Абстрактни класове и интерфейси

титуляр на курса: д-р Тодор Цонков (ttsonkov@gmail.com)

От какво ще се състои настоящата лекция?

Абстрактни класове и интерфейси. Pure virtual functions. Object slicing, type casts. Type Erasure (std::function, std::any C++ 11). Сравнение с езика Java.

Синтаксис

Какво е интерфейс?

В C++ няма ключова дума interface.

Интерфейсът е дизайнерска концепция, която се реализира чрез клас, който съдържа само pure virtual функции без състояние (data members) с виртуален деструктор. Трябва всички методи да са публични.

Целта е да дефинира какво може обектът, а не как е реализиран.

Следва Dependency Inversion Principle (DIP),който ще разгледаме в лекция 15.

```
class Person {
public:
    virtual ~Person() = default; // ЗАДЪЛЖИТЕЛНО
    virtual std::string role() const = 0; // pure virtual
};
class Student : public Person {
public:
    std::string role() const override {
        return "Student";
    }
};
```

Какво е абстрактен клас?

Абстрактен клас:

1)Не може да се инстанцира
2)Служи като интерфейс + базова логика
3)Има поне една pure virtual функция

Гарантира, че всички наследници:
Имплементират нужните функции и имат един и същ публичен достъп.

Може да има и член данни за разлика от интерфейса!

Синтаксис

```
class Person {

protected:

    std::string name_;

public:

    Person(std::string name) :
name_(std::move(name)) {}

    virtual void print() const {

        std::cout << "Name: " << name_ << "\n";

    }

    virtual std::string role() const = 0;

};
```

Добри практики

Интерфейс се прави когато има реална нужда от runtime полиморфизъм и типовете се сменят по време на изпълнение.
Множествено наследяване почти винаги трябва да е с интерфейси, а не класове със стейт!

Не трябва да се започва с интерфейс, а с конкретен клас и след това да се въведат интерфейси при нужда. В много случаи се препоръчват std::function и std::any, които ще бъдат разгледани в настоящата лекция.

Пример

```
struct Vehicle {
    virtual void start() = 0;
    virtual void stop() = 0;
    virtual void fly() = 0;
    virtual void sail() = 0;
}; - Лош пример. Правилно:
struct Drivable {
    virtual void drive() = 0;
};
struct Flyable {
    virtual void fly() = 0;
};
```

Object Slicing

Получава се при копиране по стойност
Базовият клас не знае за допълнителните членове

Решение: указатели или референции

```
void process(const Person& p);   // OK
void process(Person* p);         // OK
```

❌ Никога: `void process(Person p);`

Синтаксис

```
class Person {
public:
    std::string name;
};

class Student : public Person {
public:
    int facultyNumber;
};

Person p = Student{"Ivan", 123}; //
slicing!
```

Type casts в езика C++

Обяснение

Type cast = изрично преобразуване на тип.В C++ има 4 различни оператора, всеки със строго определено предназначение. Различните cast-ове имат различна степен на безопасност.

Пример от езика C

Student* s = (Student*)p; //

1) може да означава static_cast, reinterpret_cast или const_cast, но не е ясно от кода, че се транслира до него

2)няма яснота какво реално прави

3)трудно се ревюира

4)прикрива грешки

static_cast

double d = 3.14;
int i = static_cast<int>(d);
✔ проверява се по време на компилация
✔ без runtime overhead
❌ не проверява реалния тип на обекта

Type casts в езика C++

Проблем

```
Person* p = new Student{};
Student*s=static_cast<Student*
>(p);
```

Опасно, ако p не сочи към Student

Решение

```
Person* p = new Student{};

Student* s =
dynamic_cast<Student*>(
p);
if (s) {
    // безопасно
}
```

проверява реалния тип
връща nullptr при грешка
runtime overhead

dynamic_cast

Класът трябва да е polymorphic (да има поне една virtual функция)

```
class Person {
public:
    virtual ~Person() = default;
};
```

const_cast

const_cast е единственият cast в C++, който:
може да премахва (const) и (volatile) квалификатори
2)не променя типа, само cv-квалификациите
```
const int x = 10;
int& y = const_cast<int&>(x);
```
3) Това не означава, че е безопасно да се модифицира x.

