Лекция 9. Динамический полиморфизм. Часть 2.

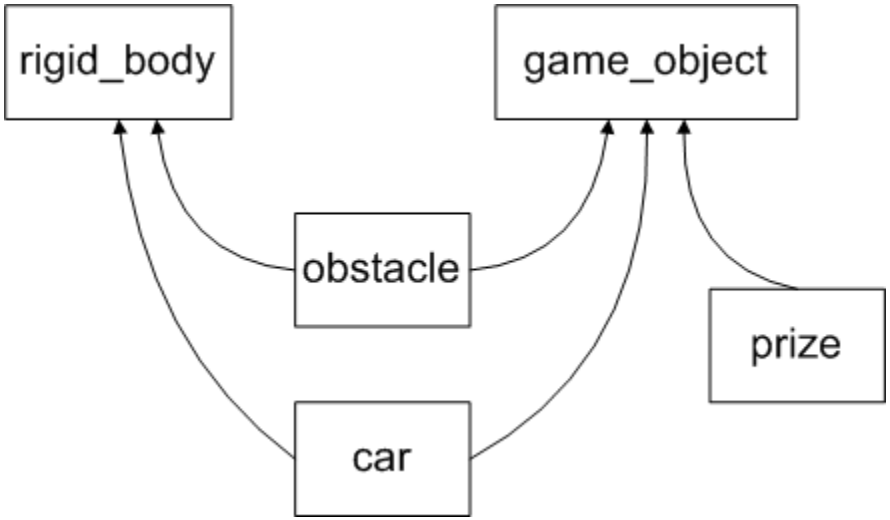
Множественное наследование и RTTI

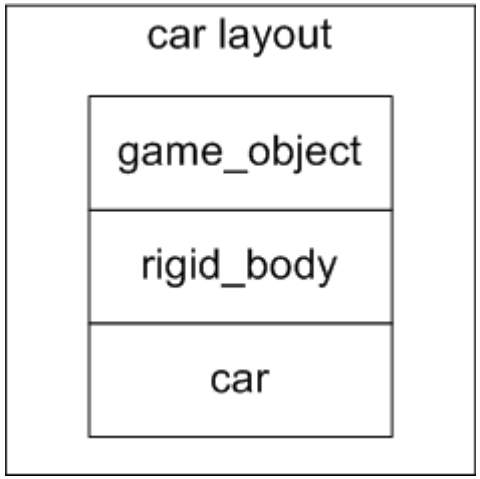
Аркадная игра «MiniMachines»

- **Описание**: по игровому полю проложена трасса. По ней едут машинки. Могут сталкиваться между собой и препятствиями (бочки, деревья). Могут собирать призы. Задача – приехать первым.

- Игровые объекты:
  - машинки,
  - препятствия,
  - призы (монетки).

Множественное наследование





Множественное наследование

```
1.  struct game_object { /*...*/ };
2.  struct rigid_body  { /*...*/ };
3.
4.  struct car
5.      : game_object
6.      , rigid_body
7.  {/*...*/
8.  };
9.
10. void hit   (rigid_body& rb);
11. void render(game_object const& go);
12.
13. ////////////////
14. car c;
15. hit(c);
16. //...
17. render(c);
```

Разрешение неоднозначности

```
1.  struct game_object { virtual point pos() const; /*...*/ };
2.  struct rigid_body  { virtual point pos() const; /*...*/ };
3.
4.  car c;
5.  auto p = c.pos(); // error: what pos?
6.
7.  ////////////////////////////////////
8.  struct car : game_object, rigid_body
9.  {
10.     point pos() const override
11.     {
12.         return rigid_body::pos();
13.     }
14. };
15.
16. auto p = c.pos(); // now ok!
```

- Поиск виртуальной функции происходит рекурсивно в базовых классах. Оказывается успешен, только если функция найдена ровно одна.

