Презентация на тему “Инкапсуляция, наследование и полиморфизм в C++”

Основные принципы ООП и зачем они нужны

Введение

Объектно-ориентированное программирование (ООП) — парадигма, где код строится вокруг объектов, сочетающих данные и поведение. ООП основано на трёх основополагающих концепциях — инкапсуляция, наследование и полиморфизм. Их задача заключается в том, чтобы решать проблемы повторяемости, безопасности и гибкости кода. Они упрощают разработку сложных систем, делая код модульным и легким в поддержке. Применение этих принципов ускоряет командную работу и снижает количество ошибок.

Что такое инкапсуляция?

Инкапсуляция (encapsulation) – это механизм, который объединяет данные и код, манипулирующий этими данными, а также защищает и то, и другое от внешнего вмешательства или неправильного использования. В объектно-ориентированном программировании код и данные могут быть объединены вместе; в этом случае говорят, что создаётся так называемый "чёрный ящик". Когда коды и данные объединяются таким способом, создаётся объект (object). Другими словами, объект – это то, что поддерживает инкапсуляцию.

- Внутри объекта коды и данные могут быть закрытыми. Они доступны только для других частей объекта. Другими словами, недоступны для тех частей программы, которые существуют вне объекта.
- Если коды и данные являются открытыми, то, несмотря на то, что они заданы внутри объекта, они доступны и для других частей программы.
- Объект является переменной определённого пользователем типа. Каждый элемент данных такого типа является составной переменной.

Пример работы инкапсуляции

```
1  class BankAccount {
2  private:
3      double balance;  // скрытые данные
4
5  public:
6      // открытые методы для работы с данными
7      void deposit(double amount) {
8          if (amount > 0) balance += amount;
9      }
10
11     double getBalance() {
12         return balance;
13     }
14 };
```

Класс BankAccount скрывает внутренние данные (поле balance) в приватной секции, защищая их от прямого доступа извне. Для работы с этими данными предоставляются публичные методы deposit() и getBalance(), которые контролируют корректность операций. То есть, пользователь класса может взаимодействовать с объектом только через безопасный интерфейс, не имея возможности напрямую изменять его внутреннее состояние.

Преимущества и недостатки

Преимущества инкапсуляции:
Инкапсуляция защищает внутренние данные объекта от несанкционированного доступа и изменений, предотвращая ошибки и повышая безопасность кода. Она упрощает поддержку, позволяя менять реализацию внутри класса без влияния на внешний код, и делает интерфейс более понятным.

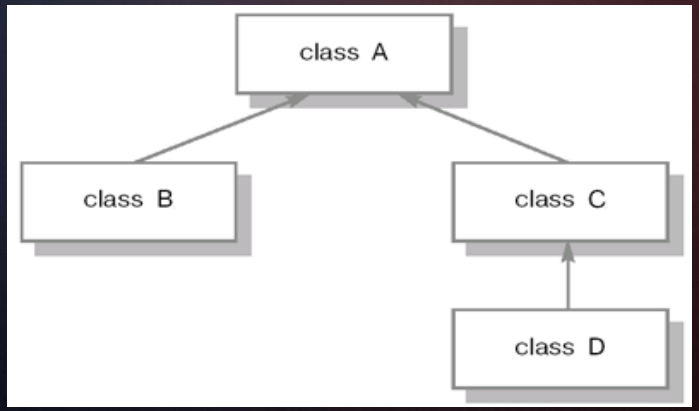
- Снижает сложность: код становится модульным, легче тестировать и отлаживать.
- Повышает переиспользуемость: объекты можно включать в другие проекты без дублирования.
- Обеспечивает контроль: геттеры и сеттеры проверяют данные перед изменениями.

Её недостатки:
Инкапсуляция увеличивает объем кода за счет дополнительных методов доступа, что усложняет чтение и сопровождение.

- Замедляет разработку: требует времени на проектирование интерфейсов.
- Риск переусложнения: избыточные getter/setter могут маскировать простую логику.

Что такое наследование?

