Professor Hans Noodles

41 уровень

01.11.2019 15513 5

# Паттерн проектирования Proxy

Статья из группы Random

1710576 участников

Вы в группе

В программировании важно правильно спланировать архитектуру приложения. Незаменимое средство для этого — шаблоны проектирования. Сегодня поговорим о Proxy, или по-другому — Заместителе.

## Для чего нужен Заместитель

Этот паттерн помогает решить проблемы, связанные с контролируемым доступом к объекту. У тебя может возникнуть вопрос: “Для чего нужен такой контролируемый доступ?”. Давай рассмотрим пару ситуаций, которые помогут тебе разобраться, что к чему.

## Пример 1

Представим, что у нас есть большой проект с кучей старого кода, где есть класс, отвечающий за выгрузку отчетов из базы данных. Класс работает синхронно, то есть вся система простаивает, пока база обрабатывает запрос. В среднем отчет генерируется за 30 минут. Из-за этой особенности его выгрузка запускается в 00:30, и руководство получает этот отчет утром.

При анализе выяснилось, что необходимо получать отчет сразу после его генерации, то есть в течение дня. Перенести время запуска нельзя, так как система будет ждать ответ от базы. Выход — изменить принцип работы, запустив выгрузку и генерацию отчета в отдельном потоке.

Такое решение позволит системе работать в обычном режиме, а руководство будет получать свежие отчеты. Однако есть

В этом случае можно ввести промежуточный прокси-класс с помощью паттерна Заместитель, который будет получать запрос на выгрузку отчета, логировать время начала и запускать отдельный поток. Когда отчет сгенерируется, поток завершит свою работу и все будут счастливы.

### Пример 2

Команда разработчиков создает сайт-афишу. Чтобы получить данные о новых мероприятиях, они обращаются к стороннему сервису, взаимодействие с которым реализовано через специальную закрытую библиотеку. При разработке появилась проблема: сторонняя система обновляет данные раз в сутки, а запрос к ней происходит каждый раз, когда пользователь обновляет страницу. Это создает большое количество запросов, и сервис перестает отвечать.

Решение — кэшировать ответ сервиса и предоставлять посетителям сохраненный результат при каждой перезагрузке, обновляя этот кэш по необходимости. В этом случае использование паттерна Заместитель — отличное решение без изменения готового функционала.

## Принцип работы паттерна

Чтобы внедрить этот паттерн, нужно создать класс-прокси. Он реализует интерфейс сервисного класса, имитируя его поведение для клиентского кода. Таким образом вместо реального объекта клиент взаимодействует с его заместителем. Как правило, все запросы передаются далее сервисному классу, но с дополнительными действиями до или после его вызова.

