МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

«ЛУГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ ВЛАДИМИРА ДАЛЯ»

Институт, факультет компьютерных систем и информационных технологий

Кафедра информационных и управляющих систем

Образовательно-квалификационный уровень бакалавр

Направление подготовки 09.03.02 Информационные системы и технологии

(шифр и название)

**З А Д А Н И Е**

**НА КУРСОВУЮ РАБОТУ СТУДЕНТУ**

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_Бардину Марку Игоревичу\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**(фамилия, имя, отчество)**

1. Тема проекта (работы)\_\_ Разработка объектно-ориентированное модели сюжета сказки «Иванушка дурачок» на основе паттернов проектирования и ее программная реализация \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

руководитель проекта (работы)\_\_\_Стоянченко С.С.\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_,

( фамилия, имя, отчество, научная степень, ученое звание)

утверждено распоряжением по кафедре от “3” марта 2022 года

2. Срок сдачи студентом проекта (работы)\_\_25 мая 2022\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

3. Исходные данные для проекта (работы) \_ 1. Сюжет художественного произведения

\_2. Спецификация языка программирования C++. 3. Паттерн проектирования \_\_”Фабричный метод”\_4. Паттерн поведения «Наблюдатель». \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Содержание расчетно-пояснительной записки (перечень вопросов, подлежащих разработке) \_\_1. Анализ предметной области. Выявление состава классов, их полей и методов\_\_\_\_\_2. Разработка диаграммы сценариев, диаграммы классов, диаграммы последовательности действий\_\_\_\_3. Выбор и обоснование используемых паттернов проектирования\_\_\_4. Разработка программного обеспечения\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

5. Перечень графического материала (с точным указанием обязательных чертежей)

1. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

2. \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

6. Дата выдачи задания\_04 марта 2022\_\_\_\_\_\_\_**\_\_**

**КАЛЕНДАРНЫЙ ПЛАН**

|  |  |  |  |
| --- | --- | --- | --- |
| №  п/п | Название этапов курсового проектирования | Срок выполнения этапов | Примечание |
| 1 | Изучение сюжета художественного произведения | 01.03.2022 |  |
|  |  |  |  |
| 2 | Разработка диаграммы сценариев | 10.03.2022 |  |
|  |  |  |  |
| 3 | Разработка диаграммы классов | 01.04.2022 |  |
|  |  |  |  |
| 4 | Выбор и обоснование языка программирования | 07.04.2022 |  |
|  |  |  |  |
| 5 | Классификация связей между объектами | 15.04.2022 |  |
|  |  |  |  |
| 6 | Разработка программного обеспечения | 01.05.2022 |  |
|  |  |  |  |
| 7 | Отладка программного обеспечения | 10.05.2022 |  |
|  |  |  |  |
| 8 | Написание пояснительной записки | 15.05.2022 |  |
|  |  |  |  |
| 9 | Разработка графической части курсовой работы | 20.05.2022 |  |
|  |  |  |  |
| 10 | Защита курсовой работы | 25.05.2022 |  |

**Студент \_\_\_\_\_\_\_\_\_ \_\_\_\_**Бардин М.И.**\_\_**

( подпись ) (фамилия и инициалы)

**Руководитель проекта (работы) \_\_\_\_\_\_\_\_\_ \_\_\_**\_Стоянченко С.С.**\_\_\_\_\_\_\_**

( подпись ) (фамилия и инициалы)

Примечание:

1. Форма предназначена для выдачи задания студенту на выполнение курсового проекта (работы) и контроля за ходом работы со стороны кафедры
2. Разрабатывается руководителем курсового проекта (работы). Выдается кафедрой в начале семестра.

Аннотация

В работе 46 листов формата А4, 2 рисунка, 3 таблицы, 4 источника.

Ключевые слова: объектно-ориентированнное программирование, шаблон, фабричный метод.

Объектом разработки является сказка «Иванушка дурачок»

Цель работы: создание программного комплекса который реализует сюжет сказки. В процессе разработки использовались методологии функционального и объектного проектирования, в частности диаграмма вариантов использования в нотации UML.

МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ

ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

ГОСУДАРСТВЕННОЕ ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ

ВЫСШЕГО ОБРАЗОВАНИЯ

ЛУГАНСКОЙ НАРОДНОЙ РЕСПУБЛИКИ

«ЛУГАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

ИМЕНИ ВЛАДИМИРА ДАЛЯ»

Кафедра информационных и управляющих систем

КУРСОВАЯ РАБОТА

по дисциплине «Объектно-ориентированное программирование»\_\_\_\_\_\_\_\_\_\_

(название дисциплины)

на тему: Разработка объектно-ориентированной модели сюжета сказки\_\_\_\_\_ \_ «Иван-Дурак» на основе паттернов проектирования и ее программная\_\_\_\_\_\_\_\_ реализация\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

**студента (ки)** \_2\_ курса \_\_\_\_\_\_\_ группы\_ИТ-902\_

**направления подготовки** 09.03.01 Информационные системы и технологии

(шифр и название направления подготовки)

\_\_\_\_\_\_\_\_\_Бардин М. И.\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_

(фамилия и инициалы) (подпись)

**Руководитель проекта**

\_Стоянченко С.С.­­­\_\_\_\_\_\_\_\_\_\_\_\_ \_\_\_\_\_\_\_\_\_\_\_\_

(должность, ученое звание, научная степень, (подпись)

фамилия и инициалы)

