

**Федеральное государственное автономное образовательное учреждение
высшего образования**

«НАЦИОНАЛЬНЫЙ ИССЛЕДОВАТЕЛЬСКИЙ УНИВЕРСИТЕТ

«ВЫСШАЯ ШКОЛА ЭКОНОМИКИ»

Московский институт электроники и математики им. Тихонова

Департамент электронной инженерии

ОТЧЕТ

О ПРАКТИЧЕСКОЙ РАБОТЕ №10

по дисциплине «Цифровая схемотехника и архитектура компьютера»

«Паттерн "Одиночка"»

Студент гр. БИБ201

Шадрунов Алексей

Дата выполнения: 24 марта 2023 г.

Преподаватель:

к. т. н., доцент кафедры

информационной безопасности

киберфизических систем

Мещеряков Я. Е.

«___» _____ 2023 г.

Москва, 2023

Содержание

1	Цель работы	3
2	Теоретические сведения	4
3	Ход работы	5
3.1	Простой синглтон	5
3.2	Синглтон Мейера	6
3.3	Синглтон в языке Python	7
4	Выводы о проделанной работе	9
	Список использованных источников	10
	Приложение А. Код simple/kitchen.h	10
	Приложение Б. Код simple/restaurant.h	12
	Приложение В. Код simple/main.cpp	13
	Приложение Г. Код meyers/kitchen.h	14
	Приложение Д. Код meyers/restaurant.h	16
	Приложение Е. Код meyers/main.cpp	17
	Приложение Ж. Код ru/main.py	18

1 Цель работы

Изучить паттерн объектно-ориентированного проектирования «Одиночка» (Singleton).

2 Теоретические сведения

Для некоторых сущностей важно, чтобы существовал только один экземпляр. Данную функцию выполняет паттерн «Одиночка».

Архитектура паттерна Singleton основана на идее использования глобальной переменной со следующими свойствами:

1. Время жизни переменной — от запуска программы до ее завершения.
2. Предоставляет глобальный доступ, то есть такая переменная может быть доступна из любой части программы.

Однако использовать глобальную переменную некоторого типа непосредственно невозможно, так как существует проблема обеспечения единственности экземпляра, а именно, возможно создание нескольких переменных того же самого типа.

Для решения этой проблемы паттерн «Одиночка» возлагает контроль над созданием единственного объекта на сам класс. Доступ к этому объекту осуществляется через статическую функцию-член класса, которая возвращает указатель или ссылку на него. Этот объект будет создан только при первом обращении к методу, а все последующие вызовы просто возвращают его адрес. Для обеспечения уникальности объекта конструкторы и оператор присваивания объявляются закрытыми.

Применимость

- экземпляр некоторого класса существует только в единственном экземпляре, к которому может обратиться любой клиент через известную точку доступа;
- единственный экземпляр должен расширяться путем порождения подклассов, а клиенты должны иметь возможность работать с расширенным экземпляром без модификации своего кода.

Реализация

Паттерн одиночка устроен так, что сам класс гарантирует, что больше одного экземпляра создать не удастся. Чаще всего для этого операция, создающая экземпляры, скрывается за операцией класса (то есть за статической функцией или методом класса), которая гарантирует создание не более одного экземпляра. Обычно такая функция называется Instance или GetInstance. Данная операция имеет доступ к переменной, где хранится уникальный экземпляр, и гарантирует инициализацию переменной этим экземпляром перед возвратом ее клиенту. При таком подходе гарантируется, что «одиночка» будет создан и инициализирован перед первым использованием.

