

Tema 14 | Type casting. SOLID principles.

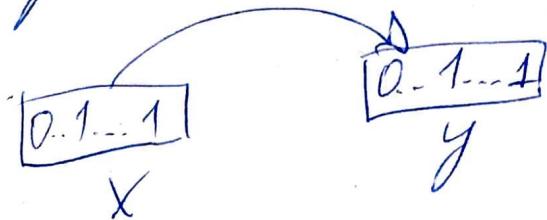
Design patterns -任务 в примере (Singleton, factory, prototype, composite, flyweight, iterator, command, visitor)

Type casting

1. Преобразование от типа к типу

Пример: $\text{int } x = 7$

$\text{double } y = x;$



⚠ $\text{int } x = -1$

$\text{unsigned } y = x$

$y = \text{INT_MAX};$

2. Конвертация типа на указатель

Пример: $\text{Der }^* d = \text{new Der}();$

$\text{Base }^* b = d; // C-style cast$

static_cast

- изначально это какая-то другая библиотека
которое преобразование

- не требует runtime check

- при грешке std::terminate

Пример:
 $\text{Base }^* \text{ptr} = \text{static_cast}<\text{Base }^*>(\text{d}); // upcasting$

I downcasting

if ($\text{ptr} \rightarrow \text{getType}() == 1$)

$\text{Der}^* \text{obj} = \text{static_cast} < \text{Der}^* > (\text{ptr})$

II dynamic-cast

- ako je one curyma & tma
- uznoscane so za downcasting
- uma runtime check (slower than static-cast)
- npu zpuska bpolje nullptr
- uznoscia vtable, zavjoro ta on naga tma
- spada noise i virt fizyczne

Пример:

$f(\text{Base}^* \text{ptr})$

if ($A^* \text{ptr} = \text{dynamic_cast} < A^* > \text{ptr}$) {

// blizgna ako ptr ima crinoc

} else {

// blizgna ako ptr=nullptr

}

⚠ Ako racvane peob(&) → throw std::bad_cast

III const-cast

- za prenajbave na konstantnosti
- legacy nog

(2)

~~#P:~~ Не подори върху данни, които са били инициализирани като константи. Това е така, защото const и static инициализиране съдържа "неизменно" място в паметта.

Пример:

```
f(const int* value){  
    int* h = const_cast<int*>(value);  
    // тук се комбинира, но не е  
    // undefined behaviour  
}
```

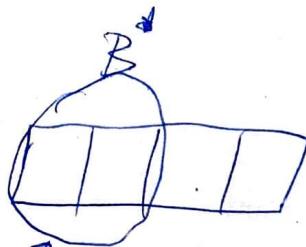
```
int main(){  
    const int x=10  
    f(&x);  
}
```

IV Reinterpret-cast

- от един указател към всеки указател
- репартерптира му паметта

Пример:

A - 4 байта, B - 2 байта

$A^* \text{ptr} = \text{new } A();$ A^* \rightarrow 

$B^* \text{ptr2} = \text{reinterpret_cast}<B^*>(\text{ptr});$ B^*

записване в паметта

```
ofs.write(reinterpret_cast<const char*>(ptr), size);  
ofs.read(reinterpret_cast<char*>(ptr), size);
```

SOLID principles - 5 принципа за добър код

S-Single responsibility

- един компонент има ТОЧНО 1 отговорност
- висока cohesion

Пример

1) Наричана - Уединяващо носе

2) Person не споделя зас как да конкретизира
char*

Пример

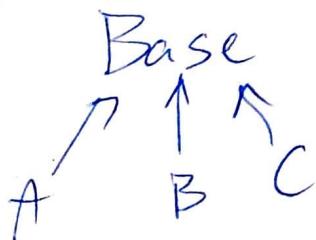
read And Write(); X

read(); ✓ write(); ✓

O-Open/Closed

- open for extension, closed for modification

Пример:



f(const Base*) не споделя га променя свойта функционалност при добавяне на нов наследник

L-Liskov's substitution

- споделя га използване указател към базовия клас дез га се интерпретира от конкретния тип

#include: class Car
get(cabinSize());

class RaceCar: public Car
get(cabinSize()) override —(throw)
get(CockpitSize())

funcame

```
f(Car & c){  
    c.get(cabinSize());  
}
```

Peremne:

```
//class Vechicle  
//peremnye vse
```

⚠ dynamic_cast<Car*>(ptr)
напомина
напомина !

I - Interface segregation

- Torpedine ist ja ke e gorden ga ummenytyra
n ustebeic, konta ne iproba ga ce. uzhazba

Пример:

```
1) struct OfficeMachine  
    virtual print()  
    virtual fax()  
    virtual scan()
```

```
2) struct Printer: OfficeMachine
```

```
    print()  
    fax  
    scan{throw}
```

```
3) struct Fax: OfficeMachine  
    print{throw}  
    scan{throw}  
    fax{}
```

Peremne:

IOfficeMachine
Iprint, Iscan, IFax

Fax: IFax Printer: IPrint,
 fax()

IScan,
print
scan

IScan: virtual IOfficeMachine
IPrint: virtual IOfficeMachine

5

D-Dependency inversion

- Могунаре от високо ниво, не трошка џа ^{инв.}
от могунаре на ниско ниво.
// Ниско ниво - конкретен гојчин џа гравите.