**Результаты защиты:**

Оценка: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

Члены комиссии:

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(фамилия и инициалы) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (фамилия и инициалы) (подпись)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ (фамилия и инициалы) (подпись)

Дата\_\_\_\_\_\_\_\_\_

**Луганск - 2022**

Содержание

ВВЕДЕНИЕ………………………………………………………………………… …..4

# АнаЛИТИЧЕСКИЙ ОБЗОР СОВРЕМЕННОГО СОСТОЯНИЯ ВОПРОСА… …5

1. Объектно-ориентированное программирование…………………..… … …5
2. Фабрики и паттерны …..………………………………………… …… ….…6

# ОСНОВНАЯ ЧАСТЬ………………………………………...…………………. …..11

1. Особенности реализации программы…………………..……………… …11
2. Объекты, Фабрики и паттерны, которые использовались ……… …. .…..12

# ЗАКЛЮЧЕНИе………………………………………………………………...... ……14

# Список используемых источников………………………….…….. ...…15

# приложеНИе а Сюжет сказки….…………………………………….… ..…16

# приложение B Листинг программы………………………………. …..…21

# приложение С файл Команд………...………………………………. …..…46

Введение

С развитием научных технологи, стал и развиваться язык программирования. Их на данный момент большое множество, но среди них можно выделить С, С++, С#, JavaScript, Python. Данная курсовая работа была разработана на основе объектно-ориентированного программирования языка С++.

В ходе написания курсового проекта, был запрограммирован сюжет русской народной сказки – «Иванушка дурачок». В самой программе персонажи сказки будут представлены в виде объектов, которые являются экземплярами класса, а классы строят иерархию наследования. Объектно-ориентированное программирование появилось во второй половине 80-х годов, поскольку процедурное программирование уже не справлялось с поставленными задачами. На них уходило куда больше времени и ресурсов, а также привлекало большое количество программистов. Так как раньше были только ЭВМ (Электронно-вычислительные машины), процедурное программирование справлялось со своей задачей, но в ходе появления ПК (персональных компьютеров), которые стали неотъемлемой частью частных предпринимателей, а также стали популярны для развлечений, оно потеряло свою актуальность. В результате это и привело к созданию Объектно-ориентированного программирования (ООП).

# Аналитический обзор современного состояния вопроса

## Объектно-ориентированное программирование

Объектно-ориентированное программирование – это методология программирования, где программы представлены в виде совокупности объектов, каждый из которых является экземпляром определенного класса, а классы образуют иерархию наследования.

Идеологически объектно-ориентированное программирование – это подход к программированию как к моделированию информационных обьектов, которое решает на новом уровне основную задачу структурного программирования, а именно ‑ структурирование информации со стороны управляемости, что повышает управляемость процессом моделирования.

Управляемость предполагает минимизацию избыточных данных и их целостность, поэтому, удобно созданное и будет легко пониматься. Таким образом решается задача по восприятию программистом формы и ее удобства дальнейшего использования.

Основные принципы объектно-ориентированного программирования:

Абстрагирование – выделение самого важного для решения поставленной задачи, а также понимание предмета, который создается в виде класса;

Инкапсуляция – быстрота и безопасность организации иерархической управляемости, простота команд;

Наследование – для быстроты и безопасности организации родственных понятий, чтобы учитывались только изменения, а не дублировалось все что было учтено в предыдущих шагах;

Полиморфизм – гибкость программы, для определения в каком месте лучше распараллелить или наоборот собрать воедино управление.

## Фабрики и паттерны

Фабричный метод – виртуальный конструктор, порождающий шаблон проектирования, предоставляющий подклассам (дочерним классам) интерфейс для создания экземпляров некоторого класса. В момент создания наследники могут определить, какой класс создавать. Иными словам, данный шаблон делегирует создание объектов наследниками родительского класса. Это позволяет использовать в коде программы не специфические классы, а манипулировать абстрактными объектами на более высоком уровне.

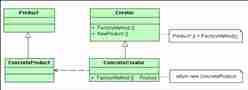


Рисунок 1.1 – Пример фабричного метода

Product – продукт, определяет интерфейс объектов, создаваемых абстрактым методом.

ConcreteProduct – конкретный продукт, реализует интерфейс Product.

Creator – создатель, объявляет фабричный метод, который возвращает объект типа Product. Может также содержать реализацию этого метода «по умолчанию»; может вызывать фабричный метод для создания объектов типа Product.

ConcreteCreator – конкретный создатель, переопределяет фабричный метод таким образом, чтобы он создавал и возвращал объекты класса ConcreteProduct.

Из достоинств можно выделить универсальность, не привязанность к конкретным классам, использование интерфейса, возможность установки связи между параллельными иерархиями классов.

Паттерн (шаблон проектирования) – повторяемая архитектурная конструкция, представляющая собой решение проблемы проектирования в рамках некоторого часто возникающего контекста. Они показывают отношения и взаимодействия между классами или объектами.

