Р О С С И Й С К А Я Ф Е Д Е Р А Ц И Я

М о с к о в с к а я о б л а с т ь

**Филиал «Котельники»**

государственного бюджетного образовательного учреждения высшего образования Московской области

**Университет «Дубна»**

**Кафедра «Информационных технологий в управлении»**

**Курсовая работа**

по дисциплине "Проектирование информационных систем"

на тему "Использование шаблона проектирования «Команда» для отмены и повторного использования операций на примере игрового приложения «Пятнашки»"

**Выполнил:**

Студент гр. ИВТ-31

Юнькин П.Д.

**Преподаватель:**

Сахаров М.В.

**Котельники 2017 г**

**Оглавление**

[Введение 3](#_Toc483669778)

[Глава 1. Анализ предметной области 4](#_Toc483669779)

[1.1 Шаблоны проектирования. Преимущества и недостатки 4](#_Toc483669780)

[1.2 Классификация шаблонов проектирования 5](#_Toc483669781)

[Выводы по Главе 1 8](#_Toc483669782)

[Шаблон «Команда» 9](#_Toc483669783)

[2.1 Основные сведения 9](#_Toc483669784)

[2.2 Применимость 9](#_Toc483669785)

[2.3 Участники 10](#_Toc483669786)

[2.4 Классовая диаграмма 11](#_Toc483669787)

[2.5 Отношения 11](#_Toc483669788)

[2.6 Результаты 12](#_Toc483669789)

[Выводы по Главе 2 12](#_Toc483669790)

[Глава 3. Реализация отмены и повторения операций на основе шаблона 13](#_Toc483669791)

[3.1 13](#_Toc483669792)

[3.2 Классовая диаграмма программы 13](#_Toc483669793)

[3.2 Кодддд 13](#_Toc483669794)

[Выводы по Главе 3 17](#_Toc483669795)

[Заключение 18](#_Toc483669796)

[Список литературы 19](#_Toc483669797)

Введение

В процессе использования многих программных продуктов пользователь достаточно часто сталкивается с тем, что требуется по некоторым причинам отменить, либо повторно выполнить уже отмененную операцию.

Данный сценарий может использоваться в разных типах приложений, таких как редакторы (текстовые, графические и.т.д.), приложения для работы с базами данных, файловые менеджеры.

Для более эффективной разработки приложений используются шаблоны проектирования.

Целью данной работы является разработка игрового приложения «Пятнашки» с использованием шаблона проектирования «Команда» для отмены и повторного использования операций.

Задачами курсовой работы являются:

1. Рассмотрение общих сведений о шаблонах проектирования;
2. Подробное изучение шаблона проектирования «Команда»;
3. Реализация программы на основе изученного шаблона;

В качестве программного инструмента был выбран язык C++ в среде Visual Studio.

Глава 1. Анализ предметной области

1.1 Шаблоны проектирования. Преимущества и недостатки

Перед тем как приступить к рассмотрению конкретного шаблона проведем общий анализ шаблонов проектирования.

Шаблон проектирования или паттерн (англ. design pattern) в разработке программного обеспечения — повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста.

Обычно шаблон не является законченным образцом, который может быть прямо преобразован в код; это лишь пример решения задачи, который можно использовать в различных ситуациях. Объектно-ориентированные шаблоны показывают отношения и взаимодействия между классами или объектами, без определения того, какие конечные классы или объекты приложения будут использоваться.

Применение шаблонов проектирования может иметь как преимущества, так и недостатки.

К преимуществам можно отнести:

1. Возможность многократного использования.

Повторное использование решений из уже завершенных успешных проектов позволяет быстро приступить к решению новых проблем и избежать типичных ошибок. Разработчик получает прямую выгоду от использования опыта других разработчиков, избежав необходимости вновь и вновь изобретать велосипед.

1. Применение единой терминологии.

