Технологии программирования

Структурные паттерны. Adapter, Facade. Valgrind. Обнаружение утечек памяти и неинициализированных переменных.

Александр Михалевич

Типы паттернов проектирования

ПОРОЖДАЮЩИЕ

СТРУКТУРНЫЕ

ПОВЕДЕНЧЕСКИЕ

Структурные паттерны

ADAPTER

FACADE

BRIDGE

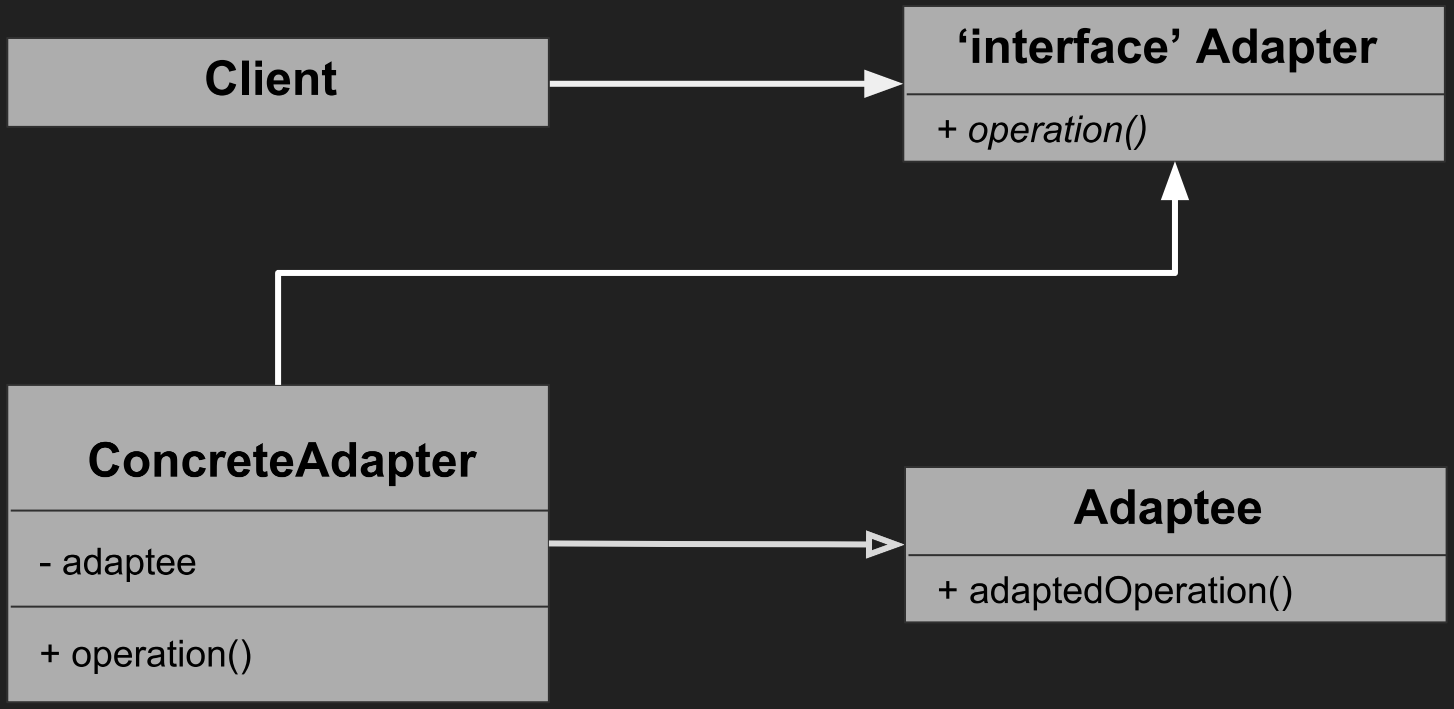
COMPOSITE

FLYWEIGHT

DECORATOR

PROXY

Adapter


Client
‘interface’ Adapter
+ *operation()*
ConcreteAdapter
- adaptee
+ operation()
Adaptee
+ adaptedOperation()