Таблица 2.1 – Основные паттерны

|  |  |  |
| --- | --- | --- |
| **Название** | **Оригинальное**  **название** | **Описание** |
| Шаблон  делегирования | Delegation  pattern | Объект внешне выражает некоторое поведение, но в реальности передает ответственность за выполнение этого поведения связанному объекту. |
| Шаблон функционального дизайна | Functional  design | Гарантирует, что каждый модуль компьютерной программы имеет только одну обязанность и исполняет ее с минимумом побочных эффектов. |
| Неизменяемый интерфейс | Immutable  interface | Создание неизменяемого объекта. |
| Интерфейс | Interface | Общий метод для структурирования компьютерных программ для того, чтобы их было проще понять. |
| Интерфейс-маркер | Marker interface | В качестве атрибута (пометки объектной сущности) применяется наличие или отсутствие реализации. |
| Контейнер свойств | Property container | Позволяет создавать дополнительные свойства для класса в контейнер(внутри класса), вместо расширения класса новыми свойствами. |
| Канал событий | Event channel | Расширяет шаблон Publish/Subscribe, создавая централизованный канал для событий. |

Порождающие шаблоны – шаблоны проектирования, которые абстрагируют процесс инстанцирования. Они позволяют сделать систему независимой от способа создания, композиции и представления объектов.

Таблица 2.2 ‑ Порождающие шаблоны

|  |  |  |
| --- | --- | --- |
| **Название** | **Оригинальное**  **название** | **Описание** |
| Абстрактная фабрика | Abstract factory | Класс, который представляет собой интерфейс для создания компонентов системы. |
| Строитель | Builder | Класс, который представляет собой интерфейс для создания сложного объекта. |
| Фабричный метод | Factoru method | Определяет интерфейс для создания объекта, но оставляет подкассам решение о том, какой класс инстанцировать. |
| Прототип | Prototype | Определяет интерфейс создания объекта через клонирование другого объекта вместо создания через конструктор. |
| Одиночка | Singleton | Класс, который имеет только один экземпляр. |
| Мультитон | Multiton | Гарантирует, что класс имеет поименованные экземпляры объекта и обеспечивает глобальную точку доступа к ним. |

Структурные шаблоны – определяют различные сложные структуры, которые изменяют интерфейс уже существующих объектов или его реализацию, позволяя облегчить разработку и оптимизировать программу.

Таблица 2.3 – Структурные шаблоны

|  |  |  |
| --- | --- | --- |
| **Название** | **Оригинальное**  **название** | **Описание** |
| Адаптер | Adapter | Объект, обеспечивающий взаимодействие двух других объектов, один из которых использует, а другой предоставляет несовместимый с первым интерфейс. |
| Мост | Bridge | Структура, позволяющая изменять интерфейс обращения и интерфейс реализации класса независимо |
| Компоновщик | Composite | Объект, который объединяет в себе объекты, подобные ему самому. |
| Декоратор | Decorator | Класс, расширяющий функциональность другого класса без использования наследования. |
| Фасад | Facade | Объект, который абстрагирует работу с несколькими классами, объединяя их в единое целое. |
| Приспособленец | Flyweight | Этот объект, представляющий себя как уникальный экземпляр в разных местах программы, но фактически не являющийся таковым. |
| Заместитель | Proxy | Объект, который является посредником между двумя другими объектами, и который реализует/ограничивает доступ к объекту, к которому обращаются через него. |

Поведенческие шаблоны – определяют взаимодействия между объектами, увеличивая таким образом его гибкость.

Таблица 2.4 – Поведенческие шаблоны

|  |  |  |
| --- | --- | --- |
| **Название** | **Оригинальное**  **название** | **Описание** |
| Цепочка обязанностей | Chain of responsibility | Предназначен для организации в системе уровней ответственности. |
| Команда | Command | Представляет действие. Объекты команды заключают в себе само действие и его параметры. |
| Интерпретатор | Interpreter | Решает часто встречающуюся, но подверженную изменениям, задачу. |
| Итератор | Iterator | Представляет собой объект, позволяющий получить последовательный доступ к элементам объекта-агрегата без использования описаний каждого из объектов, входящих в состав агрегации. |
| Посредник | Mediator | Обеспечивает взаимодействие множества объектов, формируя при этом слабую связанность и избавляя объекты от необходимости явно ссылаться друг на друга. |
| Хранитель | Memento | Позволяет не нарушая инкапсуляцию зафиксировать и сохранить внутренние состояния объекта так, чтобы позднее восстановить его в этих состояниях. |
| Наблюдатель | Observer | Определяет зависимость типа «один ко многим» между объектами таким образом, что при изменении состояния одного объекта все зависящие от него оповещаются об этом событии. |

# Основная часть

1. Особенности реализации программы

Вначале создается театр. Затем, при помощи класса TroupeCreator театр создает труппу актёров для конкретной сказки. Класс TroupeCreator аккумулирует в себе фабрики для всех актёров, предоставляя удобный интерфейс для их создания классу Theatre. Затем происходит получение данных сценария из файла. Данные в файле представлены положительными десятичными числами. Каждая новая команда начинается с новой строки. Театр использует объект класса Script, который производит парсинг текстового файла, удаляя из него все символы, кроме любых цифр, символа переноса строки ('\n'), и терминального нуля ('\0'). Также удаляются все пустые строки. В файле сценария предусмотрена возможность оставления комментариев. Комментарий в файле сценария должен начинаться с символа "точка с запятой" (';'). По итогу все команды помещаются в контейнер, откуда могут быть по очереди получены при помощи соответствующих методов.

После получения набора команд они передаются актёрам. Команды всегда передаются тому актёру, который был создан последним. Если он не смог обработать её, он передаст её следующему по цепочке ответственности. Если последний актёр не смог обработать команду, будет вызвано соответствующее исключение.