Профессиональное общение и работа в группе (команде разработчиков) требует наличия единого базового словаря и единой точки зрения на проблему. Шаблоны проектирования предоставляют подобную общую точку зрения, как на этапе анализа, так и при реализации проекта.

1. Приемлемый уровень абстракции

Шаблоны проектирования предоставляют абстрактный высокоуровневый взгляд, как на проблему, так и на весь процесс объектно-ориентированной разработки. Это помогает избежать излишней детализации на ранних стадиях проектирования.

Недостатки состоят в следующем:

Не стоит применять шаблоны ради самих шаблонов. Хорошая программа предполагает использование шаблонов. Однако не всегда шаблоы упрощают и улучшают программу.

Неоправданное их использование может привести к усложнению программного кода, уменьшению его качества. Шаблон должен быть оправданным и эффективным способом решения проблемы.

1.2 Классификация шаблонов проектирования

Существует множество различных шаблонов, которые решают разные проблемы и выполняют различные задачи. Но по своему действию их можно объединить в ряд групп. Рассмотрим некоторые группы шаблонов.

Классификация шаблонов проводится по двум основным критериям.

Первый – цель – отражает назначение шаблона (см. табл. 1).

Таблица 1 Классификация шаблонов по целям

|  |
| --- |
| **Цели** |
| **Порождающие шаблоны** — это шаблоны, которые абстрагируют процесс инстанцирования или, иными словами, процесс порождения классов и объектов  К порождающим шаблонам относят:   * Абстрактная фабрика (Abstract Factory) * Строитель (Builder) * Фабричный метод (Factory Method) * Прототип (Prototype) * Одиночка (Singleton) |
| **Структурные шаблоны** - рассматривают, как классы и объекты образуют более крупные структуры - более сложные по характеру классы и объекты.  К таким шаблонам относятся:   * Адаптер (Adapter) * Мост (Bridge) * Компоновщик (Composite) * Декоратор (Decorator) * Фасад (Facade) * Приспособленец (Flyweight) * Заместитель (Proxy) |
| **Поведенческие шаблоны** - определяют алгоритмы и взаимодействие между классами и объектами, то есть их поведение.  Среди подобных шаблонов можно выделить следующие:   * Цепочка обязанностей (Chain of responsibility) * Команда (Command) * Интерпретатор (Interpreter) * Итератор (Iterator) * Посредник (Mediator) * Хранитель (Memento) * Наблюдатель (Observer) * Состояние (State) * Стратегия (Strategy) * Шаблонный метод (Template method) * Посетитель (Visitor) |

Второй критерий – уровень – говорит о том, к чему обычно применяется шаблон: к объектам или классам (см. табл. 2).

Таблица 2 Классификация шаблонов по уровням

|  |
| --- |
| **Уровни** |
| **Шаблоы уровня классов -** описывают отношения между классами и их подклассами. Такие отношения выражаются с помощью на­ следования, поэтому они статичны, то есть, зафиксированы на этапе компиляции.  К таким шаблонам относятся:   * Фабричный метод (Factory Method) * Интерпретатор (Interpreter) * Шаблонный метод (Template Method) * Адаптер (Adapter) |
| **Шаблоны уровня объектов** - описывают отношения между объектами. Эти отношения возникают на этапе выполнения, поэтому обладают большей гибкостью.  К шаблонам объектов относят следующие:   * Абстрактная фабрика (Abstract Factory) * Строитель (Builder) * Прототип (Prototype) * Одиночка (Singleton) * Мост (Bridge) * Компоновщик (Composite) * Декоратор (Decorator) * Фасад (Facade) * Приспособленец (Flyweight) * Заместитель (Proxy) * Цепочка обязанностей (Chain of responsibility) * Команда (Command) * Итератор (Iterator) * Посредник (Mediator) * Хранитель (Memento) * Наблюдатель (Observer) * Состояние (State) * Стратегия (Strategy) * Посетитель (Visitor) |

Существуют и другие способы классификации шаблонов. Например, некоторые шаблоны часто используются вместе, а другим предлагаются альтернативные решения. Применение части шаблонов приводит к схожему дизайну, хотя изначально их назначение различно.

