Сложности при наследовании

ООП/C++: Лекция 4

Перегруженные методы, интерфейс и реализация, множественное наследование

ака «Сложности при наследовании»

О чём лекция сегодня

1. Пространства имён (namespaces)
2. Перегрузка при наследовании
3. Интерфейс и реализация
4. override и final
5. Множественное наследование

Код на сегодня

https://github.com/avasyukov/oop-2nd-term/tree/master/2019/lection04

Контест к лекции

http://judge2.vdi.mipt.ru/cgi-bin/new-client?contest_id=911140

Пространства имён (namespaces)

Пространства имён (namespaces)

- Это не специфично для ООП и не связано с наследованием
- Но полезно для организации большого объёма кода, а всё ООП во многом об этом

Namespaces: о чём и зачем это

- Проблема вида «не тот Вася»
- Разные люди пишут разные компоненты / библиотеки / модули
- Потом это всё нужно собрать в единый большой и страшный комплекс
- Как быть, если в двух компонентах уже есть class Engine, class Logger и т.д.

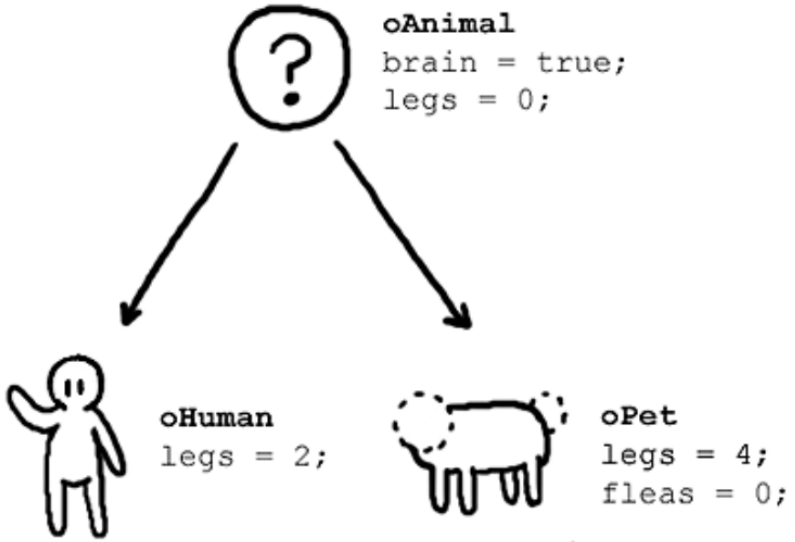
Namespaces: примеры

Разбираем примеры из 01_namespaces/

- 01_basic_namespaces.cpp
- 02_using_explained.cpp

Перегрузка при наследовании

Прошлая лекция



Перегрузка при наследовании

Как оно понимает, какую из реализаций нужно вызвать?

- Оно смотрит на тип указателя, через который инициирован вызов
- Иногда это может вызывать не то поведение класса, которое вы хотели

Перегрузка при наследовании: проблемы

Смотрим пример проблемы, заодно вспоминаем прошлую лекцию

- 01_overload_problem.cpp

Перегрузка при наследовании

И что же делать?

- Для решения этой проблемы нужны virtual функции
- Для virtual-функций явным образом строится таблица для поиска правильной реализации по всей цепочке наследования

Перегрузка при наследовании: примеры

Разбираем примеры из 02_virtual_functions_and_interfaces/

- 02_overload_problem_solved.cpp
- 03_overload_problem_completely_solved.cpp
- 04_destructor_problem.cpp
- 05_destructor_problem_solved.cpp

Перегрузка при наследовании: материалы

В этой области можно прилично закопаться. Если вдруг интересно, можно стартовать вот с этих ссылок:

- https://stackoverflow.com/questions/461203/when-to-use-virtual-destructors
- https://stackoverflow.com/questions/353817/should-every-class-have-a-virtual-destructor
- https://habrahabr.ru/post/64280/
- http://cpp-reference.ru/articles/virtual-destructor/

Интерфейс и реализация

Интерфейс и реализация

Технически в случае C++:

- интерфейс – абстрактный класс, у которого все методы виртуальные (задаёт, что должно быть, но сам это не реализует);
- реализация – какой-либо класс, унаследованный от интерфейса и реализующий все его виртуальные методы.