Класс Actor устроен таким образом, что хранит внутри себя ссылку на следующего актёра. Это необходимо для реализации цепочки ответственности. Каждый наследник класса Actor должен реализовать у себя метод bool processReplica(int replica\_id), где replica\_id - числовая команда из файла сценария. Метод возвращает true, если команда была обработана и false в ином случае. Главным методом, через который происходит взаимодействие с актёром является void handleRequest(int replica\_id), который в зависимости от возвращаемого значения метода processReplica либо передает команду следующему актёру, либо вызывает исключение (в случае, если команда не была обработана и нет ссылки на актёра, которому можно было бы её передать) либо ничего не делает (если команда была обработана и выполнена).

Всего в сюжете сказки имеются 6 персонажей: Отец (класс Father), Иван(класс Ivan), Илья Муромец(класс Illy), Федор Лыжников(класс Fedor), Царь(класс King), Добрыня (класс Dobriny). Также в ходе составления взаимоотношений между персонажами, был добавлен класс Manager, в котором реализована связь между классами: Отец, Иван, Илья Муромец, Федор Лыжников.

1. Объекты, фабрики и паттерны, которые использовались

В данной работе были применены следующие паттерны: Абстрактная фабрика, Цепочка ответственности.

Программа работает таким образом, что формируется виртуальный театр, для которого загружается сценарий и «приглашается» труппа актеров. Актеры создаются с помощью фабрик, для каждого актера она своя. Каждая фабрика является наследником класса AbstractActorFactory. Фабрики реализуют внутри себя метод createActor, который должен вернуть ссылку на соответствующего актера. Все фабрики находятся внутри класса TroupeCreator, который служит для создания трупп актеров для класса Teatr. Каждому актеру, кроме первого, необходимо передать ссылку на следующего актера(по умолчанию значение нулевой ссылки). Таким образом строится цепочка ответственности.

Театр получает команды из сценария и передает их самому последнему созданному актеру. Если он не смог обработать команду, то он передает ее следующему актеру, если и он не смог ее обработать, то будет вызвано исключение и выведено сообщение о том, что никто не смог обработать команду.

Абстрактная фабрика представляет собой интерфейс для создания семейств взаимосвязанных или взаимозависимых объектов, не специфицируя их конкретных классов.



Рисунок 2.1 Пример абстрактной фабрики

Заключение

При работе над этой курсовой работой мы на практике освоили методику проектирования программных комплексов на базе объектно-ориентированного программирования. Объектно-ориентированное программирование является одним из ведущих направлений в разработке на данный момент.

Мы провели анализ выбранной предметной области. Графически изобразили объектную модель, установили связи между объектами и классами. Также описали реализацию объектной модели на языке С++ . Мы думаем, что мы достигли поставленной цели и получили навыки проектирования программных комплексов на базе объектно-ориентированного программирования.

Список использованных источников

1. Методология объектно-ориентированного программирования – [Электронный ресурс] ‑ <http://www.znannya.org/?view=Methodology_object_oriented_programming>. (дата обращения 20.05.2019).
2. Объектно-ориентированное программирование – [Электронный ресурс] ‑ [https://ru.wikipedia.org/wiki/Объектно-ориентированное программирование #История](https://ru.wikipedia.org/wiki/Объектно-ориентированное%20%20программирование%20" \l "История). (дата обращения 20.05.2019).
3. Фабричный метод (шаблон проектирования) – [Электронный ресурс] ‑ <https://ru.wikipedia.org/wiki/Фабричный_метод_(шаблон_проектирования)>. (дата обращения 21.05.2019).
4. Шаблон проектирования – [Электронный ресурс] ‑ <https://ru.wikipedia.org/wiki/Шаблон_проектирования>. (дата обращения 21.05.2019)

Приложение А

Сюжет сказки

В некотором царстве, в некотором государстве жил-был старик со старухой. У них было три сына, третьего звали Иван-дурак. Первые двое женатые, а Иван-дурак холостой; два брата занимались делом, управляли домом, пахали и сеяли, третий же ничего не делал. Один раз отец и снохи стали Ивана посылать на поле допахать сколько-то лех пашни. Парень поехал, приехал на пашню, запряг лошадь, проехал с сохой раз ли, два ли, видит: счету нет комаров да мошек; он схватил хлыстик, стегнул по боку лошадь, убил их без сметы; ударил по другому, убил сорок паутов и думает:

- Ведь я на один замах убил сорок богатырей, а мелкой сошки сметы нет!

Взял их всех положил в кучу и завалил конским калом; сам не стал пахать, выпряг лошадь, поехал домой. Приезжает домой и говорит снохам и матери:

- Давайте мне полог и седло, а ты, батюшка, давай саблю, котора у тебя висит — на стене заржавела. Что я за мужик! У меня ничего нет.

Те посмеялись над ним и дали на смех какой-то расколотый тюрик наместо седла; парень наш приделал к нему подпруги и надел на худую кобыленку. Вместо пологу мать дала какой-то старый дубас; он и то взял, да взял саблю у отца, пошел, выточил, собрался и поехал. Доезжает до росстаней — а был еще мало-мало грамотный — написал на столбе: приезжали бы сильные богатыри Илья Муромец и Федор Лыжников в такое-то государство к сильному и могучему богатырю, который на один помах убил сорок богатырей, а мелкой сошки сметы нет, и всех их камнем привалил.

Точно, после его приезжает богатырь Илья Муромец, видит на столбе надпись:

- Ба, — говорит, — проехал сильный, могучий богатырь: ослушаться не годно.

