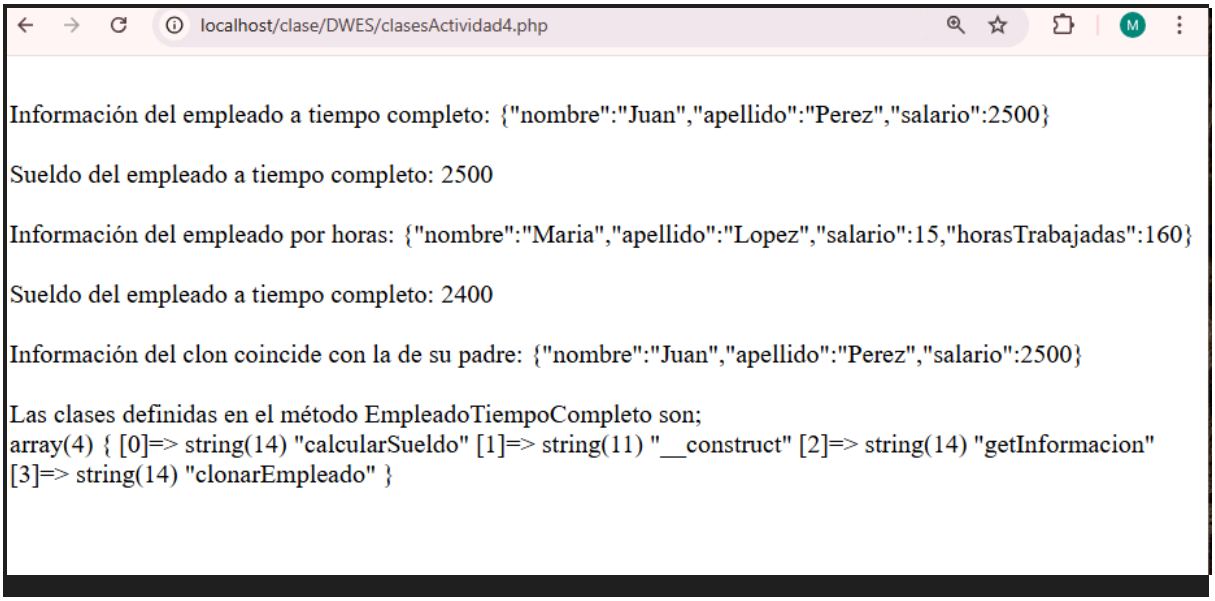
Actividad 4: Sistema de Gestión de Empleados

- Crea una clase abstracta `Empleado` con propiedades como `nombre`, `apellido`, `salario` y un método abstracto `calcularSueldo()`.
- Crea subclases como `EmpleadoTiempoCompleto` y `EmpleadoPorHoras` que hereden de `Empleado` y sobreescriban el método `calcularSueldo()` para calcular el salario de cada tipo de empleado.
- Utiliza introspección para obtener información sobre las propiedades y métodos de cada clase de empleado.
- Implementa un método `clonarEmpleado()` que permita crear una copia exacta de un empleado, incluyendo su tipo (tiempo completo o por horas).