В заключение важно отметить четыре компонента, которые должны присутствовать в описании каждого шаблона.

* Имя шаблона.
* Назначение шаблона и описание задачи, которую он призван решать
* Способ решения поставленной задачи
* Ограничения и требования, которые необходимо принимать во внимание при решении задачи

Выводы по Главе 1

В данной главе были рассмотрены общие сведения о шаблоне проектирования, их преимущества и недостатки. Также была рассмотрена классификация паттернов.

Шаблоны позволяют использовать уже готовые решения, используемые в других проектах, ускоряя разработку программных приложений. Шаблоны помогают в командной работе над проектами, так как шаблоны используют единую терминологию. Шаблоны проектирования предоставляют абстрактный высокоуровневый взгляд, что помогает избежать излишней детализации на ранних стадиях проектирования.

При этом не стоит применять шаблоны ради самих шаблонов: их использование может привести к усложнению программного кода.

Шаблон «Команда»

2.1 Основные сведения

Иногда необходимо посылать объектам запросы, ничего не зная о том, выполнение какой операции запрошено и кто является получателем. Например, в библиотеках для построения пользовательских интерфейсов встречаются такие объекты, как кнопки и меню, которые посылают запрос в ответ на действие пользователя. Но в саму библиотеку не заложена возможность обрабатывать этот запрос, так как только приложение, использующее ее, располагает информацией о том, что следует сделать. Проектировщик библиотеки не владеет никакой информацией о получателе запроса и о том, какие операции тот должен выполнить. Шаблон команда позволяет библиотечным объектам отправлять запросы неизвестным объектам приложения, преобразовав сам запрос в объект. Этот объект можно хранить и передавать, как и любой другой. В основе списываемого шаблона лежит абстрактный класс Command, в котором объявлен интерфейс для выполнения операций. В простейшей своей форме этот интерфейс состоит из одной абстрактной операции Execute. Конкретные подклассы Command определяют пару «получатель-действие», сохраняя получателя в переменной экземпляра, и реализуют операцию Execute, так чтобы она посылала запрос. У получателя есть информация, необходимая для выполнения запроса.

2.2 Применимость

Используйте шаблон команда, когда хотите:

* параметризовать объекты выполняемым действием, как в случае с пунктами меню MenuItem. В процедурном языке такую параметризацию можно выразить с помощью функции обратного вызова, то есть такой функции, которая регистрируется, чтобы быть вызванной позднее. Команды представляют собой объектно-ориентированную альтернативу функциям обратного вызова;
* определять, ставить в очередь и выполнять запросы в разное время. Время жизни объекта Command необязательно должно зависеть от времени жизни исходного запроса. Если получателя запроса удается реализовать так, что­ бы он не зависел от адресного пространства, то объект-команду можно пе­ редать другому процессу, который займется его выполнением
* поддержать отмену операций. Операция Execute объекта Command может сохранить состояние, необходимое для отката действий, выполненных командой. В этом случае в интерфейсе класса Command должна быть дополнительная операция Unexecute, которая отменяет действия, выполненные предшествующим обращением к Execute. Выполненные команды хранятся в списке истории. Для реализации произвольного числа уровней отмены и повтора команд нужно обходить этот список соответственно в обратном и прямом направлениях, вызывая при посещении каждого элемента команду Unexecute или Execute;
* поддержать протоколирование изменений, чтобы их можно было выполнить повторно после аварийной остановки системы. Дополнив интерфейс класса Command операциями сохранения и загрузки, вы сможете вести протокол изменений во внешней памяти. Для восстановления после сбоя нужно будет загрузить сохраненные команды с диска и повторно выполнить их с помощью операции Execute;
* структурировать систему на основе высокоуровневых операций, построенных из примитивных. Такая структура типична для информационных систем, поддерживающих транзакции. Транзакция инкапсулирует набор изменений данных. Шаблон команда позволяет моделировать транзакции. У всех команд есть общий интерфейс, что дает возможность работать одинаково с любыми транзакциями. С помощью этого шаблона можно легко добавлять в систему новые виды транзакций.

