Лабораторная работа 3

по курсу Технология программирования

на тему «Объектно-ориентированное программирование в Java. Часть 2»

Составил: Питикин А.Р.

кафедра ИУ-3

Москва, 2024

Содержание

Теоретическая часть

Для работы с текстовыми файлами можно использовать классы FileWriter и FileReader.

Для FileWriter в конструктор передается либо путь к файлу в виде строки, либо объект File, который ссылается на конкретный текстовый файл. Параметр append указывает, должны ли данные дозаписываться в конец файла (если параметр равен true), либо файл должен перезаписываться.

FileWriter/FileReader - это потоки, их нужно не только «открыть» (то есть объявить), но и «закрыть». Это правило работает и для других потоков - кроме стандартных System.in и System.out. Закрыть поток можно с помощью .close():

```
FileWriter fw = new FileWriter( fileName: "file.txt");
fw.write( str: "Hello World!");
fw.close();
```

Другие, тоже полезные способы работы с файлами и с файловой системой в целом можно найти тут: https://www.marcobehler.com/guides/java-files https://www.w3schools.com/java/java_files_create.asp

Для второй части лабораторной работы необходимо использовать механизмы сериализации и десериализации объектов. Их идея заключается в представлении объектов в виде последовательности байтов и дальнейшем восстановлении из нее. Наиболее полезный пример описания работы можно найти здесь: https://javarush.com/groups/posts/2022-serializacija-i-deserializacija-v-java или здесь https://metanit.com/java/tutorial/6.10.php

Практическая часть

В рамках этой лабораторной работы предлагается доработка Bauman's Gate. Т.к. технологий графического интерфейса, таких, как JavaFX или Swing мы еще не касались в нашем курсе, в работе речь вновь пойдет о консольном приложении.

Целью данной работы является закрепление на практике теоретического материала об основных идеях ООП, полученного ранее, закрепление материала о работе с файлами, применение Stream API, а также понимание сериализации данных. Результатом ЛР будет являться модифицированный прототип консольной игры.

Введение

Продолжаем ЛР2, предоставляя пользователю новые ~~проблемы~~ возможности для игры. В рамках этой ЛР добавляем свой собственный редактор карт, а также вводим дополнительную экономическую составляющую нашей игры. Технически будет необходимо использовать generics, а также более широко использовать абстрактные классы и интерфейсы.

Задание

Часть 1

В рамках части 1 необходимо разработать редактор карт с возможностью их сохранения, а также модифицировать ЛР2 так, чтобы пользователь мог запускать битвы на ранее созданных картах.

Необходимо создать отдельный проект, в котором нужно предложить пользователю интерфейс создания своих карт. При этом, в самом интерфейсе пользователя ограничений особых нет. Процесс создания карты может быть реализован любым удобным для вас способом, однако, есть некоторые технические требования, представленные в таблице ниже.

Требование	Суть
Сохранение в файл	Запрашивается имя файла для сохранения
Редактирование ранее созданных карт	Запрашивается имя файла для загрузки и предоставляется интерфейс для изменения карты
Добавление новых видов препятствий и обозначений	Нужен интерфейс, добавляющий новый тип препятствия с новой фигуркой и новым механизмом штрафов
Удаление карты	Запрашивается имя файла для удаления и удаляется карта

Сразу несколько комментариев, которые касаются создания и редактирования. При создании можно просто попросить человека вводить символы с клавиатуры построчно, тем самым, рисуя карту. Можно и любым другим способом – например, «введите координаты таких-то препятствий», а потом все остальное заполнить общими клетками. Что касается редактирования – тоже не советовал бы делать излишне сложный интерфейс, достаточно просто запросить координаты клетки, которую пользователь хочет изменить и то, на что ее можно изменить.

Что касается файлов, также, возможны некоторые допущения. Например, вы можете все сохранять в одной директории.

В основной программе необходимо добавить пункт «Загрузить карту», в рамках которого пользователю выводится список из названных, созданных ранее, им карт. При выборе карта загружается из файла, со всеми модификациями.

Часть 2

Добавим в наш замечательный, надеюсь, искренне любимый всеми вами после ЛР2, Город экономику и некоторые здания с возможностью их модификации. Остановимся подробнее на каждом из них, рассмотрим в таблице ниже.