Поехал, догонят Ванюху; далеко не доехал, снял шапку и кланятся:

- Здравствуй, сильный, могучий богатырь!

А Ванюха не ломат шапки, говорит:

- Здорово, Илюха!

Поехали вместе. Не чрез долго времени к тому же столбу приехал Федор Лыжников, видит, на столбе написано, ослушаться не годно: Илья Муромец проехал! — и он поехал туда же; так же далеко не доехал до Ванюхи — снимат шапку, говорит:

- Здравствуй, сильный, могучий богатырь!

А Ванюха шапки не ломат.

- Здорово, — говорит, — Федюнька!

Поехали вместе все трое; приезжают в одно государство, остановились на царских лугах. Богатыри поставили себе шатры, а Ванюха распялил дубас; коней два богатыря спутали шелковыми путами, а Ванюха сорвал прут с дерева, свил его и спутал свою кобылу. Вот и живут. Царь из своего терему увидел, что его любимые луга травят какие-то люди, тотчас отряжат ближнего своего спросить, что за люди? Тот приехал на луга, подошел к Илье Муромцу, спрашиват, что они за люди и как смели без спросу топтать царские луга? Илья Муромец отвечал:

- Не наше дело! Спрашивай вон старшого — сильного, могучего богатыря.

Посол подошел к Ванюхе. Тот закричал на него, не дал слова молвить:

- Убирайся, докуля жив, и скажи царю, что на его луга приехал сильный, могучий богатырь, который на один помах убил сорок богатырей, а мелкой сошки сметы нет, и камнем привалил, да Илья Муромец и Федор Лыжников с ним, и требует у царя дочь замуж.

Тот пересказал это царю. Царь хватил по записям: Илья Муромец и Федор Лыжников есть, а третьего, который убиват на один помах по сороку богатырей, нет в записях. То царь приказал собрать рать, захватить трех богатырей и привести к нему. Где захватить? Ванюха увидел, как стала подходить ближе рать; он крикнул:

- Илюха! Ступай прогони их, что за люди? — сам лежит, растянулся да поглядыват, как сыч.

Илья Муромец на того слова соскочил на коня, погнал, не столько руками бил, сколько конем топтал; всех прибил, оставил только одних язычников царю. Царь услышал эту беду, того больше собрал силы и послал поймать богатырей. Иван-дурак крикнул:

- Федюнька! Поди-ка прогони эту сволочь!

Тот соскочил на коня, всех прибил, оставил одних язычников.

Чего делать царю? Дело худо, силу побили богатыри; царь призадумался и вспомнил, что у него в царстве живет сильный богатырь Добрыня. Он посылает к нему письмо, просит приехать победить трех богатырей. Добрыня приехал; царь на третьем балконе встретил его, а Добрыня навершный подъехал к балкону вровень с царем: вот какой был! Поздоровался, поговорили. Он и поехал на царские луга. Илья Муромец и Федор Лыжников увидели, что к ним едет Добрыня, испугались, соскочили на своих лошадей да ступай-ка оттуда — угнали. А Ванюха не успел. Пока имал свою кобыленку, Добрыня и подъехал к нему, да и смеется, что это за сильный, могучий богатырь? Маленький, худенький! Согнулся головой к самому Ванюхе, смотрит на него, да и любуется. Ванюха да как-то не обробел, выхватил свою саблёшку, да и ссек ему голову.

Царь это увидел, перепугался:

- Ох, — говорит, — богатырь убил Добрыню; беда теперя! Ступайте скорее, зовите богатыря во дворец.

За Ванюхой приехал такой почет, что батюшки упаси! Кареты самолучши, люди все изжалованы.

Посадили и привезли к царю. Царь его угостил и отдал дочь; обвенчались они, и теперя живут, хлеб жуют.

Я тут был, мед пил; по усам текло, в рот не попало. Дали мне колпак, да почали толкать; дали мне кафтан, я иду домой, а синичка летат и говорит:

- Синь да хорош!

Я думал:

- Скинь да положь!

Взял скинул, да и положил. Это не сказка, а присказка, сказка впереди!

ПРИЛОЖЕНИЕ B

Листинг программы

#include <iostream>

#include <cstdlib>

#include <stdio.h>

#include <fstream>

#include <string>

#include <vector>

using namespace std;

class Exception\_ReplicaNotProcessed : public exception {

const char\* what() const throw() { return "Никто не смог обработать реплику (неверный номер)"; }

};

class fileNotOpened : public exception {

public:

const char\* what() const throw() { return "File not opened!"; }

};

// Общий класс для файлов

class File {

public:

virtual void openFile() = 0;

virtual void closeFile() = 0;

};

// Класс для текстовых файлов

class TextFile : public File {

public:

TextFile(string name) :filename(name) {}

TextFile(const char\* name) :filename(name) {}

TextFile(char\* name) :filename(name) {}

// Открывает файл для последующей обработки

void openFile() {

file.open(filename.c\_str());

if (!file.is\_open()) {

fileNotOpened e;

throw e;

}

}

// Закрывает работу с файлом (по окончании оброаботки)

void closeFile() {

file.close();

}

// Получает следующий символ из файла

bool getNextChar(char& ch) {

if (!file.get(ch)) {

return false;

}

return true;

}

private:

string filename; // Имя файла

ifstream file; // Текстовый файл

};

// Класс обработчика содержимого текстового файла

class TextFileProcessor : public TextFile {

public:

TextFileProcessor(string filename) : TextFile(filename) { }

TextFileProcessor(char\* filename) : TextFile(filename) { }

TextFileProcessor(const char\* filename) : TextFile(filename) { }

virtual void processFile() = 0;

string getFileContent() { return fileContent; }

void setFileContent(string fileContent) { this->fileContent = fileContent; }

private:

string fileContent; // Содержимое файла в виде строки

};

// Класс сценария

class Script : public TextFileProcessor {

public:

/\* КОНСТРУКТОРЫ/ДЕСТРУКТОРЫ \*/

Script() : TextFileProcessor("commands.txt") { prepareScript(); }

Script(string scriptFileName) : TextFileProcessor(scriptFileName) { prepareScript(); }

Script(char\* scriptFileName) : TextFileProcessor(scriptFileName) { prepareScript(); }

Script(const char\* scriptFileName) : TextFileProcessor(scriptFileName) { prepareScript(); }

~Script() { }

/\* КОНСТРУКТОРЫ/ДЕСТРУКТОРЫ END \*/

/\* МЕТОДЫ \*/

int getCommand(int num) {

return commands[num];

}

int size() {

return commands.size();

}

/\* МЕТОДЫ END \*/

private:

vector <int> commands; // Команды из сценария в виде чисел

// Получает содержимое файла в виде строки

// Исключает из строки все символы, кроме цифр 0-9, символа переноса строки (\n), символа терминального нуля (\0)

// Все цифры в одной строке конкатенируются. Все символы после знака комментария (точка с запятой) ";" игнорируются

void processFile() {

string fileContent = getFileContent(); // Содержимое файла сценария в виде строки

char ch;

while (getNextChar(ch)) {

if (ch != '0' && // Если текущий сивол - не число и не символ переноса/окончания строки

ch != '1' &&

ch != '2' &&

ch != '3' &&

ch != '4' &&

ch != '5' &&

ch != '6' &&

ch != '7' &&

ch != '8' &&

ch != '9' &&

ch != '\n' &&

ch != '\0') {

if (ch == ';') { // Пропуск всего, что стоит после точки с запятой (до символа переноса строки)

while (ch != '\n') {

if (!getNextChar(ch))

break;

}

fileContent += '\n';

}

continue;

}

fileContent += ch;

}

setFileContent(fileContent);

}

// Получает готовые команды в виде чисел из строки с числами, разделенными символами переноса строки ('\n')

void extractCommands() {

string fileContent = getFileContent();

string row; // Переменная для хранения строк символов (до \n)

int commandsNum = 0; // Количество команд

int command; // Переменная для хранения команды, полученной из строки

for (int i = 0; i < fileContent.length() + 1; i++) {

row += fileContent[i];

if (fileContent[i] == '\n' || fileContent[i] == '\0') {

if (command = atoi(row.c\_str())) {

commandsNum++;

//commands.resize(commandsNum);

commands.push\_back(command);

}

row.clear();

continue;

}

}

}

void prepareScript() {

try {

openFile();

processFile();

extractCommands();

closeFile();

}

catch (exception & e) {

cout << "Error: " << e.what() << endl;

}

}

};

class Manager;

class Actor {

public:

/\* КОНСТРУКТОРЫ \*/

Actor(const char\* name, Actor\* next = 0) {

setName(name);

setNextActor(next);

}

Actor(string name, Actor\* next = 0) {

setName(name);

setNextActor(next);

}

/\* КОНСТРУКТОРЫ END \*/

// Метод - обертка, который вызывается у актера

// Метод передаст необработанную реплику следующему актеру

void handleRequest(int replica\_id) {

if (!processReplica(replica\_id)) { // Если реплика не была обработана

if (nextActor == 0) { // Если следующего актера нет (текущий - последний) и реплика не была обработана

Exception\_ReplicaNotProcessed e; // То бросается исключение об этом

throw e;

}

else { // Иначе необработанная реплика передается следующему актеру

nextActor->handleRequest(replica\_id);

}

}

}

void say(const char\* phrase) {

cout << getName() << ": \"" << phrase << "\"" << endl;

}

void action(const char\* action) {

cout << getName() << ": \*" << action << "\*" << endl;

}

/\* ГЕТТЕРЫ/СЕТТЕРЫ \*/

void setName(string newName) { name = newName; }

void setName(char\* newName) { name = newName; }

void setNextActor(Actor\* next) { nextActor = next; }

string getName() { return name; }

/\* ГЕТТЕРЫ/СЕТТЕРЫ END \*/

private:

string name; //Имя актера

Actor\* nextActor; // Актер, который идет следующий в цеопчке ответственности

protected:

// Обработка реплики

// Здесь задается вся логика реакций актера на различные реплики

virtual bool processReplica(int replica\_id) = 0;

};

class Father : public Actor {

public:

Father(Actor\* next = 0) : Actor("Отец", next) { son = 0; }

friend Manager;

private:

Actor\* son;

bool processReplica(int replica\_id) {

switch (replica\_id) {

case 11: sayOtpravil(); break;

case 15: sayAcip(); break;

case 231: whoIsYourSon(); break;

default: return false; break;

}

return true;

}

void sayOtpravil() { action("Отправил Ивана в поле"); }

void sayAcip() { action("Дал седло и саблю"); }

void whoIsYourSon() {

string answer = "";

answer.append("Мой сын: ");

answer.append(son->getName());

say(answer.c\_str());

}

};

class Ivan : public Actor {

public:

Ivan(Actor\* next = 0) : Actor("Иван", next) { father = 0; }

friend Manager;

private:

Actor\* father;

vector <Actor\*> companions;

bool processReplica(int replica\_id) {

switch (replica\_id) {

case 22: sayGopole(); break;

case 23: sayKill(); break;

case 24: sayAciphelp(); break;

case 26: sayGoznak(); break;

case 211: sayHelloIlly(); break;

case 216: sayHelloFedor(); break;

case 217: sayPrival(); break;

case 220: sayDefeatposol(); break;

case 228: sayVinDobriny(); break;

case 232: whoIsYourFather(); break;

default: return false; break;

}

return true;

}

void sayGopole() { action("Пошел в поле"); }

void sayKill() { action("Убивает мошек и возвращается домой"); }

void sayAciphelp() { say("Давайте мне полог и седло, а ты, батюшка, давай саблю, котора у тебя висит — на стене заржавела. Что я за мужик! У меня ничего нет."); }

void sayGoznak() { action("Приезжает к столбу и оставляет надпись."); }

void sayHelloIlly() { say("Здорово, Илюха!"); }

void sayHelloFedor() { say("Здорово, Федюнька!"); }

void sayPrival() { action("Разбил привал на лугах царя с Ильей и Федором."); }

void sayDefeatposol() { action("Прогоняет посла и требует от Царя в жены его дочь!"); }

void sayVinDobriny() { action("Отрубает голову Добрыне"); }

void addCompanion(Actor\* newCompanion) { companions.push\_back(newCompanion); }

void whoIsYourFather() {

string answer = "";

answer.append("Мой отец: ");

answer.append(father->getName());

say(answer.c\_str());

}

};

class Illy : public Actor {

public:

Illy(Actor\* next = 0) : Actor("Илья Муромец", next) {}

friend Manager;

private:

vector <Actor\*> companions;

bool processReplica(int replica\_id) {

switch (replica\_id) {

case 37: sayRides(); break;

case 38: saySay(); break;

case 39: sayGoivan(); break;

case 310: saySayIlly(); break;

case 322: sayFight(); break;

default: return false; break;

}

return true;

}

void sayRides() { action("Читает надпись"); }

void saySay() { say("Ба, проехал сильный, могучий богатырь: ослушаться не годно."); }

void sayGoivan() { action("Догнал Ивана"); }

void saySayIlly() { say("Здравствуй, сильный, могучий богатырь!"); }

void sayFight() { action("Бьет рать"); }

void addCompanion(Actor\* newCompanion) { companions.push\_back(newCompanion); }

};

class Fedor : public Actor {

public:

Fedor(Actor\* next = 0) : Actor("Федор Лыжников", next) {}

friend Manager;

private:

vector <Actor\*> companions;

bool processReplica(int replica\_id) {

switch (replica\_id) {

case 412: sayRides(); break;

case 413: saySay(); break;

case 414: sayGoivan(); break;

case 415: saySayFedor(); break;

case 424: sayFight(); break;

default: return false; break;

}

return true;

}

void sayRides() { action("Читает надпись"); }

void saySay() { say("Илья Муромец проехал, ослушаться не гоже."); }

void sayGoivan() { action("Догнал Ивана"); }

void saySayFedor() { say("Здравствуй, сильный, могучий богатырь!"); }

void sayFight() { action("Бьет рать"); }

void addCompanion(Actor\* newCompanion) { companions.push\_back(newCompanion); }

};

class King : public Actor {

public:

King(Actor\* next = 0) : Actor("Царь", next) {}

private:

bool processReplica(int replica\_id) {

switch (replica\_id) {

case 518: saySize(); break;

case 519: sayPosol(); break;

case 521: sayRatone(); break;

case 523: sayRattwo(); break;

case 525: sayDobriny(); break;

case 529: sayStrax(); break;

case 530: saySvadba(); break;

default: return false; break;

}

return true;

}

void saySize() { action("Видит богатырей и гневается."); }

void sayPosol() { action("Отправляет посла"); }

void sayRatone() { action("Шлет рать"); }

void sayRattwo() { action("Шлет еще больше рать"); }

void sayDobriny() { action("Зовет на помощь Добрыню"); }

void sayStrax() { say("Ох, богатырь убил Добрыню; беда теперя! Ступайте скорее, зовите богатыря во дворец."); }

void saySvadba() { action("Выдает за Ивана дочь"); }

};

class Dobriny : public Actor {

public:

Dobriny(Actor\* next = 0) : Actor("Добрыня", next) {}

private:

bool processReplica(int replica\_id) {

switch (replica\_id) {

case 626: saySmex(); break;

case 627: saySay(); break;

default: return false; break;

}

return true;

}

void saySmex() { action("Смеется с Ивана"); }

void saySay() { say(" Что это за сильный, могучий богатырь? Маленький, худенький! "); }

};

class Manager : public Actor {

public:

Manager(Actor\* next = 0) : Actor("Менеджер", next) {

father = 0;

ivan = 0;

illy = 0;

fedor = 0;

}

void setTroupe(vector <Actor\*>\* troupe) {

this->troupe = troupe;

findActors();

}

private:

vector <Actor\*>\* troupe;

Father\* father;

Ivan\* ivan;

Illy\* illy;

Fedor\* fedor;

bool processReplica(int replica\_id) {

switch (replica\_id) {

case 1: start(); break;

case 233: illyJoinsIvan(); break;

case 234: fedorJoinsIvan(); break;

default: return false; break;

}

return true;

