МИНИСТЕРСТВО ЦИФРОВОГО РАЗВИТИЯ, СВЯЗИ И МАССОВЫХ КОММУНИКАЦИЙ РОССИЙСКОЙ ФЕДЕРАЦИИ
Ордена трудового Красного Знамени федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский технический университет связи и информатики»

Кафедра Математическая кибернетика и информационные технологии

Отчет по лабораторной работе № 6

Выполнил: студент группы БПИ2401
Трухина Анастасия Александровна
Проверил: Харрасов Камиль Раисович

Москва,
2025

Оглавление

Цель работы: 3
Задание: 3
Основная часть....3
Задание 1:....4
Задание 2....5
Ответы на контрольные вопросы:....**Error! Bookmark not defined.**
Заключение....**Error! Bookmark not defined.**

Цель работы:
Освоить теоретические основы и получить практические навыки работы с коллекциями (Collections Framework) в языке Java. Изучить иерархию интерфейсов (Collection, List, Set, Map, Queue) и их основные реализации (ArrayList, LinkedList, HashSet, HashMap и др.). Научиться применять дженерики для обеспечения типобезопасности кода. Сравнить характеристики и сферы применения различных коллекций для осознанного выбора оптимальной структуры данных под конкретную задачу. Приобрести умение использовать итераторы для безопасного обхода коллекций и изучить базовые принципы сортировки с помощью интерфейсов Comparable и Comparator.

Задание:.

Задание 1. Написать программу, которая считывает текстовый файл и выводит на экран топ-10 самых часто встречающихся слов в этом файле. Для решения задачи использовать коллекцию Map, где ключом будет слово, а значением — количество его повторений в файле.

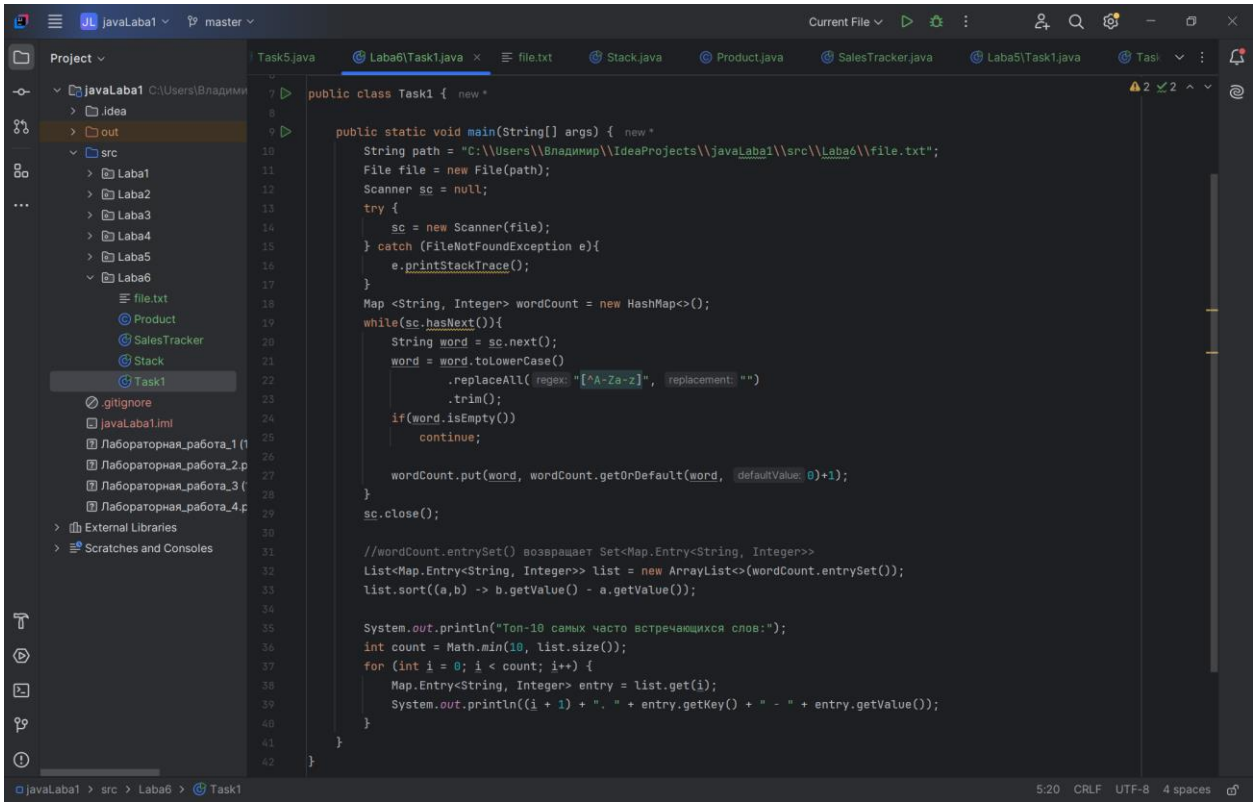
Задание 2. Написать обобщенный класс Stack<T>, который реализует стек на основе массива. Класс должен иметь методы push для добавления элемента в стек, pop для удаления элемента из стека и peek для получения верхнего элемента стека без его удаления.

Задание 3. Разработать программу для учета продаж в магазине. Программа должна позволять добавлять проданные товары в коллекцию, выводить список проданных товаров, а также считать общую сумму продаж и наиболее популярный товар.

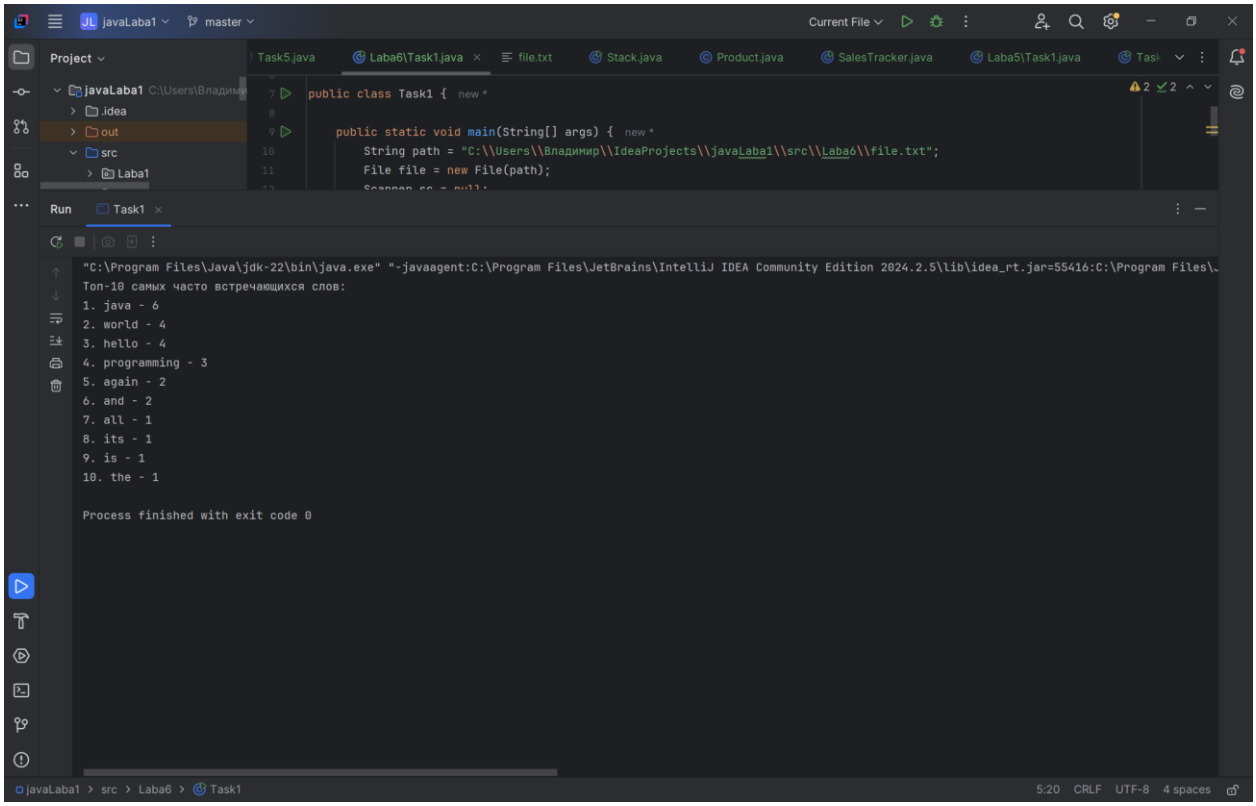
1. Использовать ArrayList для хранения списка проданных товаров.
2. Использовать LinkedList для хранения списка проданных товаров.
3. Использовать HashSet для хранения списка проданных товаров.
4. Использовать TreeSet для хранения списка проданных товаров.
5. Использовать HashMap для хранения пар «товар — количество продаж».
6. Использовать TreeMap для хранения пар «товар — количество продаж».
7. Использовать LinkedHashMap для хранения пар «товар — количество продаж».
8. Использовать ConcurrentHashMap для хранения пар «товар — количество продаж».
9. Использовать ConcurrentHashMap и AtomicInteger для счетчика количества продаж каждого товара.
10. Использовать CopyOnWriteArrayList для хранения списка проданных товаров с возможностью одновременного чтения и записи.

