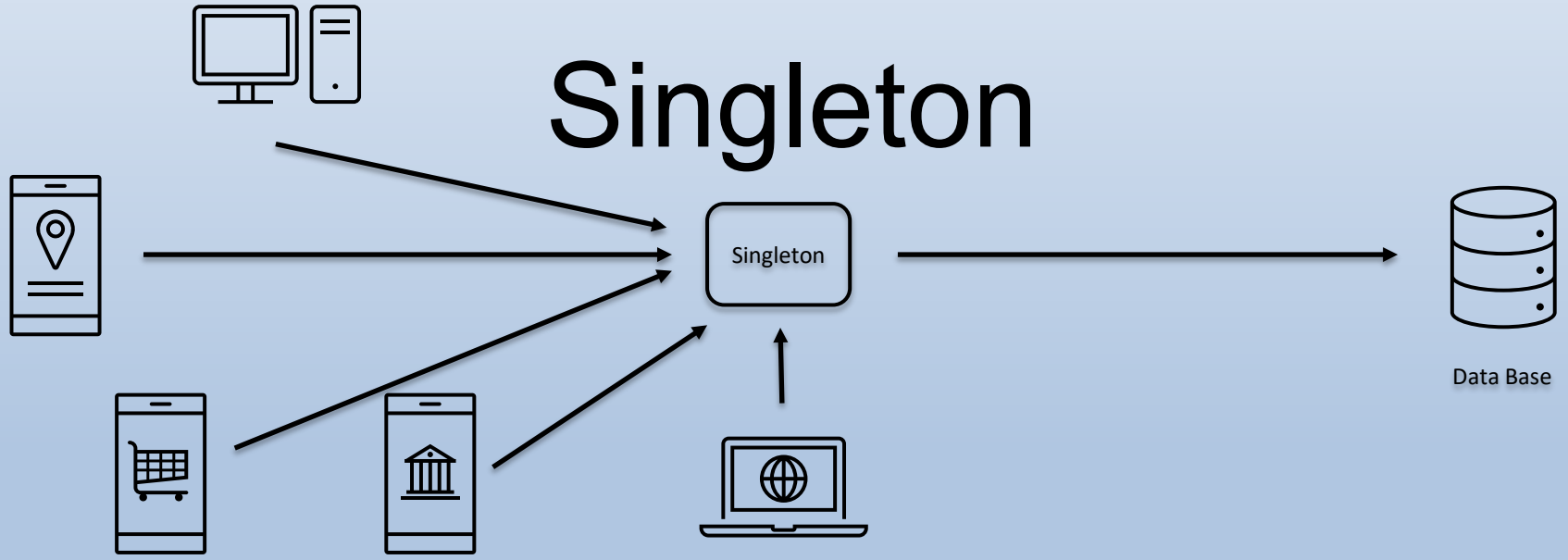
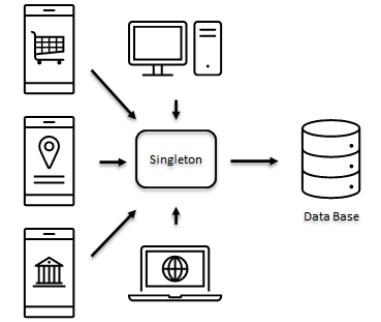
Patrón de diseño Singleton



Singleton

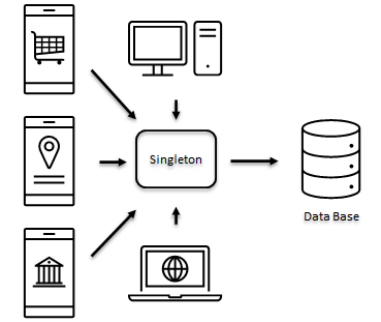


¿Por qué y qué es?

El patrón **Singleton** se usa, **y debería usarse únicamente**, cuando **una clase deba tener siempre una única instancia durante la ejecución de la aplicación**. Garantizando que en cualquier punto de la aplicación que se use esa clase lo hagamos mediante la misma instancia.