2.3 Участники

* Command – команда:

– объявляет интерфейс для выполнения операции;

* ConcreteCommand – конкретная команда:

– определяет связь между объектом-получателем Receiver и действием;

– реализует операцию Execute путем вызова соответствующих операций объекта Receiver;

* Client – клиент:

– создает объект класса ConcreteCommand и устанавливает его получателя;

* Invoker– инициатор:

– обращается к команде для выполнения запроса;

* Receiver – получатель:

– располагает информацией о способах выполнения операций, необходимых для удовлетворения запроса. В роли получателя может выступать любой класс.

2.4 Классовая диаграмма



2.5 Отношения

* клиент создает объект ConcreteCommand и устанавливает для него получателя;
* инициатор Invoker сохраняет объект ConcreteCommand;
* инициатор отправляет запрос, вызывая операцию команды Execute.
* Если поддерживается отмена выполненных действий, то ConcreteCommand перед вызовом Execute сохраняет информацию о состоянии, достаточную для выполнения отката;
* объект ConcreteCommand вызывает операции получателя для выполнения запроса. На следующей диаграмме видно, как Command разрывает связь между инициатором и получателем (а также запросом, который должен выполнить последний).

2.6 Результаты

Результаты применения шаблона команда таковы:

* команда разрывает связь между объектом, инициирующим операцию, и объектом, имеющим информацию о том, как ее выполнить;
* команды – это самые настоящие объекты. Допускается манипулировать ими и расширять их точно так же, как в случае с любыми другими объектами;
* из простых команд можно собирать составные, например класс MacroCommand, рассмотренный выше. В общем случае составные команды описываются шаблоном компоновщик;
* добавлять новые команды легко, поскольку никакие существующие классы изменять не нужно;

Выводы по Главе 2

В данной главе был изучен шаблон «Команда». Были рассмотрены основные свойства шаблона и ситуации, когда применение шаблона уместно, показана классовая диаграмма шаблона, его участники и отношения между ними. Также приведены результаты использования шаблона.

Шаблон «Команда» позволяет инкапсулировать запрос на выполнение определенного действия в виде отдельного объекта. При этом объекты, инициирующие запросы на выполнение действия, отделяются от объектов, которые выполняют это действие. Команды могут использовать параметры, которые передают ассоциированную с командой информацию. Кроме того, команды могут ставиться в очередь и также могут быть отменены.

В следующей главе будет представлена реализация шаблона на примере практической задачи.

Глава 3. Реализация отмены и повторения операций на основе шаблона

3.1 Структура приложения

В качестве приложения для демонстрации шаблона было выбрано игровое приложение Пятнашки, написанное на C++, позволяющее реализовать отмену и повторение ходов.

Для реализации данной программы используется среда Microsoft Visual Studio 2010.

Отдельно, с помощью технологии MFC, был разработан интерфейс программы, позволяющий выбрать размер поля, его внешний вид, а также содержащий кнопки для отмены и повтора операций.

Перейдем к рассмотрения структуры приложения.

Ниже, на рис.1, представлена упрощенная классовая диаграмма приложения.