Основная часть

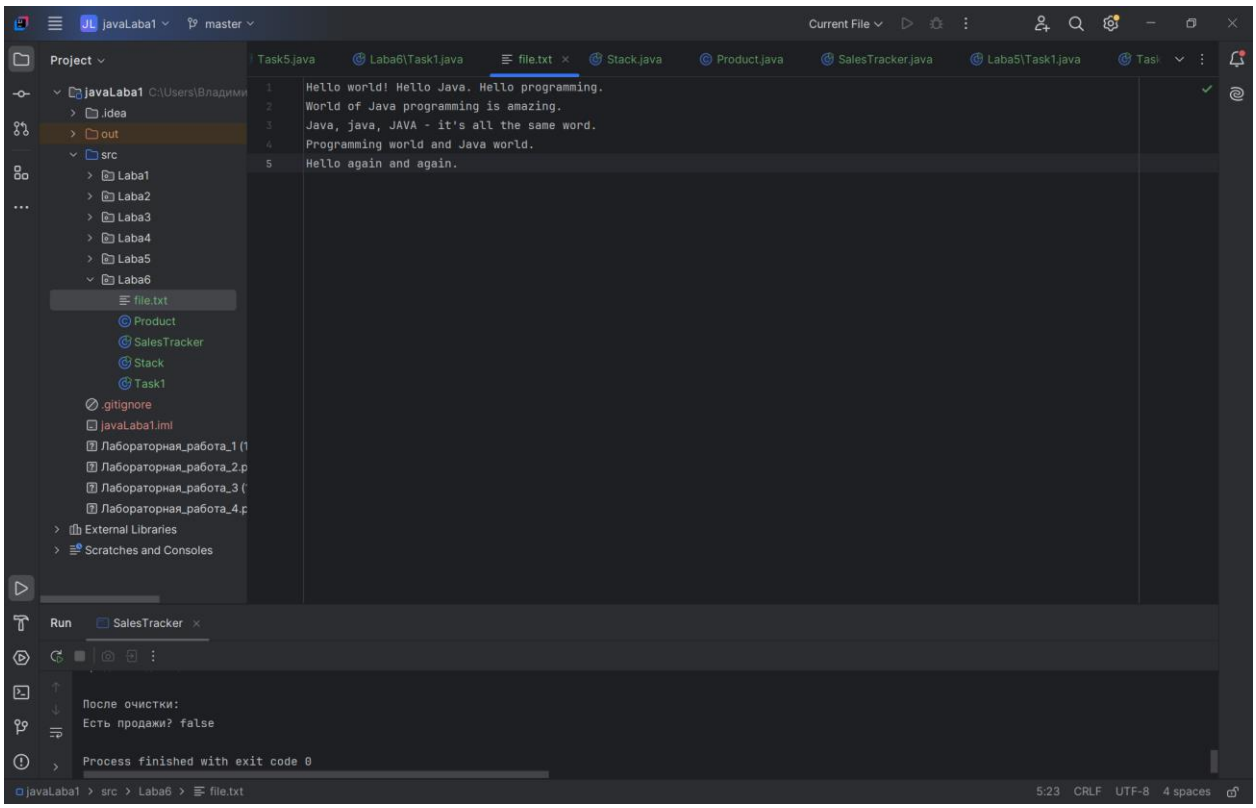
Задание 1:



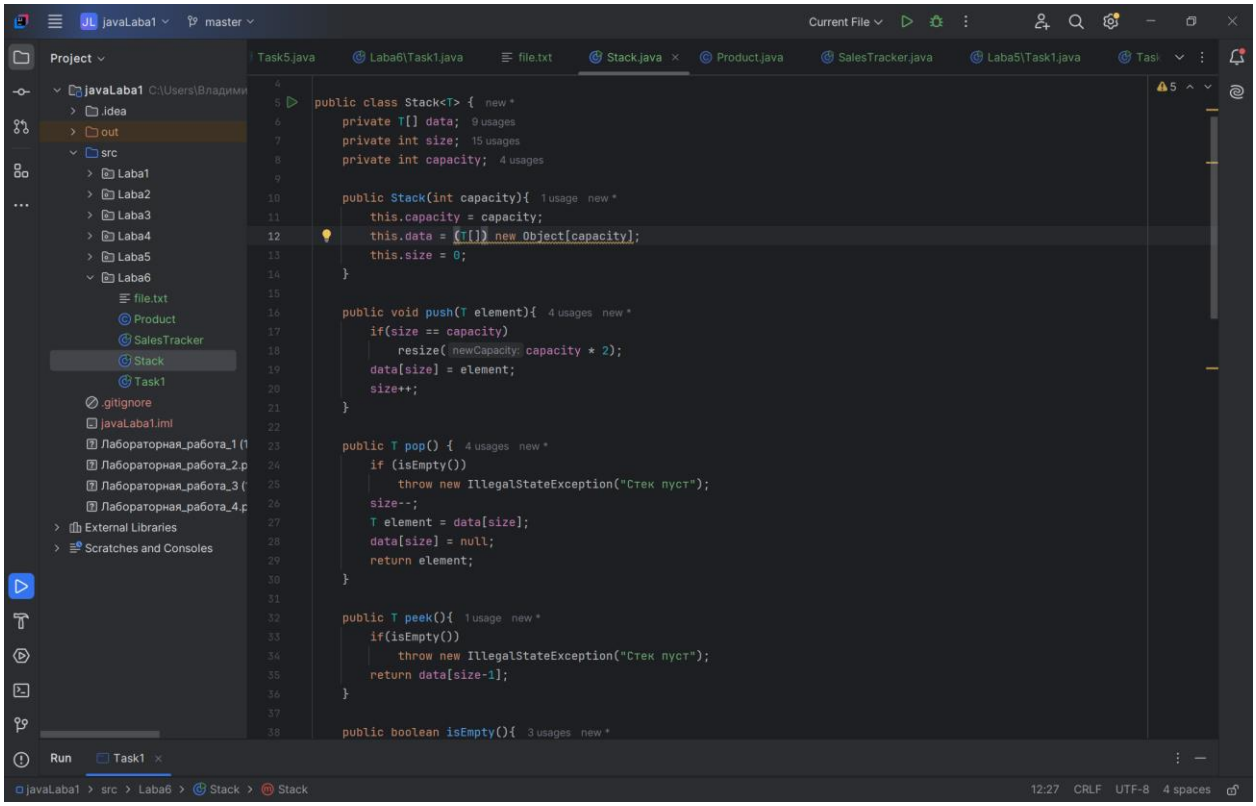
```
public class Task1 {

    public static void main(String[] args) {
        String path = "C:\\Users\\Владимир\\IdeaProjects\\javaLaba1\\src\\Laba6\\file.txt";
        File file = new File(path);
        Scanner sc = null;
        try {
            sc = new Scanner(file);
        } catch (FileNotFoundException e){
            e.printStackTrace();
        }
        Map <String, Integer> wordCount = new HashMap<>();
        while(sc.hasNext()){
            String word = sc.next();
            word = word.toLowerCase()
                    .replaceAll( regex: "[^A-Za-z]", replacement: "")
                    .trim();
            if(word.isEmpty())
                continue;

            wordCount.put(word, wordCount.getOrDefault(word,  defaultValue: 0)+1);
        }
        sc.close();

        //wordCount.entrySet() возвращает Set<Map.Entry<String, Integer>>
        List<Map.Entry<String, Integer>> list = new ArrayList<>(wordCount.entrySet());
        list.sort((a,b) -> b.getValue() - a.getValue());

        System.out.println("Топ-10 самых часто встречающихся слов:");
        int count = Math.min(10, list.size());
        for (int i = 0; i < count; i++) {
            Map.Entry<String, Integer> entry = list.get(i);
            System.out.println((i + 1) + ". " + entry.getKey() + " - " + entry.getValue());
        }
    }
}
```