3 Ход работы

3.1 Простой синглтон

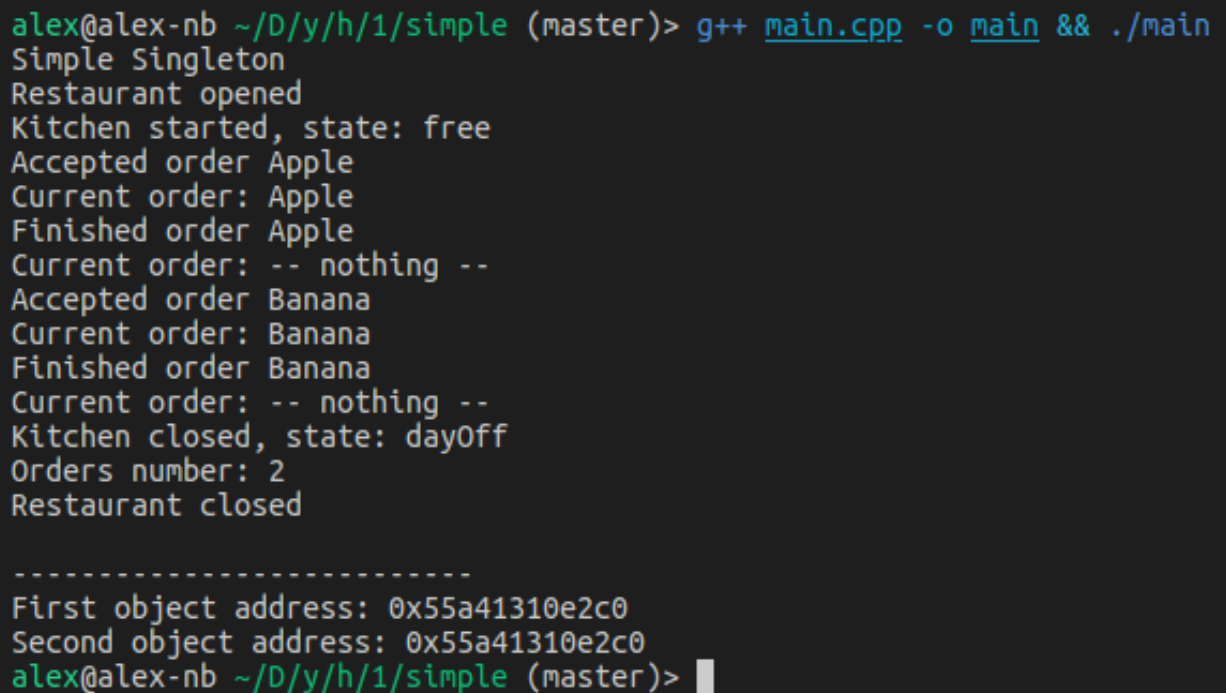
Рассмотрим простую реализацию синглтона. В этой реализации для объекта единожды выделяется область памяти, указатель на которую записывается в статическую переменную-поле класса.

Реализация показана на листинге 1.

```
1 class Restaurant : public Kitchen
2 {
3     public:
4         static Restaurant *getInstance()
5         {
6             if (!p_instance)
7                 p_instance = new Restaurant();
8             return p_instance;
9         }
10     ...
```

Листинг 1 – Простой синглтон

Результат работы программы показан на рисунке 1.



```
alex@alex-nb ~/D/y/h/1/simple (master)> g++ main.cpp -o main && ./main
Simple Singleton
Restaurant opened
Kitchen started, state: free
Accepted order Apple
Current order: Apple
Finished order Apple
Current order: -- nothing --
Accepted order Banana
Current order: Banana
Finished order Banana
Current order: -- nothing --
Kitchen closed, state: dayOff
Orders number: 2
Restaurant closed

-----
First object address: 0x55a41310e2c0
Second object address: 0x55a41310e2c0
alex@alex-nb ~/D/y/h/1/simple (master)> █
```

Рисунок 1 – Простой синглтон

Код приведён в приложении.

UML-диаграмма для классов приведена на рисунке 2.

