

avito.tech ⚡

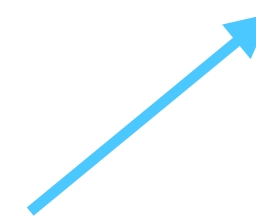
# ПРОТОТИП, ОДИНОЧКА

Афанасьев Юра

Задаёт виды создаваемых объектов  
с помощью экземпляра-прототипа  
и создаёт новые объекты путем  
копирования этого прототипа

```
$object = new SomeClass();
```


```
$anotherObject = $object;
```



Ссылка  
на одну и ту же  
область памяти

```
$object = new SomeClass();
```

```
$anotherObject = clone $object;
```



Ссылка на новый объект,  
находящийся в другой  
области памяти

Реализация  
Прототипа

# Решаемая задача

- Нужно избежать повторных усилий по созданию объекта стандартным путем
- Для создания дубликата объекта с теми же или отличающимися свойствами, ссылками на другие объекты и состоянием

```
class Prototype
{
    public int $val;
}
```

Создаём объект  
и задаём значение

```
$prototype = new Prototype();  
$prototype->val = 10;
```

Обычное  
присваивание

```
$notPrototype = $prototype;  
$clonedPrototype = clone $prototype;  
echo $notPrototype->val;    // 10  
echo $clonedPrototype->val; // 10
```

Присваивание  
через клонирование

Изменяем значение  
исходного объекта

```
$prototype->val = 99;  
echo $notPrototype->val;    // 99  
echo $clonedPrototype->val; // 10
```



Создаём объект  
и задаём значение

```
class Prototype
{
    public \stdClass $clonedObject;
    public \stdClass $simpleObject;

    public function __clone()
    {
        $this->clonedObject = clone $this->clonedObject;
    }
}
```

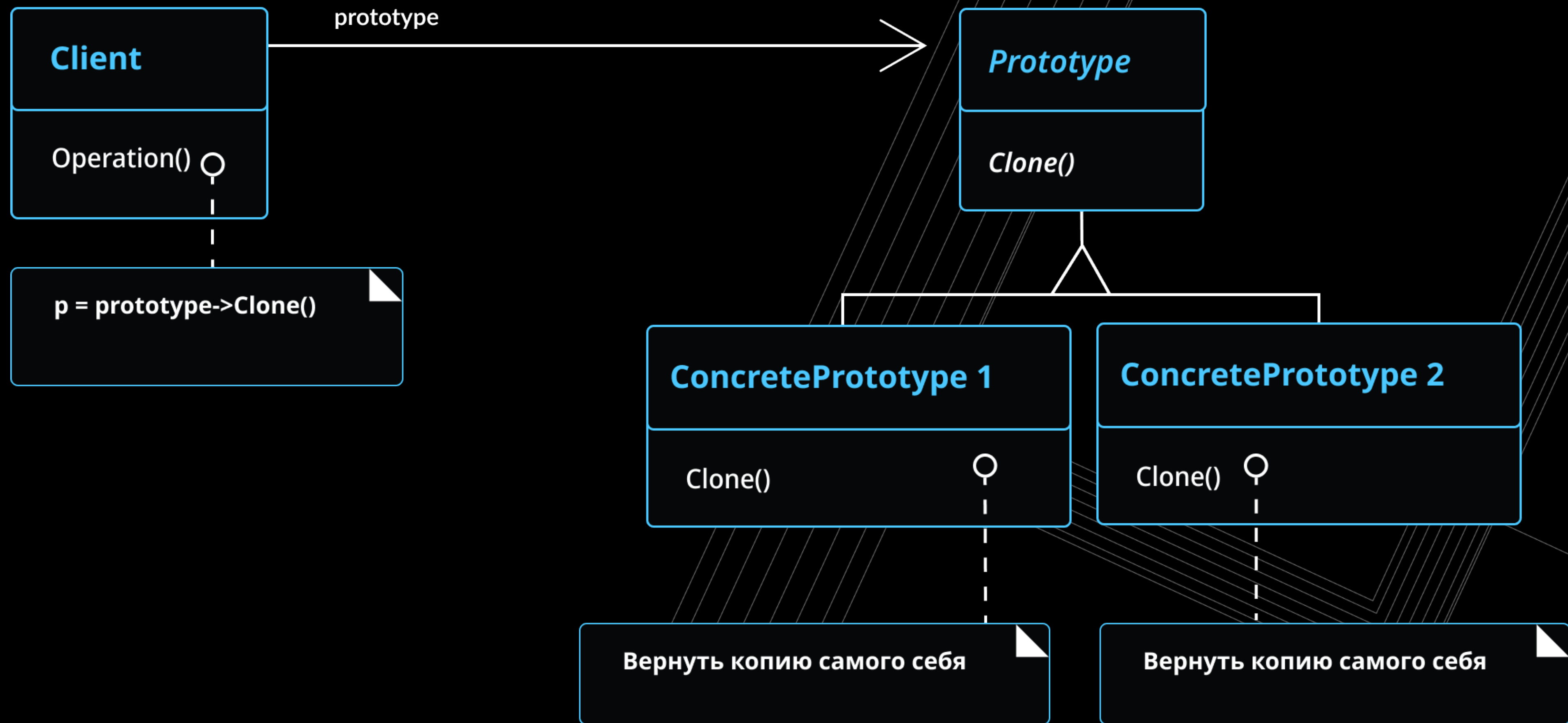
```
$prototype = new Prototype();
$prototype->clonedObject = new \stdClass();
$prototype->clonedObject->val = 1;
$prototype->simpleObject = new \stdClass();
$prototype->simpleObject->val = 2;
```

