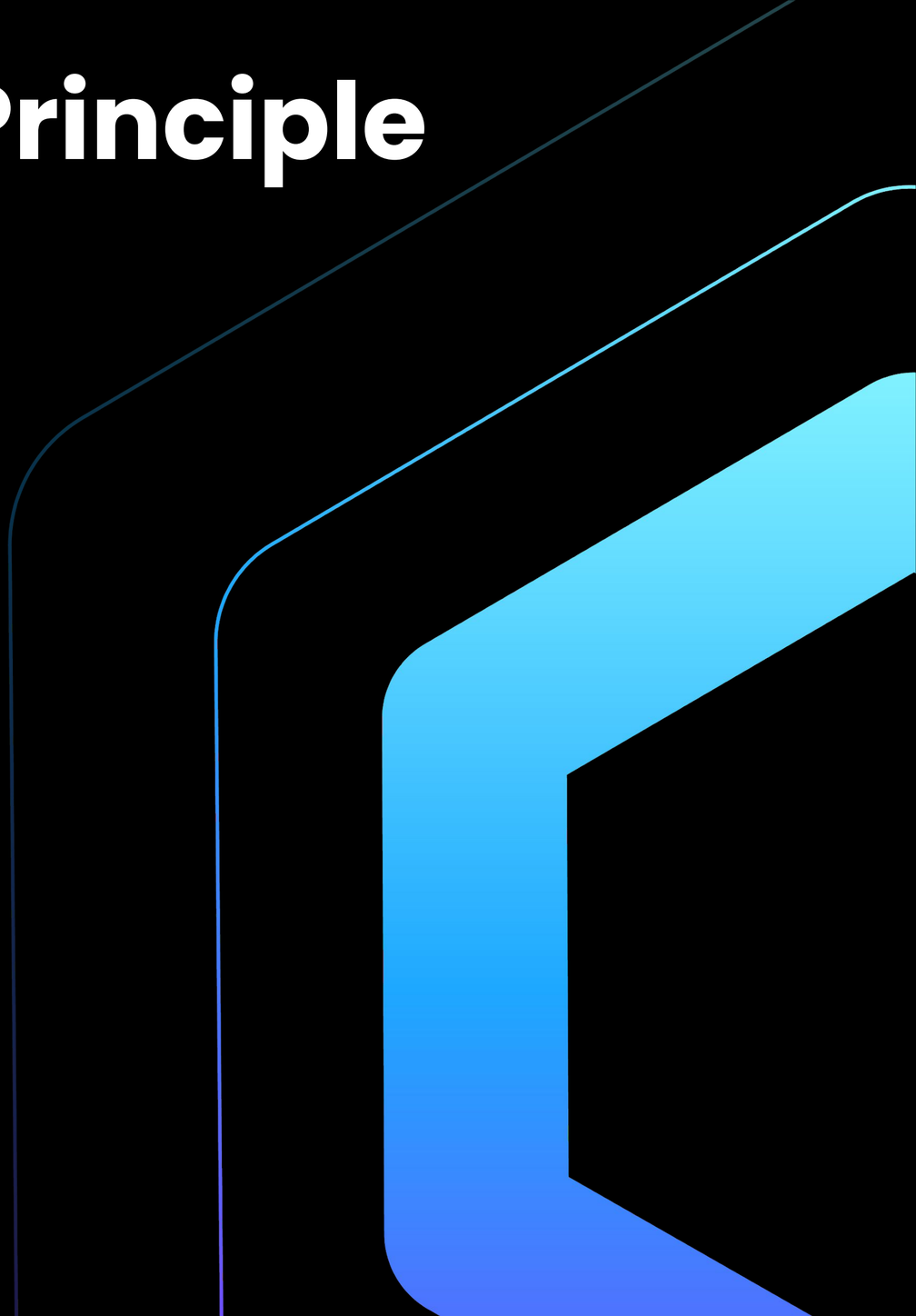
Dependency Inversion Principle

```php
class MySqlConnection {
    public function connect(): void {}
}

class Post {
    private MySqlConnection $dbConnection;
    public function __construct(MySqlConnection $dbConnection) {
        $this->dbConnection = $dbConnection;
        $this->dbConnection->connect();
    }
}
```



