**МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ**

**Ордена Трудового Красного Знамени**

**Федеральное государственное бюджетное образовательное учреждение высшего образования**

**«Московский технический университет связи и информатики»**

Кафедра «Математическая Кибернетика и Информационные технологии»

Дисциплина «Информационные технологии и программирование»

Лабораторная работа №7

«Многопоточность в Java»

Выполнила:

Студентка группы БВТ2303

Морозова Ольга

**Цель работы:**

Изучение многопоточности и применение полученных знаний на практике на языке программирования Java.

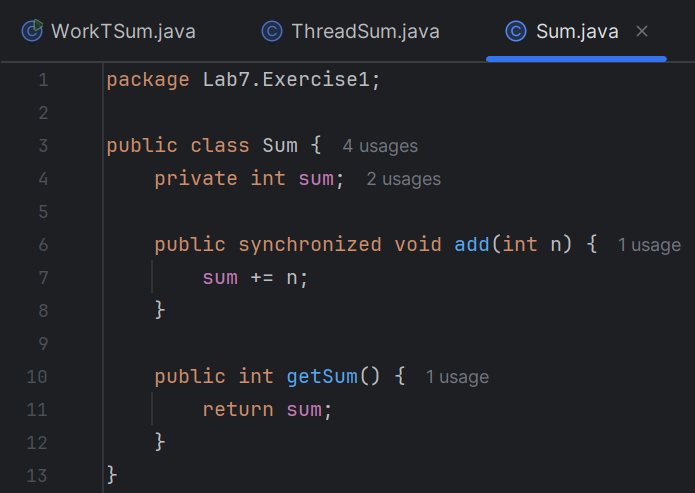
**Ход работы:**

Вариант 1.

Задание 1.

Напишем программу для вычисления суммы элементов массива.

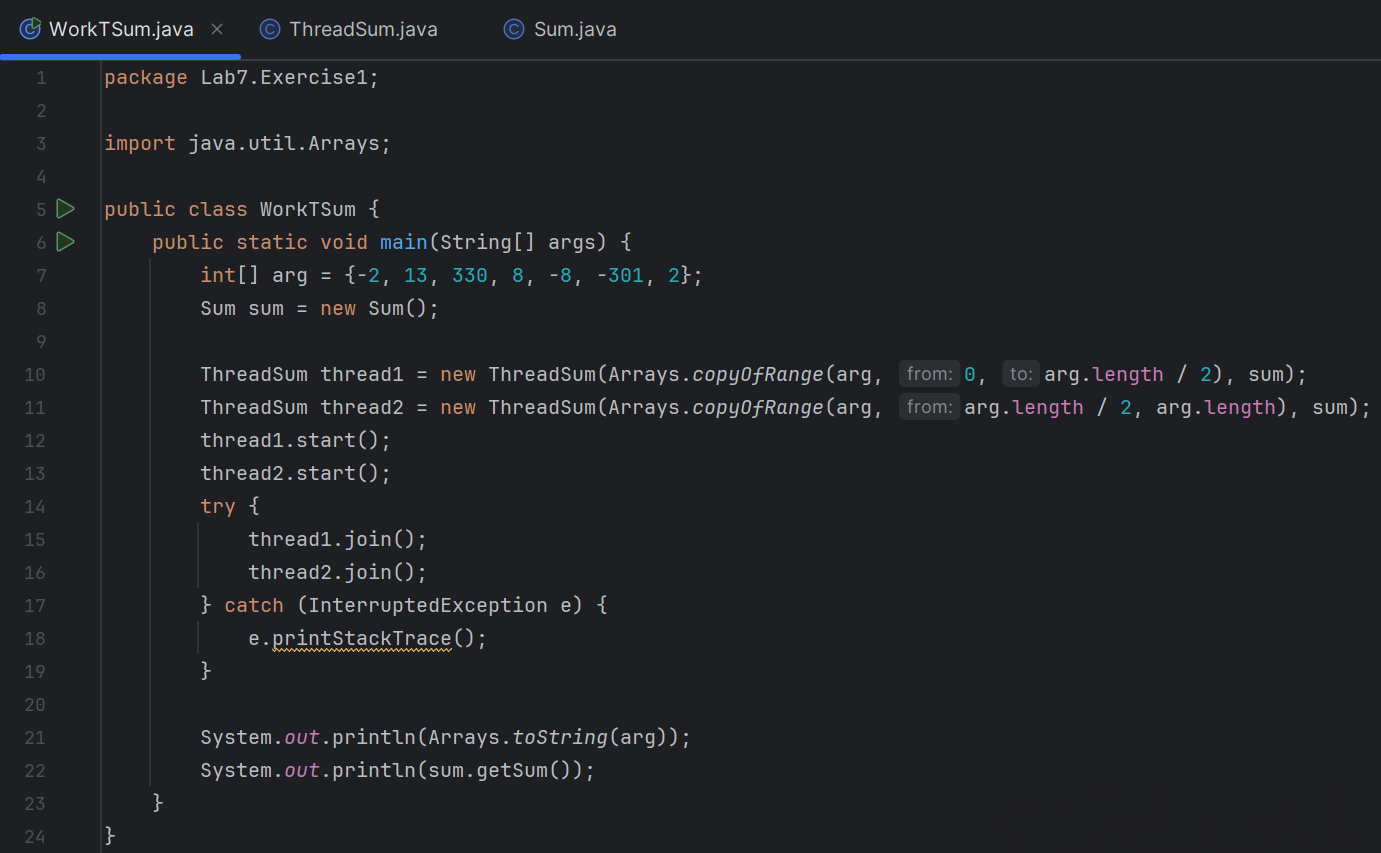
Для начала создадим класс Sum для хранения переменной суммы и создадим в нём синхронизирующийся метод, позволяющий прибавлять к переменной суммы число, а также метод для получения значения суммы.