Клонируем  
объект

```
$newPrototype = clone $prototype;
```

Меняем значение  
исходного объекта

```
$prototype->clonedObject->val = 51;
$prototype->simpleObject->val = 52;
echo $newPrototype->clonedObject->val; // 1
echo $newPrototype->simpleObject->val; // 52
```





# Плюсы

- Альтернатива созданию объектов
- Требуется меньше подготовительной работы перед созданием объекта
- Ускоряет создание объектов

# Минусы

- При наличии связей с другими объектами, требуется их переназначить

**Клонирование  
объекта с теми  
же или похожими  
данными  
и состоянием**

# SINGLETON



# ОДИНОЧКА

Гарантирует, что у класса  
есть только один экземпляр,  
и предоставляет к нему  
глобальную точку доступа



# Решаемая задача

- Ровно один экземпляр некоторого класса
- Расширение экземпляра происходит через порождение подклассов

Статический  
метод



Статическое  
свойство



## Singleton

static Instance()  
SingletonOperation()  
GetSingletonData()



return uniqueInstance

Важно!

Статическое  
свойство

Статический  
метод

`Singleton::getInstance();`

← Создание класса

```
final class Singleton
{
    private static $instance;

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

        return static::$instance;
    }

    private function __construct()
    { /* здесь возможен код */ }

    private function __clone()
    { /* тело метода пустое */ }

    private function __wakeup()
    { /* тело метода пустое */ }
}
```


↑ Первый раз создаётся класс,  
во второй - возвращает  
ссылку

← Запрещаем вызов  
магических методов  
(private)

# Плюсы

- Гарантирует единственный экземпляр класса
- Альтернатива глобальной переменной

Глобальная  
переменная



```
global $properties;  
$properties['orm'] = new SqlBuilder();  
$users = $properties['orm']->find('select * from user');
```