🔹 Какво МОЖЕ да прави const_cast

✔ маха const / volatile
✔ добавя const (рядко полезно)
✔ работи с указатели и референции

reinterpret_cast

reinterpret_cast е най-ниско-нивовият и най-опасният cast в C++.

➡️ Казва на компилатора:

„Интерпретирай тези битове по друг начин.“

```
int* p = reinterpret_cast<int*>(0x12345678);
```

Какво може да прави:

✔ преобразува между несвързани типове указатели
✔ pointer ↔ integer
✔ function pointer ↔ function pointer

🔹 Златно правило

Ако мислите, че се нуждаете от reinterpret_cast, 99% от времето дизайнът е грешен.

Liskov substitution principle (L in SOLID)

Определение:

Обект от наследен клас трябва напълно да замества базовия.

Проблемът със Square / Rectangle:

Rectangle обещава: ширина и височина са независими

Square нарушава това обещание

❗ LSP е за поведение, не за синтаксис

Практически сигнал за проблем:

if (dynamic_cast…)

проверки за тип

изключения в override

Правилен пример

```
class Shape {

public:

    virtual int area() const = 0;

    virtual ~Shape() = default;

};

class Rectangle : public Shape

class Square : public Shape,
```

а НЕ

```
class Square : public Rectangle
```

Interface Segregation Principle

Проблем

Пример

```
class Worker {
public:
   virtual void work() = 0;
   virtual void eat() = 0;
   virtual ~Worker() = default;
};
```

Пример

```
class Human : public Worker {
public:
    void work() override {
        // работи
    }

    void eat() override {
        // яде
    }
};
class Robot : public Worker {
public:
    void work() override {
        // работи
    }
    void eat() override {
        // ❌ роботите не ядат
        throw std::logic_error("Robot cannot eat");
    }
};
```

Грешка

Robot е принуден да имплементира eat()
появяват се празни методи или изключения
клиентите зависят от ненужни функции

➡ ISP е нарушен

Решение

```
class Workable {
public:
    virtual void work() = 0;
    virtual ~Workable() = default;
};
class Eatable {
public:
    virtual void eat() = 0;
    virtual ~Eatable() = default;
};
```

Проблеми и решения

❌ Проблеми:

интерфейси с много методи

override-и с празно тяло

трябва да пишем: throw NotImplemented

класът „знае твърде много“

✅ Добра практика:

интерфейси с ясна роля

често имена завършващи на -able

Какво е type erasure?

Type Erasure е техника, при която:

Скриваме конкретния тип на обект
Работим с него чрез унифициран интерфейс
👉 Идея: „Не ме интересува какъв е типът, важно е какво може да прави“

Цели:
1)Полиморфизъм без наследяване
2)Runtime гъвкавост

Какъв проблем решава?

```
template<typename F>
void run(F f) {
    f();
}
```

❌ Всеки различен F → нова инстанция
❌ Няма общ тип за съхранение
❌ Трудно предаване между модули

👉 Не можем да кажем:

```
std::vector<??> tasks;
```

Примери

std::function

std::function:

Приема всякакъв callable обект:
функция, lambda, functor, std::bind

obj(args...);
=> държи се като функция

Скрива реалния тип → type erasure
Осигурява един общ тип

Какво е std::bind -> стандартна функция (в <functional>) в C++, която създава нов callable обект, като фиксира някои от аргументите на друга функция или пренарежда тяхната последователност. Позволява частично прилагане и пренасочване на аргументи към функция.

```cpp
int add(int a, int b) {
    return a + b;
}

auto add5 = std::bind(add, 5, std::placeholders::_1);
add5(3);   // 8

void foo() { std::cout << "foo\n"; }

struct Bar {
    void operator()() const {
        std::cout << "bar\n";
    }
};

std::function<void()> f1 = foo;

std::function<void()> f2 = [] {
    std::cout << "lambda\n";
};

std::function<void()> f3 = Bar{};
```

Какво е Dependency Inversion Principle?