Adapter example

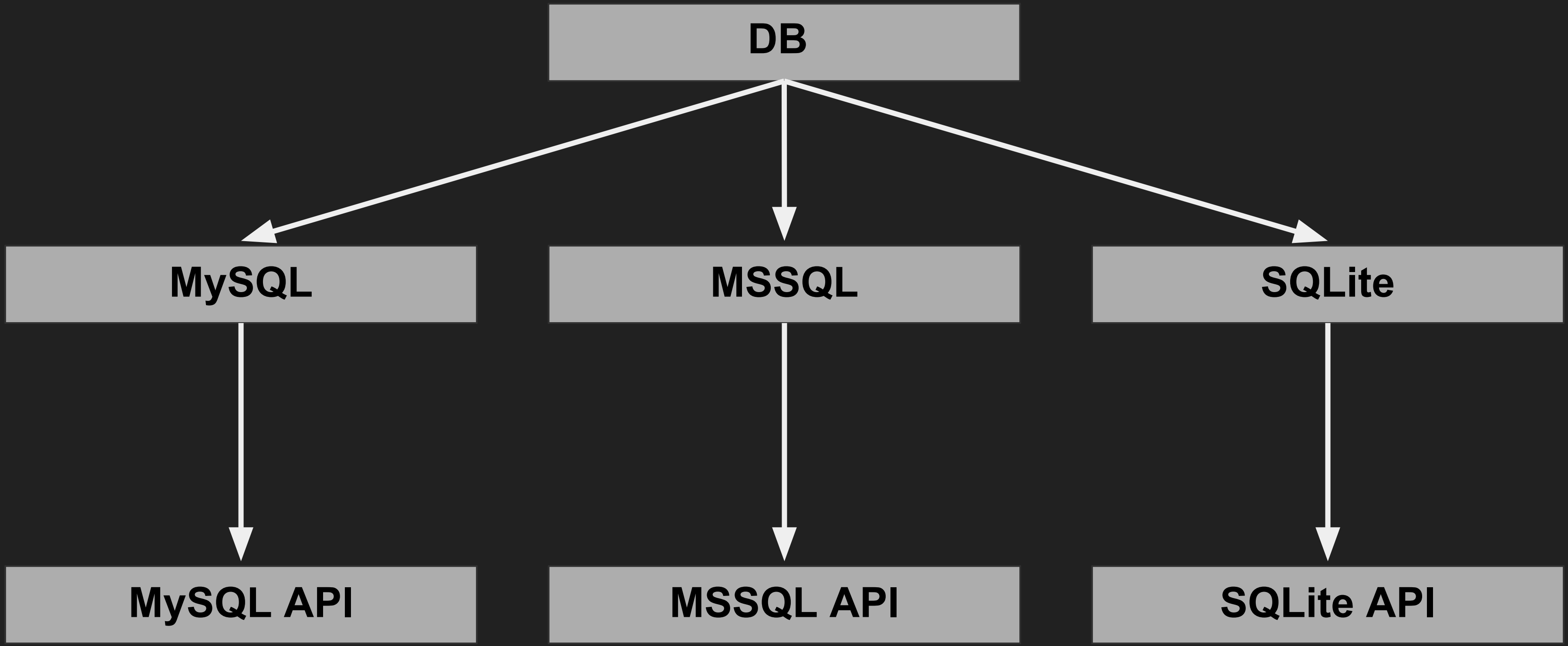
```
class IDatabase {
public:
	virtual void async_write(const id_t& id, const blob_t& data, callback_t callback) = 0;
	virtual void async_read(const id_t& id, blob_t& data, callback_t callback) = 0;
	virtual result_t sync_write(const id_t& id, const blob_t& data) = 0;
	virtual result_t sync_read(const id_t& id, blob_t& data) = 0;
};
```

