

Программирование на современном C++

Объектно-ориентированное программирование.



Отметьтесь на портале!

- Посещение необязательное, но тем, кто пришёл, следует отмечаться на портале в начале каждого занятия
- Это позволяет нам анализировать, какие занятия были более или менее интересны студентам, и менять курс в лучшую сторону
- Также это даст возможность вам оставить обратную связь по занятию после его завершения


пн, 1 ноября	вт, 2 ноября	ср, 3 ноября	чт, 4 ноября	пт, 5 ноября	сб, 6 ноября
Нет занятий	<div>18:00 Углубленный C/C++ (w... п</div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div> <div>18:00 Углубленный C/C++ (ML) п</div> <div>Обработка исключительных ситуаций. Шаблоны классов и методов. Обобщенное и безопасное программирование</div> <div>А. Халайджи</div>	Нет занятий	Нет занятий	Нет занятий	Нет занятий

Ссылка на Zoom РК сегодня в 18:00

Безопасность интернет-приложений (третий семестр)

Подключиться к конференции Zoom
<https://mailru.zoom.us/j/98586081818?pwd=eWxwSDdCbHhQNnNwQU5iblFvU2dTZz09>

Идентификатор конференции: 985 8608 1818
Код доступа: 785885

 Алексей Набережный 55 минут назад

★ 0 💬 0 ⬇ 0 ⬆

Запись прошлой лекции (+ что почитать перед РК)

Безопасность интернет-приложений (третий семестр)

Углублённый C/C++

Лекция 5

📍 Онлайн - ML

Отметьтесь, что вы пришли на занятие. Так вы улучшите свою посещаемость и вас увидит преподаватель в своём "Журнале посещений".

Отметиться

Оставьте отзыв о занятии и мы сможем улучшить учебный процесс.

Оставить отзыв

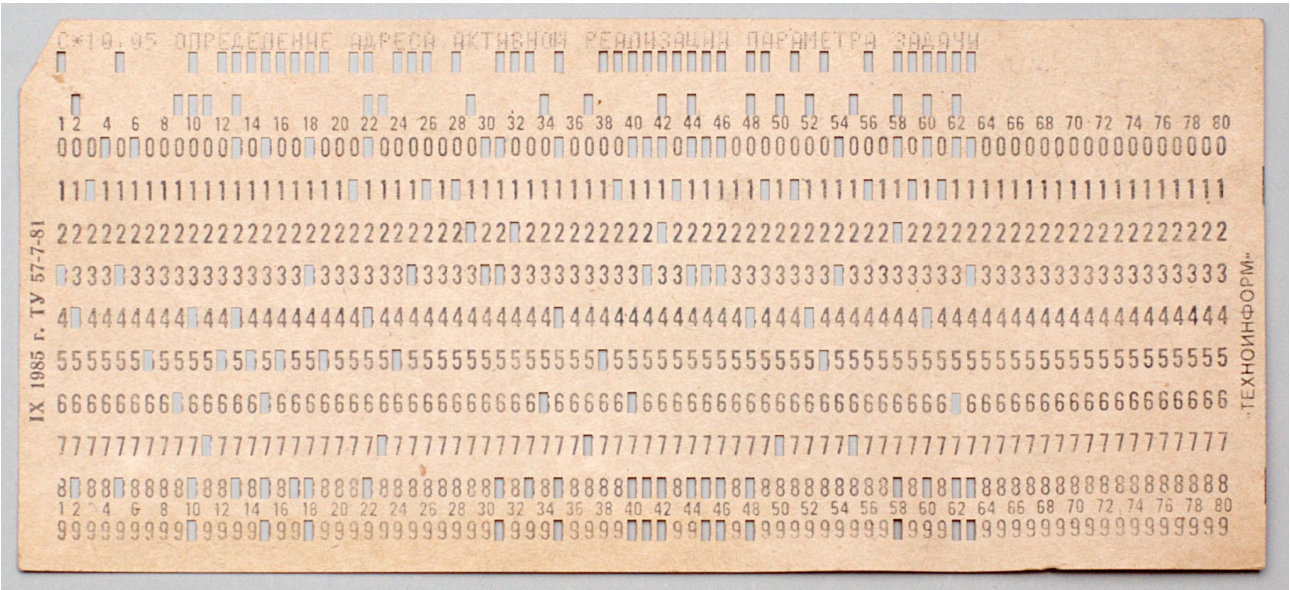
План лекции

- Развитие императивного программирования
- Инкапсуляция
- Перерыв
- Наследование
- Полиморфизм
- Приведение типов

Развитие императивного программирования

Машинные инструкции

КоП	Описание
000	запись числа
001	запись магазинная
002	обращение к спец. регистрам
003	считывание магазинное
004	арифметическое сложение
005	арифметическое вычитание
006	обратное вычитание
007	вычитание модулей
010	считывание числа
011	логическое умножение



Ассемблеры

МАШИННЫЕ КОМАНДЫ

Мнемоника в алфавитном порядке.

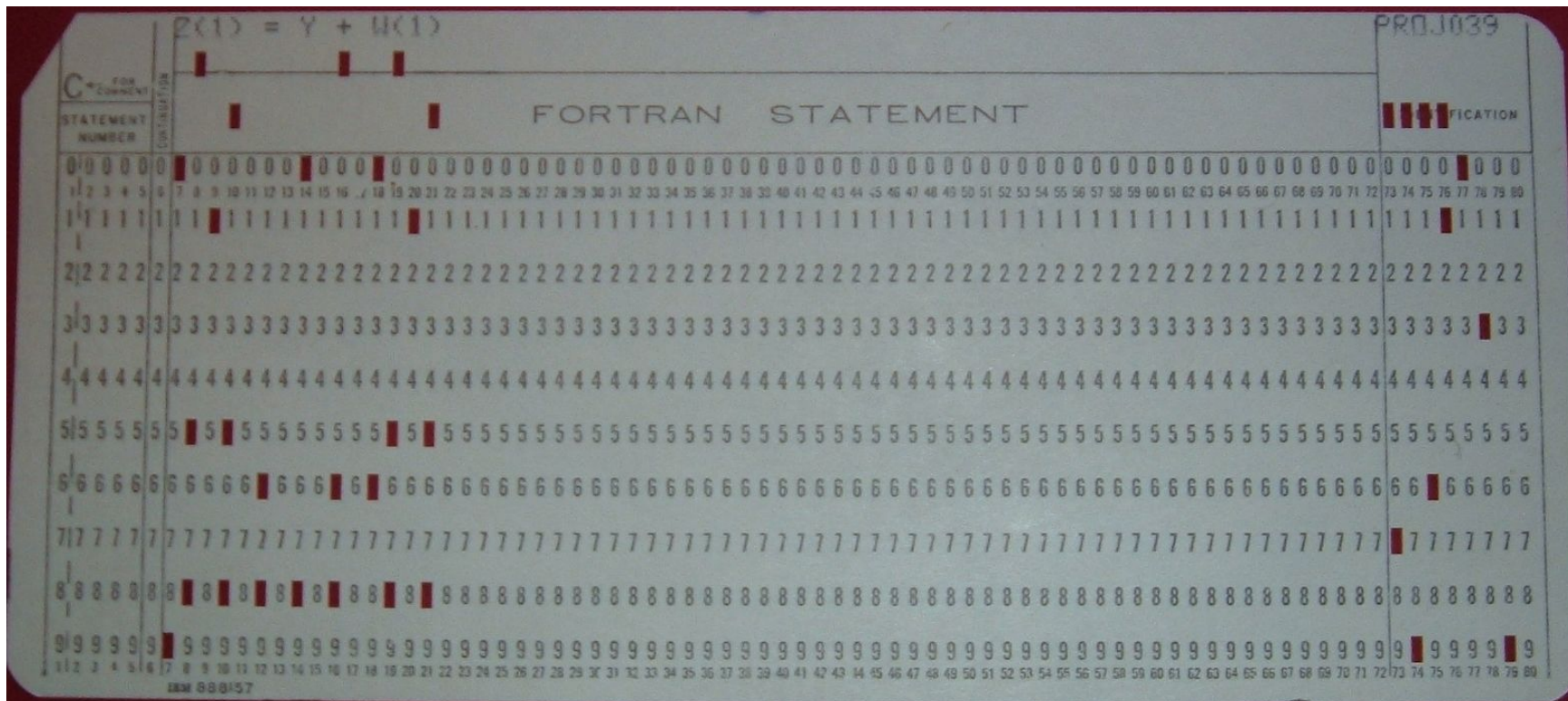
Буквой п отмечены привилегированные команды.

