

avito.tech ⚡

BRIDGE МОСТ

Афанасьев Юра

Отделяет абстракцию
от её реализации так,
чтобы и то и другое можно
было изменять независимо

Решаемая задача

- Изменения в интерфейсах не затрагивают классы, и наоборот
- Реализация класса может удовлетворять нескольким интерфейсам одновременно
- Заранее известно о реализации классом другого интерфейса, который будет определён позже
- При изменении реализации клиенты остаются неизменными

PSR-3

<https://www.php-fig.org/psr/psr-3/>



```
namespace Psr\Log;

interface LoggerInterface
{
    public function emergency($message, array $context = array());

    public function alert($message, array $context = array());

    public function critical($message, array $context = array());

    public function error($message, array $context = array());

    public function warning($message, array $context = array());

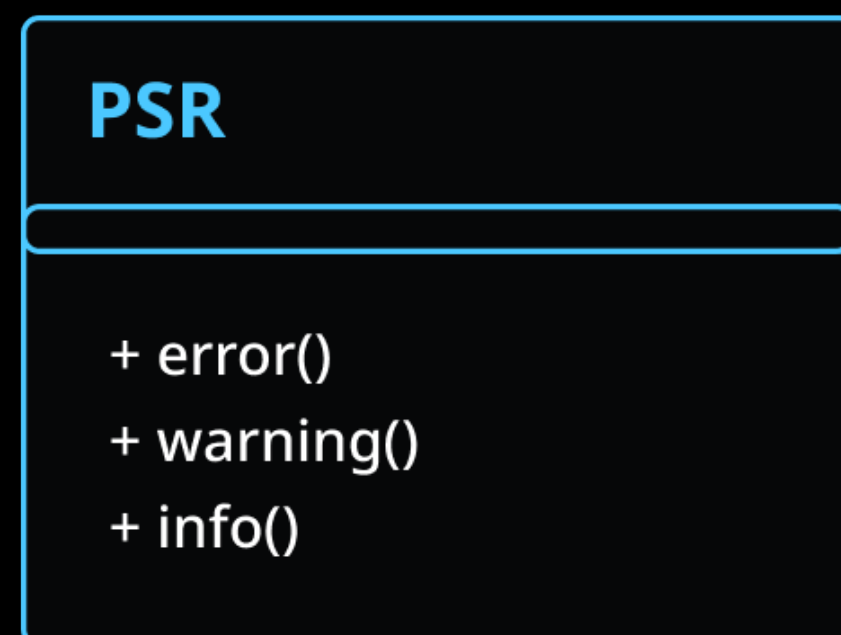
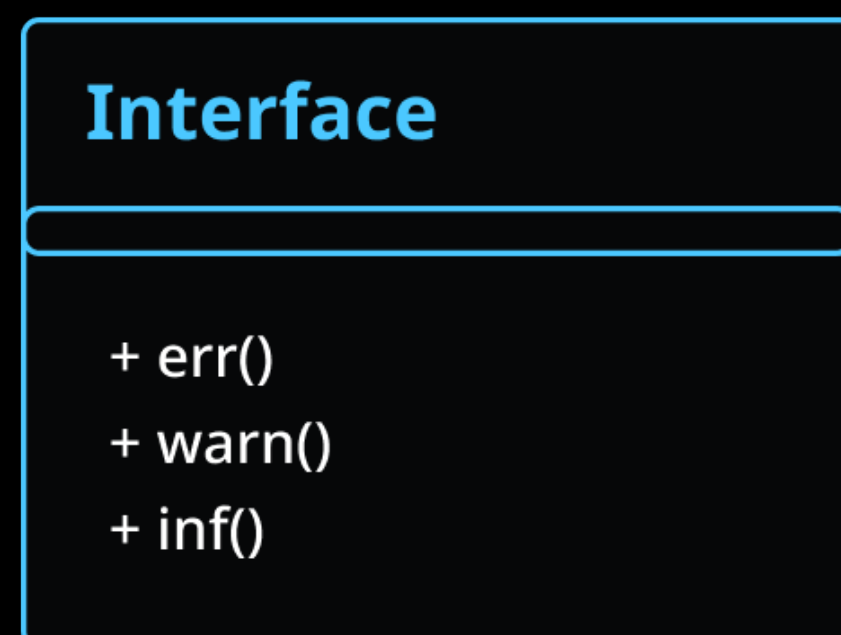
    public function notice($message, array $context = array());

    public function info($message, array $context = array());

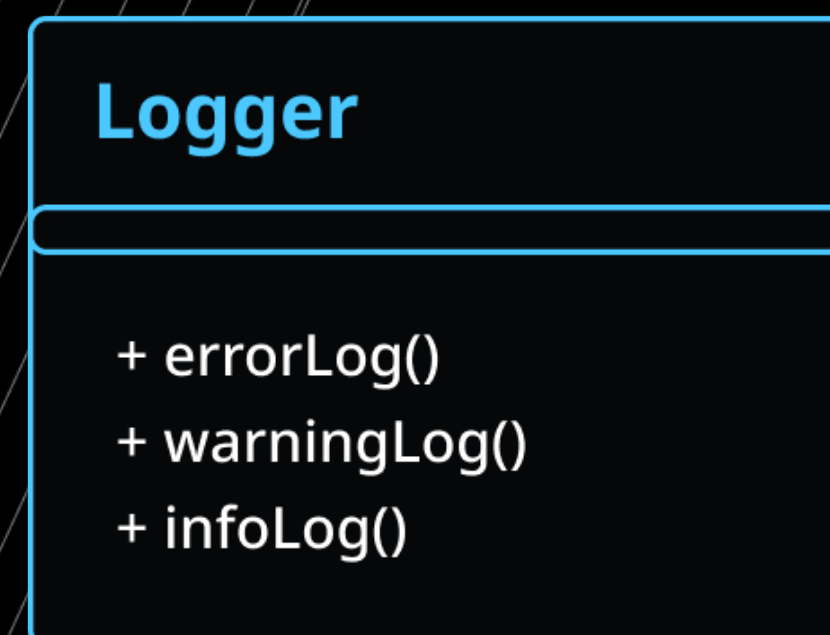
    public function debug($message, array $context = array());

    public function log($level, $message, array $context = array());
}
```