Adapter example

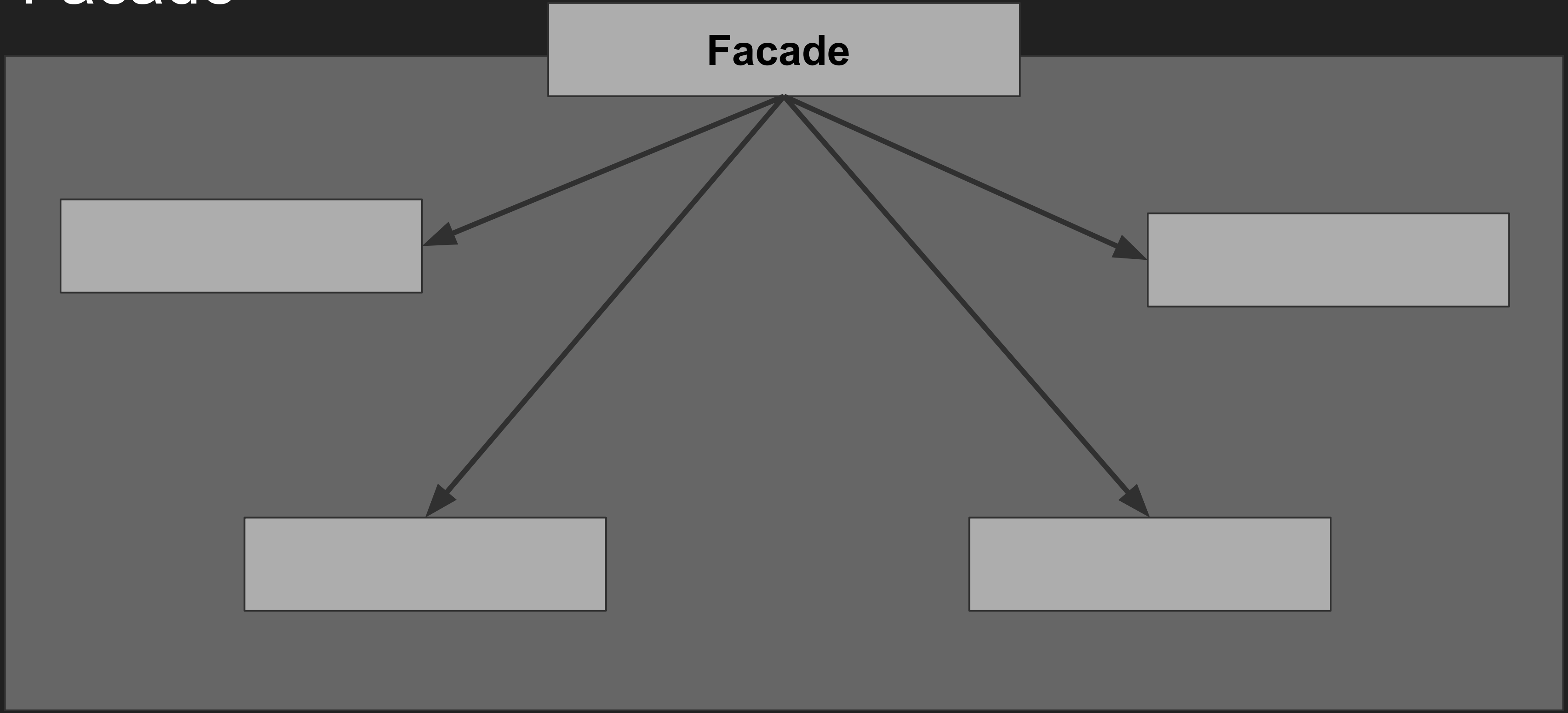
```
class CMySQLAdapter : public IDatabase {
public:
        void async_write(const id_t& id, const blob_t& data, callback_t callback) {
                // MySQL-specific code
        }
        ...
};


class CMSSQLAdapter : public IDatabase {
public:
        void async_write(const id_t& id, const blob_t& data, callback_t callback) {
                // MSSQL-specific code
        }
        ...
};
```