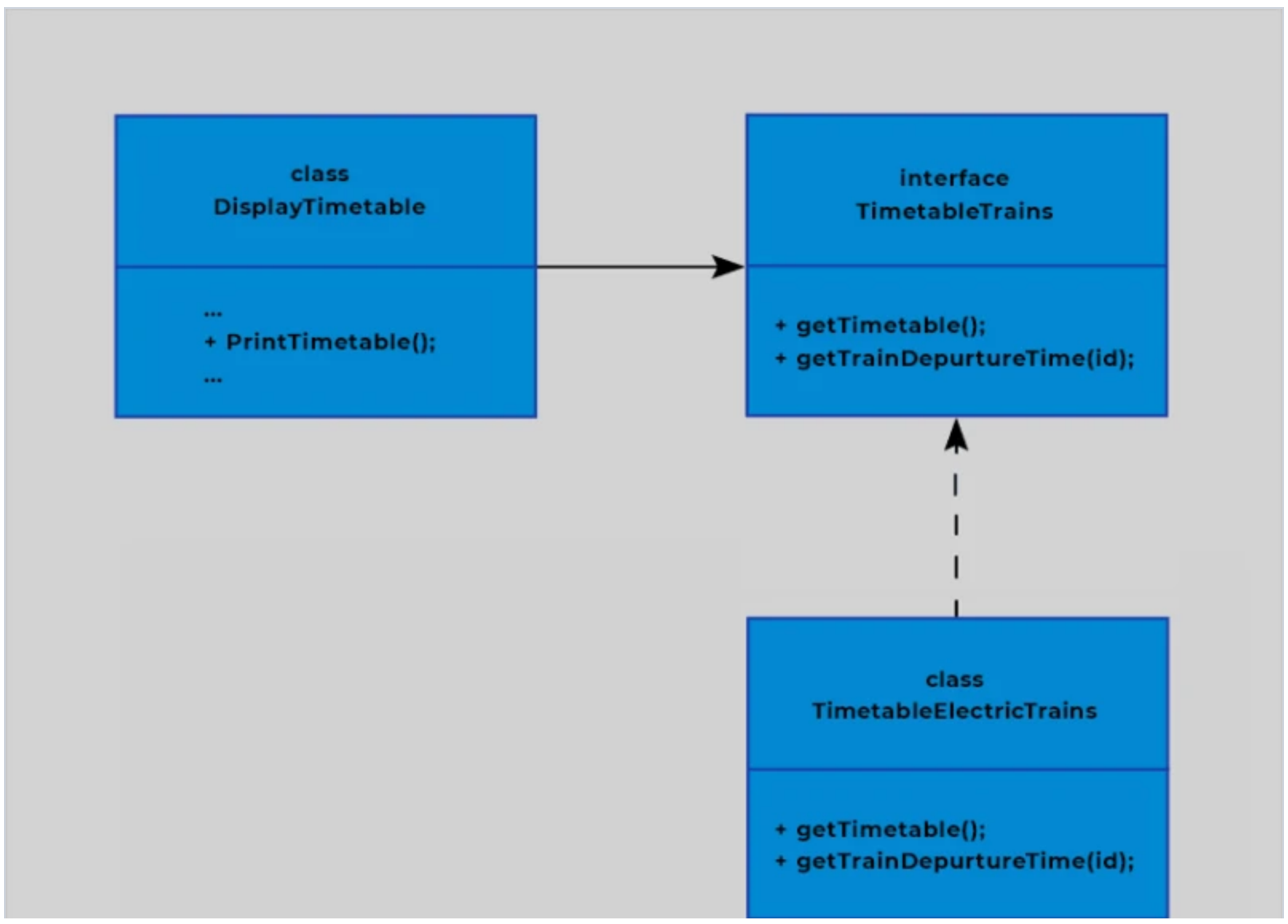
Проще говоря, этот прокси-объект — прослойка между клиентским кодом и целевым объектом.

Рассмотрим пример с кэшированием запроса из очень медленного старого диска. Пусть это будет расписание электропоездов в каком-нибудь древнем приложении, чей принцип действия нельзя изменять. Диск с обновленным расписанием вставляют каждый день в фиксированное время.

Итак, у нас есть:

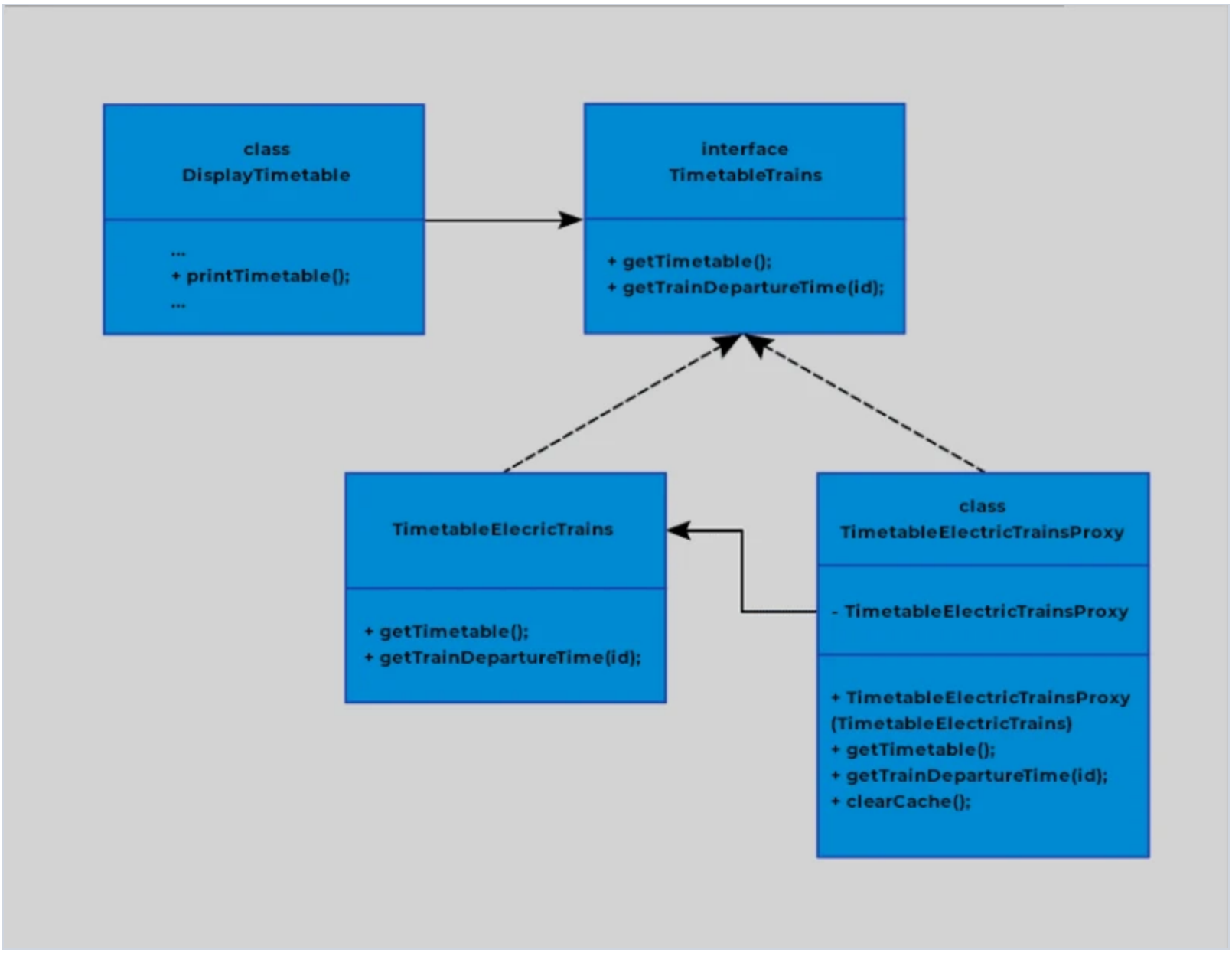
1. Интерфейс `TimetableTrains`.
2. Класс `TimetableElectricTrains`, который реализует этот интерфейс.
3. Именно через этот класс клиентский код взаимодействует с файловой системой диска.
4. Класс-клиент `DisplayTimetable`. Его метод `printTimetable()` использует методы класса `TimetableElectricTrains`.

Схема простая:



В данный момент при каждом вызове метода `printTimetable()` класс `TimetableElectricTrains` обращается на диск, выгружает данные и предоставляет их клиенту. Эта система функционирует хорошо, но очень медленно. Поэтому было решено увеличить производительность системы, добавив механизм кэширования.

Это можно сделать с использованием паттерна Proxy:




Таким образом класс `DisplayTimetable` даже не заметит, что взаимодействует с классом `TimetableElectricTrainsProxy`, а не с предыдущим.

Новая реализация загружает расписание один раз в день, а при повторных запросах возвращает уже загруженный объект из памяти.

## Для каких задач лучше использовать Proxy

Вот несколько ситуаций, в которых тебе точно пригодится этот паттерн:

1. Кэширование.
2. Отложенная реализация, также известная как ленивая. Зачем загружать объект сразу, если можно загрузить его по мере необходимости?
3. Логирование запросов.
4. Промежуточные проверки данных и доступа.
5. Запуск параллельных потоков обработки.
6. Запись или подсчет истории обращения.

Есть и другие сценарии использования. Понимая принцип работы этого паттерна, ты сам сможешь найти для него удачное применение.

На первый взгляд, **Заместитель** делает то же, что и **Фасад**, но это не так. У **Заместителя** есть тот же интерфейс, что и у сервисного объекта.

Также не нужно путать паттерн с **Декоратором** или **Адаптером**. **Декоратор** предоставляет расширенный интерфейс, а **Адаптер** — альтернативный.