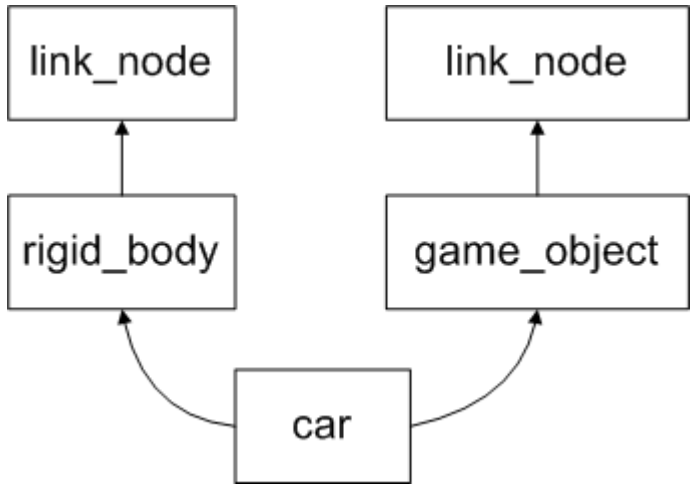
Сокрытие функций

```
1.  struct game_object { virtual point pos()         const; };
2.  struct rigid_body  { virtual point pos(polygon*) const; };
3.
4.  ////////////////////////////////////
5.  struct car : game_object, rigid_body
6.  {
7.      using rigid_body::pos;
8.
9.      point pos() const override
10.     {
11.         return game_object::pos();
12.     }
13. };
14.
15. polygon pol;
16.
17. auto p = c.pos(); // both functions
18. c.pos(&pol);      // available
```

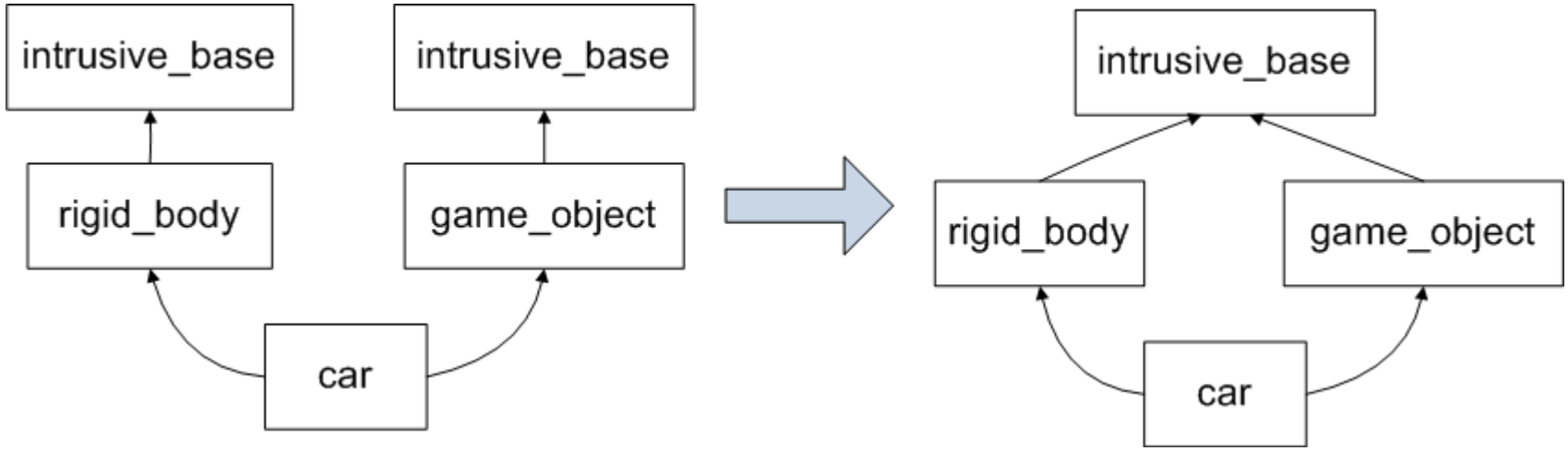
- using-объявление применимо только к членам базового класса
- Не удастся использовать using-директиву
- using-объявление не может увеличить уровень доступа

Повтор базового класса

```
1.  struct link_node
2.  {
3.      link_node* next;
4.  };
5.
6.  //...
7.  void car::process()
8.  {
9.      next = 0;                 // error
10.     link_node::next   = 0; // error
11.     game_object::next = 0; // ok!
12. }
```



Виртуальное наследование




```
1.  struct intrusive_base
2.  {
3.      size_t ref_counter_;
4.
5.      void add_ref() { ++ref_counter_; }
6.      void dec_ref() { if (--ref_counter_ == 0) delete this; }
7.  };
8.
9.  struct game_object : virtual intrusive_base {/*...*/};
10. struct rigid_body  : virtual intrusive_base {/*...*/};
11.
12. struct car : game_object, rigid_body {/*...*/};
```