После создадим класс ThreadSum, наследующий классу Thread, с помощью которого и будем создавать потоки. Укажем конструктор, а также переопределим метод run(), в котором и будем проходиться по элементам массива и прибавлять их значения к переменной суммы.



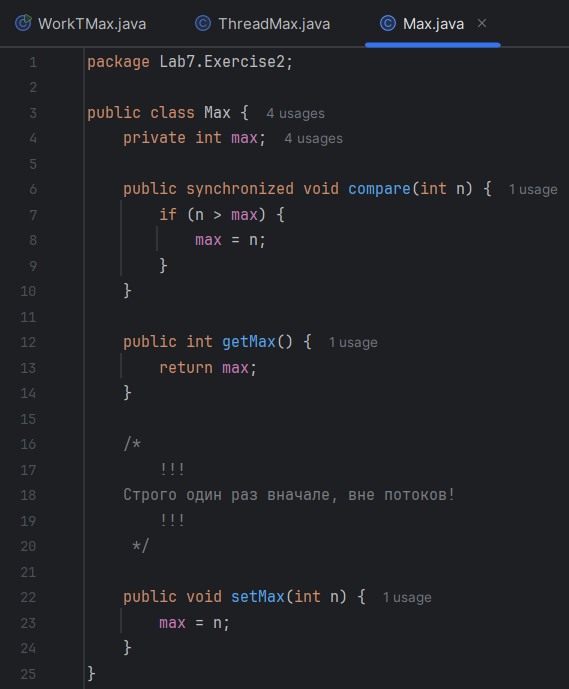
Наконец создадим основной класс, в котором и будем запускать потоки с помощью метода .strart(), а после завершения из работы сливать обратно методом .join().