+ Можно как угодно контролировать доступ к сервисному объекту;
+ Дополнительные возможности управления жизненным циклом сервисного объекта;
+ Работает без сервисного объекта;
+ Повышает быстродействие и безопасность кода.
- Есть риск ухудшения производительности из-за дополнительных обработок;
- Усложняет структуру классов программы.

## Паттерн Заместитель на практике

Давай реализуем с тобой систему, которая читает расписание поездов с диска:

```
public interface TimetableTrains {
   String[] getTimetable();
   String getTrainDepartureTime();
}
```

Класс, реализующий основной интерфейс:

```
public class TimetableElectricTrains implements TimetableTrains {

   @Override
   public String[] getTimetable() {
       ArrayList<String> list = new ArrayList<>();
       try {
           Scanner scanner = new Scanner(new FileReader(new File("/tmp/electric_trains.csv")));
           while (scanner.hasNextLine()) {
               String line = scanner.nextLine();
               list.add(line);
           }
       } catch (IOException e) {
           System.err.println("Error:  " + e);
       }
       return list.toArray(new String[list.size()]);
   }

   @Override
   public String getTrainDepartureTime(String trainId) {
       String[] timetable = getTimetable();
       for(int i = 0; i<timetable.length; i++) {
           if(timetable[i].startsWith(trainId+";")) return timetable[i];
       }
       return "";
   }
}
```

Каждый раз при попытке получить расписание всех поездов программа читает файл с диска. Но это еще цветочки. Файл также считывается каждый раз, когда нужно получить расписание только по одному поезду! Хорошо, что такой код существует только в плохих примерах :)

Клиентский класс:

```
public class DisplayTimetable {
   private TimetableTrains timetableTrains = new TimetableElectricTrains();
```

```
        String[] timetable = timetableTrains.getTimetable();
        String[] tmpArr;
        System.out.println("Поезд\tОткуда\tКуда\t\tВремя отправления\tВремя прибытия\tВремя в пути");
        for(int i = 0; i < timetable.length; i++) {
            tmpArr = timetable[i].split(";");
            System.out.printf("%s\t%s\t%s\t\t%s\t\t\t\t%s\t\t\t%s\n", tmpArr[0], tmpArr[1], tmpArr[2],
        }
    }
}
```

Пример файла:

```
9B-6854;Лондон;Прага;13:43;21:15;07:32
BA-1404;Париж;Грац;14:25;21:25;07:00
9B-8710;Прага;Вена;04:48;08:49;04:01;
9B-8122;Прага;Грац;04:48;08:49;04:01
```

Протестируем:

```
public static void main(String[] args) {
   DisplayTimetable displayTimetable = new DisplayTimetable();
   displayTimetable.printTimetable();
}
```

**Вывод:**

```
Поезд  Откуда  Куда   Время отправления Время прибытия    Время в пути
9B-6854  Лондон  Прага    13:43         21:15         07:32
BA-1404  Париж   Грац   14:25         21:25         07:00
9B-8710  Прага   Вена   04:48         08:49         04:01
9B-8122  Прага   Грац   04:48         08:49         04:01
```

**Теперь пойдем по шагам внедрения нашего паттерна:**

1. Определить интерфейс, который позволяет использовать вместо оригинального объекта новый заместитель. В нашем примере это `TimetableTrains`.

2. Создать класс заместителя. В нем должна быть ссылка на сервисный объект (создать в классе или передать в конструкторе);

   Вот наш класс-заместитель:

```
public class TimetableElectricTrainsProxy implements TimetableTrains {
   // Ссылка на оригинальный объект
   private TimetableTrains timetableTrains = new TimetableElectricTrains();

   private String[] timetableCache = null

   @Override
   public String[] getTimetable() {
```

```
11
12      @Override
13      public String getTrainDepartureTime(String trainId) {
14          return timetableTrains.getTrainDepartureTime(trainId);
15      }
16
17      public void clearCache() {
18          timetableTrains = null;
19      }
20  }
```

На этом этапе просто создаем класс со ссылкой на оригинальный объект и передаем все вызовы ему.