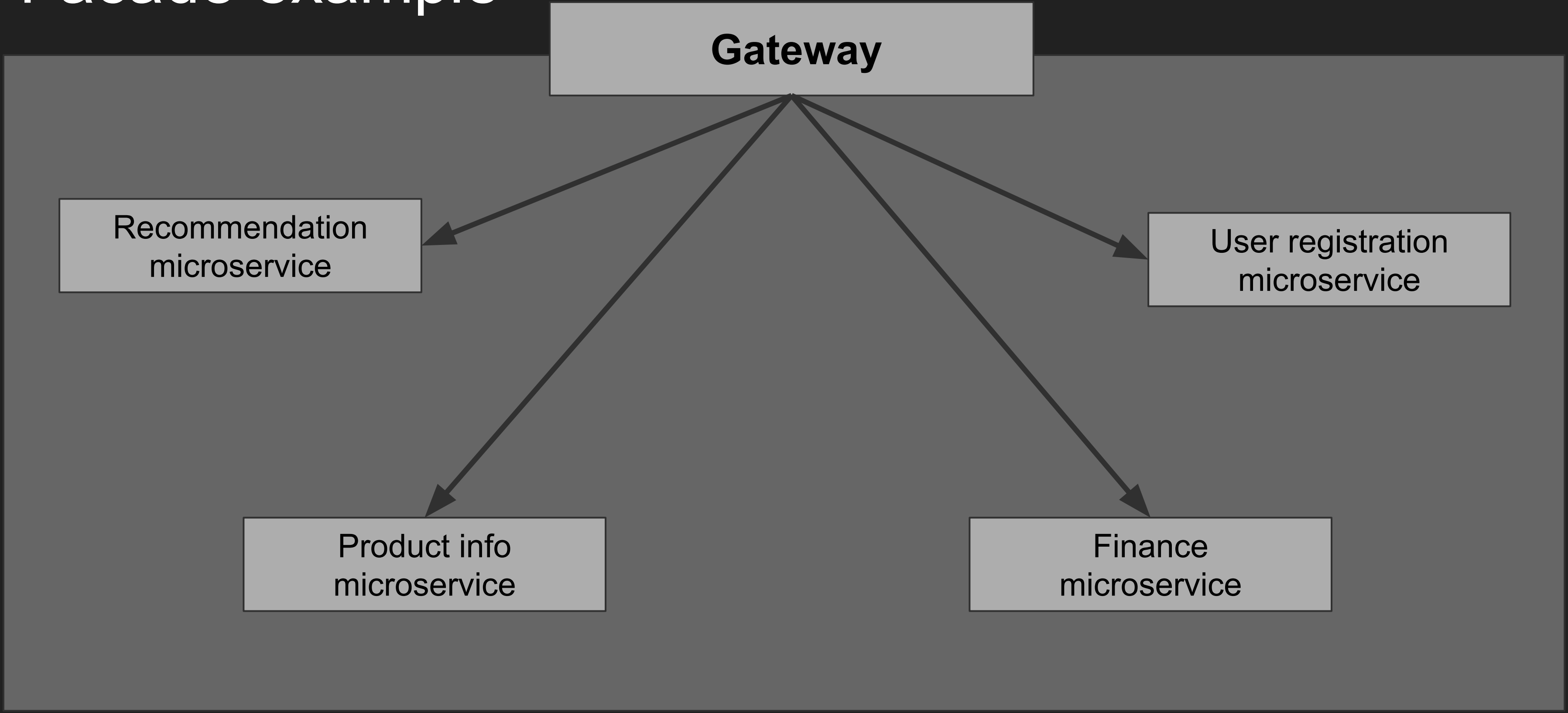
Adapter example



Facade



Facade example


Gateway
Recommendation microservice
User registration microservice
Product info microservice
Finance microservice

Valgrind. Обнаружение утечек памяти

```
#include <iostream>

int main() {
	const size_t SIZE = 20;
	int* arr = new int[20];
	for (size_t i = 0; i < SIZE; ++i) {
		arr[i] = i;
		std::cout << i << " ";
	}
	std::cout << std::endl;
	return 0;
}
```