Здание	Суть
Дом лекаря	В Доме лекаря исследуются новые алхимические элементы. Цель – повышение HP юнитов
Таверна	В Таверне проходят различные праздники. Цель – юниты получают повышенные очки перемещения или уменьшение штрафов
Кузница	В Кузнице живет Кузнец. Цель – увеличение атаки юнитов
Арсенал	В Арсенале изготавливают новые виды брони. Цель – увеличение брони юнитов
Академия	В Академии лучшие умы города, тактики и стратеги, разрабатывают новейшие приспособления для различных задач. Цель – создание новых Юнитов
Рынок	Рынок – удивительное место. Чего только тут нет. Однако, мы ограничимся – будем менять тут

	Ресурсы, требуемые для постройки зданий
Ремесленная мастерская	Ремесленники любят Ваш город почти также сильно, как и Вы сами. Они с удовольствием поселятся у Вас и будут платить некоторую сумму золотыми за аренду

Первоначально Город абсолютно пуст, никаких Зданий нет. Каждое здание стоит не деньги, а некоторое количество Ресурсов. Условимся, что ресурсов всего 2 вида – Камень и Дерево. Первоначально пусть у Игрока будут они в некотором количестве. Каждого здания игрок может купить только 1 шт, кроме Ремесленных мастерских, их может быть до 4-х штук.

Ресурсы добываются в рамках сражений, в некотором количестве они падают с убитых врагов, а также большое количество каждого вида ресурса Игрок получает за победу в Сражении. Далее, идея чрезвычайно проста. До начала каждой битвы Игрок может приобрести какое-либо Здание или улучшить его. Для зданий Дом лекаря, Кузница, Арсенал механика идентична. При постройке этих зданий Игрок получает +1 к одной из характеристик всех Юнитов, при их улучшении – еще +1. Например, Кузница 3го уровня дает +3 к атаке.

Таверна работает сходственным образом, но Игрока спрашивают, какую характеристику он хочет изменить. Если он хочет увеличить перемещение – пусть 1 уровень Таверны дает +0.5 очков. Если он хочет снизить штраф – пусть все штрафы за 1 уровень снижаются на 0.5. Одновременно за 1 уровень получить оба преимущества нельзя. Например, у Игрока есть Таверна 3го уровня. На 1м уровне он выбрал перемещение, на

втором – перемещение, на третьем – штраф. В итоге, у него перемещение для всех юнитов увеличилось на 1, а штрафы для всех уменьшились на 0.5.

Академия в прокачке не нуждается. При постройке игрока просят задать параметры будущего Юнита. На основании параметров и жутко сложной формулы (придумайте сами или возьмите из физики) считается цена в монетах, необходимая для разработки нового юнита. Если денег хватает и пользователь заказывает исследование такого Юнита – он может его нанять перед следующей битвой.

Рынок также не нуждается в прокачке. Это просто здание, в котором можно выгодно (или нет, ведь курс меняется каждый день) обменять Дерево на Камень, Камень на Дерево или же что-то из этого на монеты. Короче, рынок как рынок 😊.

Важный технический аспект. Будем считать, что у Игрока есть своего рода «учетная запись» и его прогресс сохраняется. При повторном запуске игры все уровни зданий должны быть сохранены. Также, новые типы Юнитов, если они были исследованы в Академии, тоже должны быть сохранены.

Требования

Общие требования аналогичны ЛР2 – стоит избегать публичных полей, все поля должны иметь модификатор доступа private, обращение и доступ к ним должен осуществляться посредством геттеров и сеттеров.

Таблица 4. Технические требования

Суть	Требование
Хранение данных	Использование объектов и классов
Хранение массивов данных	Использование коллекций, обоснованный выбор типа коллекции

Реализация функционала	1 задача = 1 маленький метод, избегать длинных методов, выполняющих большое количество действий
Хранение сущностей(Меню, Поле битвы и тд)	Использование объектов и классов
Дублирование кода	По максимуму избегание дублирования, использование механизмов ООП
Алгоритмы	Использование готовых решений или своих, но с обязательным обоснованием

Под использованием механизмов ООП следует понимать использование основных принципов, ведущих к снижению связности кода и уменьшению его дубляжа. Каждое свое решение студент должен быть в состоянии обосновать.

Также, очень настоятельно рекомендуется использовать некоторые идеи generics. Это позволит вам существенно сократить код и сделать его значительно более элегантным.

Глобальная идея таблицы сводится к тому, что каждая сущность программы должна быть представлена в виде отдельного класса. Каждое действие, которое может совершать сущность, представлено в виде отдельного метода.

С точки зрения бизнес требований, функционал должен быть удобным настолько, насколько это возможно в рамках консоли☺. Можно использовать возможности кастомизации консоли, почитайте об этом самостоятельно при желании.

Тестирование работоспособности

Важный момент с точки зрения оптимизации тестирования программы и защиты индивидуального задания лабораторной работы. Необходимо реализовать класс и/или метод, который при старте программы

автоматически инициализирует всех юнитов игрока и Бота. Например, при запуске такого метода игрок автоматически получает 2-х Мечников и Рыцаря, а Бот – Арбалетчика и Копейщика.

Также, с целью ускорения тестирования предлагается сократить размер поля, например, до 10x10, а также ограничить количество юнитов, чтобы игра была быстрее.

Важное замечание – для ускорения проверки **необходимо** создать сверхюнита – условного супер-лучника с большой атакой, большой дальностью. Это позволит очень быстро проверить работоспособность программы, корректность окончания игры и тд.