МНЕМО- НИКА	КОП	НАЗВАНИЕ
ВЧ	005	вычитание
ВЧАБ	007	вычитание абсолютных величин
ВЧОБ	006	вычитание обратное
ВЧП	025	вычитание порядков
ВЧПА	035	вычитание из порядка адреса
ВЫПР п	32	выход из прерывания
ДЕЛ	016	деление
ЗНАК	014	изменение знака числа
ЗП	000	запись
ЗПМ	001	запись в магазинном режиме
И	011	"И" логическое поразрядное
ИЛИ	015	"ИЛИ" логическое поразрядное

Высокоуровневые языки программирования

Fortran - первый язык программирования высокого уровня (1957).

Появляются, именованные переменные, подпрограммы и функции



Парадигмы программирования

Парадигма программирования -

идеи и понятия, определяющие стиль написания программ

Процедурное программирование

Программные инструкции собираются в подпрограммы, которые выполняют определенную задачу.

Процедурное программирование

Программные инструкции собираются в подпрограммы, которые выполняют определенную задачу.

Структурная парадигма

Теорема Бёма — Якопини (1966) доказательство того, что любой исполняемый алгоритм может быть преобразован к структурированному виду из последовательностей, ветвлений и циклов.

Процедурное программирование

Программные инструкции собираются в подпрограммы, которые выполняют определенную задачу.

Структурная парадигма

Теорема Бёма — Якопини (1966) доказательство того, что любой исполняемый алгоритм может быть преобразован к структурированному виду из последовательностей, ветвлений и циклов.



Go To Statement Considered Harmful - (1968) Эдсгер Вибе Дейкстра

Объектно-ориентированное программирование

Методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Объектно-ориентированное программирование

Методология программирования, основанная на представлении программы в виде совокупности взаимодействующих объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

- Более естественная декомпозиция программы.
- Удобство моделирования предметной области.

Принципы ООП

- Инкапсуляция
- Наследование
- Полиморфизм

Инкапсуляция

Объединение данных и методов, которые их обрабатывают.

```
char *strstr( const char *str, const char *substr );
```

```
class std::string {  
    size_type find( const char* s, size_type pos = 0 ) const;  
};
```

Инкапсуляция

Объединение данных и методов, которые их обрабатывают.

```
char *strstr( const char *str, const char *substr );
```

```
class std::string {  
    size_type find( const char* s, size_type pos = 0 ) const;  
};
```

```
char *pos = strstr(str, "123");
```

```
std::size_t pos = str.find("123");
```

Инкапсуляция

Состояние объекта - члены данных и их значения

Поведение объекта - методы и результат их вызова

```
class List {  
    void Push(...);  
    void Pop();  
    Node* head_;  
    size_t size_;  
};
```

Инкапсуляция - сокрытие

Разграничение доступа частей программы к внутренним компонентам друг друга.

Инкапсуляция - сокрытие

Разграничение доступа частей программы к внутренним компонентам друг друга.

Реализуется через модификаторы доступа:

- **private** доступны только для объектов этого класса
- **protected** доступны только для объектов этого класса и его наследников
- **public** доступны всем

Инкапсуляция - сокрытие

```
class List {  
public:  
    void Push(...);  
    void Pop();  
private:  
    Node* head_;  
    size_t size_;  
};
```


Инкапсуляция

- Скрытие деталей реализации (“Черный ящик”)
- Защита от приведения объекта в невалидное состояние
- Программа представляется как совокупность взаимодействующих друг с другом через интерфейс объектов

Методы

- Конструкторы - вызываются при создании объекта
- Деструкторы - вызываются при уничтожении объекта
- Операторы - позволяют переопределять поведение оператора
- Функции преобразования - позволяют добавлять явные/неявные преобразования

Явные/неявные Методы

- До C++11 конструкторы из одного параметра без модификатора `explicit` назывались конвертирующими конструкторами. (Конструктор с одним параметром скорее всего лучше сделать `explicit` явным)
- Функции преобразования лучше сделать явными

Явные/неявные Методы

- До C++11 конструкторы из одного параметра без модификатора `explicit` назывались конвертирующими конструкторами. (Конструктор с одним параметром скорее всего лучше сделать `explicit` явным)
- Функции преобразования лучше сделать явными

пример

Специальные методы

Компилятор может предоставить реализацию по-умолчанию

- Конструктор по-умолчанию
- Конструктор копирования
- Конструктор перемещения
- Оператор присваивания копированием
- Оператор присваивания перемещением
- Деструктор

Специальные методы

Компилятор неявно объявляет

Пользователь объявляет

	Конструктор по-умолчанию	Деструктор	Конструктор копирования	Присваивание копированием	Конструктор перемещения	Присваивание перемещением
Ничего	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию
Любой конструктор	не объявлен	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию
Конструктор по-умолчанию		по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию
Деструктор	по-умолчанию		по-умолчанию	по-умолчанию	не объявлен	не объявлен
Конструктор копирования	не объявлен	по-умолчанию		по-умолчанию	не объявлен	не объявлен
Присваивание копированием	по-умолчанию	по-умолчанию	по-умолчанию		не объявлен	не объявлен
Конструктор перемещения	не объявлен	по-умолчанию	удален	удален		не объявлен
Присваивание перемещением	по-умолчанию	по-умолчанию	удален	удален	не объявлен	

Специальные методы

Компилятор неявно объявляет

Пользователь объявляет

	Конструктор по-умолчанию	Деструктор	Конструктор копирования	Присваивание копированием	Конструктор перемещения	Присваивание перемещением
Ничего	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию
Любой конструктор	не объявлен	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию
Конструктор по-умолчанию		по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию
Деструктор	по-умолчанию		по-умолчанию	по-умолчанию	не объявлен	не объявлен
Конструктор копирования	не объявлен	по-умолчанию		по-умолчанию	не объявлен	не объявлен
Присваивание копированием	по-умолчанию	по-умолчанию	по-умолчанию		не объявлен	не объявлен
Конструктор перемещения	не объявлен	по-умолчанию	удален	удален		не объявлен
Присваивание перемещением	по-умолчанию	по-умолчанию	удален	удален	не объявлен	