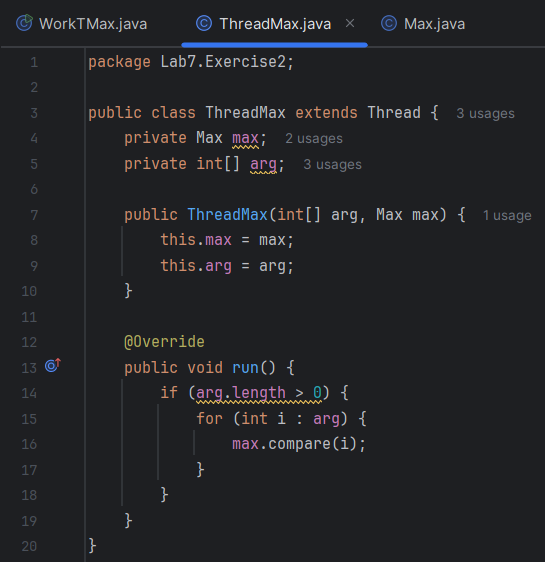
Задание 2.

Напишем программу для поиска наибольшего элемента в матрице.

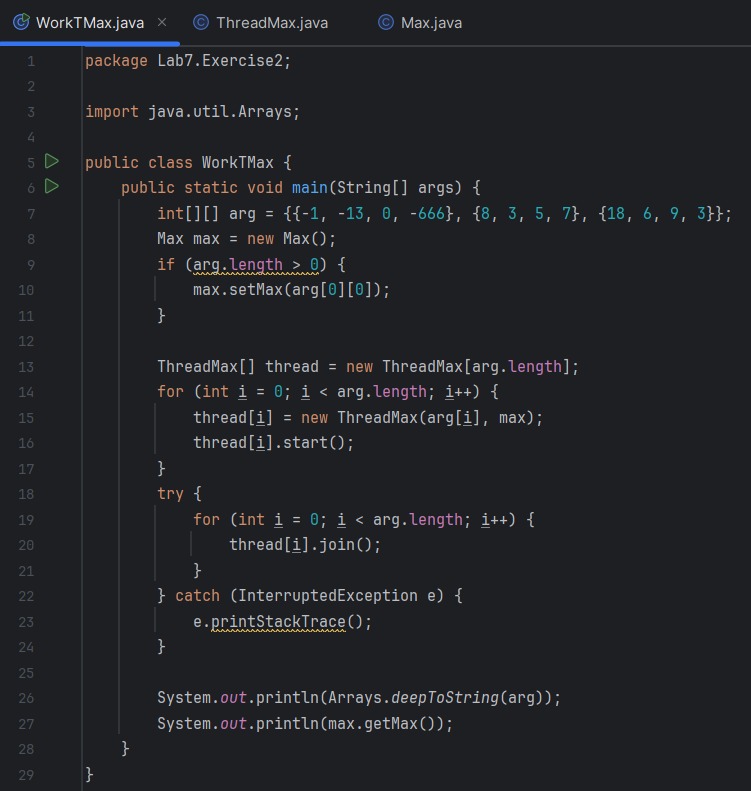
Для начала создадим класс Max для хранения переменной максимума и создадим в нём синхронизирующийся метод, в котором будем сравнивать значение с нынешним максимумом и при необходимости записывать его в переменную, а также геттер и сеттер.



После создадим класс ThreadMax, наследующий классу Thread, с помощью которого и будем создавать потоки. Укажем конструктор, а также переопределим метод run(), в котором и будем проходиться по элементам строки матрицы для нахождения максимума.



Наконец создадим основной класс, в котором и будем запускать потоки с помощью метода .strart(), а после завершения из работы сливать обратно методом .join().