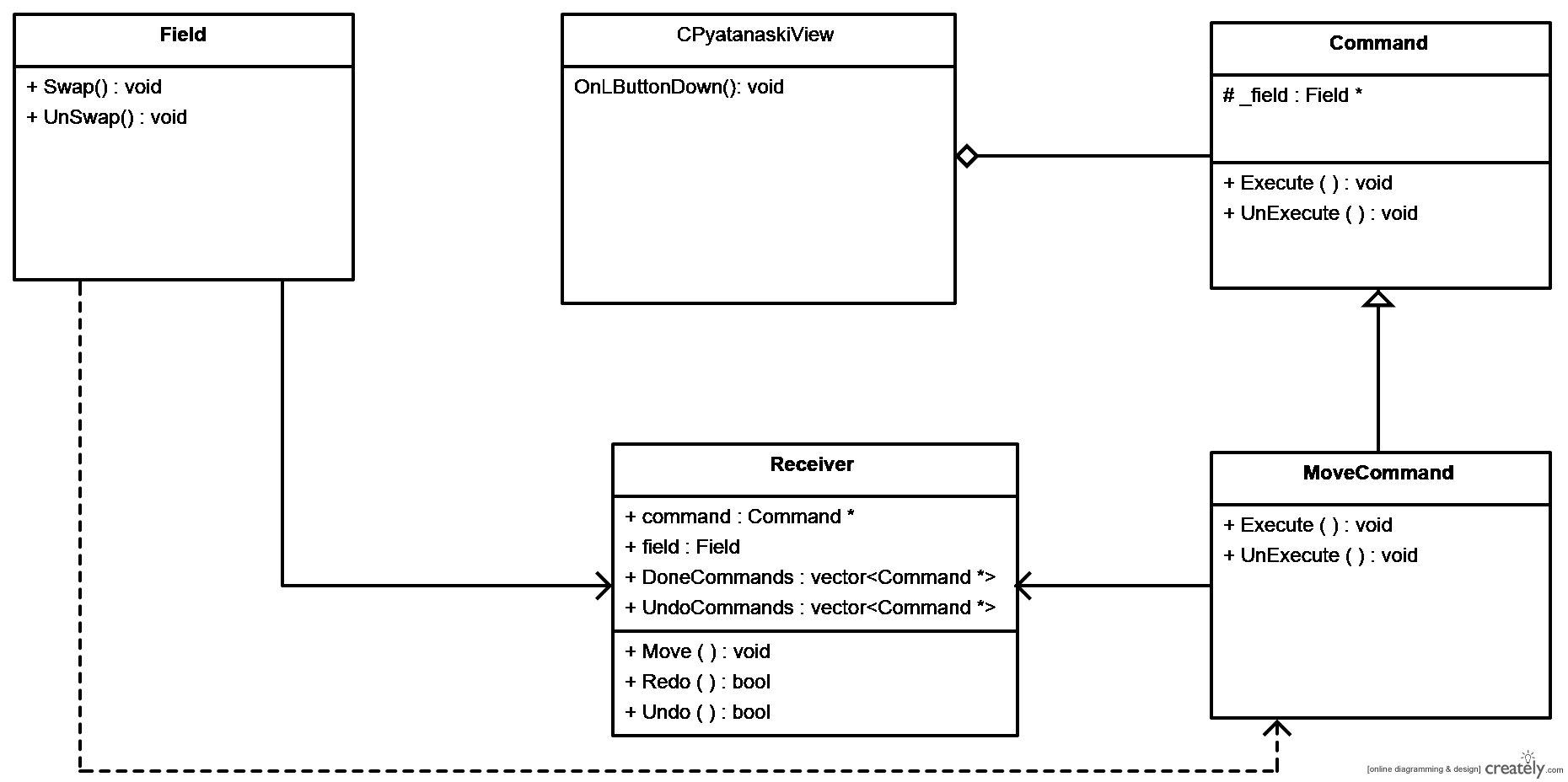


Рис.1 Классовая диаграмма приложения

Рассмотрим каждый класс и его основные методы поподробнее.

3.2 Реализация основных классов

Первым будет рассмотрен класс Field, реализующий игровое поле и действия, совершаемые над ним.

Закрытые поля класса содержат двумерный целочисленный массив \_arr, являющийся полем и его размерность \_size.

Класс имеет 2 конструктора, один – по умолчанию, второй – для создания массива поля требуемого размера.