правило 0

Специальные методы

Компилятор неявно объявляет

Пользователь объявляет

	Конструктор по-умолчанию	Деструктор	Конструктор копирования	Присваивание копированием	Конструктор перемещения	Присваивание перемещением	
Ничего	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	правило 0
Любой конструктор	не объявлен	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	
Конструктор по-умолчанию		по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	
Деструктор	по-умолчанию		по-умолчанию	по-умолчанию	не объявлен	не объявлен	правило 3
Конструктор копирования	не объявлен	по-умолчанию		по-умолчанию	не объявлен	не объявлен	
Присваивание копированием	по-умолчанию	по-умолчанию	по-умолчанию		не объявлен	не объявлен	
Конструктор перемещения	не объявлен	по-умолчанию	удален	удален		не объявлен	
Присваивание перемещением	по-умолчанию	по-умолчанию	удален	удален	не объявлен		

Специальные методы

Компилятор неявно объявляет

Пользователь объявляет

	Конструктор по-умолчанию	Деструктор	Конструктор копирования	Присваивание копированием	Конструктор перемещения	Присваивание перемещением	
Ничего	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	правило 0
Любой конструктор	не объявлен	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	
Конструктор по-умолчанию		по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	по-умолчанию	
Деструктор	по-умолчанию		по-умолчанию	по-умолчанию	не объявлен	не объявлен	правило 3
Конструктор копирования	не объявлен	по-умолчанию		по-умолчанию	не объявлен	не объявлен	
Присваивание копированием	по-умолчанию	по-умолчанию	по-умолчанию		не объявлен	не объявлен	
Конструктор перемещения	не объявлен	по-умолчанию	удален	удален		не объявлен	
Присваивание перемещением	по-умолчанию	по-умолчанию	удален	удален	не объявлен		

правило 5

Специальные методы

Пример

Перегрузка операторов

Можем перегружать различные операторы

```
struct Record {  
    std::string name;  
    std::string surname;  
    bool operator==(const Record& other) {  
        return name == other.name && surname == other.surname;  
    }  
};
```

Перегрузка операторов

Лучше руководствоваться здравым смыслом

Перегрузка операторов

Лучше руководствоваться здравым смыслом

```
po::options_description desc("Allowed options");  
desc.add_options()  
("help", "produce help message")  
("include-path,I", po::value< vector<string>>(), "include path")  
("input-file", po::value< vector<string>>(), "input file");
```

ПОТЕНЦИАЛЬНО ОПАСНЫЙ КОД

Перегрузка операторов

Лучше руководствоваться здравым смыслом

```
po::options_description desc("Allowed options");
desc.add_options()
("help", "produce help message")
("include-path,I", po::value< vector<string>>(), "include path")
("input-file", po::value< vector<string>>(), "input file");

std::filesystem::path path{"/usr"};
std::cout << std::filesystem::exists(path/"share"/"cmake") << std::endl;
```

АНАЛОГИЧНО: НЕ СЛИШКОМ БЕЗОПАСНЫЕ ВАРИАНТЫ
ПЕРЕГРУЗОК

Получение ресурса есть инициализация

RAII - Resource Acquisition Is Initialization - захват ресурса в конструкторе и освобождение в деструкторе.

Получение ресурса есть инициализация

RAII - Resource Acquisition Is Initialization - захват ресурса в конструкторе и освобождение в деструкторе.

пример

Получение ресурса есть инициализация

RAI - Resource Acquisition Is Initialization - захват ресурса в конструкторе и освобождение в деструкторе.

Java 7 try-with-resources:

```
try (FileReader fr = new FileReader(path);  
    BufferedReader br = new BufferedReader(fr)) {  
    return br.readLine();  
}
```

Python 2.5 with statement:

```
with open('file_path', 'w') as file:  
    file.write('hello world !')
```

Статические методы и члены данных

Статические методы не ассоциированы с каким-либо объектом (нет this)

Статические члены данных не ассоциированы с каким-либо объектом

Статические методы и члены данных

Статические методы не ассоциированы с каким-либо объектом (нет this)

Статические члены данных не ассоциированы с каким-либо объектом

```
class String {  
    static std::string FromInt(int i); -> auto s = String::FromInt(10);  
};
```

Статические методы и члены данных

Статические методы не ассоциированы с каким-либо объектом (нет this)

Статические члены данных не ассоциированы с каким-либо объектом

```
class String {  
    static std::string FromInt(int i); -> auto s = String::FromInt(10);  
};
```

```
auto s = std::to_string(10);
```


Статические методы и члены данных

Статические методы не ассоциированы с каким-либо объектом (нет this)

Статические члены данных не ассоциированы с каким-либо объектом

```
class String {  
    static std::string FromInt(int i); -> auto s = String::FromInt(10);  
};
```

НЕ НУЖДАЕМСЯ В СОЗДАНИИ
ОБЪЕКТА

```
auto s = std::to_string(10);
```

```
class String {  
    String() { total_string_alive += 1; }  
    ~String() { total_string_alive -= 1; }  
    inline static size_t total_string_alive = 0;  
};
```

Наследование

Класс может наследовать члены данных и методы другого класса.

Наследование

Класс может наследовать члены данных и методы другого класса.

- Переиспользование кода

Наследование

Класс может наследовать члены данных и методы другого класса.