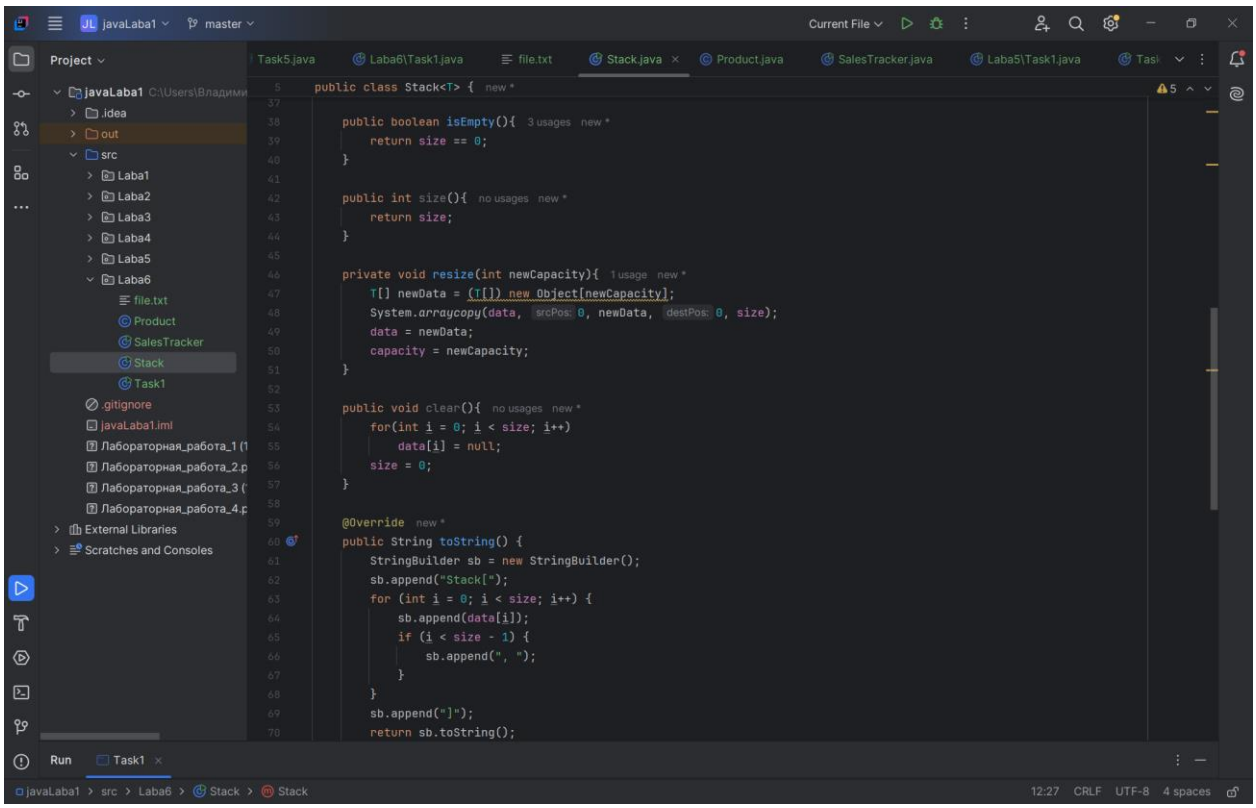


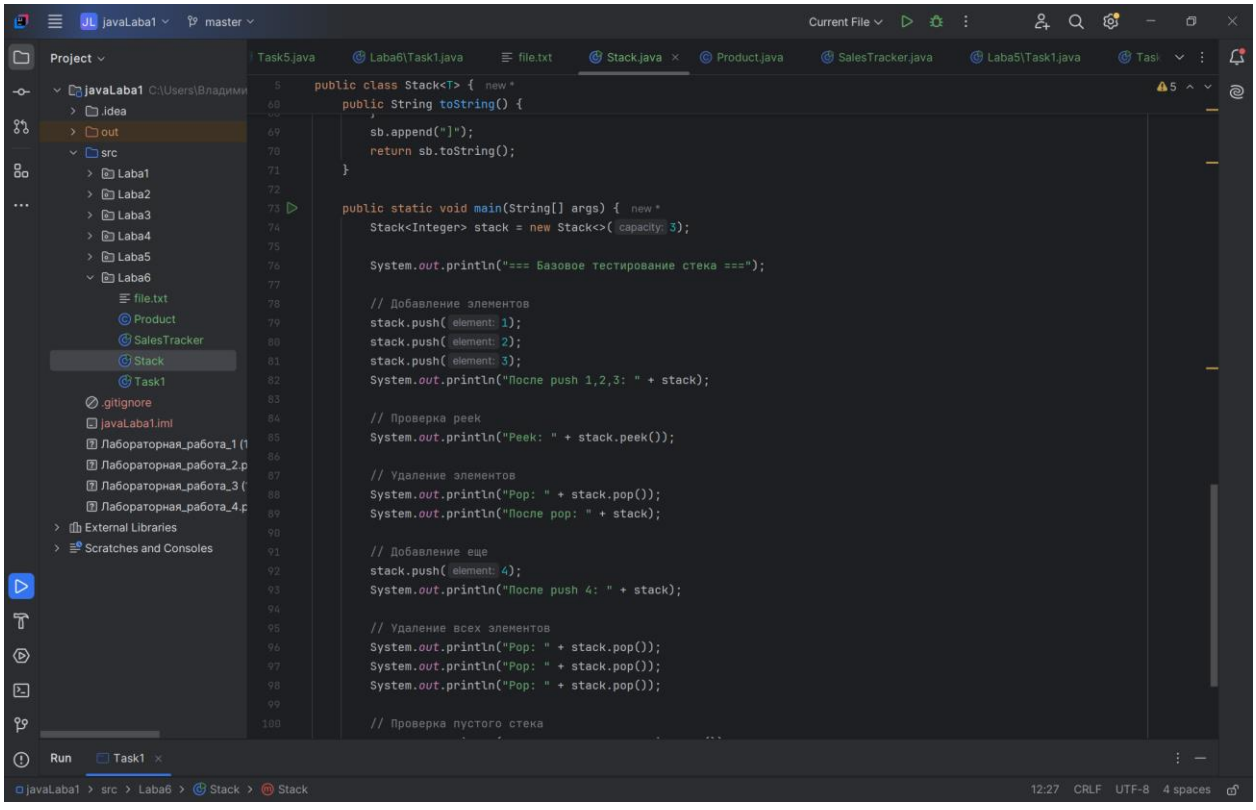
```
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=55416:C:\Program Files\
Топ-10 самых часто встречающихся слов:
1. java - 6
2. world - 4
3. hello - 4
4. programming - 3
5. again - 2
6. and - 2
7. all - 1
8. its - 1
9. is - 1
10. the - 1

Process finished with exit code 0
```