Valgrind. Обнаружение утечек памяти

```
$ g++ -g test.cpp
$ valgrind --leak-check=full ./a.out
...
==2246== Command: ./a.out
==2246==
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
==2246==
==2246== HEAP SUMMARY:
==2246==     in use at exit: 80 bytes in 1 blocks
==2246==   total heap usage: 3 allocs, 2 frees, 73,808 bytes allocated
==2246==
==2246== 80 bytes in 1 blocks are definitely lost in loss record 1 of 1
==2246==    at 0x4C2CC6F: operator new[](unsigned long) (in /usr/lib64/valgrind/vgpreload_memcheck-amd64-linux.so)
==2246==    by 0x40082F: main (test.cpp:5)
==2246==
==2246== LEAK SUMMARY:
==2246==    definitely lost: 80 bytes in 1 blocks
==2246==    indirectly lost: 0 bytes in 0 blocks
==2246==      possibly lost: 0 bytes in 0 blocks
==2246==    still reachable: 0 bytes in 0 blocks
==2246==         suppressed: 0 bytes in 0 blocks
==2246== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Valgrind. Обнаружение утечек памяти

```
#include <iostream>

int main() {
	const size_t SIZE = 20;
	int* arr = new int[20];
	for (size_t i = 0; i < SIZE; ++i) {
		arr[i] = i;
		std::cout << i << " ";
	}
	std::cout << std::endl;
	delete[] arr;
	return 0;
}
```

Valgrind. Обнаружение утечек памяти

```
$ g++ -g test.cpp
$ valgrind --leak-check=full ./a.out
==16298== Memcheck, a memory error detector
==16298== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==16298== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==16298== Command: ./a.out
==16298==
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
==16298==
==16298== HEAP SUMMARY:
==16298==     in use at exit: 0 bytes in 0 blocks
==16298==   total heap usage: 3 allocs, 3 frees, 73,808 bytes allocated
==16298==
==16298== All heap blocks were freed -- no leaks are possible
==16298==
==16298== For counts of detected and suppressed errors, rerun with: -v
==16298== ERROR SUMMARY: 0 errors from 0 contexts (suppressed: 0 from 0)
```

Valgrind. Неинициализированные переменные

```
#include <iostream>
int main() {
	const size_t SIZE = 20;
	int* arr = new int[20];
	int tmp;
	for (size_t i = 0; i < SIZE; ++i) {
		arr[i] = i;
		std::cout << arr[i] << " ";
	}
	if (tmp != 0)
		arr[SIZE - 1] = -1;

	for (size_t i = 0; i < SIZE; ++i)
		std::cout << arr[i] << " ";
	std::cout << std::endl;
	delete[] arr;
	return 0;
}
```

Valgrind. Обнаружение утечек памяти

```
$ valgrind --track-origins=yes ./a.out
==24185== Memcheck, a memory error detector
==24185== Copyright (C) 2002-2015, and GNU GPL'd, by Julian Seward et al.
==24185== Using Valgrind-3.12.0 and LibVEX; rerun with -h for copyright info
==24185== Command: ./a.out
==24185==
==24185== Conditional jump or move depends on uninitialised value(s)
==24185==    at 0x4008D5: main (test.cpp:12)
==24185==  Uninitialised value was created by a stack allocation
==24185==    at 0x400856: main (test.cpp:3)
==24185==
0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19
==24185== HEAP SUMMARY:
==24185==     in use at exit: 0 bytes in 0 blocks
==24185==   total heap usage: 3 allocs, 3 frees, 73,808 bytes allocated
==24185== All heap blocks were freed -- no leaks are possible
==24185==
==24185== ERROR SUMMARY: 1 errors from 1 contexts (suppressed: 0 from 0)
```

Valgrind. Неинициализированные переменные

```
#include <iostream>
int main() {
        const size_t SIZE = 20;
        int* arr = new int[20];
        int tmp = 1;
        for (size_t i = 0; i < SIZE; ++i) {
                arr[i] = i;
                std::cout << arr[i] << " ";
        }
        if (tmp != 0)
                arr[SIZE - 1] = -1;

        for (size_t i = 0; i < SIZE; ++i)
                std::cout << arr[i] << " ";
        std::cout << std::endl;
        delete[] arr;
        return 0;
}
```

Обращайте внимание на warning’и!

```
$ g++ -g -Wall test.cpp
test.cpp: In function ‘int main()’:
test.cpp:12:2: warning: ‘tmp’ is used uninitialized in this function [-Wuninitialized]
  if (tmp != 0)
  ^~
```