**Урок 1. Принципы ООП: Инкапсуляция, наследование, полиморфизм**

При решении этих задач следуйте принципам абстракции, инкапсуляции, наследования, полиморфизма.

1. Создайте унаследованный класс ГорячийНапиток с дополнительным полем int температура.

2. Создайте класс АвтоматГорячихНапитков, реализующий интерфейс ТорговыйАвтомат и реализуйте перегруженный метод getProduct(int name, int volume, int temperature), выдающий продукт, соответствующий имени, объему и температуре.

3. В main проинициализируйте несколько ГорячихНапитков и АвтоматГорячихНапитков и позвольте покупателю купить товар.

**Урок 2. Принципы ООП Абстракция и интерфейсы. Пример проектирования**

Имеются данные о группе студентов. У каждого студента есть средний балл (например, 4.6).

Создайте родительский класс ЗаписывательВФайл. Создайте три детских класса, которые записывают в файл информацию о студентах в разных формах:

Просто текст:

Иван Иванов=4.8

Мария Кузнецова=5.0

Степан Кузьмин=3.6

JSON:

{

"Иван Иванов": 4.8,

"Мария Кузнецова": 5.0,

"Степан Кузьмин": 3.6

}

XML:

<?xml version="1.0" encoding="utf-8" ?>

<students>

<student>

<name>Иван Иванов</name>

<grade>4.8</grade>

</student>

<student>

<name>Мария Кузнецова</name>

<grade>5.0</grade>

</student>

<student>

<name>Степан Кузьмин</name>

<grade>3.6</grade>

</student>

</students>

**Урок 3. Некоторые стандартные интерфейсы Java и примеры их использования**

1. Класс Группа содержит в себе коллекцию Студентов. По группе можно перемещаться с помощью итератора. (Это повторяет 1-ю задачу из классной работы). Остальные задачи надстраиваются поверх неё.

2. Создайте класс GroupListIterator, который позволяет перемещаться по группе в обоих направлениях. Он реализует интерфейс ListIterator<Student>.

Добавьте в Группу метод listIterator (), который позволяет начать итерацию с помощью GroupListIterator.

3. Создайте класс ReverseIterator, который работает как итератор по группе в обратном направлении:

• Он отслеживает текущую позицию в переборе

• Он изначально выставлен на конечную позицию

• Он движется справа налево

Проверьте, как он работает.

**Урок 4. ООП: Обобщения. ч1**

1. Создайте интерфейс IsGood<T>. Внутри него содержится единственная функция:

boolean isGood (T item);

Смысл этого интерфейса: ему дают элемент, он его одобряет или не одобряет.

Создайте следующие детские классы:

• IsEven — ему дают целое число, он одобряет его, если оно чётное

• IsPositive — ему дают целое число, он одобряет его, если оно положительное

• BeginsWithA — ему дают строку, он одобряет её, если она начинается с буквы A

• BeginsWith — в конструкторе запоминает строку. Ему дают строку, он проверяет, что она начинается с того, что он запомнил

2. Создайте обобщённую функцию filter. Ей дают любую коллекцию любого типа, и одобрятель IsGood.

Функция возвращает новую коллекцию, куда входят только одобренные элементы из коллекции.

Продемонстрируйте, что это работает.

**Урок 5. От простого к практике**

1. Напишите программу, позволяющую анализировать посещаемость студентов. Используйте паттерн MVP.

2. Есть группа студентов. Для каждого студента есть журнал его посещаемости: список дат занятий и для каждой даты — посетил студент занятие или нет. Создайте класс AttendanceService (сервис посещаемости), в котором хранится информация обо всех студентах.

3. Создайте класс AttendanceView, который позволяет отображать студентов и их посещаемость.

4. Создайте класс presenter со следующими функциями:

• Распечатать всех студентов и посещаемость каждого в процентах

• Распечатать студентов, отсортировав их по убыванию посещаемости (вверху самые посещающие)

• Распечатать студентов с посещаемостью ниже 25%

Проверьте, как это работает.

***\*\* Комментарий ревьюрера:***

***Привет! Спасибо за сданную работу. Задание выполнено верно, но его можно улучшить:***

***1. Сделать поля всех классов приватными***

***2. Добавить геттеры и сеттеры, соответственно***

***3. В папке с моделями должны содержаться только модели***

Урок 6. ООП Дизайн и Solid

Задача 1. Чтобы разблокировать телефон, пользователь может выбрать один из способов:

• Без пароля

• С пин-кодом (4-значное число)

• По отпечатку пальца (кодируется строкой)

• По распознаванию лица (кодируется строкой)

Программист написал класс Unlocker, в котором хранятся поля от всех способов сразу:

class Unlocker {

private int mode;

// режим private int pin; // на случай пин-кода

private String fingerprint; // на случай отпечатка пальца

private String faceID; // на случай лица

}

Здесь нарушен принцип SRP: класс имеет несколько незаивисимых причин меняться.

Напишите решение, которое будет соответствовать SRP и OCP (мы хотим в будущем добавлять новые способы разблокировки).

Задача 2. Есть два самодельных класса коллекций:

ImmutableList<T> — коллекция, которая никогда ни при каких обстоятельствах не меняется. Методы:

• getSize ()

• get (int i)

MutableList<T> — коллекция, которая допускает изменения. Методы:

• getSize ()

• get (int i)

• set (int i, T newValue)

• add (T newValue)

• remove (T value)

Реализуйте такую схему наследования между двумя этими классами, которая будет соответствовать принципу подстановки Лисков.