Реализованы такие методы, как: Shake() – для перемешивания поля перед началом игры, Swap(int source\_x,int source\_y) – для перемещения по полю, UnSwap() – для обратного перемещения и isFinished() – реализующий проверку условий победы.

class Field

{

int \_size;

int \*\*\_arr;

public:

Field(){}

Field(int size);

void Shake();

void Swap(int source\_x,int source\_y);

void UnSwap();

bool isFinished();

// Остальные методы

};

Абстрактный класс Command содержит указатель на переменную класса поля Field и метод для SetField(Field \*field) для связывания команды с конкретным полем.

Также объявлены два виртуальных метода Execute() и UnExecute() – для последующей реализации операции и ее обратного эквивалента.

class Command

{

protected:

Field \* \_field;

public:

virtual void Execute()=0;

virtual void UnExecute()=0;

void SetField(Field\* field){\_field=field;}

};

Класс MoveCommand реализует команду хода игрока. Он наследуется от класса Command.

Поля класса \_x и\_y содержат координаты клетки поля, нажатой пользователем. Эти поля заполняются в конструкторе MoveCommand(int x,int y).

Методы Execute() и UnExecute() выполняют перемещение и обратное перемещение соответственно.

class MoveCommand : public Command

{

int \_x,\_y;

public:

MoveCommand(int x,int y) : \_x(x) , \_y(y) {}

void Execute()

{

\_field->Swap(\_x,\_y);

}

void UnExecute()

{

\_field->UnSwap();}

};

Инициатором выполнения команды является класс CPytnashkiShablonView, содержащий функцию OnLButtonDown(UINT nFlags, CPoint point) для обработки нажатия ЛКМ по полю и кнопкам.

Класс Reciever служит для хранения выполненных и отмененных команд. Кроме этого в нем происходит инициализация команд и связывания их с объектом.

Поля класса DoneCommands и UndoCommands содержат вектора для хранения выполненных и отмененных команд. Присутствуют поля для хранения информации о команде и объекте.

Конструктор Reciever(int size) создает игровое поле и перемешивает его.

В методе Move(int x, int y) происходит инициализация команды и присвоение ей объекта. После команда отправляется на выполнение методом Execute() и сохраняется в векторе выполненных команд.

Метод Undo() оттеняет предыдущую команду. При не пустом веторе DoneCommands он извлекает из него команду,

class Reciever

{

*vector*<Command\*> DoneCommands;

*vector*<Command\*> UndoCommands;

Field field;

Command\* command;

public:

Reciever(){}

Reciever(int size);

void Move(int x, int y)

{

command = new MoveCommand(x,y);

command->SetField(&field);

command->Execute();

DoneCommands.*push\_back*( command );

}

bool Undo()

{

if( DoneCommands.*size*() == 0 )

{

return false;

}

else

{

command = DoneCommands.*back*();

DoneCommands.*pop\_back*();

UndoCommands.*push\_back*(command);

command->UnExecute();

return true;

}

}

bool Redo()

{

if( UndoCommands.*size*() == 0 )

{

return false;

}

else

{

command = UndoCommands.*back*();

DoneCommands.*push\_back*(command);

UndoCommands.*pop\_back*();

command->Execute();

return true;

}

}

bool Finish()

{

if(field.isFinished())

{

return true;

}

return false;

}

};

Reciever::Reciever(int size)

{

field = Field(size);

field.Shake();

}

Выводы по Главе 3

Заключение

Список литературы

1. Гамма Э., Хелм Р., Джонсон Р., «Влиссидес Д. Приемы объектно-ориентированного проектирования», Питер, 368 стр., 2017
2. Алан Шаллоуей, Джеймс Р. Тротт «Шаблоны проектирования. Новый подход к объектно-ориентированному анализу и проектированию», Вильямс, 288 стр., 2002
3. https://metanit.com/sharp/patterns/1.1.php