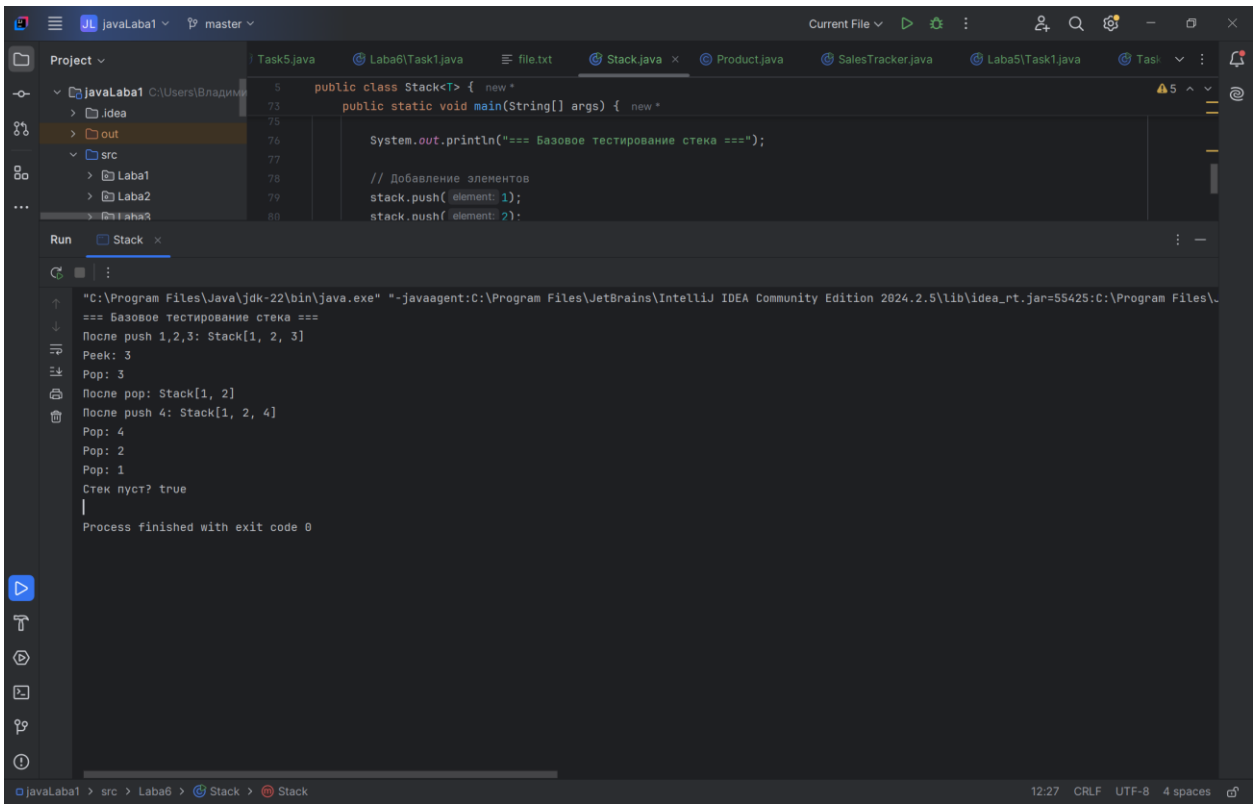

javaLaba1
master
Current File
Project
Task5.java
Laba6\Task1.java
file.txt
Stack.java
Product.java
SalesTracker.java
Laba5\Task1.java
Hello world! Hello Java. Hello programming.
World of Java programming is amazing.
Java, java, JAVA - it's all the same word.
Programming world and Java world.
Hello again and again.
Run
SalesTracker
После очистки:
Есть продажи? false
Process finished with exit code 0
5:23
CRLF
UTF-8
4 spaces