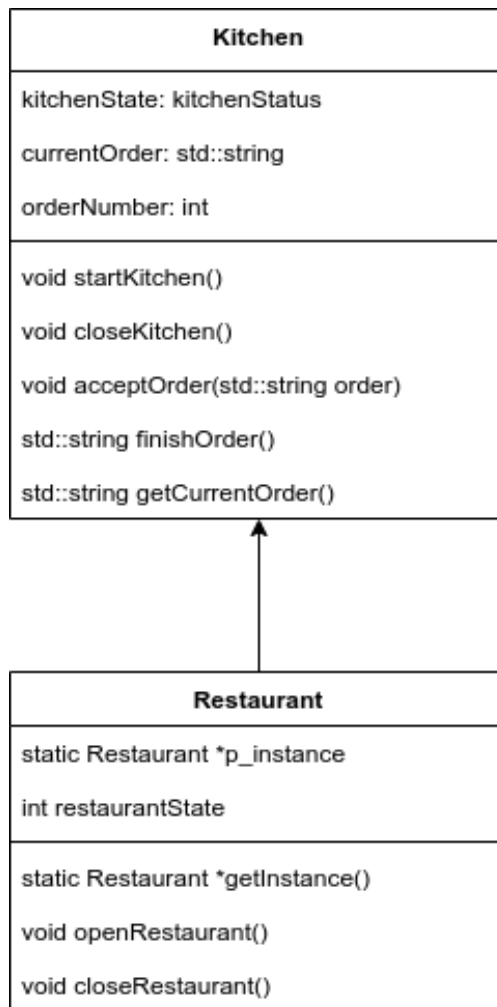


Рисунок 2 – UML-диаграмма

3.2 Синглтон Мейера

Рассмотрим более продвинутую реализацию синглтона — синглтон Мейера.

Реализация показана на листинге 2.

```

1 class Restaurant : public Kitchen
2 {
3 public:
4     static Restaurant *getInstance()
5     {
6         static Restaurant inst;
7         return &inst;
8     }
9     ...
  
```

Листинг 2 – Синглтон Мейера

Результат работы программы показан на рисунке 3.

```
alex@alex-nb ~/D/y/h/1/meyers (master)> g++ main.cpp -o main && ./main
Meyer's Singleton
Restaurant opened
Kitchen started, state: free
Accepted order Apple
Current order: Apple
Finished order Apple
Current order: -- nothing --
Accepted order Banana
Current order: Banana
Finished order Banana
Current order: -- nothing --
Kitchen closed, state: dayOff
Orders number: 2
Restaurant closed

-----
First object address: 0x55a655499260
Second object address: 0x55a655499260
alex@alex-nb ~/D/y/h/1/meyers (master)> █
```

Рисунок 3 – Синглтон Мейера

Код приведён в приложении.

3.3 Синглтон в языке Python

На языке Python синглтон реализовать ещё проще.

Реализация показана на листинге 3.

```
1 class Restaurant(Kitchen):
2     def __new__(cls):
3         if not hasattr(cls, "instance"):
4             cls.instance = super(Restaurant, cls).__new__(cls)
5     ...
```

Листинг 3 – Синглтон в Python

Результат работы программы показан на рисунке 4.

```
alex@alex-nb ~/D/y/h/1/py (master)> python main.py
Python Singleton
Restaurant opened
Accepted order: Apple
Current order: Apple
Finished order: Apple
Current order: -- nothing --
Accepted order: Banana
Current order: Banana
Finished order: Banana
Current order: -- nothing --
Kitchen closed, state: dayOff
Orders number: 2
Restaurant closed
alex@alex-nb ~/D/y/h/1/py (master)> █
```

Рисунок 4 – Синглтон в Python

Код приведён в приложении.

4 Выводы о проделанной работе

В рамках данной работы изучен паттерн объектно-ориентированного проектирования «Одиночка» (Singleton).