3. Реализовываем логику класса-заместителя. В основном вызов всегда перенаправляется оригинальному объекту.

```
1   public class TimetableElectricTrainsProxy implements TimetableTrains {
2       // Ссылка на оригинальный объект
3       private TimetableTrains timetableTrains = new TimetableElectricTrains();
4
5       private String[] timetableCache = null
6
7       @Override
8       public String[] getTimetable() {
9           if(timetableCache == null) {
10              timetableCache = timetableTrains.getTimetable();
11          }
12          return timetableCache;
13      }
14
15      @Override
16      public String getTrainDepartureTime(String trainId) {
17          if(timetableCache == null) {
18              timetableCache = timetableTrains.getTimetable();
19          }
20          for(int i = 0; i < timetableCache.length; i++) {
21              if(timetableCache[i].startsWith(trainId+";")) return timetableCache[i];
22          }
23          return "";
24      }
25
26      public void clearCache() {
27          timetableTrains = null;
28      }
29  }
```

Метод `getTimetable()` проверяет, закэширован ли массив расписания в память. Если нет, он посылает запрос для загрузки данных с диска, сохраняя результат. Если же запрос уже выполняется, он быстро вернет объект из памяти.

Благодаря простому функционалу, метод getTrainDepartireTime() не пришлось перенаправлять в оригинальный объект. Мы просто дублировали его функционал в новый метод.

Так делать нельзя. Если пришлось дублировать код или производить подобные манипуляции, значит что-то пошло не так, и нужно посмотреть на проблему под другим углом. В нашем простом примере иного пути нет, но в реальных проектах, скорее всего, код будет написан более корректно.

```java
public class DisplayTimetable {
    // Измененная ссылка
    private TimetableTrains timetableTrains = new TimetableElectricTrainsProxy();

    public void printTimetable() {
        String[] timetable = timetableTrains.getTimetable();
        String[] tmpArr;
        System.out.println("Поезд\tОткуда\tКуда\t\tВремя отправления\tВремя прибытия\tВремя в пути
        for(int i = 0; i<timetable.length; i++) {
            tmpArr = timetable[i].split(";");
            System.out.printf("%s\t%s\t%s\t\t%s\t\t\t\t%s\t\t\t%s\n", tmpArr[0], tmpArr[1], tmpArr
        }
    }
}
```

Проверка

```
Поезд  Откуда  Куда   Время отправления Время прибытия    Время в пути
9B-6854  Лондон  Прага    13:43          21:15          07:32
BA-1404  Париж   Грац   14:25          21:25          07:00
9B-8710  Прага   Вена   04:48          08:49          04:01
9B-8122  Прага   Грац   04:48          08:49          04:01
```

Отлично, работает корректно.

Можно также рассмотреть вариант с фабрикой, которая будет создавать как оригинальный объект, так и объект-заместитель в зависимости от определенных условий.



## Пара полезных ссылок вместо точки

1. Отличная статья о паттернах и немного о “Заместителе”

2. Реализация паттерна Заместитель на Java (статья с хабра)

На сегодня все! Неплохо бы вернуться к обучению и проверить новые знания на практике :)

+34

**Комментарии (5)** популярные новые старые

**JavaCoder**

Введите текст комментария

**Valua Sinicyn** Уровень 41, Харьков, Украина 24 февраля 2021, 13:20

Proxy на человеческом.

Ответить +3

**LuneFox** инженер по сопровождению в **BIFIT** EXPERT 5 марта, 22:02

Вот сразу бы так.

Ответить 0

**Валентин Кудинов** Уровень 41, Самара, Россия 3 апреля 2020, 10:19

В этой схеме тоже опечатка.

Ответить 0

**Валентин Кудинов** Уровень 41, Самара, Россия 3 апреля 2020, 10:07

Схема должна быть такая.

Ответить 0

**Bargu3in** Уровень 28, Санкт-Петербург, Россия 8 ноября 2019, 05:06

Спасибо, Кэрри.
Для меня, как для новичка все очень понравилось и все очень доступно.
Хотя меня смутили схемы. Возможно я ошибаюсь, но мне кажется что должно быть так:

Возможно я ошибаюсь, поправьте меня, если так.

Спасибо за статью!

Ответить +7

ПОДПИСЫВАЙТЕСЬ

ЯЗЫК ИНТЕРФЕЙСА

Русский