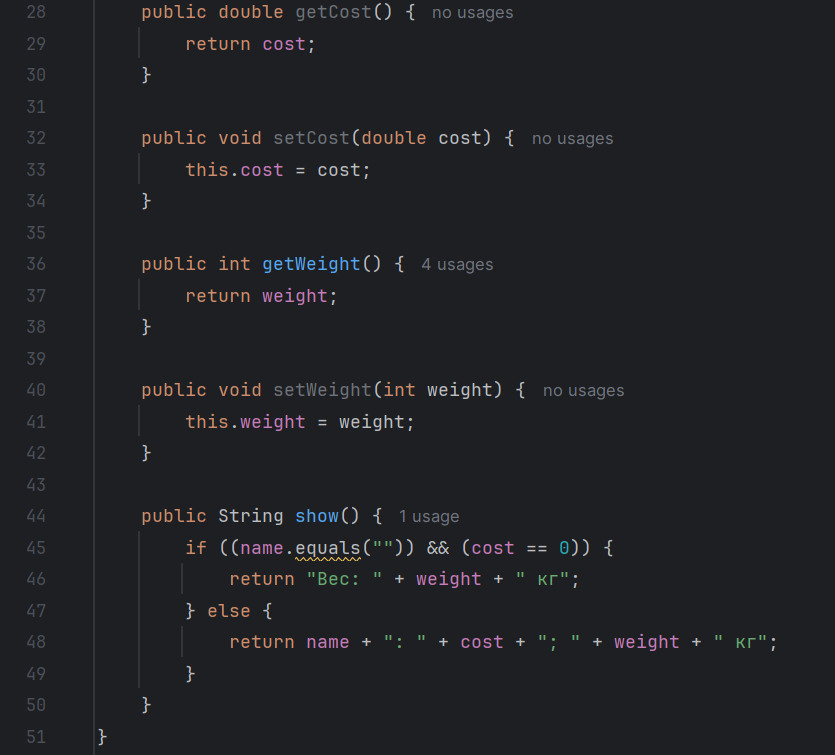


Задание 3.

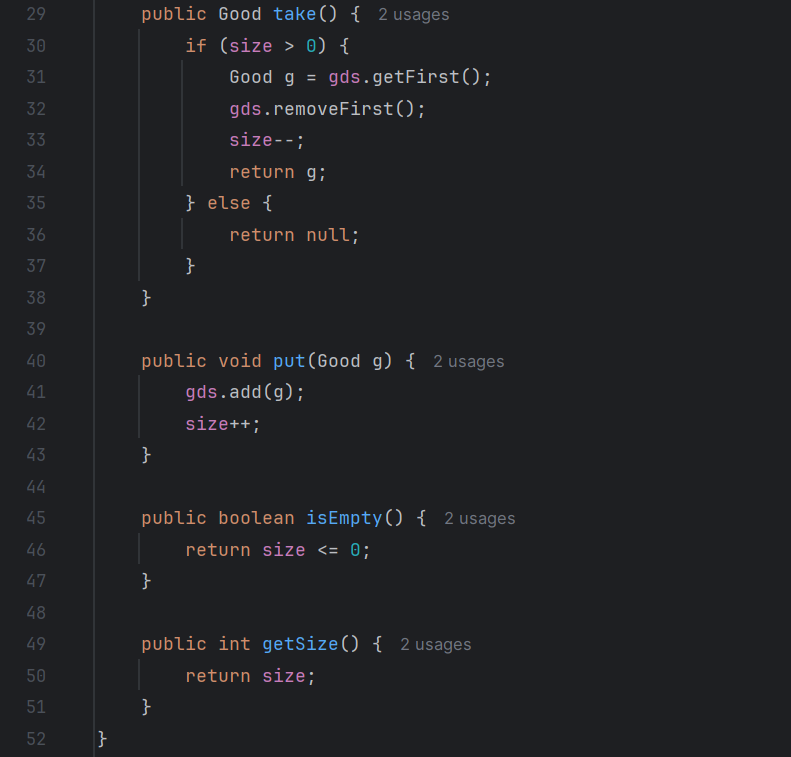
Напишем программу для реализации следующей задачи:

У вас есть склад с товарами, которые нужно перенести на другой склад. У каждого товара есть свой вес. На складе работают 3 грузчика. Грузчики могут переносить товары одновременно, но суммарный вес товаров, которые они переносят, не может превышать 150 кг. Как только грузчики соберут 150 кг товаров, они отправятся на другой склад и начнут разгружать товары.

