SINGLETON

```php
class Singleton
{
    public static function getInstance()
    {
        static $instance = null;

        if ($instance === null) {
            $instance = new static();
        }
        return $instance;
    }

    /**
     * Private to disabled instantiation.
     */
    private function __construct() {}

    /**
     * Cannot clone a singleton.
     */
    private function __clone() {}

    /**
     * Cannot serialize a singleton.
     */
    private function __sleep() {}

    private function __wakeup()
    {
        throw new \Exception("Cannot unserialize a singleton.");
    }
}

class Database extends Singleton
{
    protected $userName;

    public function setUserName(string $userName): void
    {
        $this->userName = $userName;
    }

    public function printUserName(): void
    {
        echo $this->userName . PHP_EOL;
    }
}


$database1 = Database::getInstance();
$database1->setUserName('John');
$database1->printUserName();

$database2 = Database::getInstance();
$database2->printUserName();

$database2->setUserName('Jane');
$database1->printUserName();
$database2->printUserName();
```

FACTORY SIMPLE

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

class Square implements Shape
{
    public function draw()
    {
        echo 'Inside Square::draw() method.';
    }
}

class Circle implements Shape
{
    public function draw()
    {
        echo 'Inside Circle::draw() method.';
    }
}

class ShapeFactory
{
    public static function getShape(String $shapeType): ?Shape
    {
        if ($shapeType === "circle") {
            return new Circle();
        }

        if($shapeType === "square") {
            return new Square();
        }

        throw new Exception('Shape type not found.');
    }
}


$circle = ShapeFactory::getShape("circle");
$circle->draw();

$square = ShapeFactory::getShape("square");
$square->draw();
```

FACTORY METHOD

```php
abstract class Creator
{
    abstract public function factoryMethod(): Product;

    public function someOperation(): string
    {
        $product = $this->factoryMethod();

        return "Creator: " . $product->operation();
    }
}

class ConcreteCreator1 extends Creator
{
    public function factoryMethod(): Product
    {
        return new ConcreteProduct1();
    }
}

class ConcreteCreator2 extends Creator
{
    public function factoryMethod(): Product
    {
        return new ConcreteProduct2();
    }
}

interface Product
{
    public function operation(): string;
}

class ConcreteProduct1 implements Product
{
    public function operation(): string
    {
        return "{Result of the ConcreteProduct1}";
    }
}

class ConcreteProduct2 implements Product
{
    public function operation(): string
    {
        return "{Result of the ConcreteProduct2}";
    }
}


function clientCode(Creator $creator)
{
    echo "Client: " . $creator->someOperation();
}

clientCode(new ConcreteCreator1());
clientCode(new ConcreteCreator2());
```

Ejemplo

```php
abstract class SocialNetworkPoster
{
    abstract public function getSocialNetwork(): SocialNetworkConnector;

    public function post($content): void
    {
        $network = $this->getSocialNetwork();

        $network->logIn();
        $network->createPost($content);
        $network->logout();
    }
}

class FacebookPoster extends SocialNetworkPoster
{
    private $login, $password;

    public function __construct(string $login, string $password)
    {
        $this->login = $login;
        $this->password = $password;
    }

    public function getSocialNetwork(): SocialNetworkConnector
    {
        return new FacebookConnector($this->login, $this->password);
    }
}

class LinkedInPoster extends SocialNetworkPoster
{
    private $email, $password;

    public function __construct(string $email, string $password)
    {
        $this->email = $email;
        $this->password = $password;
    }

    public function getSocialNetwork(): SocialNetworkConnector
    {
        return new LinkedInConnector($this->email, $this->password);
    }
}

interface SocialNetworkConnector
{
    public function logIn(): void;

    public function logOut(): void;

    public function createPost($content): void;
}
```

```php
class FacebookConnector implements SocialNetworkConnector
{
    private $login, $password;

    public function __construct(string $login, string $password)
    {
        $this->login = $login;
        $this->password = $password;
    }

    public function logIn(): void
    {
        echo "Send HTTP API request to log in user $this->login\n";
    }

    public function logOut(): void
    {
        echo "Send HTTP API request to log out user $this->login\n";
    }

    public function createPost($content): void
    {
        echo "Send HTTP API requests to create a post in Facebook timeline.\n";
    }
}

class LinkedInConnector implements SocialNetworkConnector
{
    private $email, $password;

    public function __construct(string $email, string $password)
    {
        $this->email = $email;
        $this->password = $password;
    }

    public function logIn(): void
    {
        echo "Send HTTP API request to log in user $this->email\n";
    }

    public function logOut(): void
    {
        echo "Send HTTP API request to log out user $this->email\n";
    }

    public function createPost($content): void
    {
        echo "Send HTTP API requests to create a post in LinkedIn timeline.\n";
    }
}


function clientCode(SocialNetworkPoster $creator)
{
    $creator->post("Hello world!");
    $creator->post("I had a large hamburger this morning!");
}

clientCode(new FacebookPoster("john_smith", "******"));
clientCode(new LinkedInPoster("john_smith@example.com", "******"));
```

FACTORY ABSTRACT

```
interface AbstractFactory
{
    public function createSquare(): AbstractProductA;
    public function createTriangle(): AbstractProductA;
}

class ShapeFactory implements AbstractFactory
{
    public function createSquare(): Shape
    {
        return new Square();
    }

    public function createTriangle(): Shape { // ... }
}

class RoudedShapeFactory implements AbstractFactory
{
    public function createSquare(): Shape
    {
        return new RoundedSquare();
    }

    public function createTriangle(): Shape { // ... }
}

interface Shape
{
    public function draw();
}

class Square implements Shape
{
    public function draw()
    {
        echo 'Inside Square::draw() method.';
    }
}

class RoundedSquare implements Shape
{
    public function draw()
    {
        echo 'Inside RoundedSquare::draw() method.';
    }
}

function clientCode(AbstractFactory $factory)
{
    $square = $factory->createSquare();
    $square->draw();
}

clientCode(new ShapeFactory());
clientCode(new RoudedShapeFactory());
```