SOLUCIÓN

```php
<?php
abstract class Empleado {
    protected $nombre;
    protected $apellido;
    protected $salario;

    public function __construct($nombre, $apellido, $salario) {
        $this->nombre = $nombre;
        $this->apellido = $apellido;
        $this->salario = $salario;
    }

    abstract public function calcularSueldo();

    public function getInformacion() {
        $propiedades = get_object_vars($this);
        return json_encode($propiedades);
    }

    public function clonarEmpleado() {
        return clone $this;
    }
}

class EmpleadoTiempoCompleto extends Empleado {
    public function calcularSueldo() {
        // Lógica para calcular el salario de un empleado a tiempo
completo
        return $this->salario;
```

```
    }
}

class EmpleadoPorHoras extends Empleado {
    protected $horasTrabajadas;

    public function __construct($nombre, $apellido, $salario,
$horasTrabajadas) {
        parent::__construct($nombre, $apellido, $salario);
        $this->horasTrabajadas = $horasTrabajadas;
    }

    public function calcularSueldo() {
        // Lógica para calcular el salario de un empleado por horas
        return $this->salario * $this->horasTrabajadas;
    }
}

// Ejemplo de uso
$empleadoTiempoCompleto = new EmpleadoTiempoCompleto("Juan", "Perez",
2500);
$empleadoPorHoras = new EmpleadoPorHoras("Maria", "Lopez", 15, 160);

echo "<br>Información del empleado a tiempo completo: " .
$empleadoTiempoCompleto->getInformacion() . "<br>";
echo "<br>Sueldo del empleado a tiempo completo: " .
$empleadoTiempoCompleto->calcularSueldo() . "<br>";

echo "<br>Información del empleado por horas: " .
$empleadoPorHoras->getInformacion() . "<br>";
echo "<br>Sueldo del empleado a tiempo completo: " .
$empleadoPorHoras->calcularSueldo() . "<br>";

$clon = $empleadoTiempoCompleto->clonarEmpleado();
echo "<br>Información del clon coincide con la de su padre: " .
$clon->getInformacion() . "<br>";

// Utilizando introspección para obtener información sobre las clases
echo "<br>Las clases definidas en el método EmpleadoTiempoCompleto
son;<br>";
var_dump(get_class_methods('EmpleadoTiempoCompleto'));
?>
```


localhost/clase/DWES/clasesActividad4.php
Información del empleado a tiempo completo: {"nombre":"Juan","apellido":"Perez","salario":2500}
Sueldo del empleado a tiempo completo: 2500
Información del empleado por horas: {"nombre":"Maria","apellido":"Lopez","salario":15,"horasTrabajadas":160}
Sueldo del empleado a tiempo completo: 2400
Información del clon coincide con la de su padre: {"nombre":"Juan","apellido":"Perez","salario":2500}
Las clases definidas en el método EmpleadoTiempoCompleto son;
array(4) { [0]=> string(14) "calcularSueldo" [1]=> string(11) "__construct" [2]=> string(14) "getInformacion"
[3]=> string(14) "clonarEmpleado" }

Actividad 5: Sistema de Gestión de Vehículos

- Crea una clase abstracta `Vehiculo` con propiedades como `marca`, `modelo`, `año` y un método abstracto `calcularImpuesto()`.
- Crea subclases como `Coche`, `Moto`, `Camion` que hereden de `Vehiculo` y sobreescriban el método `calcularImpuesto()` para calcular el impuesto correspondiente a cada tipo de vehículo.
- Utiliza introspección para obtener información sobre las propiedades y métodos de cada vehículo.
- Implementa un método `clonarVehiculo()` que permita crear una copia exacta de un vehículo.