Задание 2

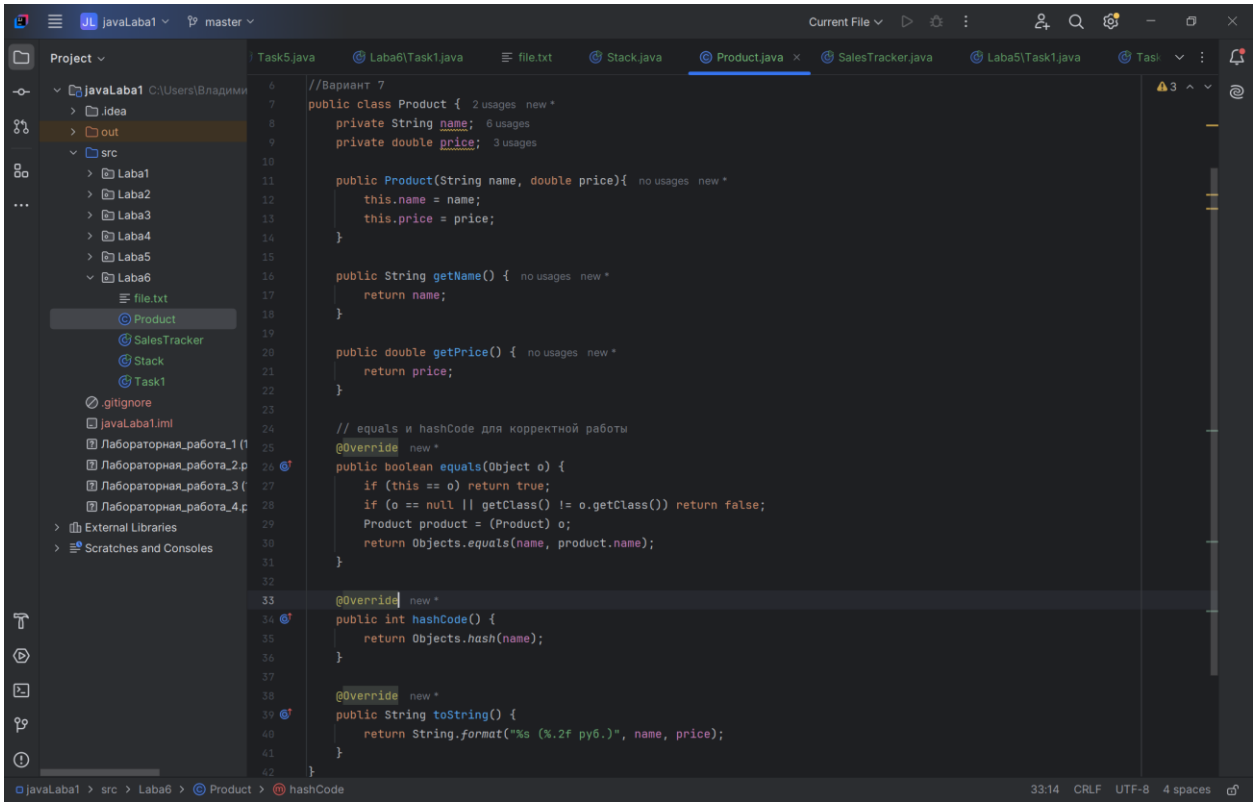

javaLaba1
master
Current File
Project
Task5.java
Laba6\Task1.java
file.txt
Stack.java
Product.java
SalesTracker.java
Laba5\Task1.java
public class Stack<T> {
private T[] data;
private int size;
private int capacity;
public Stack(int capacity){
this.capacity = capacity;
this.data = (T[]) new Object[capacity];
this.size = 0;
public void push(T element){
if(size == capacity)
resize(newCapacity: capacity * 2);
data[size] = element;
size++;
public T pop() {
if (isEmpty())
throw new IllegalStateException("Стек пуст");
size--;
T element = data[size];
data[size] = null;
return element;
public T peek(){
if(isEmpty())
throw new IllegalStateException("Стек пуст");
return data[size-1];
public boolean isEmpty(){
Run
Task1
12:27
CRLF
UTF-8
4 spaces

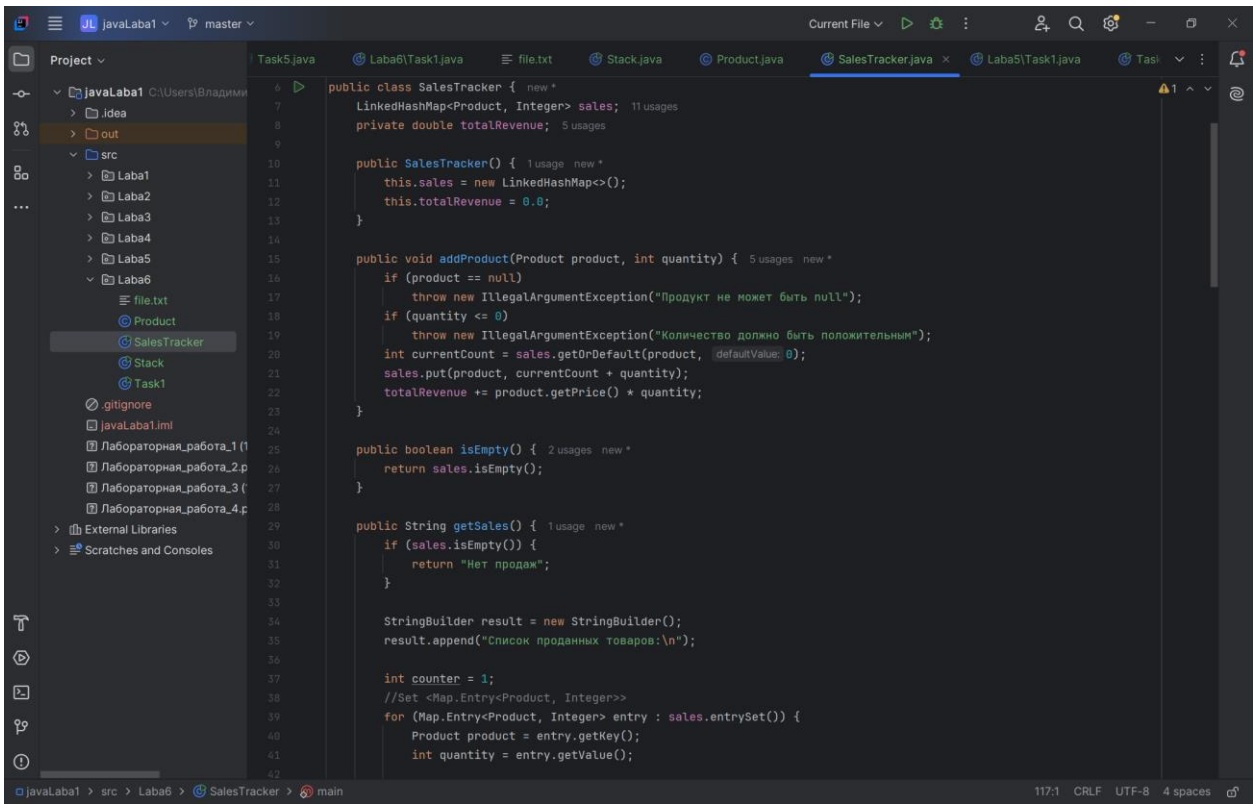

javaLaba1
master
Current File
Project
Task5.java
Laba6\Task1.java
file.txt
Stack.java
Product.java
SalesTracker.java
Laba5\Task1.java
javaLaba1 C:\Users\Владими
.idea
out
src
Laba1
Laba2
Laba3
Laba4
Laba5
Laba6
file.txt
Product
SalesTracker
Stack
Task1
.gitignore
javaLaba1.iml
Лабораторная_работа_1 (1
Лабораторная_работа_2.p
Лабораторная_работа_3 (
Лабораторная_работа_4.p
External Libraries
Scratches and Consoles
public class Stack<T> { new *
public boolean isEmpty(){ 3 usages new *
return size == 0;
public int size(){ no usages new *
return size;
private void resize(int newCapacity){ 1 usage new *
T[] newData = (T[]) new Object[newCapacity];
System.arraycopy(data, srcPos: 0, newData, destPos: 0, size);
data = newData;
capacity = newCapacity;
public void clear(){ no usages new *
for(int i = 0; i < size; i++)
data[i] = null;
size = 0;
@Override new *
public String toString() {
StringBuilder sb = new StringBuilder();
sb.append("Stack[");
for (int i = 0; i < size; i++) {
sb.append(data[i]);
if (i < size - 1) {
sb.append(", ");
sb.append("]");
return sb.toString();
Run
Task1
javaLaba1 > src > Laba6 > Stack > Stack
12:27
CRLF
UTF-8
4 spaces


javaLaba1
master
Current File
Project
Task5.java
Laba6\Task1.java
file.txt
Stack.java
Product.java
SalesTracker.java
Laba5\Task1.java
javaLaba1 C:\Users\Владими
.idea
out
src
Laba1
Laba2
Laba3
Laba4
Laba5
Laba6
file.txt
Product
SalesTracker
Stack
Task1
.gitignore
javaLaba1.iml
Лабораторная_работа_1 (1
Лабораторная_работа_2.p
Лабораторная_работа_3 (
Лабораторная_работа_4.p
External Libraries
Scratches and Consoles
public class Stack<T> { new *
public String toString() {
sb.append("]");
return sb.toString();
public static void main(String[] args) { new *
Stack<Integer> stack = new Stack<>(capacity: 3);
System.out.println("=== Базовое тестирование стека ===");
// Добавление элементов
stack.push(element: 1);
stack.push(element: 2);
stack.push(element: 3);
System.out.println("После push 1,2,3: " + stack);
// Проверка peek
System.out.println("Peek: " + stack.peek());
// Удаление элементов
System.out.println("Pop: " + stack.pop());
System.out.println("После pop: " + stack);
// Добавление еще
stack.push(element: 4);
System.out.println("После push 4: " + stack);
// Удаление всех элементов
System.out.println("Pop: " + stack.pop());
System.out.println("Pop: " + stack.pop());
System.out.println("Pop: " + stack.pop());
// Проверка пустого стека
Run
Task1
javaLaba1 > src > Laba6 > Stack > Stack
12:27
CRLF
UTF-8
4 spaces