Интерфейсы



Реализации



Применимость

- Нет конечной реализации библиотеки или понимания финальной абстракции, но код уже необходимо написать
- Реализация библиотеки или модуля не изменяется, после изменения интерфейса
- Библиотечный класс должен реализовывать методы двух взаимно противоречивых интерфейсов



Абстракция
(перенаправляет запросы
в конкретные реализации)

Ожидаем класс
с конкретной реализацией

```
abstract class Abstraction
{
    public function __construct(protected Implementor $implementor)
    {}

    public function operation(): string
    {
        return 'Abstraction result: ' . $this->implementor->operationImp();
    }
}
```

Вызываем конкретную
реализацию и дополняем
её доп. действиями

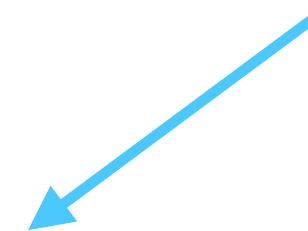
Расширяет
исходный интерфейс

Уточнённая абстракция

```
class RefinedAbsraction extends Abstraction
{
    public function operation(): string
    {
        return 'RefinedAbsraction result: ' . $this->implementor->operationImp();
    }
}
```

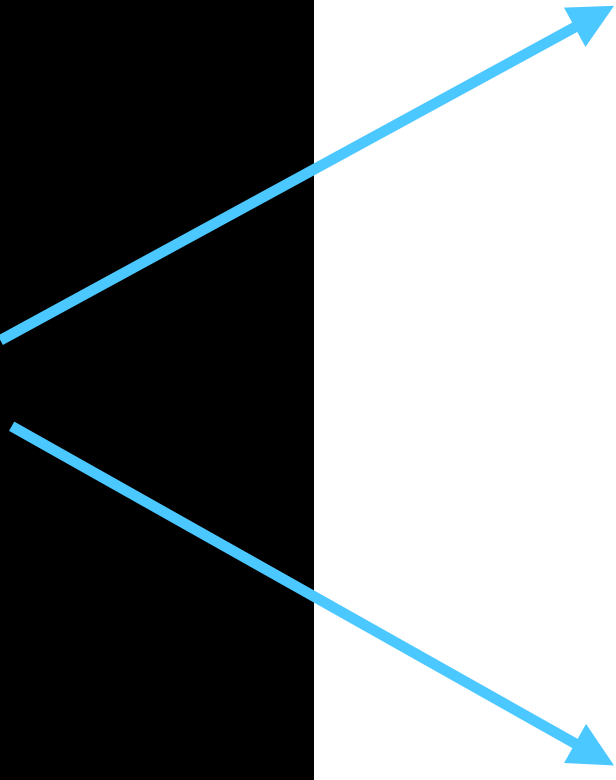
Расширяет поведение
родительского класса

Реализатор
(общий интерфейс
реализаций)



```
interface Implementor
{
    public function operationImp();
}
```

Конкретные
реализаторы



```
class ConcreteImplementorA implements Implementor
{
    public function operationImp(): string
    {
        return 'ConcreteImplementorA' . PHP_EOL;
    }
}
```

```
class ConcreteImplementorB implements Implementor
{
    public function operationImp(): string
    {
        return 'ConcreteImplementorB' . PHP_EOL;
    }
}
```

Пробрасываем
абстракцию

Пробрасываем
конкретную реализацию

```
testBridge(new RefinedAbstraction(new ConcreteImplementorA()));  
// testBridge(new RefinedAbsraction(new ConcreteImplementorB()));
```

```
function testBridge(Abstraction $abstraction)  
{  
    echo $abstraction->operation();  
}
```

Можно заменить
любым другим
расширением
абстракции

Вызываем любую
доступную операцию

Позволяет классам
и абстракциям
развиваться
независимо

Adapter

- Устраняет несовместимость между двумя существующими интерфейсами
- Работает с существующими классами в проекте

Bridge

- Позволяет легко модифицировать классы, не изменяя абстракции
- Классы и абстракции развиваются независимо