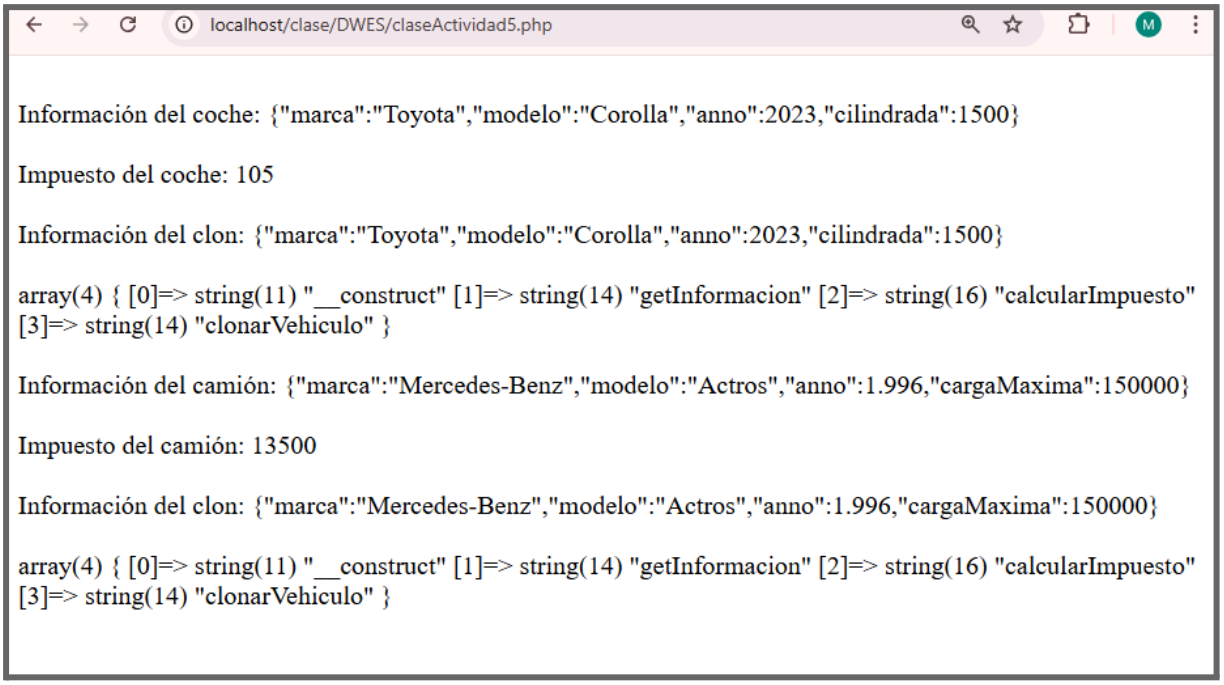
SOLUCIÓN

```php
<?php
abstract class Vehiculo {
    protected $marca;
    protected $modelo;
    protected $anno;
    public function __construct($marca, $modelo, $anno) {
        $this->marca = $marca;
        $this->modelo = $modelo;
        $this->anno = $anno;
    }
    abstract public function calcularImpuesto();
    public function getInformacion() {
        $propiedades = get_object_vars($this);
        return json_encode($propiedades);
    }
    public function clonarVehiculo() {
        return clone $this;
    }
}

class Coche extends Vehiculo {
    private $cilindrada;

    public function __construct($marca, $modelo, $anno, $cilindrada) {
        parent::__construct($marca, $modelo, $anno);
        $this->cilindrada = $cilindrada;
    }
    public function getInformacion() {
        $propiedades = get_object_vars($this);
        // Agregamos la cilindrada a la salida
```

```
        $propiedades['cilindrada'] = $this->cilindrada;
        return json_encode($propiedades);
    }
    public function calcularImpuesto() {
        $impuesto = 0;
        // Ejemplo simplificado basado en tramos de cilindrada (ajustar
a la normativa)
        if ($this->cilindrada <= 1000) {
            $impuesto = $this->cilindrada * 0.05; // 5% para
cilindradas menores o iguales a 1000cc
        } else if ($this->cilindrada <= 1500) {
            $impuesto = $this->cilindrada * 0.07; // 7% para
cilindradas entre 1001cc y 1500cc
        } else {
            $impuesto = $this->cilindrada * 0.09; // 9% para
cilindradas superiores a 1500cc
        }
        return $impuesto;
    }
}

class Moto extends Vehiculo {
    private $cilindrada;
    public function __construct($marca, $modelo, $anno, $cilindrada) {
        parent::__construct($marca, $modelo, $anno);
        $this->cilindrada = $cilindrada;
    }
    public function getInformacion() {
        $propiedades = get_object_vars($this);
        // Agregamos la cilindrada a la salida
        $propiedades['cilindrada'] = $this->cilindrada;
        return json_encode($propiedades);
    }
    public function calcularImpuesto() {
        $impuesto = 0;
        // Ejemplo simplificado basado en tramos de cilindrada (ajustar
a la normativa)
        if ($this->cilindrada <= 1000) {
            $impuesto = $this->cilindrada * 0.05; // 5% para
cilindradas menores o iguales a 1000cc
        } else if ($this->cilindrada <= 1500) {
            $impuesto = $this->cilindrada * 0.07; // 7% para
cilindradas entre 1001cc y 1500cc
```

```
        } else {
            $impuesto = $this->cilindrada * 0.09; // 9% para
cilindradas superiores a 1500cc
        }
        return $impuesto;
    }
}

class Camion extends Vehiculo {
    private $cargaMaxima;
    public function __construct($marca, $modelo, $anno, $cargaMaxima) {
        parent::__construct($marca, $modelo, $anno);
        $this->cargaMaxima = $cargaMaxima;
    }
    public function getInformacion() {
        $propiedades = get_object_vars($this);
        // Agregamos la carga máxima a la salida
        $propiedades['cargaMaxima'] = $this->cargaMaxima;
        return json_encode($propiedades);
    }
    public function calcularImpuesto() {
        $impuesto = 0;
        // Ejemplo simplificado basado en tramos de cilindrada (ajustar
a la normativa)
        if ($this->cargaMaxima <= 10000) {
            $impuesto = $this->cargaMaxima * 0.05; // 5% para
cilindradas menores o iguales a 1000cc
        } else if ($this->cargaMaxima <= 15000) {
            $impuesto = $this->cargaMaxima * 0.07; // 7% para
cilindradas entre 1001cc y 1500cc
        } else {
            $impuesto = $this->cargaMaxima * 0.09; // 9% para
cilindradas superiores a 1500cc
        }
        return $impuesto;
    }
}

// Ejemplo de uso
$coche = new Coche("Toyota", "Corolla", 2023, 1500);
echo "<br>Información del coche: " . $coche->getInformacion() . "<br>";
echo "<br>Impuesto del coche: " . $coche->calcularImpuesto() . "<br>";
$clon = $coche->clonarVehiculo();
```

```php
echo "<br>Información del clon: " . $clon->getInformacion() . "<br>";
// Utilizando introspección para obtener información sobre las clases
echo "<br>";
var_dump(get_class_methods('Coche'));
echo "<br>";

$camion = new Camion("Mercedes-Benz", "Actros", 1.996, 150000);
echo "<br>Información del camión: " . $camion->getInformacion() .
"<br>";
echo "<br>Impuesto del camión: " . $camion->calcularImpuesto() .
"<br>";
$clon = $camion->clonarVehiculo();
echo "<br>Información del clon: " . $clon->getInformacion() . "<br>";
// Utilizando introspección para obtener información sobre las clases
echo "<br>";
var_dump(get_class_methods('Camion'));
echo "<br>";
?>
```



