**Паттерн Command (команда)**

**Назначение паттерна Command**

Используйте паттерн Command если

* Система управляется событиями. При появлении такого события (запроса) необходимо выполнить определенную последовательность действий.
* Необходимо параметризировать объекты выполняемым действием, ставить запросы в очередь или поддерживать операции отмены (undo) и повтора (redo) действий.
* Нужен объектно-ориентированный аналог функции обратного вызова в процедурном программировании.

Пример событийно-управляемой системы – приложение с пользовательским интерфейсом. При выборе некоторого пункта меню пользователем вырабатывается запрос на выполнение определенного действия (например, открытия файла).

**Описание паттерна Command**

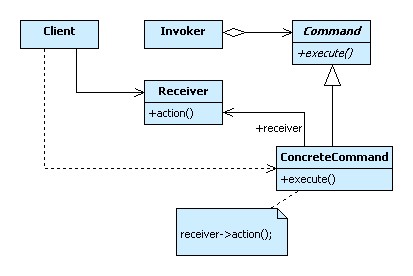
Паттерн Command преобразовывает запрос на выполнение действия в отдельный объект-команду. Такая инкапсуляция позволяет передавать эти действия другим функциям и объектам в качестве параметра, приказывая им выполнить запрошенную операцию. Команда – это объект, поэтому над ней допустимы любые операции, что и над объектом.

Интерфейс командного объекта определяется абстрактным базовым классом Command и в самом простом случае имеет единственный метод execute(). Производные классы определяют получателя запроса (указатель на объект-получатель) и необходимую для выполнения операцию (метод этого объекта). Метод execute() подклассов Command просто вызывает нужную операцию получателя.

В паттерне Command может быть до трех участников:

* Клиент, создающий экземпляр командного объекта.
* Инициатор запроса, использующий командный объект.
* Получатель запроса.

**UML-диаграмма классов паттерна Command**



Сначала клиент создает объект ConcreteCommand, конфигурируя его получателем запроса. Этот объект также доступен инициатору. Инициатор использует его при отправке запроса, вызывая метод execute(). Этот алгоритм напоминает работу функции обратного вызова в процедурном программировании – функция регистрируется, чтобы быть вызванной позднее.

Паттерн Command отделяет объект, инициирующий операцию, от объекта, который знает, как ее выполнить. Единственное, что должен знать инициатор, это как отправить команду. Это придает системе гибкость: позволяет осуществлять динамическую замену команд, использовать сложные составные команды, осуществлять отмену операций.

**Реализация паттерна Command**

Рассмотрим реализацию паттерна Command на примере игры «Шахматы». Имитируем возможность выполнения следующих операций:

* Создать новую игру.
* Открыть существующую игру.
* Сохранить игру.
* Сделать очередной ход.
* Отменить последний ход.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90  91  92  93  94  95  96  97  98  99 | #include<iostream>  #include<vector>  #include<string>    class Game  {    public:      void create( ) {        cout << "Create game " << endl;      }      void open( string file ) {        cout << "Open game from " << file << endl;      }      void save( string file ) {        cout << "Save game in " << file << endl;      }      void make\_move( string move ) {        cout << "Make move " << move << endl;      }  };    string getPlayerInput( string prompt ) {    string input;    cout << prompt;    cin >> input;    return input;  }    // Базовый класс  class Command  {    public:      virtual ~Command() {}      virtual void execute() = 0;    protected:      Command( Game\* p ): pgame( p) {}      Game \* pgame;  };    class CreateGameCommand: public Command  {    public:      CreateGameCommand( Game \* p ) : Command( p) {}      void execute() {        pgame->create( );      }  };    class OpenGameCommand: public Command  {    public:      OpenGameCommand( Game \* p ) : Command( p) {}      void execute() {        string file\_name;        file\_name = getPlayerInput( "Enter file name:");        pgame->open( file\_name);      }  };    class SaveGameCommand: public Command  {    public:      SaveGameCommand( Game \* p ) : Command( p) {}      void execute( ) {        string file\_name;        file\_name = getPlayerInput( "Enter file name:");        pgame->save( file\_name);      }  };    class MakeMoveCommand: public Command  {    public:      MakeMoveCommand( Game \* p) : Command( p) {}      void execute() {        // Сохраним игру для возможного последующего отката        pgame->save( "TEMP\_FILE");        string move;        move = getPlayerInput( "Enter your move:");        pgame->make\_move( move);      }  };    class UndoCommand: public Command  {    public:      UndoCommand( Game \* p ) : Command( p) {}      void execute() {        // Восстановим игру из временного файла        pgame->open( "TEMP\_FILE");      }  };      int main()  {    Game game;    // Имитация действий игрока    vector<Command\*> v;    // Создаем новую игру    v.push\_back( new CreateGameCommand( &game));    // Делаем несколько ходов    v.push\_back( new MakeMoveCommand( &game));    v.push\_back( new MakeMoveCommand( &game));    // Последний ход отменяем    v.push\_back( new UndoCommand( &game));    // Сохраняем игру    v.push\_back( new SaveGameCommand( &game));      for (size\_t i=0; i<v.size(); ++i)      v[i]->execute();      for (size\_t i=0; i<v.size(); ++i)      delete v[i];      return 0;  } |

Вывод программы:

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10 | Create game  Save game in TEMP\_FILE  Enter your move: E2-E4  Make move E2-E4  Save game in TEMP\_FILE  Enter your move: D2-D3  Make move D2-D3  Open game from TEMP\_FILE  Enter file name: game1.sav  Save game in game1.sav |

**Результаты применения паттерна Command**

**Достоинства паттерна Command**

* Придает системе гибкость, отделяя инициатора запроса от его получателя.

Источник: <http://sourcemaking.com/design_patterns/decorator/>

**http://cpp-reference.ru/patterns/structural-patterns/flyweight/**