PROXY

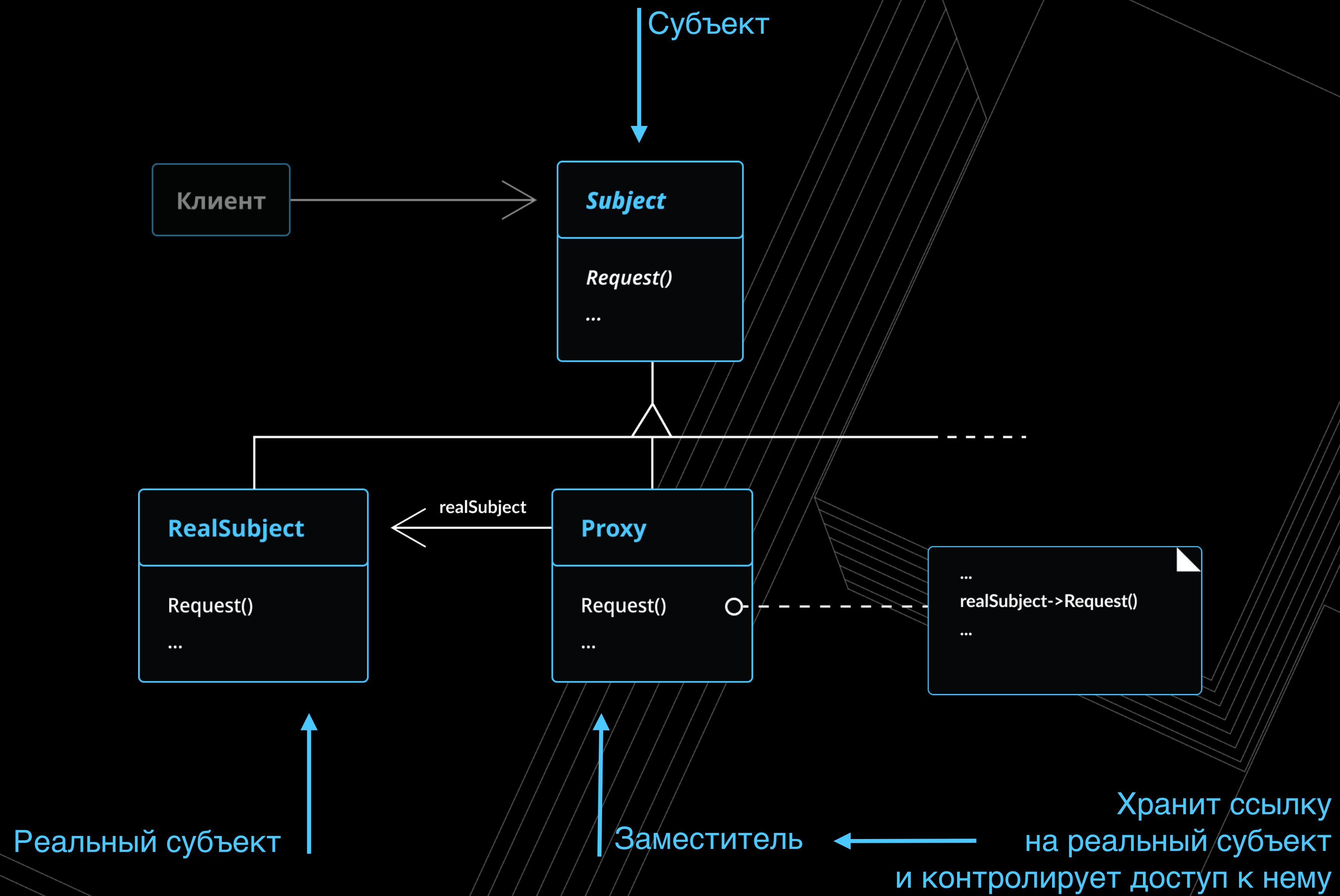


ЗАМЕСТИТЕЛЬ

Является суррогатом
другого объекта
и контролирует
доступ к нему

Решаемая задача

- Локальный представитель объекта, находящийся в другом адресном пространстве (на другом сервере)
- Контролирует доступ к объекту
- Выполнение дополнительных действий при доступе к объекту
- Создание тяжёлых объектов по требованию



Субъект

```
interface Subject
{
    public function request(): string;
}
```

Определяет общие
методы для иерархии

Заместитель

Субъект
(общий интерфейс)

```
class Proxy implements Subject
{
    public function __construct(private Subject $subject)
    {}

    public function request(): string
    {
        return 'Proxy result: ' . $this->subject->request();
    }
}
```

Выполняем задачу заместителя
и вызываем реальный объект

Реальный
субъект

Субъект
(общий интерфейс)

```
class RealSubject implements Subject
{
    public function request(): string
    {
        return 'Real result.' . PHP_EOL;
    }
}
```

Выполняем
необходимые действия

Реальный
субъект

```
function testProxy()  
{  
    $realSubject = new RealSubject();  
    $proxy = new Proxy($realSubject);  
    echo $proxy->request();  
}
```

Заместитель

Выполняем
действие

Суррогат
другого объекта,
контролирующий
доступ к нему

Adapter

- Представляет отличный от существующего интерфейс

Proxy

- Повторяет интерфейс своего субъекта
- Используется для ограничения доступа к реальному объекту

Decorator

- Добавляет новые обязанности объекту

Proxy

- Контролирует доступ к объекту

FLYWEIGHT



ПРИСПОСОБЛЕНЕЦ

Использует разделение
для эффективной поддержки
множества мелких объектов

Когда использовать?

