Санкт-Петербургский Национальный Исследовательский Университет ИТМО
Факультет Программной Инженерии и Компьютерной Техники

Вариант № 3109701.3
Лабораторная работа № 3-4
По дисциплине
Программирование

Выполнил студент группы P3109:
Михальченков Александр

Преподаватель:
Наумова Надежда Александровна

Санкт-Петербург 2025 г.

1. Текст задания

Лабораторная работа #3-4

В соответствии с выданным вариантом на основе предложенного текстового отрывка из литературного произведения создать объектную модель реального или воображаемого мира, описываемого данным текстом. Должны быть выделены основные персонажи и предметы со свойственным им состоянием и поведением. На основе модели написать программу на языке Java.

Введите вариант: 3109701.3

Описание предметной области, по которой должна быть построена объектная модель:

> Вскоре туман рассеялся, и Скуперфильд обнаружил, что шагает по рыхлой земле, усаженной какими-то темно-зелеными, ломкими кустиками, достигавшими ему до колен. Выдернув из земли один кустик, он увидел несколько прицепившихся к корням желтоватых клубней. Осмотрев клубни внимательно, Скуперфильд начал догадываться, что перед ним самый обыкновенный картофель. Впрочем, он далеко не был уверен в своей догадке, так как до этого видел картофель только в жареном или вареном виде и к тому же почему-то воображал, что картофель растет на деревьях. Отряхнув от земли один клубень, Скуперфильд откусил кусочек и попробовал его разжевать. Сырой картофель показался ему страшно невкусным, даже противным. Сообразив, однако, что никто не стал бы выращивать совершенно бесполезных плодов, он сунул вытащенные из земли полдесятка картофелин в карман пиджака и отправился дальше. Шагать по рыхлой земле, беспрерывно путаясь ногами в картофельной ботве, было очень утомительно. Скуперфильд на все лады проклинал коротышек, вздумавших, словно ему назло, взрыхлить вокруг землю и насадить на его пути все эти кустики.

Этапы выполнения работы:

1. Получить вариант
2. Нарисовать UML-диаграмму, представляющую классы и интерфейсы объектной модели и их взаимосвязи;
3. Придумать сценарий, содержащий действия персонажей, аналогичные приведенным в исходном тексте;
4. Согласовать диаграмму классов и сценарий с преподавателем.
5. Написать программу на языке Java, реализующую разработанные объектную модель и сценарий взаимодействия и изменения состояния объектов. При запуске программа должна проигрывать сценарий и выводить в стандартный вывод текст, отражающий изменение состояния объектов, приблизительно напоминающий исходный текст полученного отрывка.
6. Для одного из методов класса персонажа (или объекта), согласованного с преподавателем, создать промпты для 2-3 ИИ-ассистентов (Gigachat, DeepSeek, Qwen, или др.), получить от них код метода, **реализовать метод самостоятельно**, и включить в отчет сравнительный анализ кода, написанного ИИ-ассистентами, и своего варианта. Оценить качество кода, полученного от ИИ-ассистентов, указать на возможные недоработки.
7. Продемонстрировать выполнение программы на сервере `helios`.
8. Ответить на контрольные вопросы и выполнить дополнительное задание.

Текст, выводящийся в результате выполнения программы не обязан дословно повторять текст, полученный в исходном задании. Также не обязательно реализовывать грамматическое согласование форм и падежей слов выводимого текста.

Стоит отметить, что цель разработки объектной модели состоит не в выводе текста, а в эмуляции объектов предметной области, а именно их состояния (поля) и поведения (методы). Методы в разработанных классах должны изменять состояние объектов, а выводимый текст должен являться побочным эффектом, отражающим эти изменения.

Требования к объектной модели, сценарию и программе:

1. В модели должны быть представлены основные персонажи и предметы, описанные в исходном тексте. Они должны иметь необходимые атрибуты и характеристики (состояние) и уметь выполнять свойственные им действия (поведение), а также должны образовывать корректную иерархию наследования классов.
2. Объектная модель должна реализовывать основные принципе ООП - инкапсуляцию, наследование и полиморфизм. Модель должна соответствовать принципам SOLID, быть расширяемой без глобального изменения структуры модели.
3. Сценарий должен быть вариативным, то есть при изменении начальных характеристик персонажей, предметов или окружающей среды, их действия могут изменяться и отклоняться от базового сценария, приведенного в исходном тексте. Кроме того, сценарий должен поддерживать элементы случайности (при генерации персонажей, при задании исходного состояния, при выполнении методов).
4. Объектная модель должна содержать как минимум один корректно использованный элемент каждого типа из списка:
    - абстрактный класс как минимум с одним абстрактным методом;
    - интерфейс;
    - перечисление (enum);
    - запись (record);
    - массив или ArrayList для хранения однотипных объектов;
    - проверяемое исключение.
5. В созданных классах основных персонажей и предметов должны быть корректно переопределены методы `equals()`, `hashCode()` и `toString()`. Для классов-исключений необходимо переопределить метод `getMessage()`.
6. Созданные в программе классы-исключения должны быть использованы и обработаны. Кроме того, должно быть использовано и обработано хотя бы одно unchecked исключение (можно свое, можно из стандартной библиотеки).
7. При необходимости можно добавить внутренние, локальные и анонимные классы.