Продакт:

Product Catalog

Mongo Provider - db; // Продакт употребува ниска
void Print() {
 - db.getAllData();
}

от SAL Provider

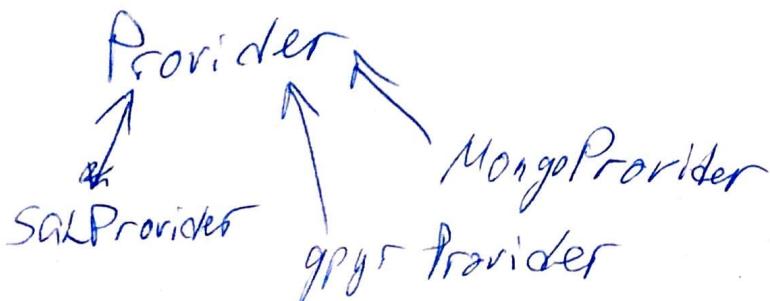
Потребник:

Product Catalog

Provider * db;
ProductCatalog(Provider * db); // dependency
void Print {
 - db.getAllData();
}

injection = начин
за подавување на
този принцип

- закачете на битум
забелешката, а не
богатство објектите



Design patterns

- решенија на често сречани проблеми
- добри дизајн практики

I Creational

- група за создаване на обекти, като серви, логиката за техното создаване

II Structural

- создаване на по-сложни обекти
(наследяване и полиморфизъм)

III Behavioural

- комуникация между обекти

Patterns:

Singleton

Осигурява една искателна на един клас

- глобален достъп
- използва се за тексли за создаване, гордо използвани се класове

Пример: Една колекция с базара, StringPool, Factory

...
...

...

1

Singleton

public:

```
static Singleton & getInstance()
```

```
    static Singleton instance;
```

```
    return instance;
```

```
}
```

private:

```
SG();
```

```
SG(const SG&) = delete;
```

```
SG& op = (const SG&) = delete;
```

```
~SG();
```

Недостатки:

- многопоточное программирование

- антипаттерн

- общий с конкретной инициализацией

2. Factory

- упрощение MACRO за изграждане на обект

- . factory - една обща create()

- . factory method

Factory

Base create();

AFactory

Base * create();

BFactory

Base * create();

8

Abstract factory

BaseFactory

A* createA();

B* createB();

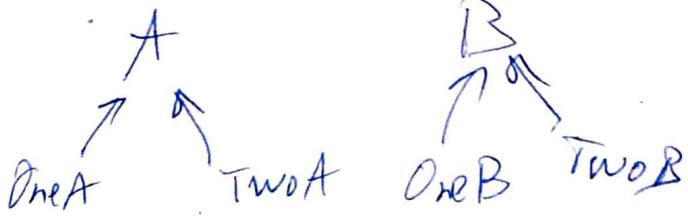
OneFactory

A* createA()

new OneA;

B* createB();

new OneB;



TwoFactory

A* createA();

new TwoA;

B create

new TwoB;

- + генерация на съсъд 3^g
- ограждане
- подобни експансионисти

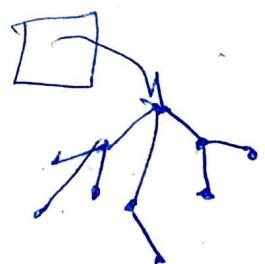
- много ниско на общи ресурси
- грижа за хипотеза на тези factory-ти

3. Prototype

- ограждане на конекти на обекта докато се инициализира
- класът е тип на обект от нов. тип.
- clone()

4. Composite factory

- композиране на обекти в деривираща структура

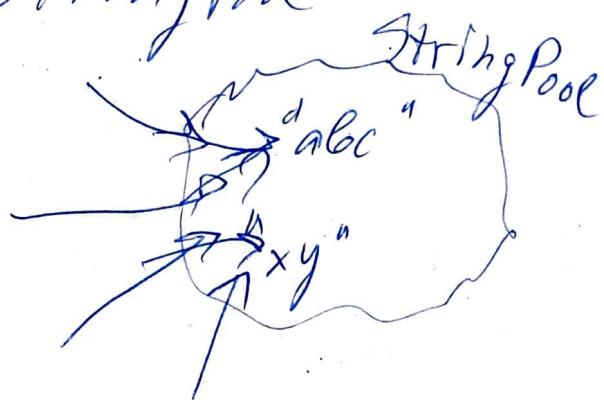


- единична входна точка
- листа и деривирани типове

5. Flyweight

- Обща обект и наследва като съществува
одинакъв рефрен
- Намалява използваната памет
- immutable обекти

Пример: StringPool



6. Iterator

- начин за работа с коллекции без да се изпълняват какъв е този
- изпратор-указател към конкретен element
- нужен изпратор

Iterator

- Op^{*} - geprob.
- ++ - next()
- - back()
- ==, !=

Collection

- begin();
- end();