Виртуальное наследование

- Конструктор вызывается из конечного класса иерархии

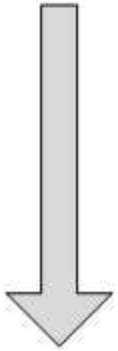
```
1.  struct A { A(int v) { cout << v << endl; } };
2.
3.  struct B : virtual A { B(int v) : A(v){} };
4.  struct C : virtual A { C(int v) : A(v){} };
5.
6.  struct D : B, C { D(int v) : A(v), B(0), C(1){} };
7.  struct E : D    { E()      : A(2), D(3){}       };
8.
9.  //
10. E e; // out: 2
```

- Виртуальное наследование используется крайне редко. Особый случай: использование при ромбовидном наследовании исключений. Без виртуального наследование не перехватить исключение по базовому классу*.

Управление доступом

Уровень доступа

всем	all
всем наследникам	all_der
непосредственным наследникам	dir_der
member'ам и друзьям	mem



member \ inheritance	public	protected	private
public	all	all_der	dir_der
protected	all_der	all_der	dir_der
private	mem	mem	mem

- Доступ к защищенным полям базового класса только для объектов своего типа

```
1.  struct interface { protected: void process() {/*...*/}};
2.
3.  struct uno : interface {/*...*/};
4.  struct due : interface
5.  {
6.      void foo(interface* other)
7.      {
8.          process();         // ok!
9.          other->process(); // error!
10.     }
11. };
```

Использование защищенных методов

```
1.  struct rigid_body
2.  {
3.      virtual point  pos () const = 0;
4.      virtual double mass() const = 0;
5.      /*...*/
6.  };
7.
8.  struct rigid_body_implulse_impl
9.      : rigid_body
10. {
11.     point pos() const override
12.     {
13.         /*...*/
14.     }
15.     /*...*/
16. protected:
17.     /*interface for derived classes*/
18.     double curr_torque() const {/*...*/}
19. };
20.
21. struct car : rigid_body_implulse_impl {};
```

Динамическое приведение типов

```
1.  struct collision_manager
2.  {
3.      void on_new_object(game_object* go)
4.      {
5.          if (auto ptr = dynamic_cast<rigid_body*>(go))
6.              add(*ptr);
7.      }
8.  };
```

- Проверяется в момент выполнения через информацию о типе из таблицы виртуальных функций.
- Можно применить только к полиморфному типу. Результат может быть неполиморфным.
- Если преобразование невозможно, вернет 0.
- Позволяет делать upcast, downcast, crosscast.
- Не снимает `const`, не приводит к `private` базе, не разрешает неоднозначность.
- `dynamic_cast<void*>` - начало объекта в памяти.

`static_cast` vs `dynamic_cast`

- `static_cast`: точно знаю, что приведение возможно, типы из одной ветки иерархии
- `static_cast` используем для неполиморфных типов (dynamic_cast неприменим). В том числе из `void*`
- `dynamic_cast` проверяет в процессе выполнения может ли быть приведение
- `dynamic_cast` позволяет делать crosscast

Оператор `typeid`

```
1.  const type_info& typeid(type_name) throw();
2.  const type_info& typeid(expression) throw(bad_typeid);
3.
4.  struct type_info
5.  {
6.      bool operator==(typeinfo const&) const;
7.      bool operator!=(typeinfo const&) const;
8.
9.      bool before(typeinfo const&) const;
10.     const char* name() const;
11. };
```

- Для полиморфных типов выдает тип объекта, а не указателя. Иначе определяет тип при компиляции.
- Если значение полиморфного указателя 0, генерирует исключение `bad_typeid`

Динамическое приведение ссылок*

```
1.  struct collision_manager
2.  {
3.      void on_new_car(game_object& go)
4.      {
5.          try
6.          {
7.              add(dynamic_cast<rigid_body&>(go));
8.              /*...*/
9.          }
10.         catch(std::bad_cast const& e)
11.         {
12.             cerr << e.what();
13.         }
14.     }
15. };
```

Вопросы?

Упражнение для самостоятельной работы

- Разработайте библиотеку создания мультиметодов