Приложение А. Код simple/kitchen.h

```
1 #pragma once
2 #include <string>
3 #include <iostream>
4
5 class Kitchen
6 {
7 public:
8     enum kitchenStatus
9     {
10         busy = 1,
11         free = 2,
12         dayOff = 3
13     };
14
15     void startKitchen()
16     {
17         if ((this->kitchenState != kitchenStatus::busy) &&
18             (this->kitchenState != kitchenStatus::dayOff))
19         {
20             this->kitchenState = kitchenStatus::free;
21             std::cout << "Kitchen started, state: free" << std::endl;
22         }
23     }
24
25     void closeKitchen()
26     {
27         if (this->kitchenState == kitchenStatus::free)
28         {
29             this->kitchenState = kitchenStatus::dayOff;
30             std::cout << "Kitchen closed, state: dayOff" << std::endl;
31             std::cout << "Orders number: " << this->orderNumber <<
32             std::endl;
33             this->orderNumber = 0;
34         }
35     }
36
37     void acceptOrder(std::string order)
38     {
39         if (this->kitchenState != kitchenStatus::free)
40             std::cout << "Can't accept order, kitchen is already cooking"
41             << std::endl;
42         else
43         {
44             std::cout << "Accepted order " << order << std::endl;
45             this->kitchenState = kitchenStatus::busy;
46             this->currentOrder = order;
47         }
48     }
49
50     std::string finishOrder()
51     {
52         if (this->kitchenState != kitchenStatus::busy)
53             std::cout << "Can't finish order, kitchen currently is not
54             cooking" << std::endl;
55         else
56         {
57             return nullptr;
58             std::string _currentOrder = this->currentOrder;
59             this->kitchenState = kitchenStatus::free;
60             this->currentOrder = "";
61             std::cout << "Finished order " << _currentOrder << std::endl;
62             this->orderNumber++;
63             return _currentOrder;
64         }
65     }
66
67     std::string getCurrentOrder() const
68     {
69     }
```

```

64         if (currentOrder != "")
65             return currentOrder;
66         return "-- nothing --";
67     }
68
69 protected:
70     kitchenStatus kitchenState;
71     std::string currentOrder;
72     int orderNumber;
73     Kitchen()
74     {
75         this->kitchenState = kitchenStatus::free;
76         this->currentOrder = "";
77         this->orderNumber = 0;
78     };
79     ~Kitchen() = default;
80 };

```

Приложение Б. Код simple/restaurant.h

```
1 #include <string>
2 #include <utility>
3 #include "kitchen.h"
4
5 class Restaurant : public Kitchen
6 {
7 public:
8     static Restaurant *getInstance()
9     {
10         if (!p_instance)
11             p_instance = new Restaurant();
12         return p_instance;
13     }
14
15     // other methods
16     enum restaurantStatus
17     {
18         open = 1,
19         closed = 2
20     };
21
22     void openRestaurant()
23     {
24         if (this->restaurantState != restaurantStatus::open)
25         {
26             this->restaurantState = restaurantStatus::open;
27             std::cout << "Restaurant opened" << std::endl;
28         }
29     }
30
31     void closeRestaurant()
32     {
33         if (this->restaurantState != restaurantStatus::closed)
34         {
35             this->restaurantState = restaurantStatus::closed;
36             std::cout << "Restaurant closed" << std::endl;
37         }
38     }
39
40 private:
41     static Restaurant *p_instance;
42     int restaurantState;
43     Restaurant(){}; // Private constructor
44     ~Restaurant(){}; // Private destructor
45     Restaurant(const Restaurant &); // Prevent copy constructor
46     Restaurant &operator=(const Restaurant &); // Prevent assignment
47 };
48
49 Restaurant *Restaurant::p_instance = nullptr;
```

Приложение В. Код simple/main.cpp

```
1 #include <iostream>
2 #include "kitchen.h"
3 #include "restaurant.h"
4
5 int main()
6 {
7     std::cout << "Simple Singleton" << std::endl;
8
9     Restaurant *rest = Restaurant::getInstance(); // Указатель на синглтон
10    rest->openRestaurant();
11    rest->startKitchen();
12
13    rest->acceptOrder("Apple");
14    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
15    rest->finishOrder();
16    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
17
18    rest->acceptOrder("Banana");
19    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
20    rest->finishOrder();
21    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
22
23    rest->closeKitchen();
24    rest->closeRestaurant();
25
26    Restaurant *rest1 = Restaurant::getInstance();
27    printf("\n-----\n");
28    printf("First object address: %p\n", (void *)rest);
29    printf("Second object address: %p\n", (void *)rest1);
30
31    return 0;
32 }
```

