

**Федеральное государственное автономное образовательное учреждение  
высшего образования**

**«Национальный исследовательский университет ИТМО»**



**Лабораторная работа #3**

**Вариант #30612**

По дисциплине

**Программирование**

Выполнил студент группы R3138

**Глазков В.Н.**

Преподаватель:

**Кобелев Р.П.**

г. Санкт-Петербург

2025 г.

## Текст задания

Нужно сказать, что подражание трем бывшим ослам не ограничивалось одной одеждой. Некоторые коротышки так усердствовали в соблюдении моды, что хотели во всем быть похожими на Калигулу, Брыкуна и Пегасика. Часто можно было видеть какого-нибудь коротышку, который часами торчал перед зеркалом и одной рукой нажимал на свой собственный нос, а другой оттягивал книзу верхнюю губу, добиваясь, чтобы нос стал как можно короче, а губа как можно длиннее. Были среди них и такие, которые, нарядившись в модные пиджаки и брюки, бесцельно шатались по улицам, никому не уступали дороги и поминутно плевались по сторонам. В газетах между тем иногда стали появляться сообщения о том, что где-нибудь кого-нибудь облили водой из шланга, где-нибудь кто-нибудь споткнулся о веревку и разбил себе лоб, где-нибудь в кого-нибудь бросили из окна каким-нибудь твердым предметом, и тому подобное.

## Диаграмма классов реализованной объектной модели

<https://github.com/RYAZHENKA-11/itmo-java-labs/blob/main/lab3-30612/uml.png>

## Исходный код программы

<https://github.com/RYAZHENKA-11/itmo-java-labs/tree/main/lab3-30612/src>

## Результат работы программы

>>> Запуск симуляции <<<

== ДЕНЬ 1 ==

--- Новый день в городе Цветочный ---

Незнайка не нашел подходящей жертвы.

Козлик бесцельно шатается по улицам, никому не уступая дороги и поминутно плюясь.

Винтик занимается полезным делом.

[Новости в конце дня]:

--- ГАЗЕТА: Неделя Моды ---

Новостей нет. В городе спокойно.

--- ГАЗЕТА: Криминальная Хроника ---

Новостей нет. В городе спокойно.

=== ДЕНЬ 2 ===

--- Новый день в городе Цветочный ---

Незнайка любит себя.

Строит рожи перед зеркалом: нос сжат на 0,9 мм, губа оттянута на 1,2 мм.

НОВОСТЬ!: Модный приговор

Винтик занимается полезным делом.

[Новости в конце дня]:

--- ГАЗЕТА: Неделя Моды ---

NewsReport[headline=Модный приговор, description=Винтик часами торчал перед зеркалом,  
type=FASHION\_VICTIM]

--- ГАЗЕТА: Криминальная Хроника ---

NewsReport[headline=Модный приговор, description=Винтик часами торчал перед зеркалом,  
type=FASHION\_VICTIM]

=== ДЕНЬ 3 ===

--- Новый день в городе Цветочный ---

Незнайка бесцельно шатается по улицам, никому не уступая дороги и поминутно плюясь.

Козлик любит себя.

Строит рожи перед зеркалом: нос сжат на 1,1 мм, губа оттянута на 1,2 мм.

Винтик занимается полезным делом.

[Новости в конце дня]:

--- ГАЗЕТА: Неделя Моды ---

Новостей нет. В городе спокойно.

--- ГАЗЕТА: Криминальная Хроника ---

Новостей нет. В городе спокойно.

## III

Для анализа был выбран метод use класса Hose (Шланг). Задача метода: Реализовать использование шланга одним персонажем против другого. С вероятностью 10% предмет должен сломаться (выбросить исключение), иначе — вернуть отчет о происшествии (NewsReport).

Промпт:

Напиши реализацию метода `public NewsReport use(Shorty user, Shorty target)` на Java для класса `Hose`, который наследуется от `AbstractItem`.

Логика:

1. С вероятностью 10% предмет ломается — нужно выбросить кастомное проверяемое исключение `ItemMalfunctionException` с сообщением об ошибке.
2. Если предмет сработал, вернуть объект `record NewsReport` с заголовком "Водная атака", описанием (кто кого облил) и типом инцидента `IncidentType.WATER_ATTACK`.