- Переиспользование кода
- Легкое расширение функциональности

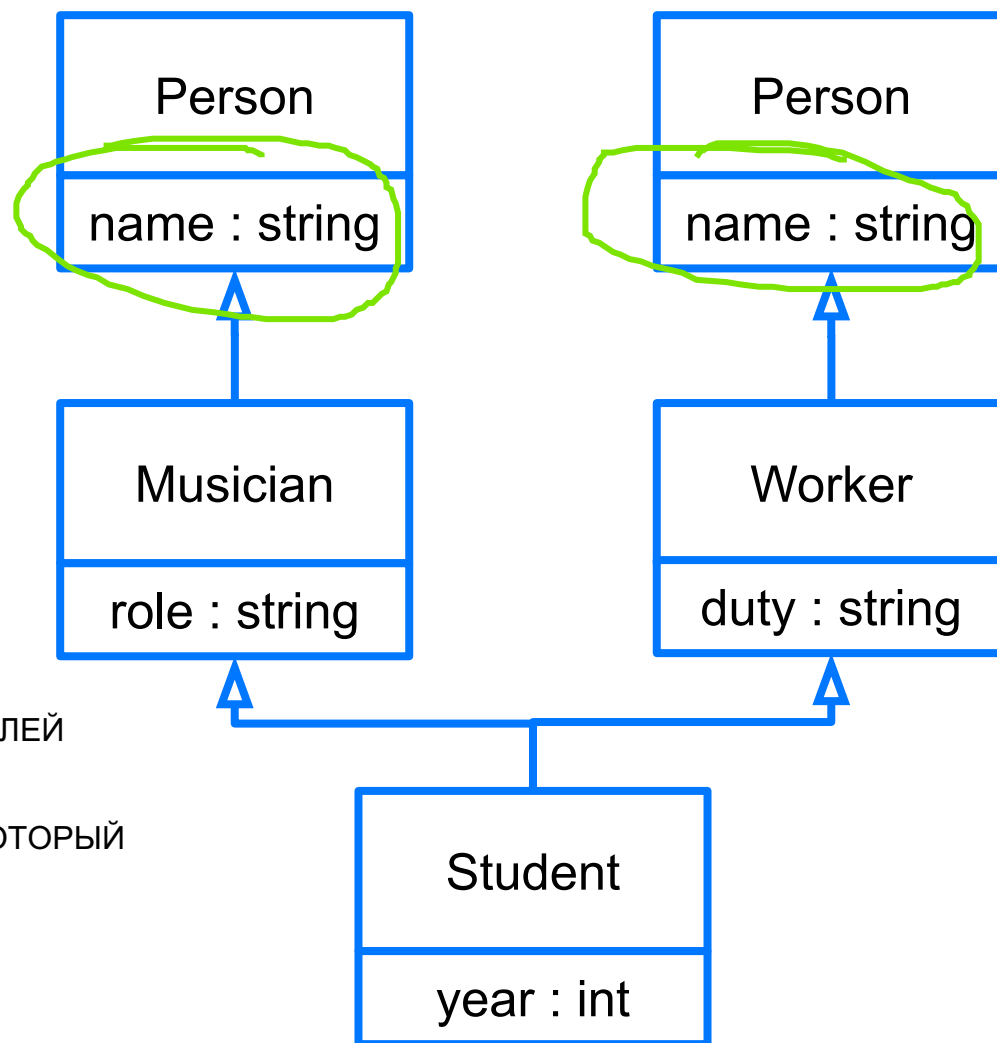
Наследование

Класс может наследовать члены данных и методы другого класса.

- Переиспользование кода
- Легкое расширение функциональности

Пример

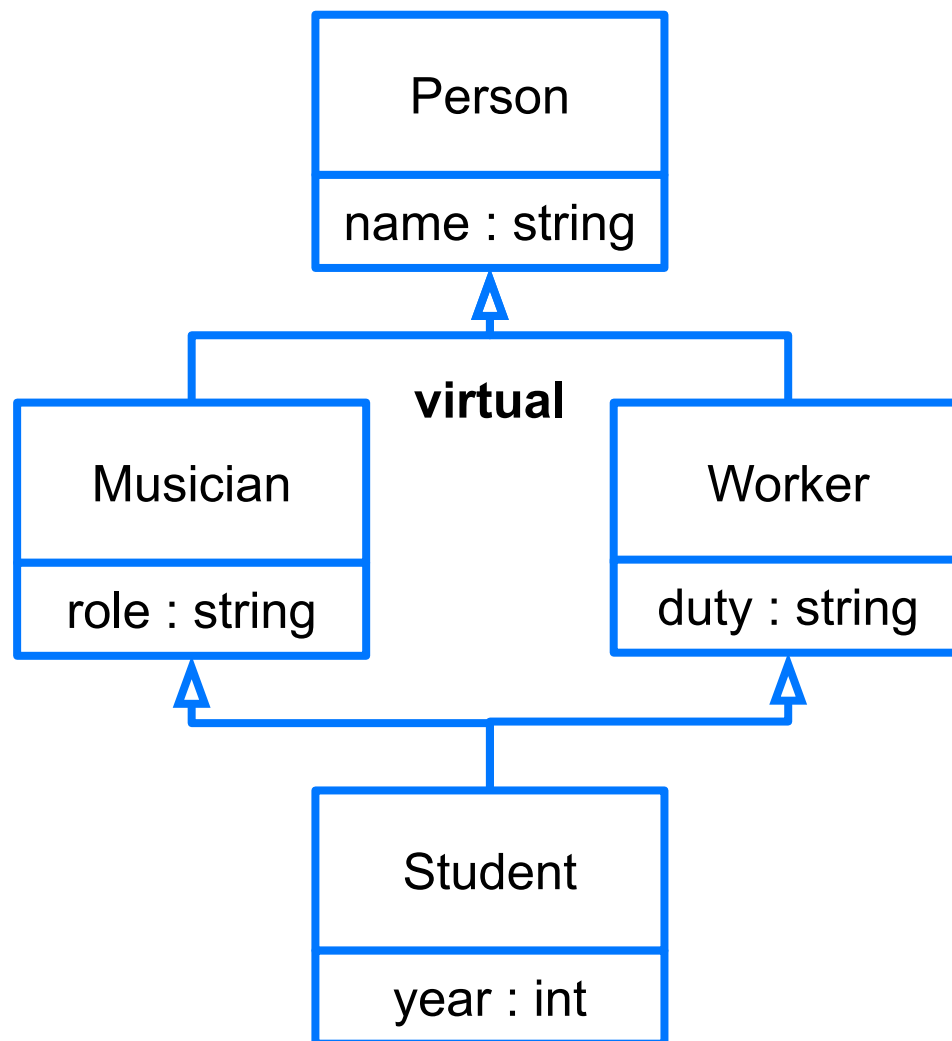
Множественное наследование



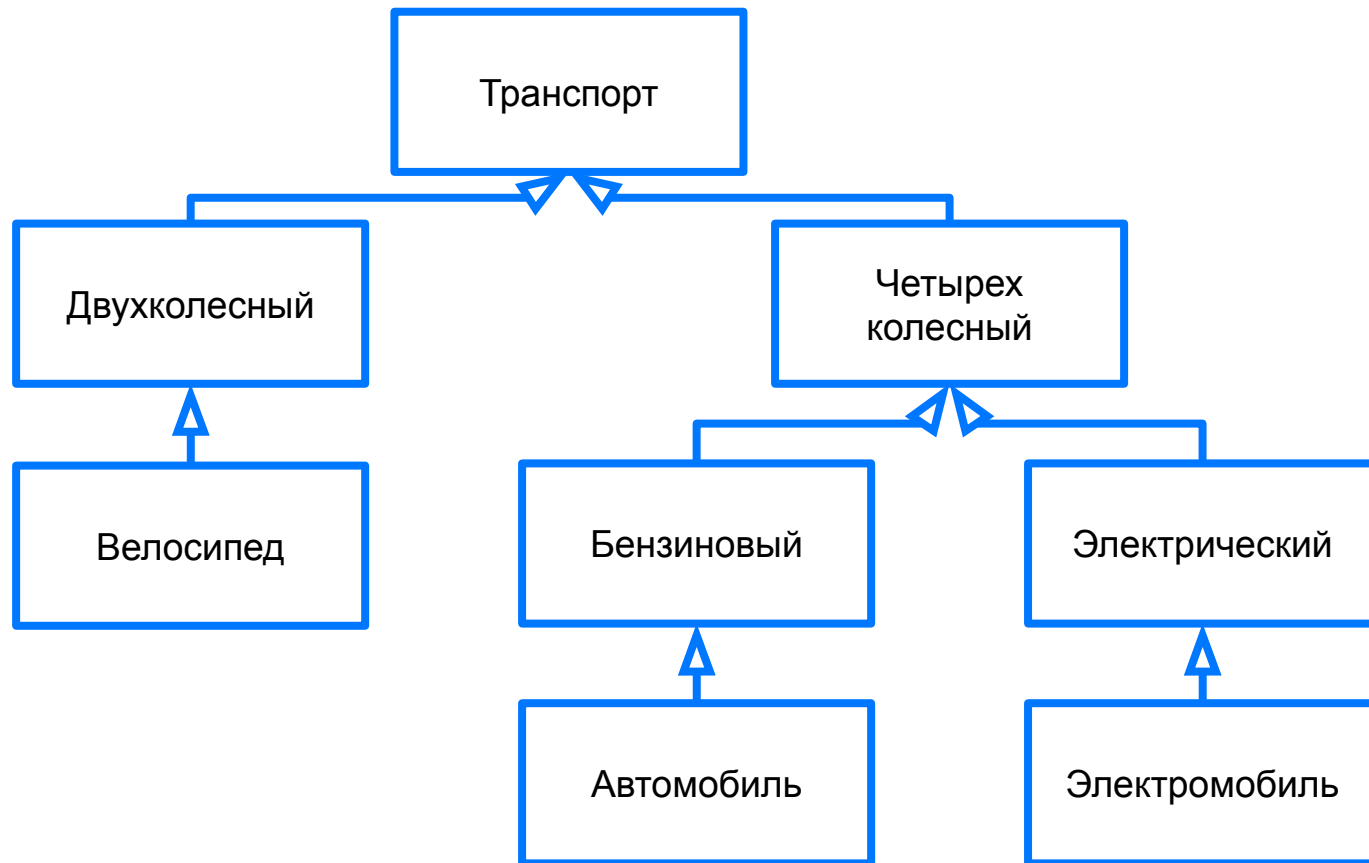
+ КОНФЛИКТ ПРИ ВОЗВРАЩЕНИИ ПОЛЕЙ
ЧЕРЕЗ
ОПРЕДЕЛЕННЫЕ МЕТОДЫ
+ ПРИ ИСПОЛЬЗОВАНИИ МЕТОДА, КОТОРЫЙ
ЕСТЬ У ОБОИХ РОДИТЕЛЕЙ