Содержание отчёта по работе:

1. Текст задания.
2. Диаграмма классов объектной модели.
3. Исходный код программы (можно в виде ссылки на репозиторий).
4. Результат работы программы.
5. Промпты для ИИ-ассистентов, примеры полученного кода, сравнительный анализ.
6. Выводы по работе.

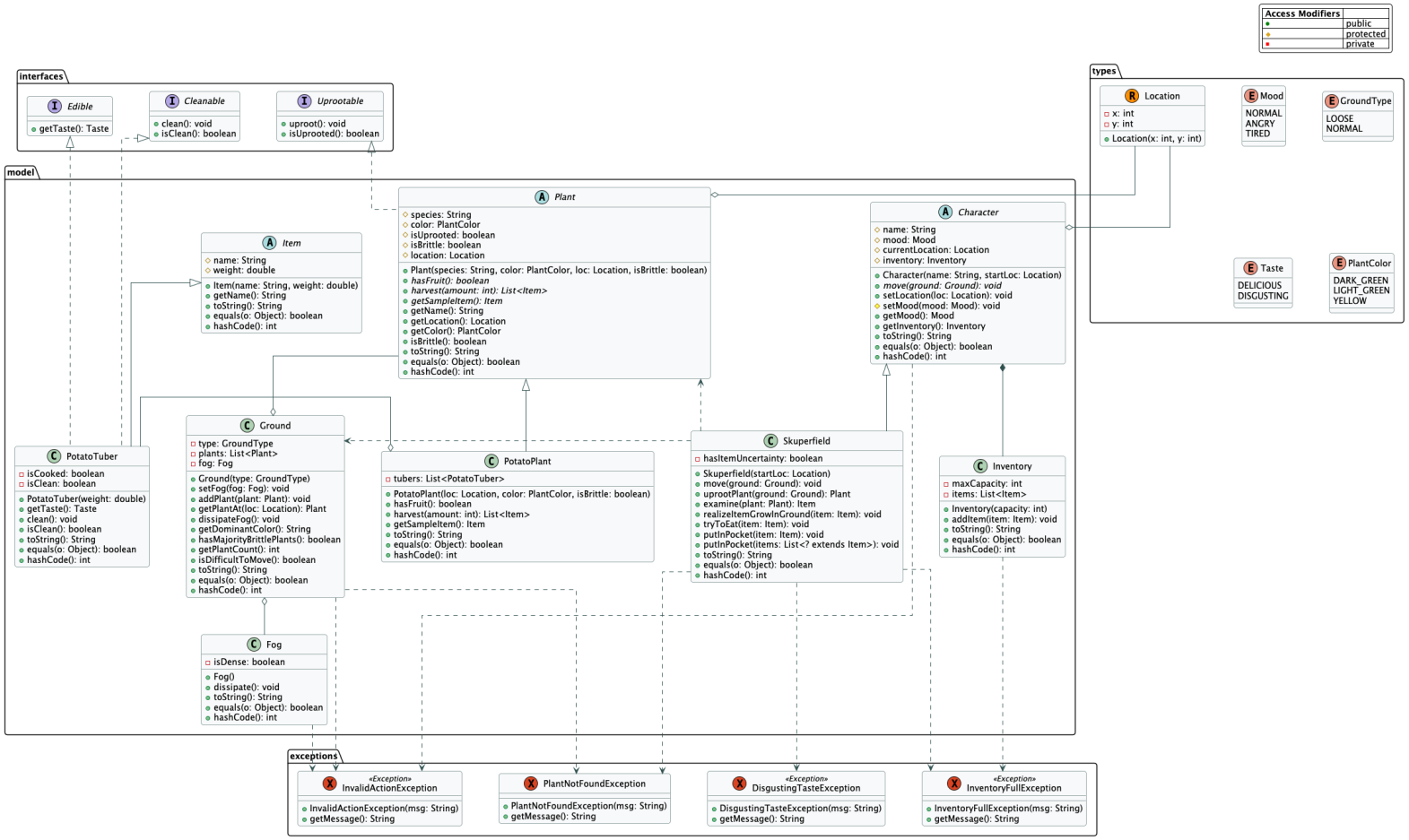
Вопросы к защите лабораторной работы:

1. Принципы объектно-ориентированного программирования SOLID и STUPID.
2. Класс `Object`. Реализация его методов по умолчанию.
3. Простое и множественное наследование. Особенности реализации наследования в Java.
4. Понятие абстрактного класса. Модификатор `abstract`.
5. Понятие интерфейса. Реализация интерфейсов в Java. Отличие интерфейсов от абстрактных классов.
6. Модификаторы `default`, `static` и `private` для методов интерфейса.
7. Перечисляемый тип данных (enum) в Java. Особенности реализации и использования.
8. Тип запись (record) в Java. Особенности использования.
9. Методы и поля с модификаторами `static` и `final`.
10. Перегрузка и переопределение методов.
11. Обработка исключительных ситуаций, три типа исключений.
12. Стандартный массив и динамический массив (ArrayList). Основные различия.
13. Вложенные, локальные и анонимные классы.

2. Исходный код программы

Репозиторий на GitHub: https://github.com/mikhalexandr/itmo-cse-programming/tree/main/lab3-4

3. Диаграмма классов реализованной объектной модели



Ссылка на UML-диаграмму: https://github.com/mikhalexandr/itmo-cse-programming/blob/main/lab3-4/docs/diagram.png

4. Результат работы программы

```
~/IdeaProjects/itmo-cse-programming/lab3-4 main !2
task run
Туман рассеялся.
Скуперфильд обнаружил, что шагает по рыхлой земле, усаженной какими-то тёмно-зелёными, ломкими кустиками, достигавшими ему до колен.
Выдернув из земли куст, он увидел что-то на корнях.
Скуперфильд внимательно осмотрел куст. Он начал догадываться, что перед ним самый обыкновенный картофель.
Впрочем, он далеко не был уверен в своей догадке, так как до этого видел картофель только в жареном или вареном виде и к тому же почему-то воображал, что картофель растёт на деревьях.
Отряхнув от земли картофель, Скуперфильд откусил кусочек и попробовал его разжевать. Какая гадость, сырой картофель ужасен на вкус.
Сообразив, что никто не стал бы выращивать совершенно бесполезных плодов, он сунул вытащенные из земли 5 шт. [картофель] в карман пиджака.
Наш герой отправился дальше.
Шагать по рыхлой земле, беспрерывно путаясь ногами в надоедливых кустиках, было очень утомительно.
Скуперфильд на все лады проклинал коротышек, вздумавших, словно ему назло, взрыхлить вокруг землю и насадить на его пути все эти кустики.
```

5. Инженерия промптов и анализ результатов ИИ-ассистентов

Промпты для ИИ-ассистентов

- Промпт для GPT-5.1

Ты выступаешь в роли Java-разработчика.

Необходимо реализовать метод tryToEat(Item item) для класса персонажа в объектно-ориентированной модели.

Item - базовый класс предметов, имеет метод getName().

Edible - интерфейс съедобных предметов, содержит метод getTaste().

Taste - перечисление со значениями, включая DISGUSTING.

Cleanable - интерфейс предметов, которые могут быть загрязнены, с методами isClean() и clean().

DisgustingTasteException - пользовательское проверяемое исключение.

Если предмет не является съедобным, персонаж должен вывести сообщение о неудачной попытке поесть и завершить выполнение метода.

Если предмет можно очистить и он загрязнён, необходимо выполнить очистку.

Персонаж должен попытаться откусить предмет и попробовать его на вкус.

Если вкус предмета является отвратительным, метод должен выбросить исключение DisgustingTasteException.

В остальных случаях необходимо вывести сообщение об успешной попытке.

Используй стандартный синтаксис Java без привязки к конкретной версии.

Реализация должна быть корректной с точки зрения компиляции.

- Промпт для Grok Code Fast 1

Сгенерируй Java-метод tryToEat(Item item) для игрового персонажа.

Метод моделирует попытку съесть предмет в рамках объектной модели.

Не все предметы являются съедобными.

Некоторые предметы могут быть загрязнены и поддерживают интерфейс Cleanable.

У съедобных предметов присутствует характеристика вкуса (Taste).

При отвратительном вкусе необходимо выбрасывать исключение DisgustingTasteException.

Метод должен выводить текстовые сообщения о действиях персонажа.

В случае несъедобного предмета выполнение должно корректно завершаться.

Код должен быть простым и читаемым.