Синглтон

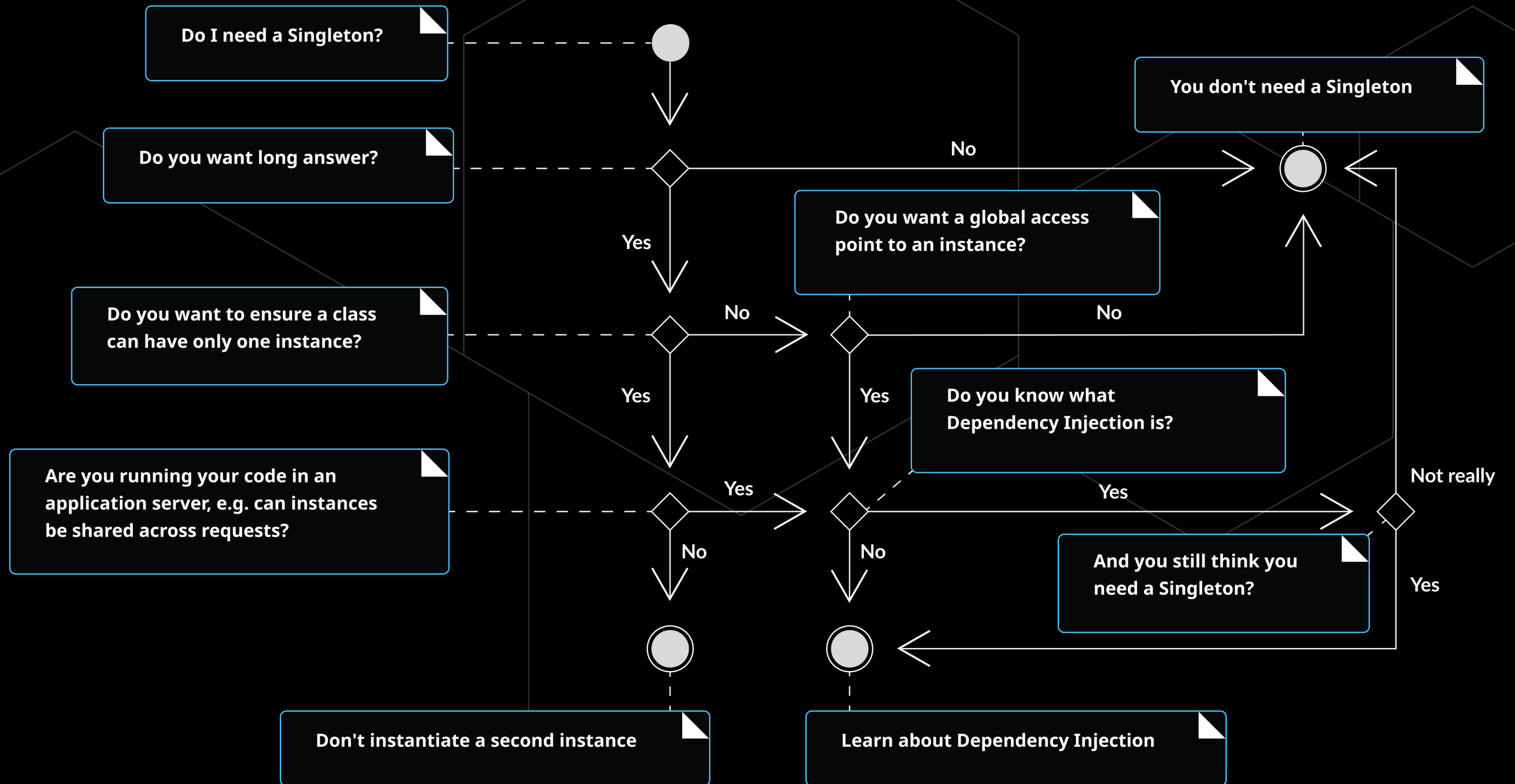


```
Properties::getInstance()->setOrm(new SqlBuilder());  
$users = Properties::getInstance()->getOrm()->find('select * from user');
```



# Минусы

- После инстанцирования класса невозможно создать его с другими параметрами
- Несовместим с юнит-тестами
- Нарушается принцип одной ответственности из SOLID
- Использование паттерна ведёт к антипаттерну «Сплошное одиночество» (Singletonitis)



Гарантирует  
создание одного  
экземпляра  
объекта

avito.tech 