Множественное наследование

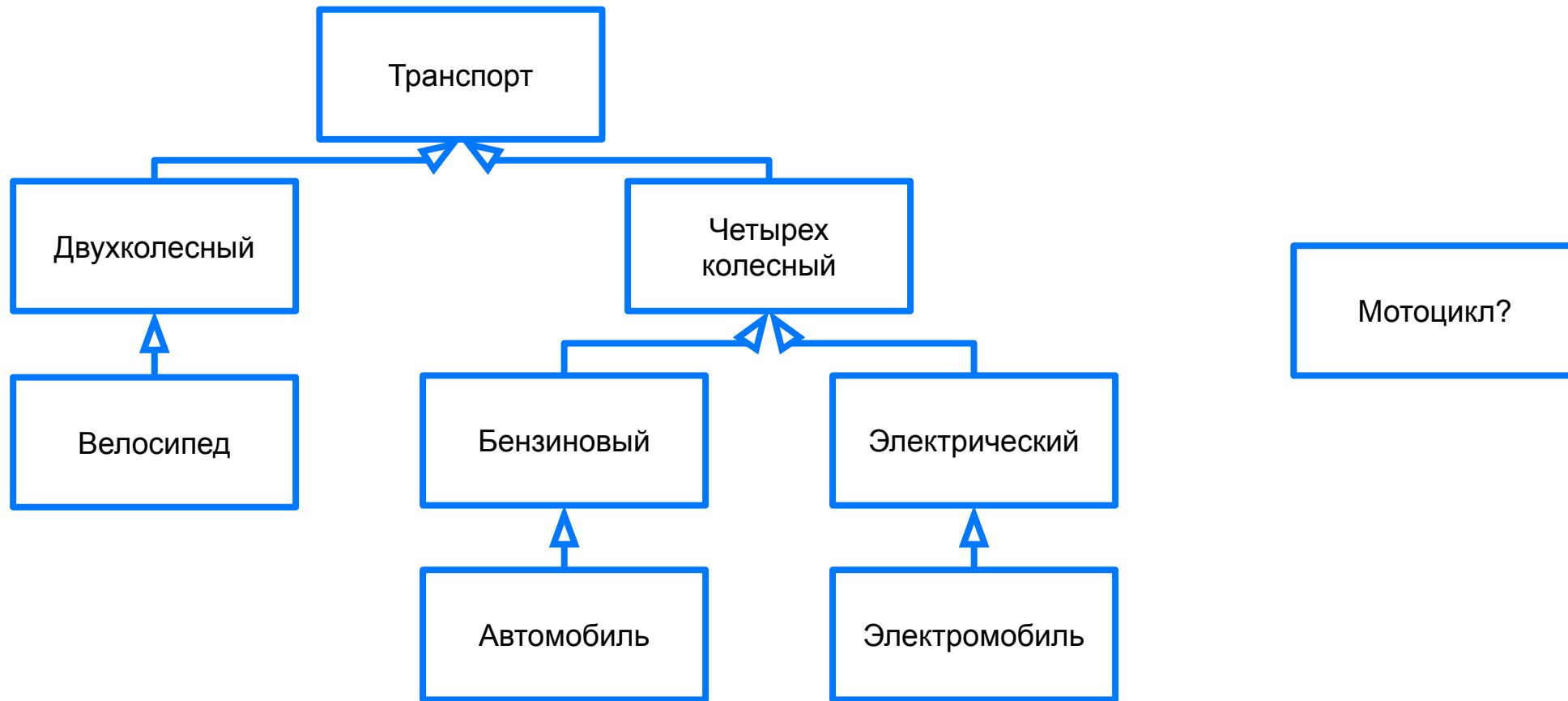
ПРОБЛЕМА РОМБА



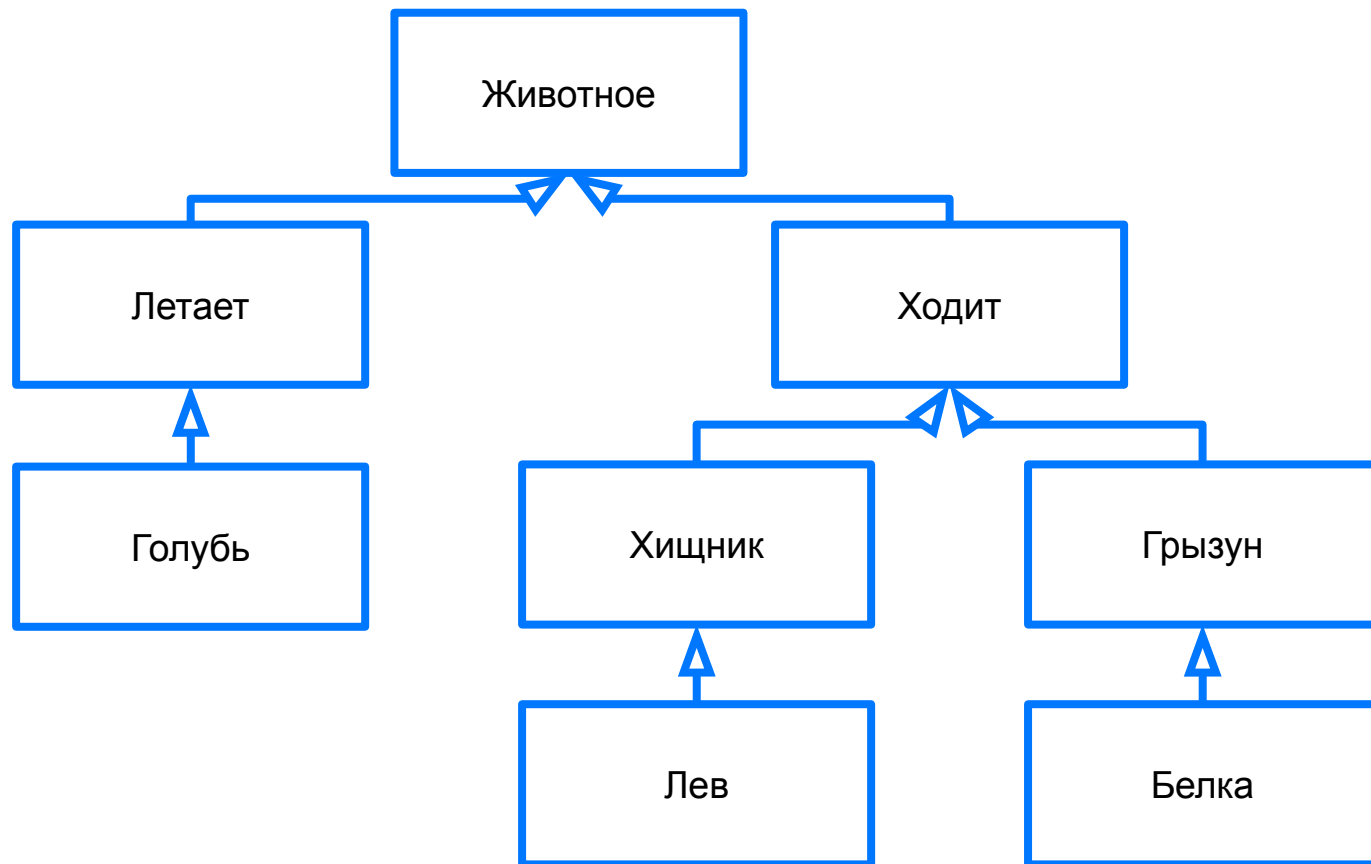
Иерархии наследования



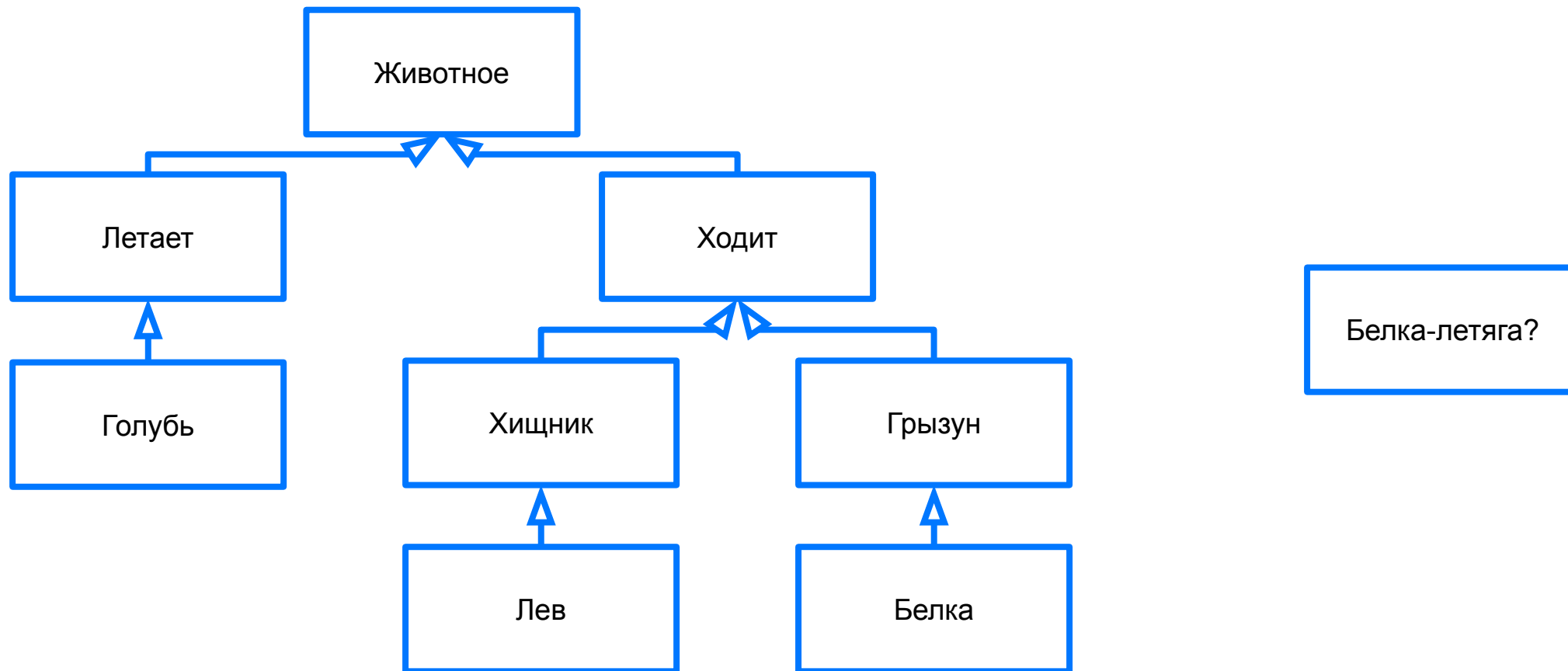
Иерархии наследования



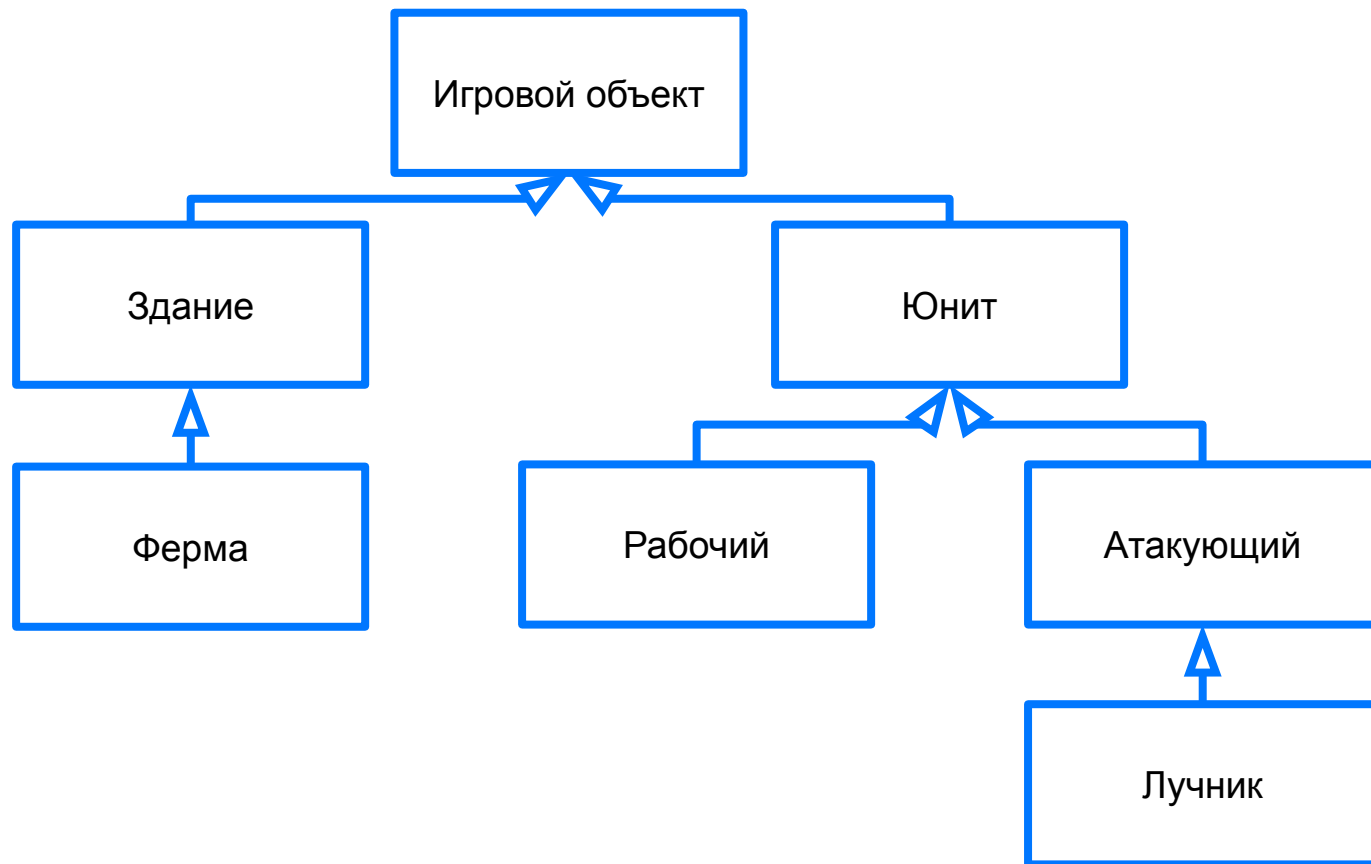
Иерархии наследования



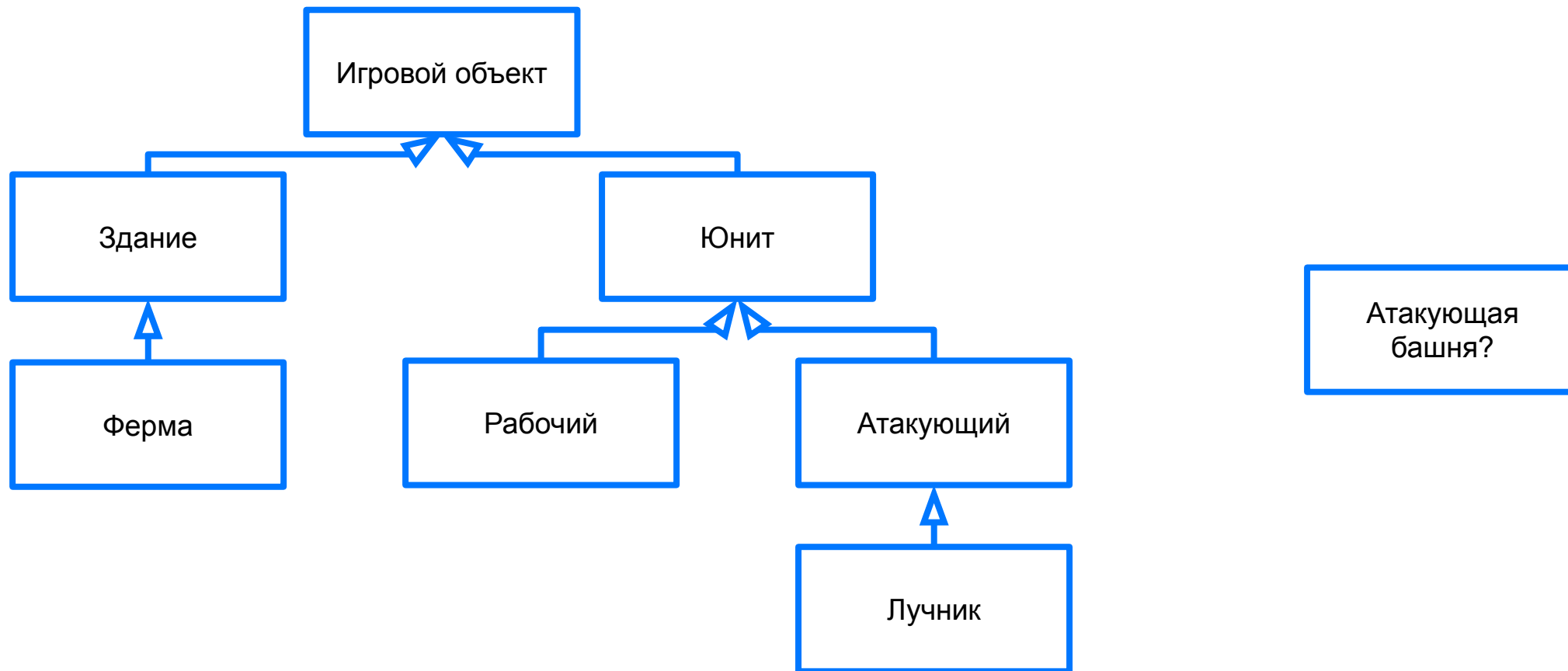
Иерархии наследования



Иерархии наследования