Пример

Модулите от високо ниво не трябва да зависят от модулите от ниско ниво.И двата вида модули трябва да зависят от абстракции. Абстракциите не трябва да зависят от детайли.Детайлите трябва да зависят от абстракциите.
Пример:

```
class FileLogger {
public:
    void log(const std::string& msg) {
        std::cout << "File: " << msg << std::endl;
    }
};

class OrderService {
    FileLogger logger;   // ❌ директна зависимост
public:
    void createOrder() {
        logger.log("Order created");
    }
}; //OrderService знае точния клас, не можем лесно да използваме ConsoleLogger, DBLogger
```

Правилен дизайн

Пример

```
class Logger {
public:
    virtual void log(const std::string& msg) = 0;
    virtual ~Logger() = default;
}; //интерфейс
class FileLogger : public Logger {
public:
    void log(const std::string& msg) override {
        std::cout << "File: " << msg << std::endl;
    }
};

class OrderService {
    Logger& logger;   // ✅ зависимост от абстракция
public:
    OrderService(Logger& l) : logger(l) {}

    void createOrder() {
        logger.log("Order created");
    }
};
```

Пример предметната област School

```cpp
class Person {
public:
    virtual ~Person() {}
    virtual std::string name() const = 0;   // pure
virtual
    virtual void print() const = 0;
};

class Student : public Person {
    std::string m_name;
    double m_gpa;

public:
    Student(std::string name, double gpa)
        : m_name(name), m_gpa(gpa) {}

    std::string name() const override { return m_name; }
    double gpa() const { return m_gpa; }

    void print() const override {
        std::cout << "Student: " << m_name
                  << ", GPA=" << m_gpa << "\n";
    }
};
```

Пример предметната област School

```cpp
class School {
    std::vector<IPerson*> students; // raw pointers (без
ownership)

public:
    void add(IPerson* s) {
        students.push_back(s);
    }

    // Type erasure – приема всякакъв callable
    void for_each(const std::function<void(const
IPerson&)>& action) const {
        for (const IPerson* s : students)
            action(*s);
    }
};

Използване:
    Student s1("Ivan", 3.4);
    Student s2("Maria", 4.0);

    School school;
    school.add(&s1);
    school.add(&s2);

    // Lambda
    school.for_each([](const IStudent& s) {
        s.print();
    });
```

Пример на Java

```java
public interface Student {
    String getName();
    void printInfo();
}

public class RegularStudent implements Student {
    private String name;
    private double gpa;

    public RegularStudent(String name, double gpa) {
        this.name = name;
        this.gpa = gpa;
    }

    public String getName() {
        return name;
    }

    public double getGpa() {
        return gpa;
    }

    @Override
    public void printInfo() {
        System.out.println("Student: " + name + ", GPA="
+ gpa);
    }
}
```

Пример на Java

```java
import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;
import java.util.function.Predicate;
public class School {
    private List<Student> students = new ArrayList<>();

    public void add(Student s) {
        students.add(s);
    }

    // Consumer<Student> = еквивалент на
std::function<void(Student)>
    public void forEachStudent(Consumer<Student> action)
{
        for (Student s : students) {
            action.accept(s);
        }
    }

    public void filterAndApply(Predicate<Student> pred,
                               Consumer<Student> action)
{
        for (Student s : students) {
            if (pred.test(s)) {
                action.accept(s);
            }
        }
    }
}
```

Въпроси?

Благодаря за вниманието!