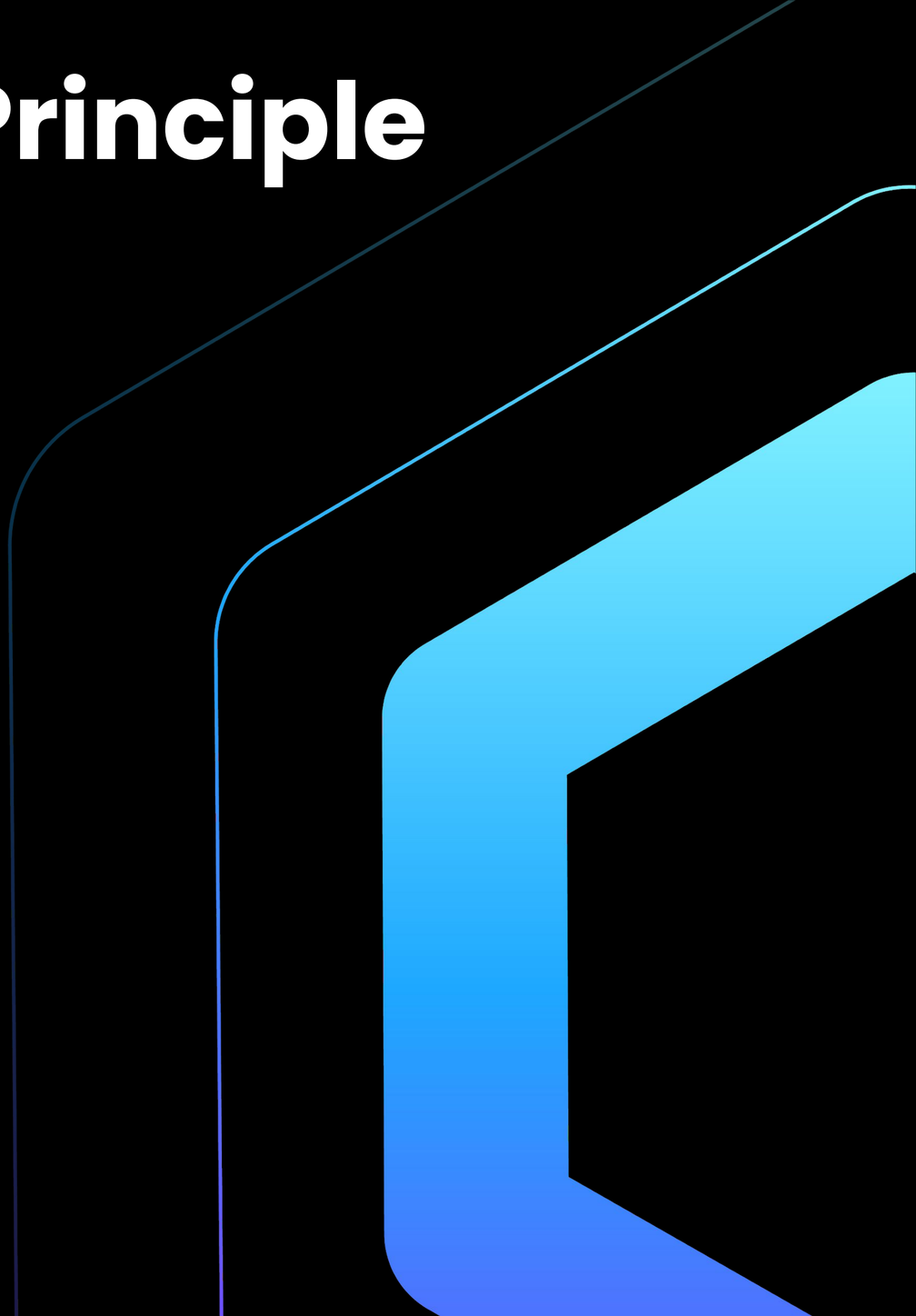
Dependency Inversion Principle

```php
interface DbConnectionInterface {
    public function connect(): void;
}

class MySqlConnection implements DbConnectionInterface {
    public function connect(): void {}
}

class Post {
    private DbConnectionInterface $dbConnection;
    public function __construct(DbConnectionInterface $dbConnection) {
        $this->dbConnection = $dbConnection;
        $this->dbConnection->connect();
    }
}
```



Dependency Inversion Principle

```php
interface DbConnectionInterface {
    public function connect(): void;
}

class MySqlConnection implements DbConnectionInterface {
    public function connect(): void {}
}

class SqlLiteConnection implements DbConnectionInterface {
    public function connect(): void {}
}

class Post {
    private DbConnectionInterface $dbConnection;
    public function __construct(DbConnectionInterface $dbConnection) {
        $this->dbConnection = $dbConnection;
        $this->dbConnection->connect();
    }
}
```



TeleSoftas

Questions?

TeleSoftas

Nerijus Žutautas

Nerjuz