Иерархии наследования



Полиморфизм

Один интерфейс — много реализаций.

Возможность переопределить поведение метода в потомке.

Полиморфизм

Один интерфейс — много реализаций.

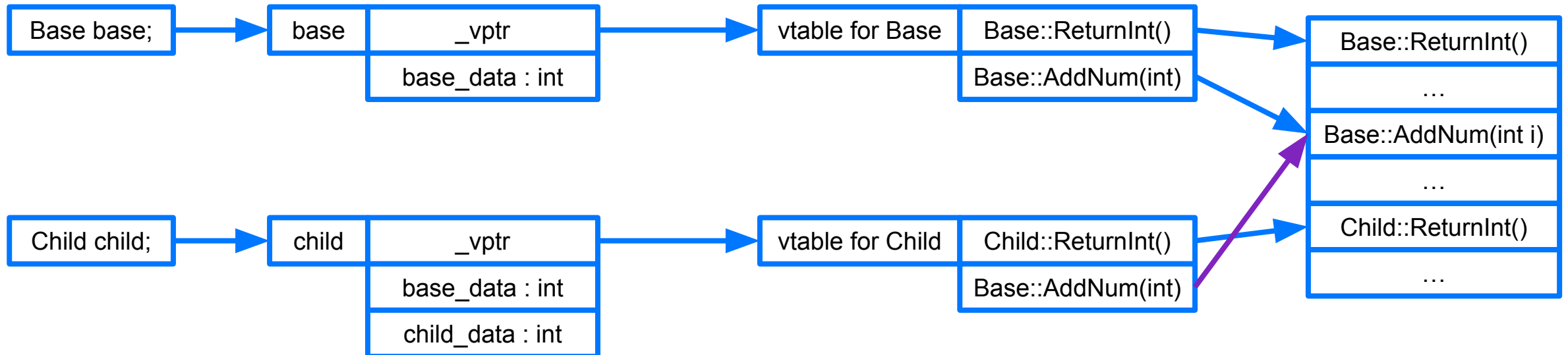
Возможность переопределить поведение метода в потомке.

```
class Base {  
    virtual void Print() { std::cout << "Base" << std::endl; }  
};  
  
class Child : public Base {  
    void Print() override { std::cout << "Child" << std::endl; }  
};
```

Полиморфизм

```
struct Base {  
    virtual int ReturnInt() { return 1; }  
    virtual int AddNum(int a) { return a + 10; }  
    int base_data;  
};
```

```
struct Child : Base {  
    int ReturnInt() override { return 2; }  
    int child_data;  
};
```