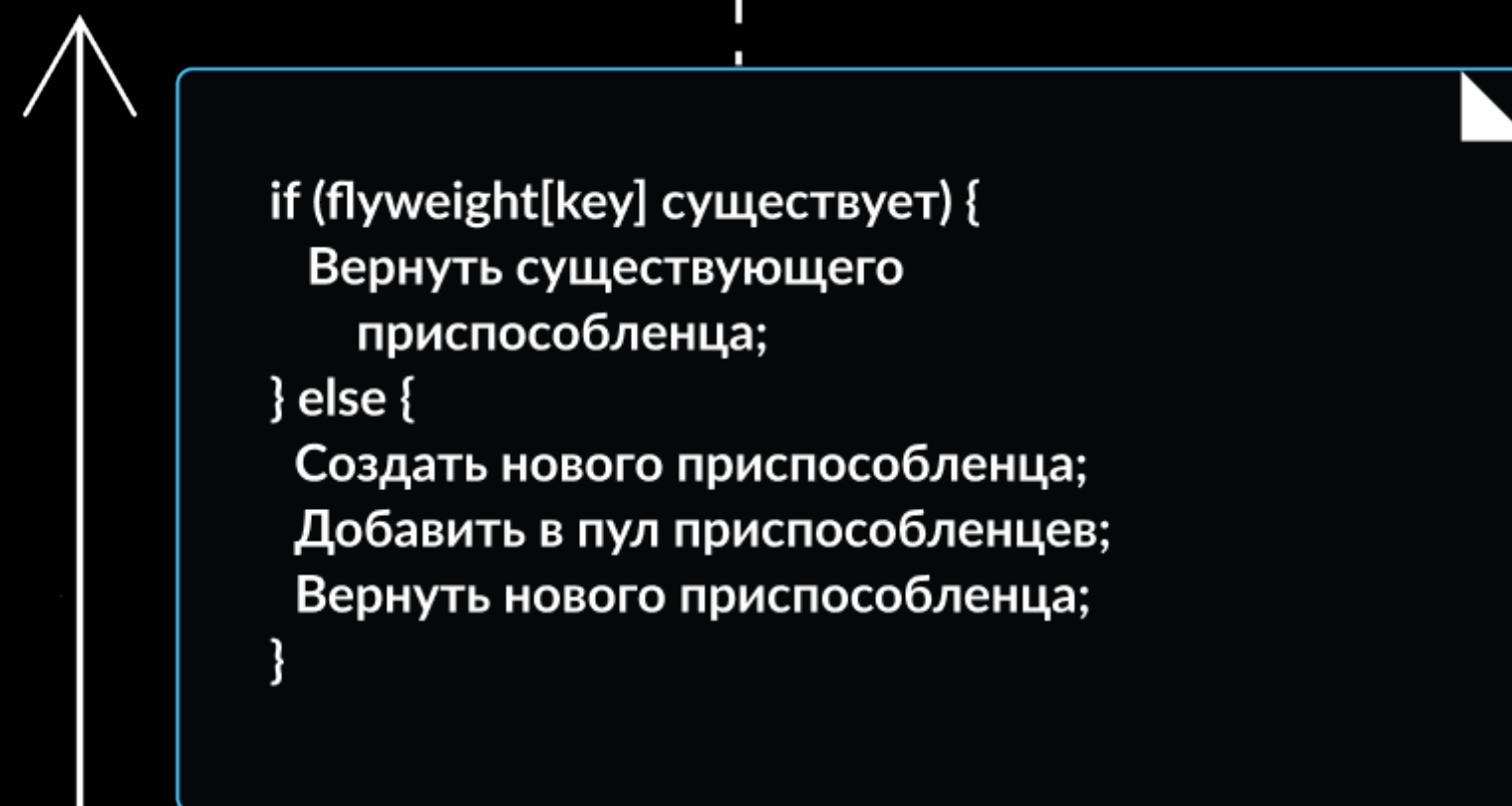
Когда выполняются
все условия

- Создаётся много однотипных объектов
- Расходы на потребление памяти объектами высоки
- Большую часть состояния объектов можно вынести вовне
- Большие группы объектов можно заменить относительно небольшим количеством разделяемых объектов, поскольку внешнее состояние вынесено

Решаемая задача

- Экономит ресурсы (память) для создания множества однотипных небольших объектов
- Хранит в себе внутреннее состояние объектов

Фабрика
приспособленцев
(создаёт
и управляет ими)



Хранит ссылку
на одного или несколько
приспособленцев

flyweights



Приспособленец
(интерфейс)



Конкретный
приспособленец
(хранит внутреннее
состояние)



Не разделяемый
конкретный
приспособленец

Разделяет очень
мелкие объекты
без недопустимо
высоких издержек

avito.tech 