}

void findActors() {

for (int i = 0; i < troupe->size(); i++) {

if (troupe->data()[i]->getName() == "Отец") {

father = (Father\*)troupe->data()[i];

}

else if (troupe->data()[i]->getName() == "Иван") {

ivan = (Ivan\*)troupe->data()[i];

}

else if (troupe->data()[i]->getName() == "Илья Муромец") {

illy = (Illy\*)troupe->data()[i];

}

else if (troupe->data()[i]->getName() == "Федор Лыжников") {

fedor = (Fedor\*)troupe->data()[i];

}

}

}

void start() {

father->son = ivan;

ivan->father = father;

}

void illyJoinsIvan() {

ivan->addCompanion(illy);

illy->addCompanion(ivan);

}

void fedorJoinsIvan() {

ivan->addCompanion(fedor);

fedor->addCompanion(ivan);

illy->addCompanion(fedor);

fedor->addCompanion(illy);

}

};

// Абстрактная фабрика набора актеров

class AbstractActorFactory {

public:

virtual Actor\* createActor(Actor\* next) = 0;

};

class FatherFactory : public AbstractActorFactory {

public:

Actor\* createActor(Actor\* next = 0) { return new Father(next); }

};

class IvanFactory : public AbstractActorFactory {

public:

Actor\* createActor(Actor\* next = 0) { return new Ivan(next); }

};

class IllyFactory : public AbstractActorFactory {

public:

Actor\* createActor(Actor\* next = 0) { return new Illy(next); }

};

class FedorFactory : public AbstractActorFactory {

public:

Actor\* createActor(Actor\* next = 0) { return new Fedor(next); }

};

class KingFactory : public AbstractActorFactory {

public:

Actor\* createActor(Actor\* next = 0) { return new King(next); }

};

class DobrinyFactory : public AbstractActorFactory {

public:

Actor\* createActor(Actor\* next = 0) { return new Dobriny(next); }

};

class ManagerFactory : public AbstractActorFactory {

public:

Actor\* createActor(Actor\* next = 0) { return new Manager(next); }

};

// Фабрика конкретного актеров

class TroupeCreateor {

public:

Actor\* createFather(Actor\* next = 0) { return father\_f.createActor(next); }

Actor\* createIvan(Actor\* next = 0) { return ivan\_f.createActor(next); }

Actor\* createIlly(Actor\* next = 0) { return illy\_f.createActor(next); }

Actor\* createFedor(Actor\* next = 0) { return fedor\_f.createActor(next); }

Actor\* createKing(Actor\* next = 0) { return king\_f.createActor(next); }

Actor\* createDobriny(Actor\* next = 0) { return dobriny\_f.createActor(next); }

Actor\* createManager(Actor\* next = 0) { return manager\_f.createActor(next); }

private:

FatherFactory father\_f;

IvanFactory ivan\_f;

IllyFactory illy\_f;

FedorFactory fedor\_f;

KingFactory king\_f;

DobrinyFactory dobriny\_f;

ManagerFactory manager\_f;

};

// Абстрактный театр

class AbstractTheatre {

public:

virtual void play() = 0;

};

// Конкретный театр со своими актерами и сценарием

class Theatre : public AbstractTheatre {

public:

// Конструктор создает фабрику актеров и вызывает метод создания труппы актеров

Theatre() {

TroupeCreateor factory;

inviteTroupe(&factory);

script = new Script; // Сценарий по-умолчанию берется из файла "commands.txt", однако можно в качестве аргумента конструктору указать любое другое название

}

// Примечание: Фабрика актеров для театра и сам театр получаются "связанными" (т.к. выделяет память фабрика, а освобождает память театр)

~Theatre() {

for (int i = 0; i < troupe.size(); i++) delete troupe[i];

}

void play() {

for (int i = 0; i < script->size(); i++) {

troupe.back()->handleRequest(script->getCommand(i));

}

}

private:

vector <Actor\*> troupe;

Script\* script;

Manager\* m;

// Метод вызывает метод создания актеров у фабрики и сохраняет готовых актеров в труппу (вектор troupe)

void inviteTroupe(TroupeCreateor\* actor\_factory) {

troupe.push\_back(actor\_factory->createFather());

troupe.push\_back(actor\_factory->createIvan(troupe.back()));

troupe.push\_back(actor\_factory->createIlly(troupe.back()));

troupe.push\_back(actor\_factory->createFedor(troupe.back()));

troupe.push\_back(actor\_factory->createKing(troupe.back()));

troupe.push\_back(actor\_factory->createDobriny(troupe.back()));

troupe.push\_back(actor\_factory->createManager(troupe.back()));

m = (Manager\*)troupe.back();

m->setTroupe(&troupe);

}

};

int main() {

setlocale(LC\_ALL, "Russian");

try {

Theatre theatre;

theatre.play();

}

catch (exception & e) {

cout << e.what() << endl;

}

printf("\

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n\

\* Курсовой проект по дисциплине \*\n\

\* Объектно-ориентированное программирование \*\n\

\* Тема: Сказка \"Иван-дурак\" \*\n\

\* выполнил студент группы ИТ - 902 \*\n\

\* Бардин Марк Игоревич \*\n\

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\n");

system("pause");

return 0;

}

ПРИЛОЖЕНИЕ С

Файл комманд

1

11

22

23

24

15

26

37

38

39

310

211

412

413

414

415

216

217

518

519

220

521

322

523

424

525

626

627

228

529

530

231

232

233

234