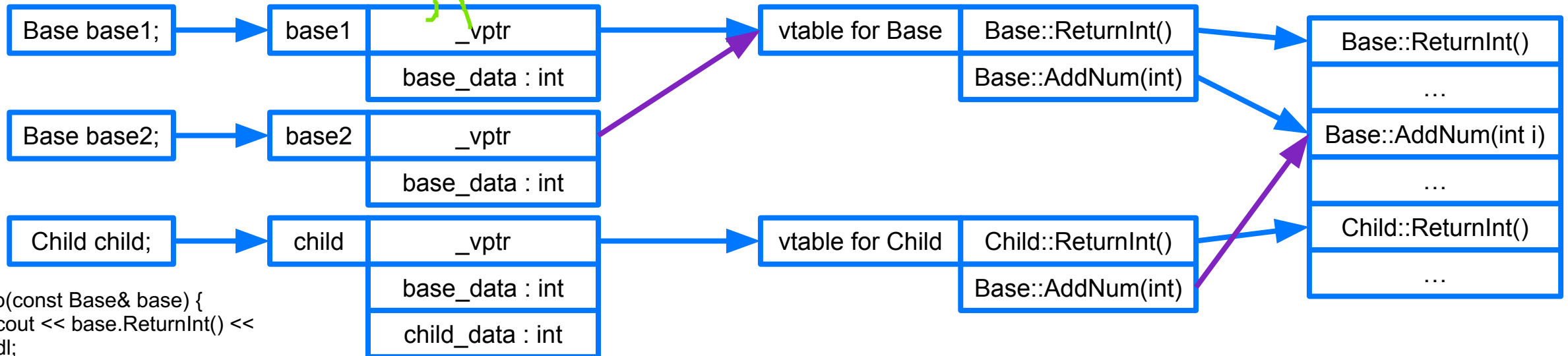
Полиморфизм

VERRIDE В НАСЛЕДНИКАХ - ПРОВЕРКА НА СОВПАДЕНИЕ СИГНАТУРЫ ПЕРЕОПРЕДЕЛЕННОЙ ФУНКЦИИ(ЕСЛИ НЕ СОВПАДАЕТ - ТО ОШИБКА, ПОЛУЧАЕТСЯ, МЫ НИЧЕГО НЕ ПЕРЕОПРЕДЕЛИЛИ)

```
struct Base {  
    virtual int ReturnInt() { return 1; }  
    virtual int AddNum(int a) { return a + 10; }  
    int base_data;  
};
```

если есть min 1 virtual метод
ptr на таблицу virtual методов

```
struct Child : Base {  
    int ReturnInt() override { return 2; }  
    int child_data;  
};
```



```
void foo(const Base& base) {  
    std::cout << base.ReturnInt() <<  
    std::endl;  
}  
Base b; Child c;  
foo(b); // 1  
foo(c); // 2
```

Виртуальный деструктор

Деструктор класса, который предполагает наследование должен быть публичным виртуальным или защищенным.

- При уничтожении объекта по указателю на базовый класс, публичный виртуальный деструктор корректно уничтожит объект.
- Если деструктор будет защищенным или приватным, компилятор не даст вызвать деструктор по указателю на базовый класс.

Виртуальный деструктор

Деструктор класса, который предполагает наследование должен быть публичным виртуальным или защищенным.

- При уничтожении объекта по указателю на базовый класс, публичный виртуальный деструктор корректно уничтожит объект.
- Если деструктор будет защищенным или приватным, компилятор не даст вызвать деструктор по указателю на базовый класс.

Пример

Абстрактные классы

Можно не предоставлять реализацию в базовом классе - чистая виртуальная функция.

Класс с хотя бы одной чистой виртуальной функцией называется абстрактным классом.

Невозможно создать объект абстрактного класса.

Абстрактные классы

Можно не предоставлять реализацию в базовом классе - чистая виртуальная функция.

Класс с хотя бы одной чистой виртуальной функцией называется абстрактным классом.

Невозможно создать объект абстрактного класса.

Пример

Интерфейсный класс

В C++ нет интерфейсов как в Java/C#.

Но функционально они очень похожи на абстрактный класс из только чистых виртуальных функций, без членов данных.

Интерфейсный класс

В C++ нет интерфейсов как в Java/C#.

Но функционально они очень похожи на абстрактный класс из только чистых виртуальных функций, без членов данных.

```
struct ISerializable() {  
    virtual ~ISerializable() {}  
    virtual std::string Serialize() = 0;  
};
```

Интерфейсный класс

В C++ нет интерфейсов как в Java/C#.

Но функционально они очень похожи на абстрактный класс из только чистых виртуальных функций, без членов данных.

```
struct ISerializable() {  
    virtual ~ISerializable() {}  
    virtual std::string Serialize() = 0;  
};
```

Такие классы не имеют минусов множественного наследования:

- Нет дублирования членов данных
- Нет неопределенности реализации методов

Интерфейсный класс

Интерфейс наиболее гибкий метод установления взаимоотношений между объектами.

Интерфейсный класс

Интерфейс наиболее гибкий метод установления взаимоотношений между объектами.

Интерфейс устанавливает обязательства иметь реализацию определенных методов
(Контрактное программирование)

Интерфейсный класс

Интерфейс наиболее гибкий метод установления взаимоотношений между объектами.

Интерфейс устанавливает обязательства иметь реализацию определенных методов
(Контрактное программирование)

Пример

Приведение ТИПОВ

Приведение типов

`static_cast` - сравнительно безопасный.

1. Downcast базовый класс в наследника
2. Функции преобразований и конвертирующие конструкторы
3. Числа в перечисление, перечисление в числа, перечисление в перечисление
4. `void *` в указатель на другой тип

Приведение типов

`dynamic_cast` - безопасный downcast родителя в потомка.

Если указатель не указывает на класс к которому приводим - вернется `nullptr`

Если ссылка не ссылается на класс к которому приводим - выбросится `std::bad_cast`

Приведение типов

`dynamic_cast` - безопасный downcast родителя в потомка.

Если указатель не указывает на класс к которому приводим - вернется `nullptr`

Если ссылка не ссылается на класс к которому приводим - выбросится `std::bad_cast`

`dynamic_cast` накладывает дополнительные расходы на проверку возможности сделать преобразование во время работы программы

Приведение типов

`dynamic_cast` - безопасный downcast родителя в потомка.

Если указатель не указывает на класс к которому приводим - вернется `nullptr`

Если ссылка не ссылается на класс к которому приводим - выбросится `std::bad_cast`

`dynamic_cast` накладывает дополнительные расходы на проверку возможности сделать преобразование во время работы программы

`dynamic_cast` работает только на полиморфных типах

Приведение типов

`reinterpret_cast` - самый небезопасный каст.

Прямое указание компилятору трактовать выражение, как имеющее другой тип.

Может преобразовывать указатели любого типа и целочисленные на указатели другого типа.

Приведение типов

`reinterpret_cast` - самый небезопасный каст.

Прямое указание компилятору трактовать выражение, как имеющее другой тип.

Может преобразовывать указатели любого типа и целочисленные на указатели другого типа.

Пример

Приведение типов

`const_cast` - небезопасный.

Позволяет добавлять/убирать `const/volatile` типа у ссылки или указателя.

Практика

Спасибо за внимание!