(В других языках программирования интерфейс может быть «классоподобной», но отдельной сущностью.)

Для чего придуманы интерфейсы?

Логически интерфейс – строгое задание требований, что должны уметь все сущности в определённой предметной области.

Для чего придуманы интерфейсы?

- Если вы пишите весь проект в одиночку – можно и без интерфейсов (скорее всего, базовые классы всё равно будут, но они вряд ли будут чисто абстрактными).
- Если вы работаете над большим проектом – интерфейс позволяет до начала разработки технически строго определить точки стыковки компонентов.

Для чего придуманы интерфейсы?

- Если в проекте вы пишите реализацию компонента – интерфейс позволяет быть уверенным, что вы ничего не забыли реализовать.
- Если в проекте вы используете компоненты – интерфейс позволяет заранее знать, что вам точно предоставят все реализации.
- Если интерфейс было решено изменить – сразу станут видны все места проекта, требующие доработки.

override и final

Директивы override и final

- Придуманы, чтобы контролировать потенциальные проблемы с наследованием / реализацией интерфейсов.
- override – «здесь мы точно хотим перегрузить базовый класс» (и оно не скомпилится, если мы никого не перегружаем).
- final – «эту штуку перегружать уже нельзя» (и попытка перегрузить не скомпилится).
- Нужно в длинных проектах для борьбы с трудно обнаруживаемыми ошибками.

Директивы override и final: пример

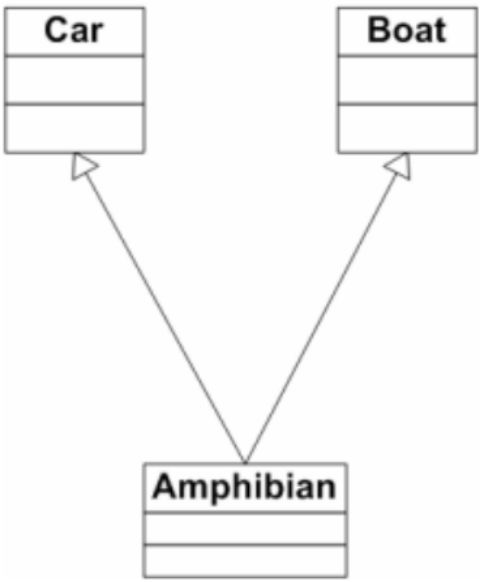
Разбираем пример из 02_virtual_functions_and_interfaces/

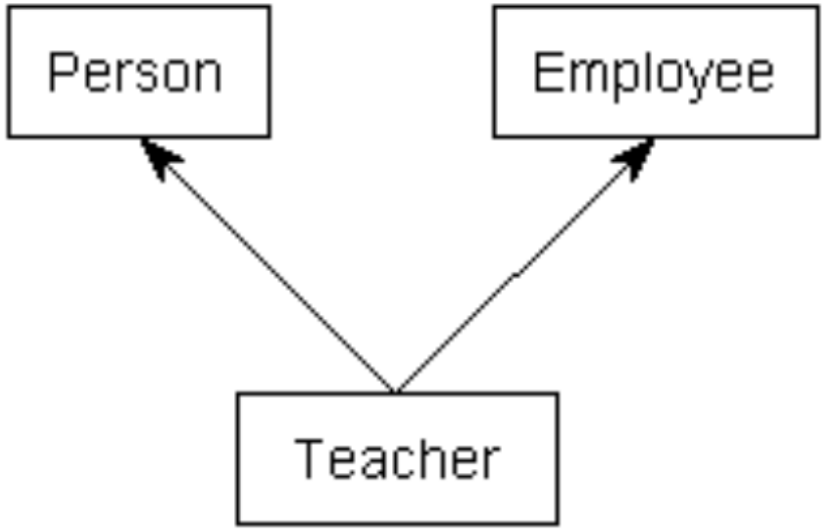
- 06_override_and_final.cpp

Множественное наследование

Множественное наследование

Иногда вы хотите, чтобы класс реализовал несколько интерфейсов. Это нормально.