El principal motivo es evitar tener demasiadas instancias que tengan un recurso pesado o, mantener siempre el estado idéntico en todas las partes, haciéndolo global. Normalmente hablamos de conexiones a las BB.DD., controladores de configuraciones o similares.

Singleton



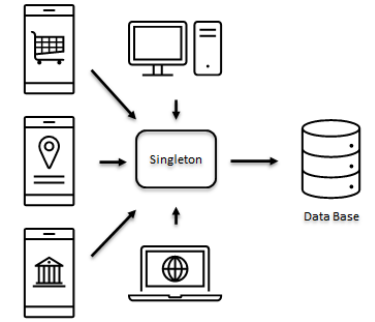
¿Por qué no usarlo?

Cuando desarrollamos **tendemos a complicar el código demasiado**, pensando que con ello vamos a convertirnos en grandes desarrolladores. Y eso es muy habitual en el uso de muchos patrones, y también lo es en el Singleton.

El patrón Singleton tiene un propósito muy específico, y es muy útil, pero no siempre se usa de forma adecuada. Porque si no necesitamos una única instancia, no es necesario usarlo.

Al ser muy fácil de usar se tiende a implementar por todas partes, complicando el código, haciéndolo insostenible o dificultando su mantenimiento de forma innecesarea. Por ello, mucha gente lo define como el antipatrón.

Singleton



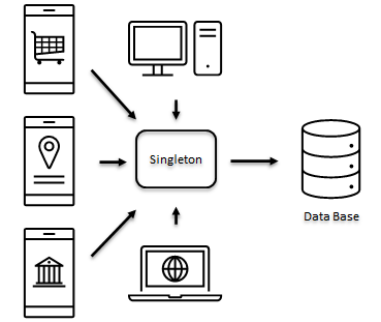
Ejemplo

Un ejemplo de uso podría ser la carga de una configuración en una aplicación, por ejemplo, de carga de credenciales para un gestor de URL.

1. Cargamos la configuración desde un sistema de archivos.
2. Luego lo almacenarlo en una matriz o estructura de datos, por ejemplo un JSON.
3. Y posteriormente para usarlo en cada una de las instancias, crearíamos un nuevo objeto que se encargaría de conectar y leer los datos del JSON.

Con esto cada vez que lo instanciamos hacemos estos pasos continuamente, leemos, pasamos a JSON y guardamos en un **nuevo** objeto.

Singleton



Ejemplo II

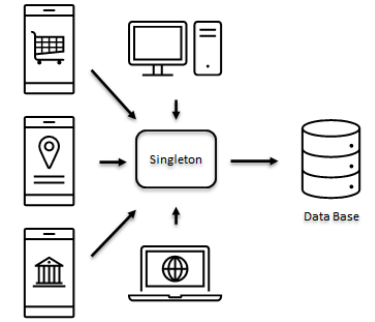
Imaginaros que nuestra aplicación tiene un archivo/s de configuración muy pesado y grande. **Convirtiendo el uso de esta carga de credenciales en algo muy "tedioso" y consumiendo recursos innecesariamente**.

Lo más optimo sería realizar esa carga la primera vez que accedemos a la configuración y el resto de ocasiones usar la misma carga.

Para ello, vamos a usar las propiedades y lo métodos estáticos. Cargando la configuración en una matriz estática dentro del objeto de configuración que vamos a usar durante toda la aplicación.

Lo instancias una vez, y luego llamas a un método estático que será el encargado de acceder a la propiedad estática cargada previamente.