Пример:

- Търсене на одеко в конкрайт
template <typename Iter, typename T>
bool search(Iter begin, const Iter end, const T& el)
 while (begin != end) {
 if ((*begin) == search)
 return true;
 ++begin;
 }
 return false;
}

7. Command

- изпълнение на задачи като управляващи
одекон

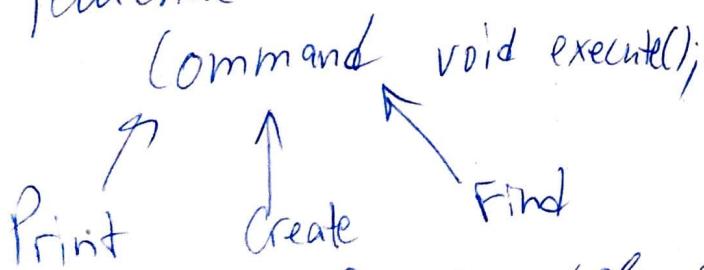
Пример:

>create

>print

>undo

Редене:



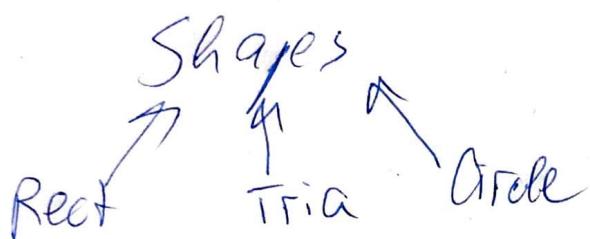
Main.cpp

Command cmd=CommandFactory::
(STR);

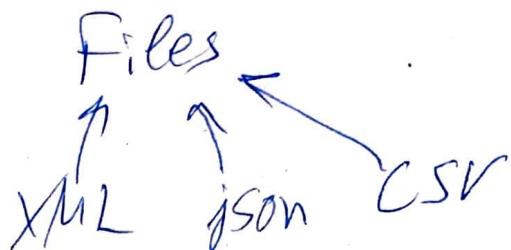
(cmd.execute();

- може да се направи изблъскване и да се избегне
менюкодинга на тези задачи

P. Visitor / duel Rock, Paper, Scissors
- programme melegy algoritmust
eggy körök padlás



virt serialize(File* f)



virt serCircle()
virt serTri()
virt serRect()

Typenep: void ser(Shape* sh, File* f)
sh->serialize(f);

```
#pragma once  
#include <iostream>  
using std::cout;
```

```
class Base
```

```
{ public:  
    virtual ~Base() = default;
```

```
}  
class A : public Base
```

```
{ public:  
    A() { cout << "A()" ; }  
    ~A() { cout << "~A()" ; }
```

```
}  
class B : public Base
```

```
{ public:  
    B() { cout << "B()" ; }  
    ~B() { cout << "~B()" ; }
```

```
}
```

Obj.h

SingletonFactory.h

```
#pragma once  
#include <iostream>  
#include "Obj.h"
```

```
class SingletonFactory
```

```
{ private:
```

```
    int * _date = nullptr;  
    size_t _numbersCount = 0;  
    size_t _currIndex = 0;
```

```
    SingletonFactory(const SF&) = delete;
```

```
    SF& operator=(const SF&) = delete;
```

```
    SF();
```

```
    ~SF();
```

```
public:
```

```
    static SingletonFactory& getInstance();
```

```
    Base * create();
```

```
}
```

```
#include "SingletonFactory.h"
```

SingletonFactory.cpp

```
SingletonFactory::SF::getInstance()
```

```
{  
    static SingletonFactory instance;  
    return instance;
```

```
}
```

```
Base* SF::create()
```

```
{
```

```
    if(-numberCount <= currIndex)
```

```
    {  
        throw std::runtime_error("All numbers used!");  
    }
```

```
}
```

```
    if(-data[-currIndex+1] % 2 == 0)
```

```
    {  
        return new A();  
    }
```

```
}
```

```
else
```

```
    {  
        return new B();  
    }
```

```
}
```

```
}  
SF::SF()
```

```
{  
    std::fstream ifs("data.txt", std::ios::in);
```

```
    if(!ifs.is_open())
```

```
        throw std::runtime_error("cannot open file");
```

```
if s >= numbersCount  
    - currIndex = 0  
    - data = new int [-numbersCount]  
  
try {  
    int i=0;  
    while(!ifs.eof()) {if s >= data[i++];}  
}  
catch (std::bad_alloc & ex)  
{ delete [-data];  
throw;  
}  
}  
SF::~SF()  
{ delete [-data];  
data = nullptr;  
numbersCount = currIndex = 0;  
}
```

```
#include "SingletonFactory.h"
#include <iostream>
#include "Obj.h"
```

main.cpp

```
int main()
```

```
{
```

```
    SingletonFactory & sf = SingletonFactory::getByName("
```

```
    Base * o1 = sf.create();
```

```
    Base * o2 = sf.create();
```

```
    Base * o3 = sf.create();
```

```
    delete o1;
```

```
    delete o2;
```

```
    delete o3;
```

```
}
```