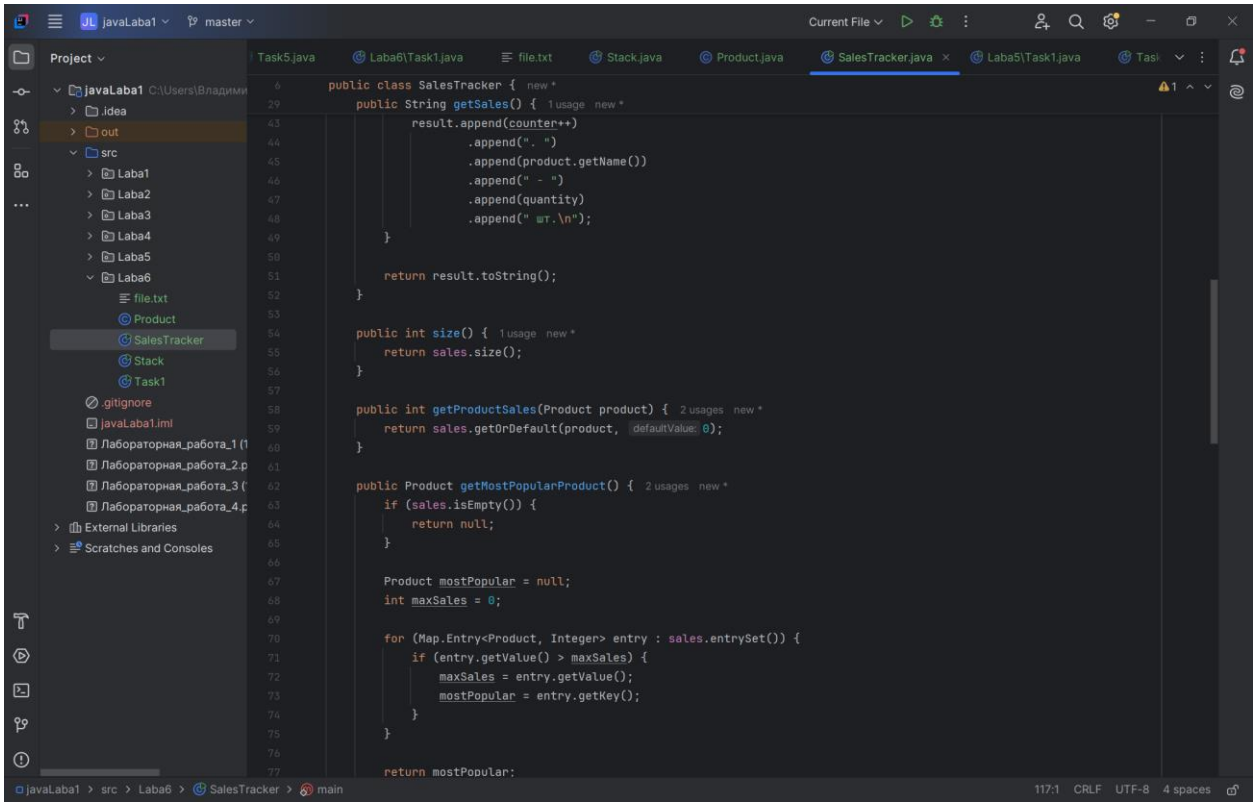

javaLaba1
master
Current File
Project
Task5.java
Laba6\Task1.java
file.txt
Stack.java
Product.java
SalesTracker.java
Laba5\Task1.java
public class Stack<T> {
public static void main(String[] args) {
System.out.println("=== Базовое тестирование стека ===");
// Добавление элементов
stack.push(element: 1);
Run
Stack
"C:\Program Files\Java\jdk-22\bin\java.exe" "-javaagent:C:\Program Files\JetBrains\IntelliJ IDEA Community Edition 2024.2.5\lib\idea_rt.jar=55425:C:\Program Files\
=== Базовое тестирование стека ===
После push 1,2,3: Stack[1, 2, 3]
Peek: 3
Pop: 3
После pop: Stack[1, 2]
После push 4: Stack[1, 2, 4]
Pop: 4
Pop: 2
Pop: 1
Стек пуст? true
Process finished with exit code 0
javaLaba1 › src › Laba6 › Stack › Stack
12:27 CRLF UTF-8 4 spaces

