

## **Содержание**

Задание .....	2
Диаграмма классов объектной модели .....	3
Исходный код программы.....	4
Результат работы программы .....	4
Сравнение кода программы с вариантом ИИ-ассистентов.....	5
Вывод .....	8

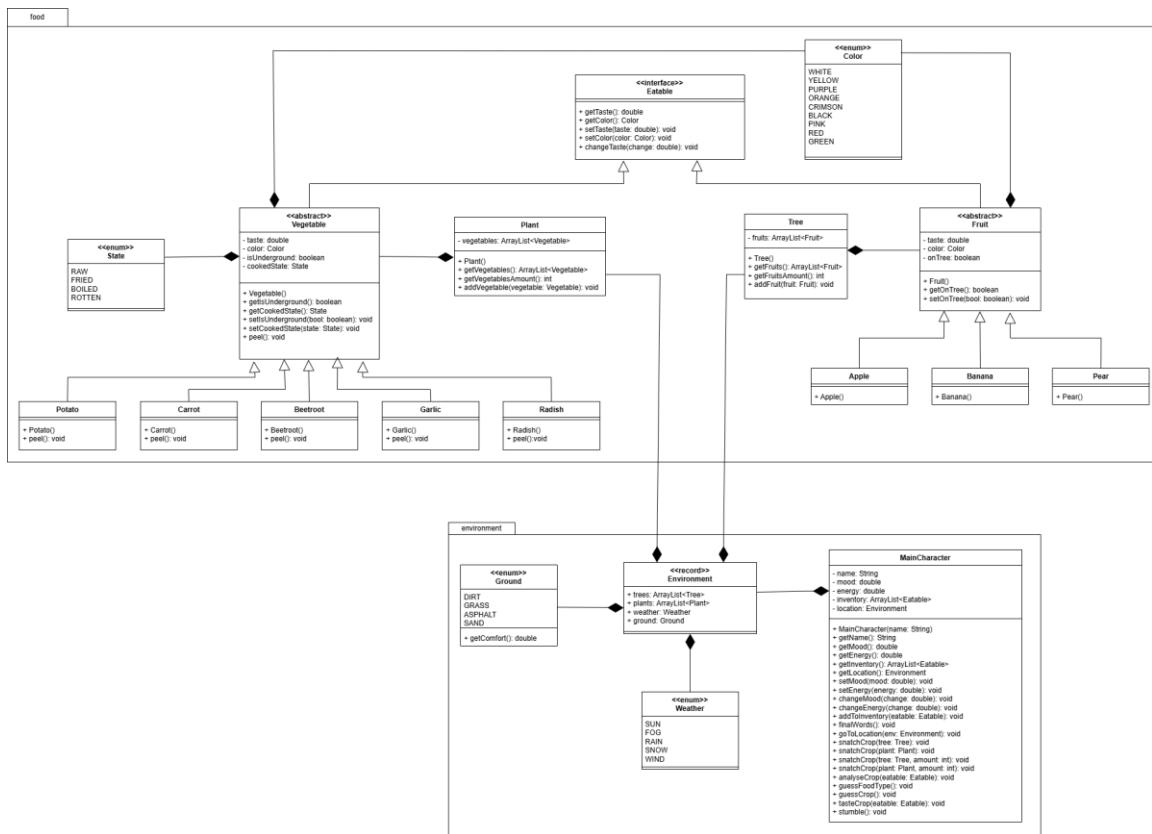
## **Задание**

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.

Данный текст:

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.

## Диаграмма классов объектной модели



## Исходный код программы

<https://github.com/LeterZP/ITMO/tree/main/Programming/ProgLab3> (ссылка на репозиторий)

## Результат работы программы

Из-за большого количества случайных переменных в программе вывод для каждого запуска может отличаться. Вот один из примеров вывода:

Суперфильд перемещается в новую локацию.  
 Суперфильд осматривается.  
 Суперфильд видит SNOW и DIRT.  
 Суперфильд не нравится идти по DIRT.  
 Суперфильд не замечает никаких посевов.  
 Суперфильд перемещается в новую локацию.  
 Суперфильд осматривается.  
 Суперфильд видит SUN и DIRT.  
 Суперфильд радуется SUN.  
 Суперфильд не нравится идти по DIRT.  
 Суперфильд замечает растение с чесноком.  
 Суперфильд срывает чеснок.

Скуперфильд пробует чеснок.  
Скуперфильд недоволен.  
Скуперфильд перемещается в новую локацию.  
Скуперфильд осматривается.  
Скуперфильд видит WIND и ASPHALT.  
Скуперфильд нравится идти по ASPHALT.  
Скуперфильд не замечает никаких посевов.  
Скуперфильд перемещается в новую локацию.  
Скуперфильд осматривается.  
Скуперфильд устал.  
Скуперфильд слегка недоволен своим небольшим путешествием.

## Сравнение кода программы с вариантом ИИ-ассистентов

Я попросил 2 ИИ-ассистента, DeepSeek и GigaChat, написать код для одного из методов программы, самого нагруженного из методов lookAround(). В качестве промпта я вводил следующее сообщение:

“Я пишу программу на языке java, основная суть которой - создание вариативной истории на основе объектной модели. Но я не могу написать один метод в классе главного персонажа, который бы объединил логику исследования локации в одном методе. Сначала этот метод должен слегка изменить настроение персонажа в зависимости от погоды и поверхности окружения. После этого он должен взять случайное дерево или растение, и для него вызвать метод snatchCrop(), при этом обработав проверяемое исключение, выдаваемое методом. Пожалуйста, помоги мне написать этот метод.”

А далее, после данного сообщения, я приводил часть кода программы, в котором содержались все классы пакета environment и класс MainCharacter, кроме самого метода, который необходимо написать.