Не используй внешние библиотеки.

Реализация должна быть сосредоточена в одном методе.

Код, сгенерированный ИИ-ассистентами

- Реализация, полученная от GPT-5.1

java Копировать код

```java
public void tryToEat(Item item) throws DisgustingTasteException {
    if (!(item instanceof Edible)) {
        System.out.println("Персонаж попытался съесть предмет, но он несъедобен.");
        return;
    }

    Edible food = (Edible) item;

    if (item instanceof Cleanable) {
        Cleanable c = (Cleanable) item;
        if (!c.isClean()) {
            c.clean();
        }
    }

    System.out.println("Персонаж попробовал откусить кусочек.");

    if (food.getTaste().equals(Taste.DISGUSTING)) {
        throw new DisgustingTasteException("Еда оказалась отвратительной на вкус.");
    }

    System.out.println("Еда оказалась съедобной.");
}
```

- Реализация, полученная от Grok Code Fast 1

```java
public void tryToEat(Item item) throws DisgustingTasteException {
    String name = item.getName();

    if (item instanceof Edible) {
        Edible food = (Edible) item;

        if (item instanceof Cleanable) {
            ((Cleanable) item).clean();
        }

        System.out.println("Скуперфильд съел " + name);

        if (food.getTaste() == Taste.DISGUSTING) {
            throw new DisgustingTasteException("Невкусно!");
        }
    } else {
        System.out.println("Это нельзя есть: " + name);
    }
}
```

- Собственная реализация метода

```java
public void tryToEat(Item item) throws DisgustingTasteException {  1 usage  mikhalexandr
    String itemName = item.getName();
    if (!(item instanceof Edible food)) {
        System.out.printf("Скуперфильд попытался укусить %s, но чуть не сломал зуб. Несъедобно.\n", itemName);
        return;
    }

    String text = "";
    if (item instanceof Cleanable cleanableItem) {
        if (!cleanableItem.isClean()) {
            cleanableItem.clean();
            text += "Отряхнув от земли " + itemName + ", ";
        }
    }
    text += "Скуперфильд откусил кусочек и попробовал его разжевать. ";
    if (food.getTaste() == Taste.DISGUSTING) {
        throw new DisgustingTasteException(text + "Какая гадость, сырой " + itemName + " ужасен на вкус.");
    }
    System.out.println(text + "Ммм, вполне съедобно.");
}
```

Сравнительный анализ

- GPT-5.1

Преимуществом данной реализации является корректная базовая логика и отсутствие синтаксических ошибок. Однако код использует явное приведение типов вместо современных возможностей Java (instanceof с привязкой переменной), а текстовые сообщения носят обобщённый характер и слабо отражают поведение конкретного персонажа.

- Grok Code Fast 1

Реализация отличается лаконичностью, однако содержит логические недочёты. Предмет очищается без проверки состояния загрязнённости, сообщение о поедании выводится до анализа вкуса, а исключение не содержит контекстной информации. Это может приводить к нарушению логики повествования.

- Собственная реализация

Собственный вариант использует современные языковые конструкции, последовательно описывает действия персонажа и формирует единый текст, который используется как для вывода, так и для сообщения об ошибке. Логика метода соответствует предметной области и требованиям задания.

Вывод

Использование ИИ-ассистентов позволяет быстро получить работоспособную заготовку метода, однако полученный код требует анализа и доработки. ИИ, как правило, не учитывает специфику предметной области и может допускать логические упрощения.

Наилучший результат достигается при осознанном применении ИИ, когда разработчик критически оценивает сгенерированный код и адаптирует его под архитектуру проекта. Таким образом, ИИ-ассистенты являются полезным инструментом, но не заменяют глубокого понимания языка программирования и принципов объектно-ориентированного проектирования.

6. Выводы по работе

В ходе лабораторной работы я разработал объектно-ориентированную модель предметной области на Java и создал UML-диаграмму классов. Я реализовал сценарий взаимодействия персонажей и предметов, использовав abstract classes, interfaces, enums, records и проверяемые исключения, а также реализовал обработку непроверяемых исключений. В коде были переопределены методы equals(), hashCode() и toString(), чтобы корректно сравнивать объекты и отображать их состояние. Программа отражает изменение состояния объектов и демонстрирует их поведение по мере выполнения сценария. При сравнении собственного кода с кодом, сгенерированным ИИ-ассистентами, стало видно, что ИИ может давать рабочие заготовки, но требует внимательной проверки и доработки. Полученный опыт помог лучше понять принципы объектно-ориентированного программирования и проектирования SOLID и будет полезен в будущем при разработке более сложных программных систем.