Приложение Г. Код meyers/kitchen.h

```
1 #pragma once
2 #include <string>
3 #include <iostream>
4
5 class Kitchen
6 {
7 public:
8     enum kitchenStatus
9     {
10         busy = 1,
11         free = 2,
12         dayOff = 3
13     };
14
15     void startKitchen()
16     {
17         if ((this->kitchenState != kitchenStatus::busy) &&
18             (this->kitchenState != kitchenStatus::dayOff))
19         {
20             this->kitchenState = kitchenStatus::free;
21             std::cout << "Kitchen started, state: free" << std::endl;
22         }
23     }
24
25     void closeKitchen()
26     {
27         if (this->kitchenState == kitchenStatus::free)
28         {
29             this->kitchenState = kitchenStatus::dayOff;
30             std::cout << "Kitchen closed, state: dayOff" << std::endl;
31             std::cout << "Orders number: " << this->orderNumber <<
32             std::endl;
33             this->orderNumber = 0;
34         }
35     }
36
37     void acceptOrder(std::string order)
38     {
39         if (this->kitchenState != kitchenStatus::free)
40             std::cout << "Can't accept order, kitchen is already cooking"
41             << std::endl;
42         else
43         {
44             std::cout << "Accepted order " << order << std::endl;
45             this->kitchenState = kitchenStatus::busy;
46             this->currentOrder = order;
47         }
48     }
49
50     std::string finishOrder()
51     {
52         if (this->kitchenState != kitchenStatus::busy)
53             std::cout << "Can't finish order, kitchen currently is not
54             cooking" << std::endl;
55         else
56         {
57             return nullptr;
58             std::string _currentOrder = this->currentOrder;
59             this->kitchenState = kitchenStatus::free;
60             this->currentOrder = "";
61             std::cout << "Finished order " << _currentOrder << std::endl;
62             this->orderNumber++;
63             return _currentOrder;
64         }
65     }
66
67     std::string getCurrentOrder() const
68     {
69     }
```

```

64         if (currentOrder != "")
65             return currentOrder;
66         return "-- nothing --";
67     }
68
69 protected:
70     kitchenStatus kitchenState;
71     std::string currentOrder;
72     int orderNumber;
73     Kitchen()
74     {
75         this->kitchenState = kitchenStatus::free;
76         this->currentOrder = "";
77         this->orderNumber = 0;
78     };
79     ~Kitchen() = default;
80 };

```

Приложение Д. Код meyers/restaurant.h

```
1 #include <string>
2 #include <utility>
3 #include "kitchen.h"
4
5 class Restaurant : public Kitchen
6 {
7 public:
8     static Restaurant *getInstance()
9     {
10         static Restaurant inst;
11         return &inst;
12     }
13
14     // other methods
15     enum restaurantStatus
16     {
17         open = 1,
18         closed = 2
19     };
20
21     void openRestaurant()
22     {
23         if (this->restaurantState != restaurantStatus::open)
24         {
25             this->restaurantState = restaurantStatus::open;
26             std::cout << "Restaurant opened" << std::endl;
27         }
28     }
29
30     void closeRestaurant()
31     {
32         if (this->restaurantState != restaurantStatus::closed)
33         {
34             this->restaurantState = restaurantStatus::closed;
35             std::cout << "Restaurant closed" << std::endl;
36         }
37     }
38
39 private:
40     static Restaurant *p_instance;
41     int restaurantState;
42     Restaurant(){}; // Private constructor
43     ~Restaurant(){}; // Private destructor
44     Restaurant(const Restaurant &); // Prevent copy constructor
45     Restaurant &operator=(const Restaurant &); // Prevent assignment
46 };
```