Задание 3

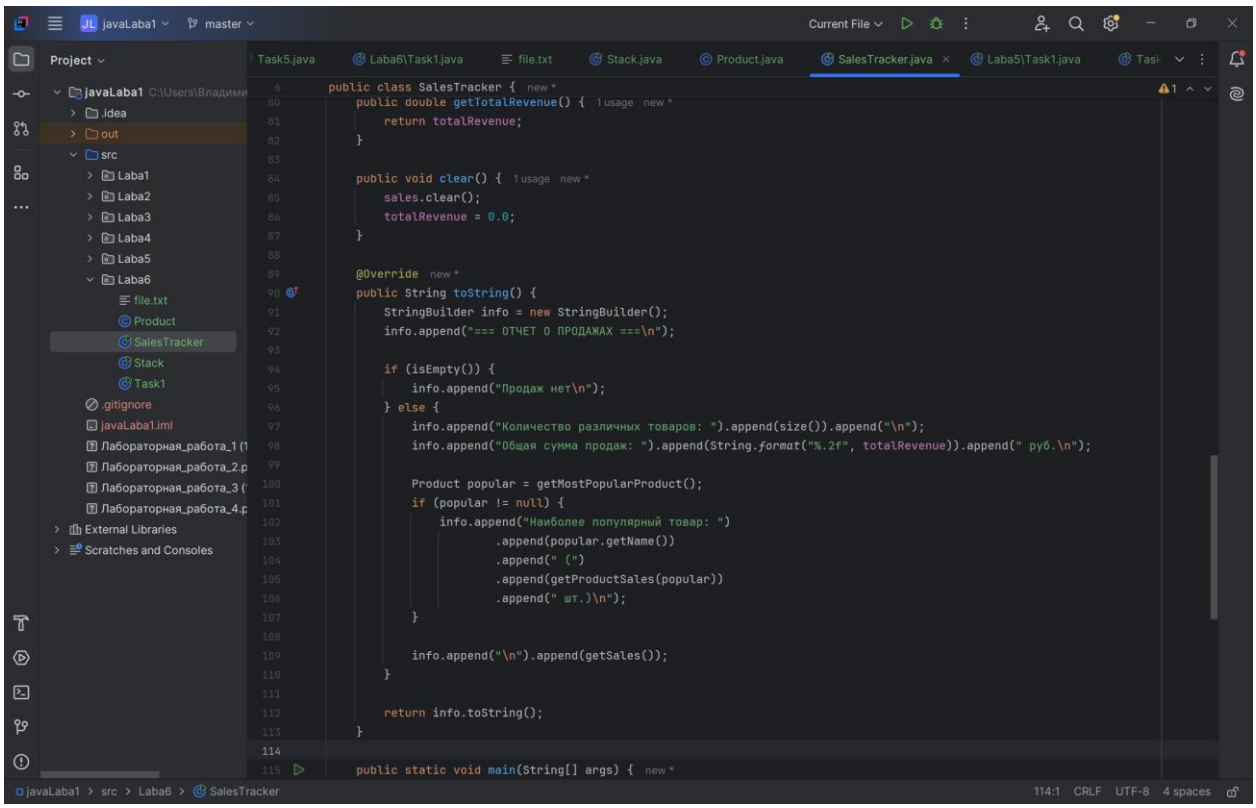

javaLaba1
master
Current File
Project
Task5.java
Laba6\Task1.java
file.txt
Stack.java
Product.java
SalesTracker.java
Laba5\Task1.java
javaLaba1 C:\Users\Владими
.idea
out
src
Laba1
Laba2
Laba3
Laba4
Laba5
Laba6
file.txt
Product
SalesTracker
Stack
Task1
.gitignore
javaLaba1.iml
Лабораторная_работа_1 (1
Лабораторная_работа_2.p
Лабораторная_работа_3 (
Лабораторная_работа_4.p
External Libraries
Scratches and Consoles
//Вариант 7
public class Product { 2 usages new *
private String name; 6 usages
private double price; 3 usages
public Product(String name, double price){ no usages new *
this.name = name;
this.price = price;
public String getName() { no usages new *
return name;
public double getPrice() { no usages new *
return price;
// equals и hashCode для корректной работы
@Override new *
public boolean equals(Object o) {
if (this == o) return true;
if (o == null || getClass() != o.getClass()) return false;
Product product = (Product) o;
return Objects.equals(name, product.name);
@Override new *
public int hashCode() {
return Objects.hash(name);
@Override new *
public String toString() {
return String.format("%s (%.2f руб.)", name, price);
javaLaba1 › src › Laba6 › Product › hashCode
33:14 CRLF UTF-8 4 spaces



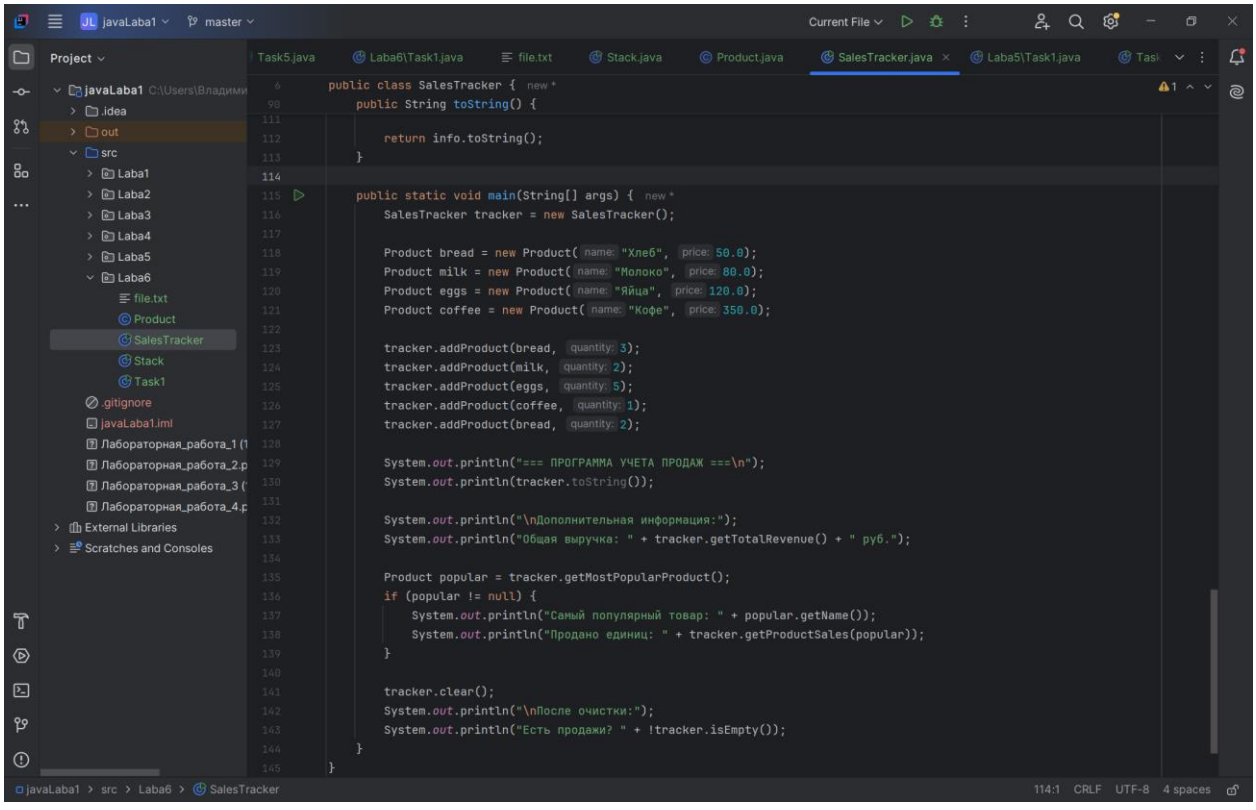
```java
public class SalesTracker {
    LinkedHashMap<Product, Integer> sales;
    private double totalRevenue;

    public SalesTracker() {
        this.sales = new LinkedHashMap<>();
        this.totalRevenue = 0.0;
    }

    public void addProduct(Product product, int quantity) {
        if (product == null)
            throw new IllegalArgumentException("Продукт не может быть null");
        if (quantity <= 0)
            throw new IllegalArgumentException("Количество должно быть положительным");
        int currentCount = sales.getOrDefault(product, 0);
        sales.put(product, currentCount + quantity);
        totalRevenue += product.getPrice() * quantity;
    }

    public boolean isEmpty() {
        return sales.isEmpty();
    }

    public String getSales() {
        if (sales.isEmpty()) {
            return "Нет продаж";
        }

        StringBuilder result = new StringBuilder();
        result.append("Список проданных товаров:\n");

        int counter = 1;
        //Set <Map.Entry<Product, Integer>>
        for (Map.Entry<Product, Integer> entry : sales.entrySet()) {
            Product product = entry.getKey();
            int quantity = entry.getValue();
```



