**МИНОБРНАУКИ РОССИИ**

**Санкт-Петербургский государственный**

**электротехнический университет**

**«ЛЭТИ» им. В.И. Ульянова (Ленина)**

**Кафедра МО ЭВМ**

отчет

**по лабораторной работе №3**

**по дисциплине «Объектно-ориентированное программирование»**

Тема: Связывание классов

|  |  |  |
| --- | --- | --- |
| Студент гр. 3341 |  | Шаповаленко Е.В. |
| Преподаватель |  | Жангиров Т.Р. |

Санкт-Петербург

2024

## Цель работы

Исследовать и проанализировать методы связывания классов в объектно-ориентированном программировании, выявить их преимущества и недостатки, а также применить полученные знания на практике, создав класс Игры.

Для достижения данной цели необходимо выполнить следующие задачи:

1. Изучить основные концепции связывания классов, такие как ассоциация, агрегация и композиция.
2. Разработать примеры кода, иллюстрирующие различные способы связывания классов и их влияние на архитектуру приложения.

Таким образом, работа будет направлена на углубленное понимание и практическое применение концепций связывания классов в объектно-ориентированном программировании.

## Задание

А) Создать класс игры, который реализует следующий игровой цикл:

1. Начало игры
2. Раунд, в котором чередуются ходы пользователя и компьютерного врага. В свой ход пользователь может применить способность и выполняет атаку. Компьютерный враг только наносит атаку.
3. В случае проигрыша пользователь начинает новую игру
4. В случае победы в раунде, начинается следующий раунд, причем состояние поля и способностей пользователя переносятся.

Класс игры должен содержать методы управления игрой, начало новой игры, выполнить ход, и т.д., чтобы в следующей лаб. работе можно было выполнять управление исходя из ввода игрока.

Б) Реализовать класс состояния игры, и переопределить операторы ввода и вывода в поток для состояния игры. Реализовать сохранение и загрузку игры. Сохраняться и загружаться можно в любой момент, когда у пользователя приоритет в игре. Должна быть возможность загружать сохранение после перезапуска всей программы.

**Примечание**:

* Класс игры может знать о игровых сущностях, но не наоборот
* Игровые сущности не должны сами порождать объекты состояния
* Для управления самое игрой можно использовать обертки над командами
* При работе с файлом используйте идиому RAII.

## Выполнение работы

Структура Participant представляет собой участника игры.

Она хранит информацию о текущем состоянии участника.

У структуры есть методы:

1) placeShip (int index, Coords coords, ShipOrientation orientation): размещает корабль с индексом index по координатам coords с ориентацией orientation.

2) attackCell(Participant\* target, Coords coords): атакует клетку цели target с координатами coords

3) isAlive(): возвращает, жив ли участник

4) ready(): возвращает, готов ли участник к игре

У структуры есть виртуальный метод isBot(), который переопределен в каждом из наследников: Bot и Player. Кроме этого, в Bot и Player переопределены конструкторы.

Эти структуры позволяют взаимодействовать с игровым полем, менеджерами кораблей и способностей как с одним целым – участником игры.

Структура GameState представляет собой состояние игры.

В ней хранится вся информация об игре, которая будет сохраняться/загружаться, и информация о том, были ли изменения в игре, которые необходимо сохранить, и была ли игра успешно загружена.

Для сохранения/загрузки у структуры переопределены операторы ввода и вывода в поток.

Для обеспечения сохранности файла сохранения разработана структура Hasher. В конструкторе она принимает строку и вычисляет для нее два полиномиальных хэша (это обеспечивает, что шанс коллизии всего 10-18). При сохранении первыми двумя строками записываются хэши. При загрузке сначала считываются записанные хэши, после чего вычисляются хэши для всей оставшейся информации. Если они не совпадают – файл поврежден.

Для реализации идиомы RAII при работе с файлами реализован класс FileManager. В конструкторе он принимает название файла и режим работы: true, если файл необходимо открыть на запись, и false, если файл необходимо открыть на чтение. В деструкторе все открытые потоки закрываются.

У класса есть 3 метода:

1) read: для чтения данных из файла в GameState

2) write: для записи данных в файл из GameState

3) getData: для получения всего содержимого файла в виде строки

Класс Game представляет собой обертку над методами участников.

Игра хранит в себе GameState, FileManager, а также индекс цели target\_index\_ (атакуемый участник).

У Game есть набор методов для:

1) Задания количества участников (игроков и ботов)

2) Создания участников

3) Инициализации участников

4) Расстановки кораблей участников

5) Старта новой игры

6) Перезапуска игры при победе игрока

7) Атаки

8) Использования способности

9) Задания цели

10) Сохранения/загрузки

11) Получения информации о текущем состоянии игры

12) Получения выбранного участника

Ниже представлена UML-диаграмма реализованных в данной работе классов:

|  |
| --- |
|  |
| Рисунок 1: UML-диаграмма классов |

Разработанный программный код см. в приложении А.

Демонстрацию работы программы см. в приложении Б.

## Выводы

В ходе исследования методов связывания классов в объектно-ориентированном программировании была поставлена цель — выявить преимущества и недостатки различных подходов к связыванию, а также применить полученные знания на практике, создав класс Игры.

Для достижения этой цели были определены ключевые задачи:

1. Изучение основных концепций связывания классов: в процессе работы были рассмотрены такие понятия, как ассоциация, агрегация и композиция. Эти концепции стали основой для понимания того, как объекты взаимодействуют друг с другом и как правильно организовать их связи для достижения максимальной эффективности и удобства в поддержке кода.

2. Разработка примеров кода: создание примеров, иллюстрирующих различные способы связывания классов, позволило на практике увидеть, как каждый метод влияет на архитектуру приложения. Это дало возможность оценить, как выбор конкретного типа связывания может улучшить или усложнить структуру программы, а также её читаемость и расширяемость.

В результате проведённого исследования было достигнуто углубленное понимание концепций связывания классов и их практического применения. Разработанный класс Игры стал иллюстрацией того, как правильная организация связей между классами может существенно улучшить архитектуру приложения, сделать его более гибким и удобным для дальнейшего развития.

Таким образом, работа подтвердила важность осознания выбора методов связывания классов в процессе проектирования программного обеспечения, что в конечном итоге приводит к созданию более качественных и устойчивых программных систем.

# Приложение А Исходный код программы

https://github.com/lastikp0/Battleship

.

# Приложение Б Демонстрация работы программы