Лицензионное соглашение

Набросок игры в шахматы в терминах классов, иерархии и композиции

Дата публикации: 15.09.2010

Источник: BINED.RU :: В помощь студенту-программисту

(Файл исходного кода и исполняемый файл во вложениях к документу)

Задание:

Представьте, что вам необходимо реализовать игру в шахматы. Вам не следует в полной мере разрабатывать игру, но описать макет этой игры в терминах классов, иерархии и композиции. Например, есть класс фигур, для которых определены операции «ходить». В шахматах есть определенные фигуры с определенным правилом хождения. Фигуры могут друг друга съедать. Игра шахматы - это частный случай игр с шахматной доской, которые в свою очередь могут быть частным случаем игр на двух игроков и т.д.

Ответить на вопросы:

- Отношения между классами «содержит» представляет ..., а отношение «является» представляет ....
- Вызов функции, обрабатываемый во время компиляции, называется ... связыванием.
- Производный класс, полученный закрытым наследованием, не является подтипом базового класса?
- Подтип всегда равен подклассу?
- Выявите различия между статическим и динамическим связыванием. Объясните использование таблиц виртуальных методов.
- Сравните преимущества и недостатки композиции и наследования.
- В чем разница между замещением и уточнением.
- Как, по-вашему, должны работать виртуальные функции в конструкторе и деструкторе?

Решение:

```
#include <iostream>

// класс "шахматная фигура"
class t_figure
{
protected:
	int m_hor;		// цифра по горизонтали
	char m_vert;		// буква по вертикали
	int m_color;		// цвет...
public:
  t_figure(char vert, int hor, int color): m_vert(vert), m_hor(hor),
m_color(color) {
	}
	t_figure(t_figure &t): m_vert(t.m_vert), m_hor(t.m_hor), m_color(t.m_color) {
	}
	t_figure operator=(t_figure &t) {
		if(this == &t)
			return *this;
		m_vert = t.m_vert;
		m_hor = t.m_hor;
		m_color = t.m_color;
		return *this;
	}
	virtual int move() = 0;
	virtual int eat(t_figure &t) = 0;
};

// класс "ладья"
class t_castle: public virtual t_figure
{
public:
	t_castle(char vert, int hor, int color): t_figure(vert, hor, color) {
	}
	t_castle(t_castle &t): t_figure(t.m_vert, t.m_hor, t.m_color) {
	}
	t_castle operator=(t_castle &t) {
		if(this == &t)
			return *this;
		m_vert = t.m_vert;
		m_hor = t.m_hor;
		m_color = t.m_color;
		return *this;
	}
	// ход. 0 - ход сделать нельзя, 1 - ход сделан
	int move(char vert, int hor) {
		if( ((vert == m_vert) && (hor!= m_hor)) || ((vert!= m_vert) && (hor ==
m_hor)) ) {
		  m_hor =  hor;
		  m_vert = vert;
		  return 1;
	    }
```

```
            return 0;
        }
        int eat(t_figure &t) {
            // ...
        }
};

// класс "слон"
class t_bishop: public virtual t_figure
{
public:
        t_bishop(char vert, int hor, int color): t_figure(vert, hor, color) {
        }
        t_bishop(t_bishop &t): t_figure(t.m_vert, t.m_hor, t.m_color) {
        }
        t_bishop operator=(t_bishop &t) {
            if(this == &t)
                return *this;
            m_vert = t.m_vert;
            m_hor = t.m_hor;
            m_color = t.m_color;
            return *this;
        }
        int move(char vert, int hor) {
            if( abs((vert - m_vert) == abs(hor - m_hor)) && (vert!= m_vert) ) {
        m_vert = vert;
              hor = m_hor;
        return 1;
            }
            return 0;
        }
        int eat(t_figure &t) {
            // ...
        }
};

// класс "ферзь"
// ...однако ходить он умеет как слон и ладья :)
class t_queen: public t_bishop, public t_castle
{
public:
        t_queen(char vert, int hor, int color): t_castle(vert, hor, color),
                            t_bishop(vert, hor, color), t_figure(vert, hor,
color) {
        }
        t_queen(t_queen &t): t_castle(t.m_vert, t.m_hor, t.m_color),
                            t_bishop(t.m_vert, t.m_hor, t.m_color),
                            t_figure(t.m_vert, t.m_hor, t.m_color) {
        }
        t_queen operator=(t_queen &t) {
            if(this == &t)
                return *this;
            m_vert = t.m_vert;
```

```
            m_hor = t.m_hor;
            m_color = t.m_color;
            return *this;
        }
        int move(char vert, int hor) {
            return t_castle::move(vert, hor) || t_bishop::move(vert, hor);
        }
        int eat(t_figure &t) {
            // ...
        }
};

// описание классов других фигур...

// класс "доска"
class t_board
{
public:
        // доска представляет собой массив фигур...
        t_figure *figures[32];
};

// класс "игра" (какой-то вид игры с доской...шашки?)
class t_game
{
public:
        // запуск игры
        virtual void init(void) = 0;
};

// класс "игра в шахматы"
class t_chess: public t_game
{
protected:
        t_board board;
public:
        // запуск игры в шахматы
        virtual void init(void) = 0;
};

// класс "игра в шахматы на одного"
class t_chess1: public t_chess
{
public:
        // запуск игры в шахматы на одного
        void init(void) {
            // ...
        }
};

// класс "игра на двоих", наследует класс игры на одноого.
// добавляются поле данных время хода и метод слежения за временем хода.
// ...
class t_chess2: public t_chess1
{
```

```
private:
	unsigned m_pass_time;			// время хода
public:
	t_chess2(unsigned pass_time = 1200): m_pass_time(pass_time) {
	}
	void set_time(unsigned sec) {
		m_pass_time = sec;
	}
	// время хода вышло?
	int time_end(void) {
		// ...
	}
	// переопределенная функция запуска игры для двоих
	void init(void) {
		// ...
	}
};

int main(void)
{
	using namespace std;
	t_chess1 chess;
	chess.init();
	cout<<"It works :)\n";
	return 0;
}
```