Actividad 6: Sistema de Gestión de Figuras Geométricas

- Crea una clase abstracta `Figura` con propiedades como `color` y un método abstracto `calcularArea()`.
- Crea subclases como `Circulo`, `Rectangulo`, `Triangulo` que hereden de `Figura` y sobreescriban el método `calcularArea()` para calcular el área de cada figura.
- Utiliza introspección para obtener información sobre las propiedades y métodos de cada figura.
- Implementa un método `clonarFigura()` que permita crear una copia exacta de una figura.

SOLUCIÓN

```php
<?php
abstract class Figura {
    public $color;

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

    abstract public function calcularArea();

    public function getInformacion() {
        $propiedades = get_object_vars($this);
        return json_encode($propiedades);
    }

    public function clonarFigura() {
        return clone $this;
    }
}

class Circulo extends Figura {
    private $radio;

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

    public function calcularArea() {
        return pi() * pow($this->radio, 2);
    }
```

```php
}

class Rectangulo extends Figura {
    private $base;
    private $altura;

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

    public function calcularArea() {
        return $this->base * $this->altura;
    }
}

class Triangulo extends Figura {
    private $base;
    private $altura;

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

    public function calcularArea() {
        return 0.5 * $this->base * $this->altura;
    }
}

// Ejemplo de uso
$circulo = new Circulo("rojo", 5);
echo "El color del círculo es: " . $circulo->color . "<br>";
echo "Información del círculo: " . $circulo->getInformacion() . "<br>";
echo "Área del círculo: " . $circulo->calcularArea() . "<br>";

$rectangulo = new Rectangulo("azul", 4, 6);
echo "El color del rectángulo es: " . $circulo->color . "<br>";
echo "Información del rectángulo: " . $rectangulo->getInformacion() .
"<br>";
echo "Área del rectángulo: " . $rectangulo->calcularArea() . "<br>";
```

```php
$triangulo = new Triangulo("verde", 3, 4);
echo "El color del triángulo elegido es: " . $circulo->color . "<br>";
echo "Información del triángulo: " . $triangulo->getInformacion() .
"<br>";
echo "Área del triángulo: " . $triangulo->calcularArea() . "<br>";
?>
```