ИИ-ассистент DeepSeek отлично понял задание, и написал неплохой код, приведенный ниже:

```
public void lookAround(Environment env) throws BadVisibilityException {  
    double moodChange = calculateMoodChange(env.weather(), env.ground());  
    this.changeMood(moodChange);  
    System.out.printf("%s оглядывается. Настроение изменено на %+.2f.%n",  
        this.getName(), moodChange);  
  
    if (env.weather() == Weather.FOG || env.weather() == Weather.SNOW) {  
        throw new BadVisibilityException("Слишком плохая видимость, чтобы  
        осматриваться.");  
    }  
  
    Object selectedCrop = selectRandomCrop(env);  
    if (selectedCrop == null) {
```

```

        System.out.println(this.getName() + " не нашёл ничего, что можно сорвать.");
        return;
    }

    try {
        if (selectedCrop instanceof Tree) {
            snatchCrop((Tree) selectedCrop);
        } else if (selectedCrop instanceof Plant) {
            snatchCrop((Plant) selectedCrop);
        }
    } catch (EmptyCropException e) {
        System.out.println(this.getName() + " пытался что-то сорвать, но на " +
                (selectedCrop instanceof Tree ? "дереве" : "растении") +
                " ничего не оказалось.");
    }
}

private double calculateMoodChange(Weather weather, Ground ground) {
    double weatherEffect = switch (weather) {
        case SUN -> 0.10;
        case FOG, SNOW -> -0.08;
        case RAIN -> -0.05;
        case WIND -> -0.03;
    };
    double groundEffect = (ground.getComfort() - 0.5) * 0.2;
    return weatherEffect + groundEffect;
}

private Object selectRandomCrop(Environment env) {
    ArrayList<Object> allCrops = new ArrayList<>();
    allCrops.addAll(env.trees());
    allCrops.addAll(env.plants());

    if (allCrops.isEmpty()) {
        return null;
    }
    int index = (int) (Math.random() * allCrops.size());
    return allCrops.get(index);
}

```

Из положительных моментов данного кода можно выделить то, что для более понятной структуры было создано два `private`-метода, хоть это и противоречило условию в промпте. Но, помимо этого, также можно выделить и отрицательные моменты данного кода. К примеру, в методе `selectRandomCrop()`, во-первых, создается массив с элементами `Object` там, где этого можно избежать, что не очень практично, и, во-вторых, создает новый массив, соединяя первые два, что является неоптимальным использованием памяти. Также, из незначительных ошибок, присутствуют “магические числа” для определения изменения настроения, и возвращение `null` в методе `selectRandomCrop()`, что не очень практично.

В качестве второго ИИ-ассистента я использовал GigaChat с тем же промптом, и вот какой результат получил:

```
public void explore() {
    Environment currentEnv = this.getLocation();

    switch(currentEnv.weather) {
        case SUN:
            this.changeMood(+0.1); break;
        case FOG:
            this.changeMood(-0.05); break;
        case RAIN:
            this.changeMood(-0.1); break;
        case SNOW:
            this.changeMood(-0.15); break;
        case WIND:
            this.changeMood(-0.05); break;
    }

    this.changeMood(currentEnv.ground.getComfort());

    if (!currentEnv.trees.isEmpty() || !currentEnv.plants.isEmpty()) {
        boolean chooseTree = Math.random() < 0.5 ? true : false;

        if ((chooseTree && !currentEnv.trees.isEmpty()) ||
            (!chooseTree && currentEnv.trees.isEmpty())) {
            Tree randomTree = currentEnv.trees.get((int)(Math.random() *
currentEnv.trees.size()));
        }

        try {
            snatchCrop(randomTree);
        } catch (EmptyCropException e) {
            System.out.println("У выбранного дерева нет плодов!");
        }
    }
}
```

```

} else {
    Plant randomPlant = currentEnv.plants.get((int)(Math.random() *
currentEnv.plants.size()));

    try {
        snatchCrop(randomPlant);
    } catch (EmptyCropException e) {
        System.out.println("У выбранного растения нет овощей!");
    }
}

} else {
    System.out.println("Здесь нечего исследовать, местность пуста.");
}
}

```

Сразу видно разницу между DeepSeek и GigaChat: второй хуже анализирует входные данные, и не смог выделить название метода из входных данных, а также не вник в суть задания, не сделав текстовые комментарии к действиям. Помимо этого, данный ИИ-ассистент не использует встроенные в класс геттеры и сеттеры, и обращается напрямую к полям, что в некоторых случаях может быть критической ошибкой. Также некоторые логические выражения были созданы неправильно, и при исполнении программы могут вызвать ошибку. В качестве последнего момента можно выделить два блока try-catch там, где можно было обойтись одним, более общим.

Таким образом, сравнивая методы, полученные от ИИ-ассистентов, и самописный метод, можно найти как положительные стороны, так и отрицательные у каждого из них. В некоторых случаях, самописный метод оказался лучше, чем код ИИ-ассистентов, в других же они предложили более простые, оптимизированные и понятные пути решения задачи. Иными словами, при правильном использовании они могут стать незаменимым инструментом для правки и улучшения кода, но полностью на них полагаться не стоит, ведь одна незначительная очевидная ошибка может сломать весь код.

## Вывод

Во время выполнения данной лабораторной работы была построена некая модель материального мира со своими предметами и персонажами, каждый из которых обладал неким поведением и состоянием. Помимо этого, были изучены некоторые принципы ООП, более подробно изучены принципы наследования и абстракции, а также частично разобраны исключения и методы их обработки. В конце работы также был проведен анализ кода, написанного ИИ-ассистентами, и указаны возможные недоработки.