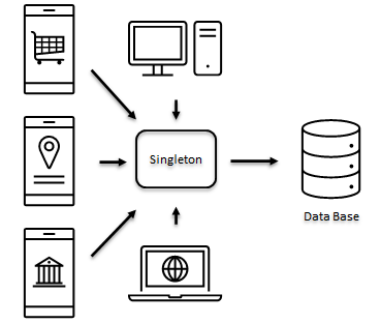
Singleton



Ejemplo III

```php
<?php

namespace Config;

class Config
{
    private static $data;
    public function __construct()
    {
        $json = file_get_contents(__DIR__.'/../config/app.json');
        self::$data = json_decode($json, true);
    }

    public static function get($key)
    {
        return self::$data($key);
    }
}

```

Singleton

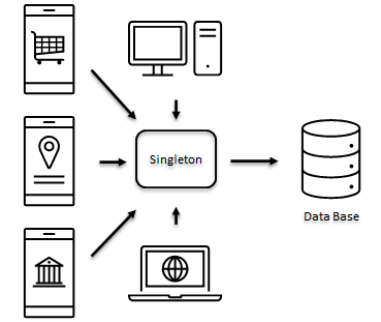


Ejemplo IV

Como hemos visto en el código anterior parece una buena idea, pero es muy peligroso, porque no controlamos si el usuario sigue instanciando la clase una y otra vez.

Y es aquí donde los Singleton son útiles, para evitar justamente que se vuelva a instanciar si ya lo ha hecho.

Singleton



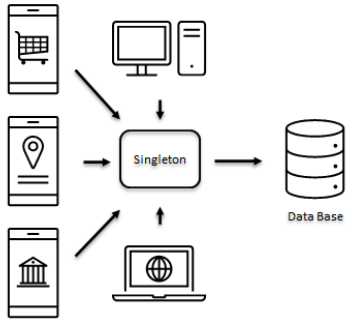
Ejemplo V

Implementar un Singleton es:

1. El constructor de la clase debe ser privado. Evitando que se instancie desde fuera de la clase.
2. Creamos una propiedad estática que se llame ***$instance***, que debe contener la instancia de la clase.
3. Creamos un método estático, ***getInstance,*** que verifica si $instance es nulo, y si así lo es, creará una nueva instancia con el contructor privado. O nos devolverá la instancia actual mediante $instance si esta definida previamente.

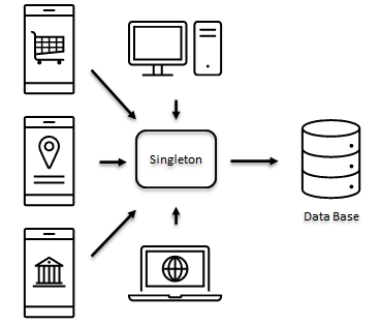
Singleton

Ejemplo VI




```php
<?php

namespace Config;

class Config
{
    private $data;
    private static $instance;

    private function __construct()
    {
        $json = file_get_contents(__DIR__.'/../config/app.json');
        $this->data = json_decode($json, true);
    }

    public static function getInstance()
    {
        if (self::$instance == null) {
            self::$instance = new Config();
        }
        return self::$instance;
    }

    public function get($key)
    {
        return self::$data($key);
    }
}
```

Singleton

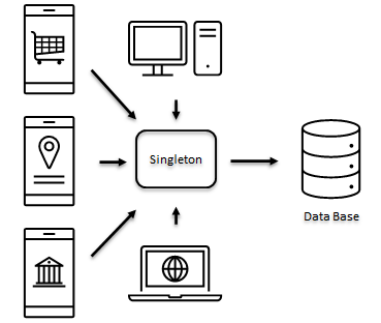


Ejemplo VII

Y deberemos hacer la llamada al singleton de la siguiente forma:

```php
$config = Config::getInstance();
$dbConfig = $config->get('db');
```

Singleton

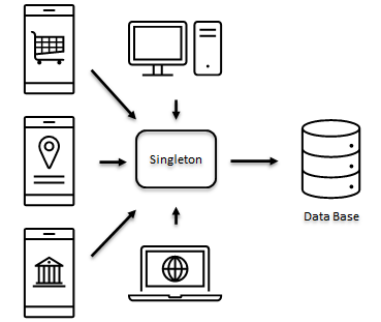


Conclusión

Debemos usar con cuidad este patrón, pues su uso es muy específico para determinados casos, como hemos visto. Y **su mal uso complica el desarrollo de la aplicación**.

No lo implementes en todas partes, si no es necesario.

MVC



Bibliografía

- Learning PHP 7 – Antonio Lopez