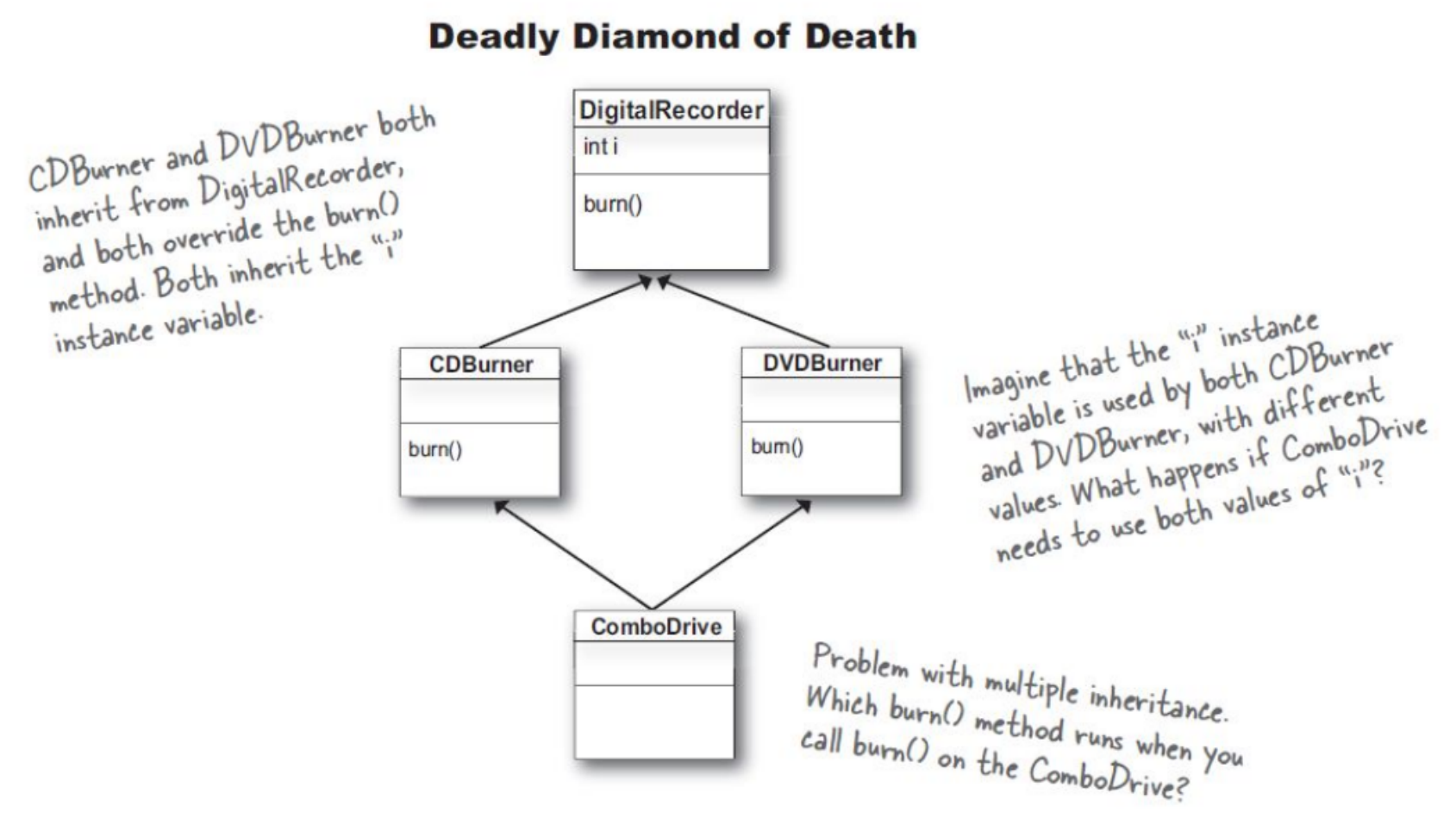
Множественное наследование: пример

Разбираем пример из 03_multiple_inheritance/

- 01_multiple_interfaces.cpp

Множественное наследование

Но иногда могут начаться проблемы.



