ООП

Федотова Наталья Петровна

ООП (+ ПОСТАНОВКА ЗАДАЧИ)

- Принципы ООП
  - Инкапсуляция
  - Наследование
  - Полиморфизм
- Отношения между классами
  - включение (клиент, содержит)
  - наследование (является)
- Множественное наследование
- Виртуальные функции
- Чистые виртуальные функции
- Абстрактные классы
- Интерфейсы

Включение

Включение (агрегирование) – один класс является полем другого

Вложение – класс определяется внутри другого класса (пр. итератор)

Классы:

- Библиотека и книга
- Университет и студент
- Студент и Дата рождения
- Книга и автор

Наследование

- Наследование — это механизм, позволяющий создавать производные классы, расширяя уже существующие.
- Производный = дочерний класс
- Базовый = родительский класс
- Дочерний класс наследует все члены базового класса и содержит собственные члены (поля и методы…).

Наследование

```
struct Person {
    string name() const { return name_; }
    int      age() const { return age_; }
private:
    string name_;
    int age_;
};

struct Student : Person {
    string university() const { return uni_; }
private:
    string uni_;
};
```

Наследование

- Объекты класса-наследника могут присваиваться объектам родительского класса:

```
Student s("Alex", 21, "Oxford");
Person p = s; // Person("Alex", 21);
```

При этом копируются только поля класса-родителя (срезка).
(Т.е. в данном случае вызывается конструктор копирования Person(Person const& p), который не знает про uni_.)

Наследование

Классы:

- Character (Персонаж),
- LongRange (Персонаж дальнего действия),
- Wizard (Маг),
- SwordsMan (Мечник),
- Archer (Лучник).

Как связаны классы?

- Море, рыба
- Волк, хищник
- Стол, крупногабаритная мебель, мебель
- Посуда, инструмент, вещь
- Оружие, молот, инструмент
- Магазин, отдел, товар
- Библиотека, читатель, книга
- ! Фигура, прямоугольник, квадрат
- Фигура, треугольник, равносторонний треугольник, призма
- Фигура, треугольник, равносторонний треугольник, призма, точка
  - Liskov Substitution Principle (LSP): Функции, работающие с базовым классом, должны иметь возможность работать с подклассами не зная об этом

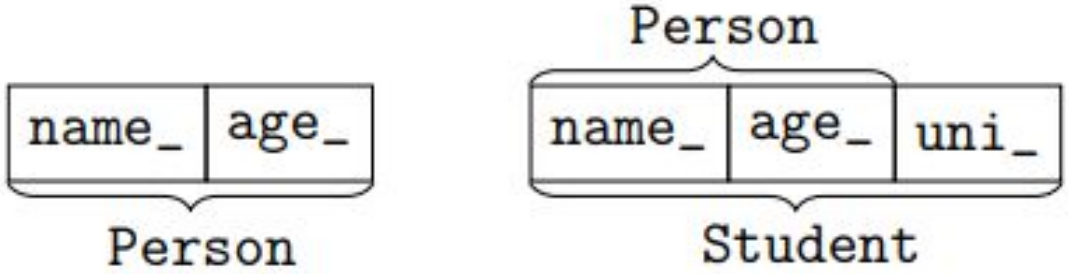
Наследование

- Модификаторы доступа
  - public
  - private
  - protected
- Модификаторы (тип) наследования
  - struct Derived : <modifier> Base { };

Наследование

- Порядок вызова конструкторов и деструкторов и создании и удалении объектов производного класса

Внутри объекта класса-наследника хранится экземпляр родительского класса.



Вызов не переопределенных методов базового класса

- Они наследуются, поэтому вызываются так, как будто определены в дочернем классе:

```
float Prism::SurfaceArea(){
    return Area() * 2 + Perimeter() * height;
}
```

Явный вызов конструктора базового класса

```
#include "IsoscelesTriangle.h"
class Prism :
    public IsoscelesTriangle{….};

Prism::Prism(float s, float a, float h) :IsoscelesTriangle(s,a)
{
    this->height = h;
}
```

Переопределение методов (overriding)

```
struct Person {
    string name() const { return name_; }
    ...
};
struct Professor : Person {
    string name() const {
        return "Prof. " + Person::name();
    }
    ...
};
```

```
    Professor pr("Stroustrup");
    cout << pr.name() << endl; // Prof. Stroustrup
    Person * p = &pr;
    cout << p->name() << endl; // Stroustrup
```

Один метод перекрывает другой

Виртуальные функции

```
struct Person {
    virtual string name() const { return name_; }
    ...
};
struct Professor : Person {
    string name() const {
        return "Prof. " + Person::name();
    }
    ...
};
```

```
    Professor pr("Stroustrup");
    cout << pr.name() << endl; // Prof. Stroustrup
    Person * p = &pr;
    cout << p->name() << endl; // Prof. Stroustrup
```

Полиморфизм

- -это возможность единообразно обрабатывать разные типы данных.
- Перегрузка функций Выбор функции происходит в момент компиляции на основе типов аргументов функции, **статический** полиморфизм.
- Виртуальные методы Выбор метода происходит в момент выполнения на основе типа объекта, у которого вызывается виртуальный метод, **динамический** полиморфизм.
- примеры

- Можно ли реализовать концепцию полиморфизма используя переопределение методов?
- Можно ли реализовать концепцию полиморфизма используя виртуальные методы?
- Какой метод лучше для классов связанных операцией включения и почему?
- Какой метод лучше для классов связанных операцией наследования и почему?

Вызов переопределенных методов базового класса

- Нужно уточнение при вызове, иначе будет рекурсивный вызов.