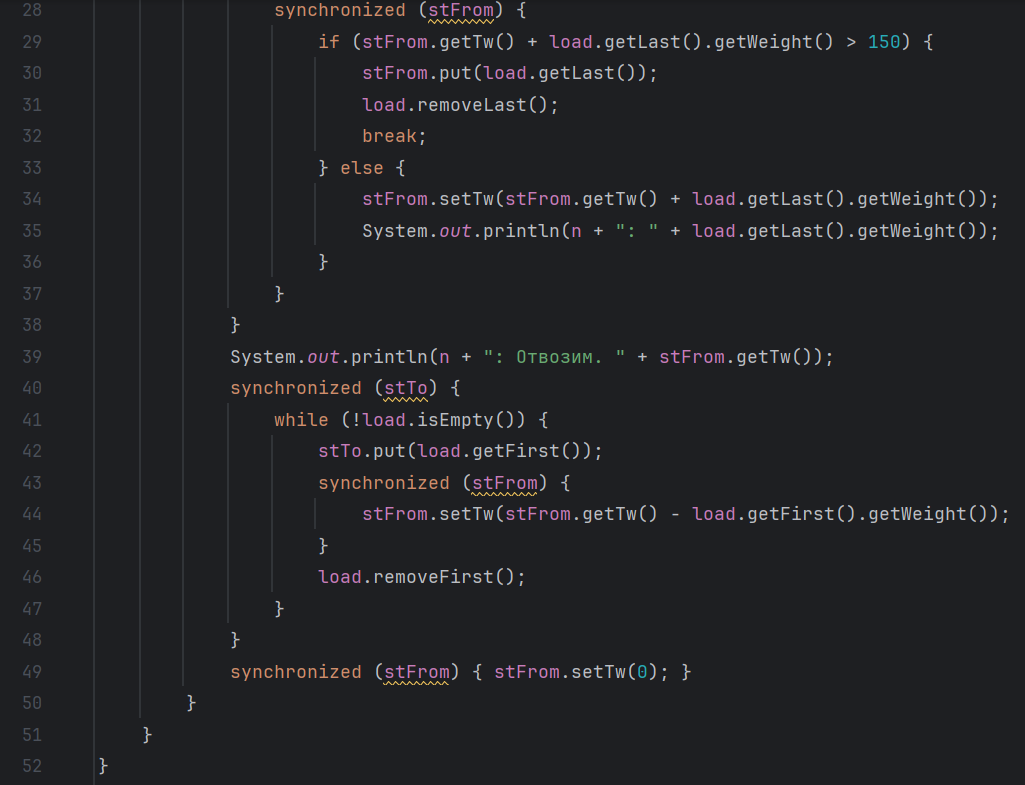
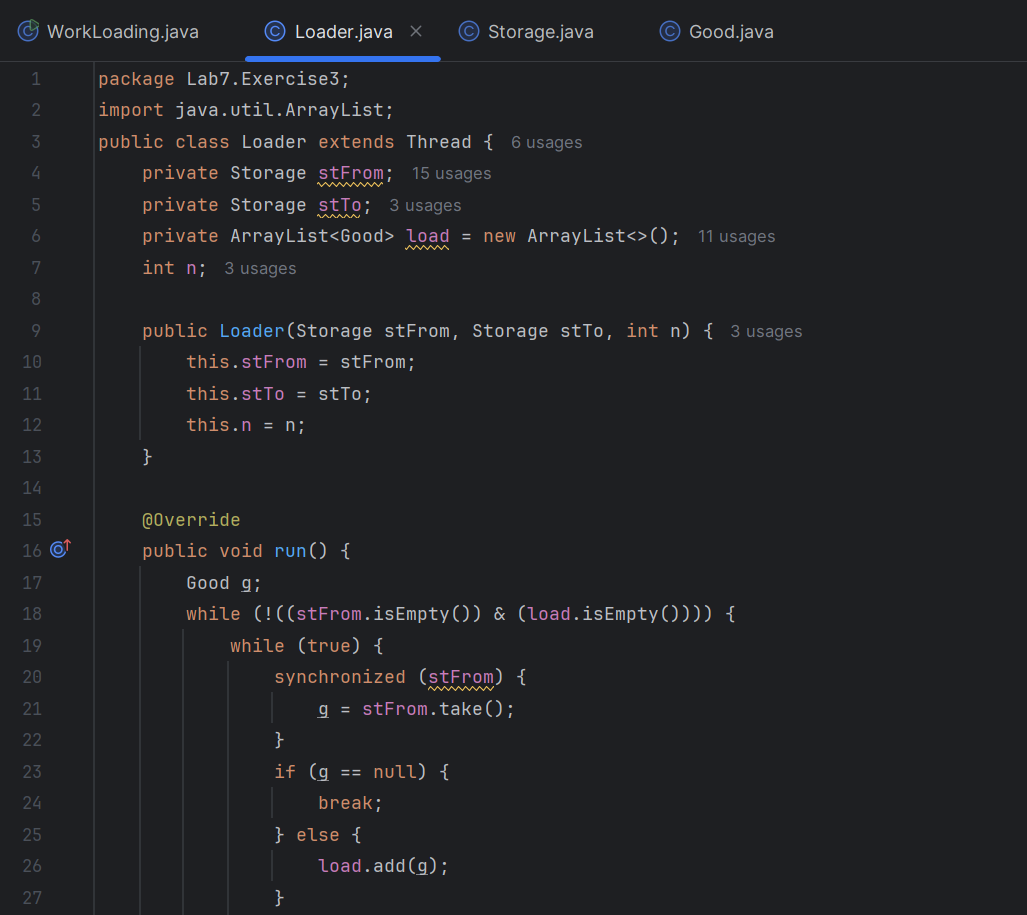
Для начала создадим класс Good для хранения информации о товаре (наименование, цена, вес) с геттерами и сеттерами.