Приложение Е. Код meyers/main.cpp

```
1 #include <iostream>
2 #include "kitchen.h"
3 #include "restaurant.h"
4
5 int main()
6 {
7     std::cout << "Meyer's Singleton" << std::endl;
8
9     Restaurant *rest = Restaurant::getInstance(); // Указатель на синглтон
10    rest->openRestaurant();
11    rest->startKitchen();
12
13    rest->acceptOrder("Apple");
14    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
15    rest->finishOrder();
16    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
17
18    rest->acceptOrder("Banana");
19    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
20    rest->finishOrder();
21    std::cout << "Current order: " << rest->getCurrentOrder() << std::endl;
22
23    rest->closeKitchen();
24    rest->closeRestaurant();
25
26    Restaurant *rest1 = Restaurant::getInstance();
27    printf("\n-----\n");
28    printf("First object address: %p\n", (void *)rest);
29    printf("Second object address: %p\n", (void *)rest1);
30
31    return 0;
32 }
```

Приложение Ж. Код ru/main.py

```
1 from enum import Enum
2
3
4 class Kitchen:
5     class kitchenStatus(Enum):
6         busy = 1
7         free = 2
8         dayOff = 3
9
10    def __init__(self) -> None:
11        self.kitchenState: Kitchen.kitchenStatus =
Kitchen.kitchenStatus.free
12        self.currentOrder: str = ""
13        self.orderNumber: int = 0
14
15    def startKitchen(self):
16        if self.kitchenState != Kitchen.kitchenStatus.free:
17            self.kitchenState = Kitchen.kitchenStatus.free
18            print("Kitchen started, state: free")
19
20    def closeKitchen(self):
21        if self.kitchenState == Kitchen.kitchenStatus.free:
22            self.kitchenState = Kitchen.kitchenStatus.dayOff
23            print("Kitchen closed, state: dayOff")
24            print("Orders number:", self.orderNumber)
25            self.orderNumber = 0
26
27    def acceptOrder(self, order: str):
28        if self.kitchenState != Kitchen.kitchenStatus.free:
29            print("Can't accept order, kitchen is already cooking")
30        else:
31            print("Accepted order:", order)
32            self.kitchenState = Kitchen.kitchenStatus.busy
33            self.currentOrder = order
34
35    def finishOrder(self):
36        if self.kitchenState != Kitchen.kitchenStatus.busy:
37            print("Can't finish order, kitchen currently is not cooking")
38            return ""
39        _currentOrder = self.currentOrder
40        self.kitchenState = Kitchen.kitchenStatus.free
41        self.currentOrder = ""
42        print("Finished order:", _currentOrder)
43        self.orderNumber += 1
44        return _currentOrder
45
46    def getCurrentOrder(self):
47        if self.currentOrder:
48            return self.currentOrder
49        return "-- nothing --"
50
51
52 class Restaurant(Kitchen):
53     def __new__(cls):
54         if not hasattr(cls, "instance"):
55             cls.instance = super(Restaurant, cls).__new__(cls)
56         return cls.instance
57
58     def __init__(self) -> None:
59         super().__init__()
60         self.restaurantState = Restaurant.restaurantStatus.closed
61
62     class restaurantStatus(Enum):
63         open = 1
64         closed = 2
65
66     def openRestaurant(self):
```

```

67         if self.restaurantState != Restaurant.restaurantStatus.open:
68             self.restaurantState = Restaurant.restaurantStatus.open
69             print("Restaurant opened")
70
71     def closeRestaurant(self):
72         if self.restaurantState != Restaurant.restaurantStatus.closed:
73             self.restaurantState = Restaurant.restaurantStatus.closed
74             print("Restaurant closed")
75
76
77 if __name__ == "__main__":
78     print("Python Singleton")
79     rest: Restaurant = Restaurant()
80
81     rest.openRestaurant()
82     rest.startKitchen()
83
84     rest.acceptOrder("Apple")
85     print("Current order: ", rest.getCurrentOrder())
86     rest.finishOrder()
87     print("Current order: ", rest.getCurrentOrder())
88
89     rest.acceptOrder("Banana")
90     print("Current order: ", rest.getCurrentOrder())
91     rest.finishOrder()
92     print("Current order: ", rest.getCurrentOrder())
93
94     rest.closeKitchen()
95     rest.closeRestaurant()

```