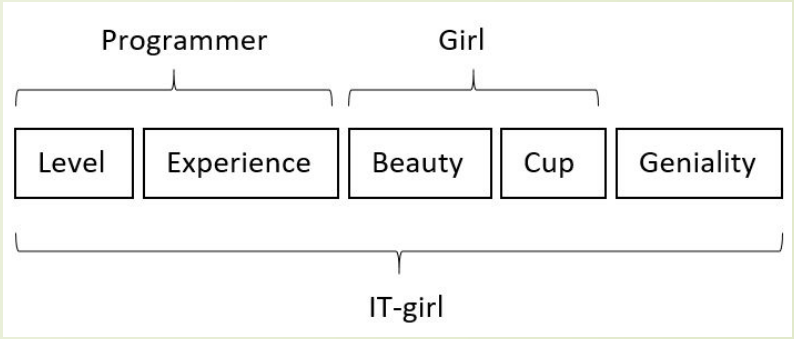
```
void Prism::Print(){
    IsoscelesTriangle::Print();
    cout << "Additional properties for prism" << endl;
    cout << "height of prism:" << height << endl;
    cout << "volume of prism:" << Volume() << endl;
    cout << "surface area of prism:" << SurfaceArea() << endl;
}
```

Множественное наследование

- Девушка-программист как морская свинка: не имеет отношения ни к морю, ни к свиньям.

Классы:

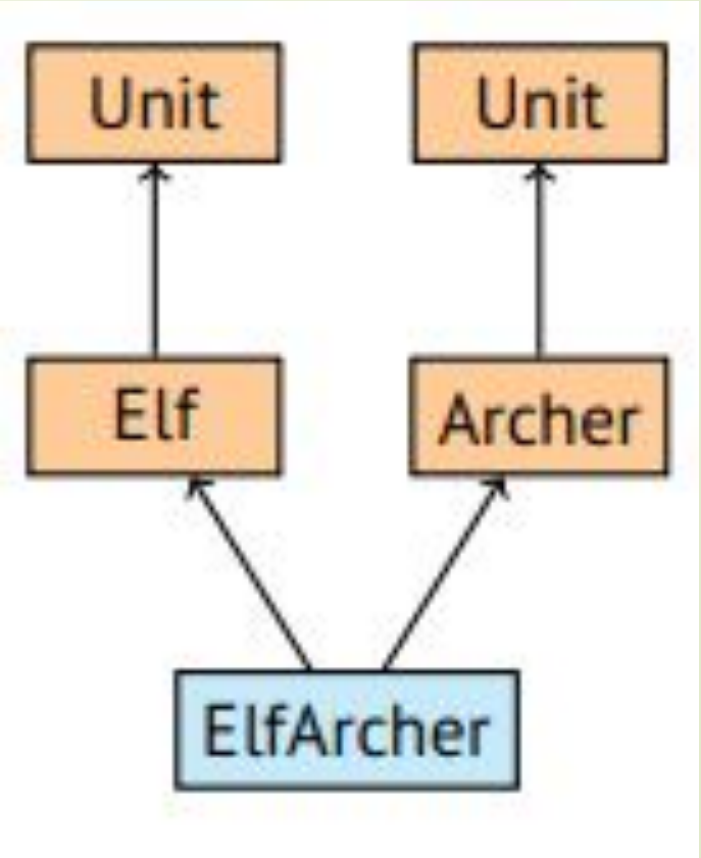
- Девушка
- Программист
- Девушка-программист



- Рекомендуется использовать интерфейсы

Множественное наследование

- Возможные проблемы:
- коллизия имен
- общие предки (дублирование)



Чистые виртуальные функции

- **Чистая виртуальная функция** (pure virtual function)- не имеют тела, их определяют только дочерние классы.
- При создании чистой виртуальной функции, вместо определения (написания тела) виртуальной функции, ей просто присваиваем ей значение 0.
- Пример

- Пример с обобщенными методами

Шаблон интерфейсного класса-коллекции может выглядеть следующим образом:

```
template <typename T>
class ICollect
{
protected:
    virtual ~ICollect() = default;

public:
    virtual ICollect<T>* Clone() const = 0;
    virtual void Delete() = 0;

    virtual bool IsEmpty() const = 0;
    virtual int GetCount() const = 0;
    virtual T& GetItem(int ind) = 0;
    virtual const T& GetItem(int ind) const = 0;
};
```

Чистые виртуальные функции. Пример

```
class  Figure
{
public:
    Figure();
    virtual void Print() = 0; //вывод полной информации об объекте : точки, площадь, периметр...
    ~Figure();
};
```

Абстрактные классы

- **Абстрактные классы** - это **классы**, которые содержат или наследуют без переопределения хотя бы одну чистую виртуальную функцию.

- ?
- Является ли класс Figure абстрактным?
- Можно ли создавать экземпляры этого класса?
- Как же тогда создать массив объектов этого класса?

Абстрактные классы

- Можно создать массив указателей на объекты, например так:

LinkedList<Figure*> l;

- Можно создать объекты дочерних классов и добавить указатели на них:

```
in >> s; in >> a;
IsoscelesTriangle * t1 = new IsoscelesTriangle(s, a);
l.PushBack(t1);
```

- Почему так можно?
- Как реализовать и вызвать полиморфный метод печати?

Приведение типов

- Необходимо пройти по контейнеру элементов базового класса и определять какого типа очередной класс:

```
prism = dynamic_cast<Prism*>(temp);
    if (prism) cout << "prism = type of the object" << endl;
```

И можно вызывать методы класса Prism:

```
    prism->SurfaceArea()
```

Интерфейсы

- **Интерфейс** — это класс, который не имеет переменных-членов и все методы которого являются чистыми виртуальными функциями!
- Они определяют поведение класса

- Пример: слайд 21

- Частый вопрос на собеседованиях

Расскажите об ООП на английском языке

Самостоятельная работа