Наследование (inheritance) – это процесс, посредством которого один объект может приобретать свойства другого. Точнее, объект может наследовать основные свойства другого объекта и добавлять к ним черты, характерные только для него. Наследование является важным, поскольку оно позволяет поддерживать концепцию иерархии классов. При использовании наследования можно описать объект путём определения того общего класса (или классов), к которому он относится, с теми специальными чертами, которые делают объект уникальным.



Пример работы наследования

```cpp
#include <iostream>
using namespace std;

class Device {
    public:
        int serial_number = 12345678;

        void turn_on() {
            cout << "Device is on" << endl;
        }
};

class Computer: private Device {
    public:
        void say_hello() {
            turn_on();
            cout << "Welcome to Windows 95!" << endl;
        }
};
```

В этом коде создается базовый класс Device с публичными полями и методом turn_on(). Класс Computer наследуется от Device через приватное наследование, что означает, что все унаследованные члены становятся приватными в классе-потомке.

Что такое полиморфизм?

Полиморфизм (polymorphism) – это свойство, которое позволяет одно и то же имя использовать для решения двух или более схожих, но технически разных задач. Целью полиморфизма, применительно к объектно-ориентированному программированию, является использование одного имени для задания общих для класса действий. Выполнение каждого конкретного действия будет определяться типом данных. В C++ существует два основных типа полиморфизма: полиморфизм времени компиляции (статический) и полиморфизм времени выполнения (динамический). Статический полиморфизм реализуется с помощью перегрузки функций и операторов, а динамический — с помощью виртуальных функций, позволяющих вызывать правильную реализацию метода во время выполнения программы.

Динамический

```cpp
class Property
{
protected:
    double worth;
public:
    Property(double worth) : worth(worth) {}
    virtual double getTax() const = 0;
};
class CountryHouse :
    public Property
{
public:
    CountryHouse(double worth) : Property(worth) {}
    double getTax() const override { return this->worth / 500; }
};

class Car :
    public Property
{
public:
    Car(double worth) : Property(worth) {}
    double getTax() const override { return this->worth / 200; }
};

class Apartment :
    public Property
{
public:
    Apartment(double worth) : Property(worth) {}
    double getTax() const override { return this->worth / 1000; }
};


void printTax(Property const& p)
{
    std::cout << p.getTax() << "\n";
}
```

Абстрактный класс Property, виртуальный метод getTax, поле worth, три класса: CountryHouse, Car, Apartment, которые реализуют данный метод

Статический

```cpp
class CountryHouse
{
private:
    double worth;
public:
    CountryHouse(double worth) : worth(worth) {}
    double getTax() const { return this->worth / 500; }
};

class Car
{
private:
    double worth;
public:
    Car(double worth) : worth(worth) {}
    double getTax() const { return this->worth / 200; }
};

class Apartment
{
private:
    unsigned worth;
public:
    Apartment(unsigned worth) : worth(worth) {}
    unsigned getTax() const { return this->worth / 1000; }
};

template <class T>
void printTax(T const& p)
{
    std::cout << p.getTax() << "\n";
}
```

Компилятор инстанцировал три различных функции и подставил нужную на этапе компиляции – поэтому полиморфизм на основе шаблонов и является статическим

Вывод

Инкапсуляция, наследование и полиморфизм — это три фундаментальных принципа ООП в C++, которые тесно взаимосвязаны:

Инкапсуляция создаёт "защищённые" классы с чётким интерфейсом, наследование позволяет расширять эти классы, создавая иерархии, а полиморфизм обеспечивает работу с объектами разных классов через единый интерфейс базового класса.

Наследование использует инкапсуляцию — производные классы наследуют не только методы, но и структуру доступа к данным. Полиморфизм же реализуется через наследование и виртуальные функции, позволяя одинаково работать с разными типами объектов.

Вместе они образуют мощную систему: инкапсуляция защищает данные, наследование организует иерархию, а полиморфизм обеспечивает гибкость использования объектов разных типов через общий интерфейс.

Источники

http://www.codenet.ru/progr/cpp/ipn.php

https://java-online.ru/java-oop.xhtml

https://sky.pro/wiki/java/chetyre-stolpa-oop-inkapsulyaciya-nasledovanie-polimorfizm-abstrakciya/

https://thecode.media/new-oop-inp/