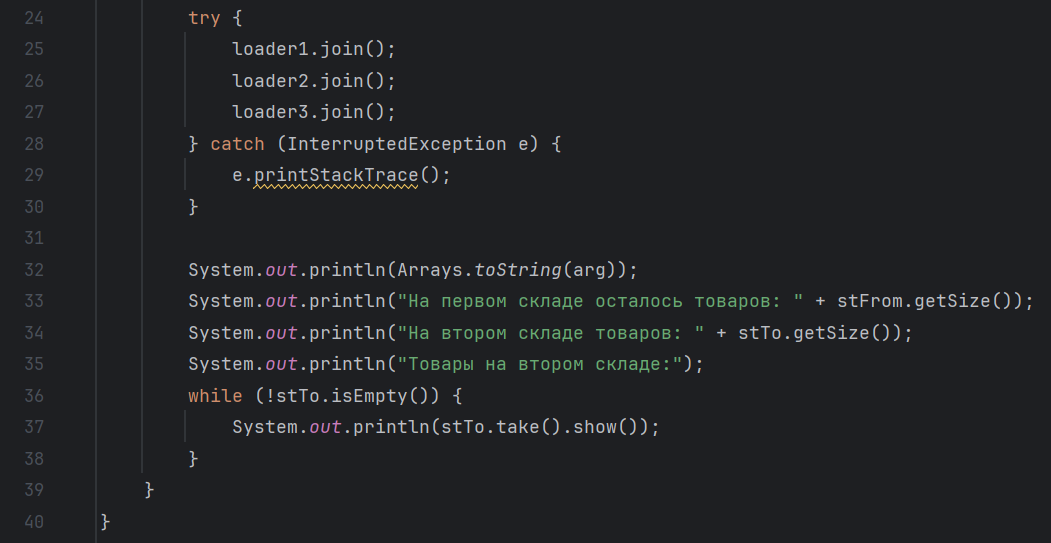
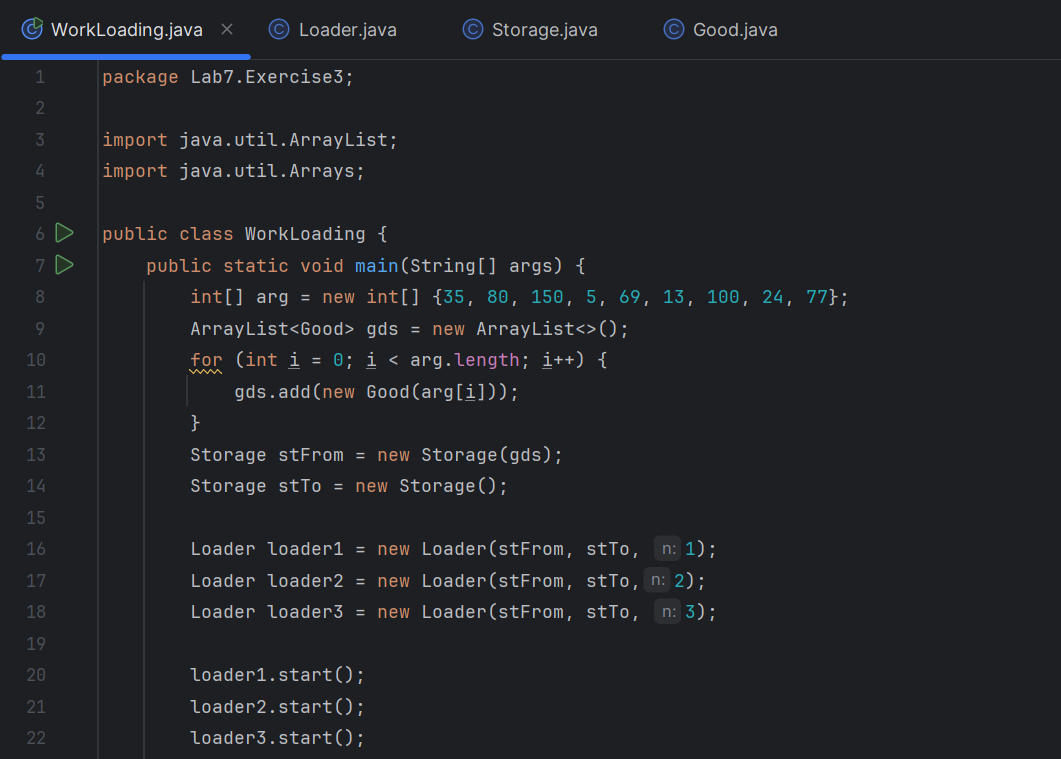
Теперь создадим класс Storage для склада, в котором будем хранить ArrayList товаров, а также вес товаров, погруженных в грузовик для перевозки. Напишем методы для того, чтобы взять товар со склада и положить обратно, а также для получения количества товаров на складе и проверку на пустоту.



После создадим класс Loader, наследующий классу Thread, для грузчиков, с помощью которого и будем создавать потоки. Укажем конструктор, а также переопределим метод run(), в котором сперва грузчик берёт товар с первого склада и, проверяя, может ли тот уместиться по весу в грузовик, либо укладывает его, либо возвращает обратно на склад. Используем синхронизацию с stFrom (первый склад) при заборе товара, а также при сверке переменной tw на возможность добавления товара в грузовик по весу. Затем же происходит выгрузка с синхронизацией с stTo (второй склад). Всё это в цикле while.



Наконец создадим основной класс, в котором и будем запускать потоки с помощью метода .strart(), а после завершения из работы сливать обратно методом .join().



**Вывод:**

Мы изучили многопоточность, существующие возможности достижения многопоточности в Java и применили полученные знания на практике.

GitHub - https://github.com/MiniLynx13/ITaP\_Lab7