Ответы:

1) Отношения между классами "содержит" представляет композицию, а отношение "является" представляет наследование.

2) Вызов функции, обрабатываемый во время компиляции, называется статическим связыванием (ранним связыванием).

3) Производный класс, полученный закрытым наследованием, не является подтипом базового класса?

Да, не является.

4) Подтип всегда равен подклассу?

Нет. Подкласс, в отличие от подтипа, не обязательно реализует интерфейс родительского класса.

5) Выявите различия между статическим и динамическим связыванием. Объясните использование таблиц виртуальных методов.

Статическое связывание реализуется во время компиляции. Следствием этого является вызов методов родительского класса, а не дочернего, не смотря на то что они (методы) перегружены в дочернем классе. Это случается, например, когда объект производного типа передается в функцию в качестве родительского типа. В данном случае внутри функции объект рассматривается как инстанция родительского класса, а не дочернего. Динамическое связывание позволяет обойти это ограничение путем связывания сообщения с получателем во время выполнения программы, используя таблицу

виртуальных функций. Каждый класс имеет таблицу виртуальных функций, которая содержит список перегруженных методов и ссылку на таблицу виртуальных функций родительского класса. Связывание происходит путем поиска подходящего метода, проходом по таблицам снизу-вверх.

6) Сравните преимущества и недостатки композиции и наследования.

- Композиция более проста. Ее преимущество заключается в том, что ясно видны какие точно операции будут выполняться. Т.е. мы смотрим на

  Set и видим, что для этого типа определены только операции добавить, проверка включения и определения размера. И не важно, какие операции определены для списков.

- При наследовании новый тип данных добавляет новое поведение. Чтобы знать все возможности типа

  Set, надо просмотреть и протокол класса ***List***.

- Краткость при создании новых типов с помощью наследования можно рассматривать и как преимущество. Нет необходимости писать весь код. Можно полностью использовать функции базового класса.
- Наследование не запрещает пользователям вызывать методы родительского класса, даже если эти методы не соответствуют идеологии потомка. Например, мы можем вызвать функцию

  addToFront, хотя и не стоит этого делать.

- При композиции трудно реализовать идею о том, что

  Set является лишь уточнением множества ***List***.

- Полиморфизм не свойственен композиции. Полиморфизм - это возможность принимать различные формы. В наследовании же полиморфизм- это мощное средство разработки программ.
- Реализация с помощью наследования имеет небольшие преимущества в качестве скорости выполнения.

Обе техники полезны при разработке программ. Вряд ли можно однозначно выбрать одну и назвать ее лучшей. Стоит поставить вопрос так: если в ответ на вопрос ***X*** является экземпляром ***Y***, вы отвечаете - Да, следует выбрать наследование. Если в ответ на вопрос ***X*** включает ***Y*** как часть, вы отвечаете - да, остановитесь на композиции.

7) В чем разница между замещением и уточнением.

Метод дочернего класса может *замещать* метод родительского класса, т.е. иметь совершенно новый код. Но метод дочернего класса может и *уточнять* метод базового класса, т.е. при выполнении метода дочернего класса выполняется и метод базового класса.

8) Как, по-вашему, должны работать виртуальные функции в конструкторе и деструкторе?

Да, будут. В конструкторах и деструкторах виртуальные методы работают как обыкновенные.