Множественное наследование: проблемы

Желающие могут посмотреть примеры в
03_multiple_inheritance/

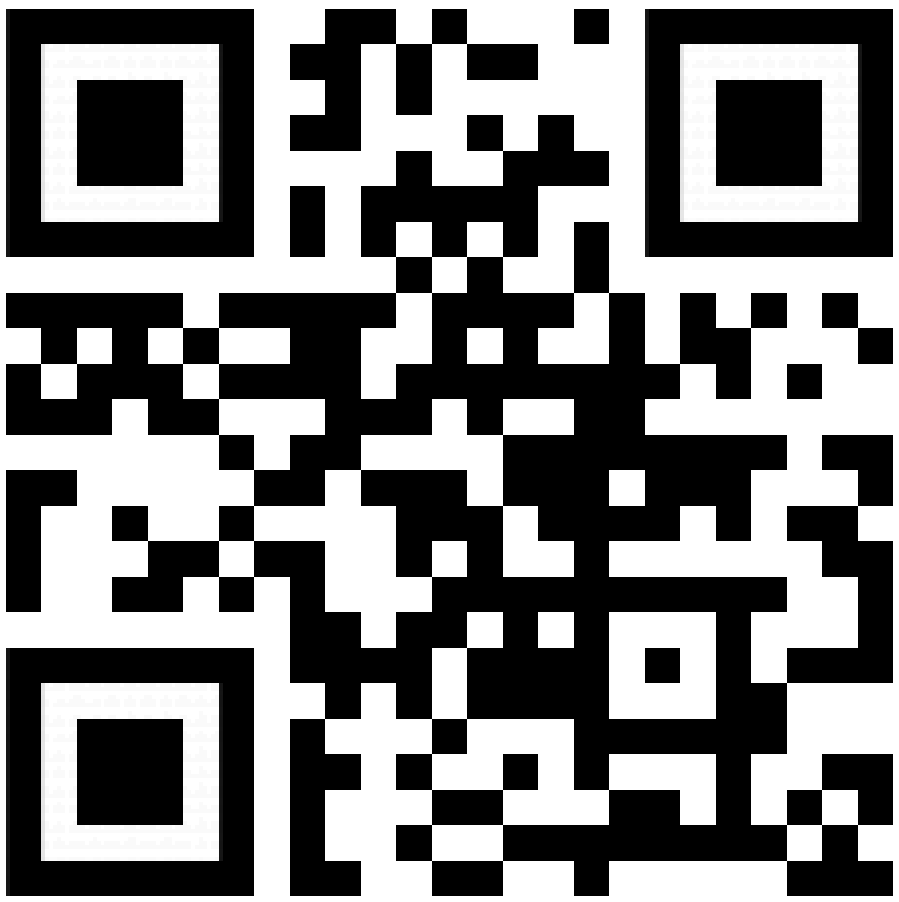
- 02_advanced_materials_diamond_problem.cpp
- 03_advanced_materials_diamond_problem_solved.cpp

Что стоит запомнить из лекции

- Для решения проблем вызова перегруженных методов нужны virtual-функции.
- Интерфейс – задание требований к сущностям в некоторой предметной области, технически строго определяет точки стыковки компонентов.
- Реализация нескольких интерфейсов – это полностью нормально. А вот наследование от нескольких неабстрактных классов может потребовать решать небанальные проблемы.

Мнение о лекции

https://tinyurl.com/y2fzkhoy



Multiple Inheritance