Вариант 1: DeepSeek (v2)

@Override

```
public NewsReport use(Shorty user, Shorty target) throws ItemMalfunctionException {  
    // Используем ThreadLocalRandom для лучшей производительности в многопоточности  
    if (java.util.concurrent.ThreadLocalRandom.current().nextDouble() < 0.1) {  
        throw new ItemMalfunctionException("Шланг лопнул в руках у " + user.getName());  
    }  
}
```

```

        String description = String.format("%s облил водой %s из шланга", user.getName(), target.getName());
        return new NewsReport("Водная атака", description, IncidentType.WATER_ATTACK);
    }

```

## Вариант 2: GigaChat (Pro)

```

public NewsReport use(Shorty user, Shorty target) throws ItemMalfunctionException {
    Random random = new Random();
    int chance = random.nextInt(100);

    if (chance < 10) {
        throw new ItemMalfunctionException("Не удалось использовать шланг: он оказался дырявым.");
    }

    return new NewsReport(
        "Водная атака",
        user.getName() + " успешно облил водой " + target.getName(),
        IncidentType.WATER_ATTACK
    );
}

```

## Вариант 3: Собственная реализация (из проекта)

```

@Override
public NewsReport use(Shorty user, Shorty target) throws ItemMalfunctionException {
    if (Math.random() < 0.1) {
        throw new ItemMalfunctionException("Шланг прохудился и намочил " + user.getName());
    }

    return new NewsReport(
        "Водная атака!",
        user.getName() + " облил водой из шланга коротышку по имени " + target.getName(),
        IncidentType.WATER_ATTACK
    );
}

```

## Анализ и оценка качества

### 1. Генерация случайных чисел

- DeepSeek: Использовал ThreadLocalRandom. Это более современный и производительный подход для многопоточных сред, однако для простой лабораторной работы это может быть избыточным усложнением (over-engineering).
- GigaChat: Создает новый экземпляр new Random() при каждом вызове метода. Это плохая практика, так как создание объекта Random ресурсоемко и при частых вызовах может привести к дублированию последовательностей, если вызовы происходят в одну миллисекунду.
- Собственная реализация: Использует Math.random(). Это статический потокобезопасный метод, который внутри использует один экземпляр Random. Для данной задачи это оптимальный по лаконичности вариант.

### 2. Форматирование строк

- DeepSeek: Использовал String.format. Это делает код чище и читаемее, чем конкатенация.
- GigaChat и Собственная реализация: Использовали конкатенацию строк (+). В Java современные компиляторы оптимизируют это через StringBuilder, так что с точки зрения производительности разницы нет, но вариант DeepSeek выглядит аккуратнее.

### 3. Обработка исключений

Все варианты справились с выбрасыванием исключения. Однако текст ошибки в собственной реализации ("Шланг прохудился и намочил...") более художественный и соответствует духу задания, в то время как GigaChat выдал слишком техническое сообщение ("Не удалось использовать...").

### 4. Возможные недоработки ИИ

1. GigaChat: Ошибка проектирования с созданием new Random() внутри метода.
2. Контекст: ИИ не видит всей структуры проекта. Например, DeepSeek предположил, что getName() есть у класса Shorty (это верно, но повезло), но мог бы попытаться использовать несуществующие методы, если бы промпт был менее детальным.

3. Стилль: ИИ часто добавляют лишние комментарии (как DeepSeek), которые засоряют код простых методов.

#### Итоговый вывод

Код, предложенный DeepSeek, оказался наиболее качественным с технической точки зрения (использование форматирования строк), но собственная реализация лучше вписывается в контекст простой лабораторной работы благодаря лаконичности (`Math.random()`) и корректному литературному описанию событий, требуемому по заданию. Код GigaChat содержит архитектурную пометку с инициализацией генератора случайных чисел.

#### Вывод

процессе выполнения лабораторной работы:

#### Изучены

- особенности работы с записями (`record`) для хранения неизменяемых данных;
- механизмы обработки исключительных ситуаций (`checked` и `unchecked` исключения).

#### Создана

- объектная модель мира, включающая иерархию персонажей (`Shorty`) и предметов (`AbstractItem`), а также среду их взаимодействия (`City`, `Newspaper`).

#### Реализованы

- классы предметов (например, `Hose`, `Rope`, `Mirror`), унаследованные от абстрактного класса `AbstractItem`, с полиморфной реализацией метода `use()`;
- интерфейс `Malleable`, описывающий способность частей лица (`Face`, `Lip`, `Nose`) к деформации;
- система генерации новостей через `NewsReport` и их трансляции в газеты.

#### Использованы

- перечисления (`enum`) `Lifestyle` и `IncidentType` для типизации состояний и событий;
- проверяемые исключения (`ItemMalfunctionException`) для реализации механики поломки предметов;

- коллекции (ArrayList) для хранения динамических списков граждан и предметов.

#### Организовано

- вариативное поведение персонажей в зависимости от их стиля жизни (Lifestyle), влияющее на выбор действий (самолюбование или хулиганство).

#### Разработана

- программа, симулирующая жизнедеятельность города коротышек, демонстрирующая случайные события, изменение социального рейтинга и последствия действий персонажей.

В результате была разработана работоспособная программа, соответствующая принципам SOLID и требованиям задания. Все цели лабораторной работы достигнуты.