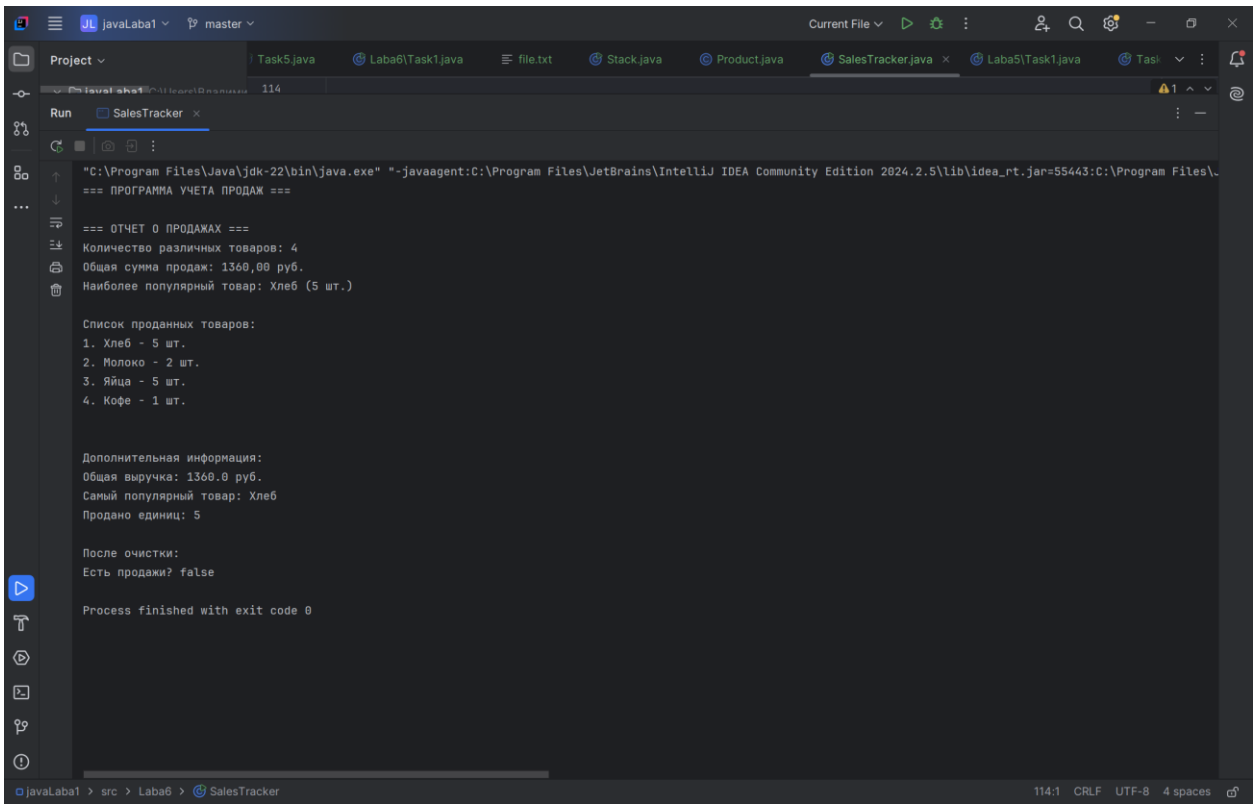
```java
public class SalesTracker {
    public String getSales() {
                result.append(counter++)
                        .append(". ")
                        .append(product.getName())
                        .append(" - ")
                        .append(quantity)
                        .append(" шт.\n");
        }

        return result.toString();
    }

    public int size() {
        return sales.size();
    }

    public int getProductSales(Product product) {
        return sales.getOrDefault(product, 0);
    }

    public Product getMostPopularProduct() {
        if (sales.isEmpty()) {
            return null;
        }

        Product mostPopular = null;
        int maxSales = 0;

        for (Map.Entry<Product, Integer> entry : sales.entrySet()) {
            if (entry.getValue() > maxSales) {
                maxSales = entry.getValue();
                mostPopular = entry.getKey();
            }
        }

        return mostPopular;
```



```java
public class SalesTracker {
    public double getTotalRevenue() {
        return totalRevenue;
    }

    public void clear() {
        sales.clear();
        totalRevenue = 0.0;
    }

    @Override
    public String toString() {
        StringBuilder info = new StringBuilder();
        info.append("=== ОТЧЕТ О ПРОДАЖАХ ===\n");

        if (isEmpty()) {
            info.append("Продаж нет\n");
        } else {
            info.append("Количество различных товаров: ").append(size()).append("\n");
            info.append("Общая сумма продаж: ").append(String.format("%.2f", totalRevenue)).append(" руб.\n");

            Product popular = getMostPopularProduct();
            if (popular != null) {
                info.append("Наиболее популярный товар: ")
                        .append(popular.getName())
                        .append(" (")
                        .append(getProductSales(popular))
                        .append(" шт.)\n");
            }

            info.append("\n").append(getSales());
        }

        return info.toString();
    }

    public static void main(String[] args) {
```



```java
public class SalesTracker {
    public String toString() {

        return info.toString();
    }

    public static void main(String[] args) {
        SalesTracker tracker = new SalesTracker();

        Product bread = new Product( name: "Хлеб",  price: 50.0);
        Product milk = new Product( name: "Молоко",  price: 80.0);
        Product eggs = new Product( name: "Яйца",  price: 120.0);
        Product coffee = new Product( name: "Кофе",  price: 350.0);

        tracker.addProduct(bread,  quantity: 3);
        tracker.addProduct(milk,  quantity: 2);
        tracker.addProduct(eggs,  quantity: 5);
        tracker.addProduct(coffee,  quantity: 1);
        tracker.addProduct(bread,  quantity: 2);

        System.out.println("=== ПРОГРАММА УЧЕТА ПРОДАЖ ===\n");
        System.out.println(tracker.toString());

        System.out.println("\nДополнительная информация:");
        System.out.println("Общая выручка: " + tracker.getTotalRevenue() + " руб.");

        Product popular = tracker.getMostPopularProduct();
        if (popular != null) {
            System.out.println("Самый популярный товар: " + popular.getName());
            System.out.println("Продано единиц: " + tracker.getProductSales(popular));
        }

        tracker.clear();
        System.out.println("\nПосле очистки:");
        System.out.println("Есть продажи? " + !tracker.isEmpty());
    }
}
```



Заключение

Вывод:

В ходе выполнения лабораторной работы №6 были глубоко изучены и практически освоены фундаментальные концепции коллекций (Collections Framework) в языке Java. Работа была сфокусирована на понимании иерархии интерфейсов (Collection, List, Set, Map, Queue) и их основных реализаций, таких как ArrayList, LinkedList, HashSet, HashMap и TreeMap. Практическая часть позволила на собственном опыте оценить различия в производительности и сценариях применения этих структур данных.

Ответы на контрольные вопросы:

Вопрос 1. Какие интерфейсы коллекций есть в Java?

Основные интерфейсы коллекций в Java образуют иерархию. Корневым интерфейсом является Iterable, который обеспечивает возможность обхода элементов в цикле for-each. От него наследуется интерфейс Collection, который представляет общую абстракцию для группы объектов. Более специализированные интерфейсы, расширяющие Collection, это: List (упорядоченная коллекция, допускающая дубликаты), Set (коллекция уникальных элементов) и Queue (очередь, обычно работающая по принципу FIFO — «первым пришел, первым ушел»). Отдельно стоит интерфейс Map, который не является наследником Collection и представляет структуру «ключ-значение». Существуют также его расширенные версии: SortedMap и NavigableMap для упорядоченных карт, а для множеств — SortedSet и NavigableSet.

Вопрос 2. Какие классы коллекций есть в Java?

Для каждого интерфейса существует несколько стандартных реализаций. Для интерфейса List: ArrayList (реализация на основе динамического массива, быстрый доступ по индексу), LinkedList (реализация на основе двусвязного списка, быстрые вставки/удаления), Vector (устаревшая синхронизированная версия, похожая на ArrayList). Для Set: HashSet (хранит элементы в хэш-таблице, не гарантирует порядок), LinkedHashSet (сохраняет порядок вставки элементов), TreeSet (хранит элементы в отсортированном виде, основан на красно-черном дереве). Для Queue: PriorityQueue (очередь с приоритетом), ArrayDeque (двусторонняя очередь на основе массива). Для Map: HashMap (основана на хэш-таблице), LinkedHashMap (сохраняет порядок вставки), TreeMap (отсортированная по ключам карта), Hashtable (устаревшая синхронизированная версия, похожая на HashMap). Также существуют синхронизированные обертки, создаваемые с помощью методов Collections.synchronizedList(), и потокобезопасные коллекции из пакета java.util.concurrent, такие как ConcurrentHashMap и CopyOnWriteArrayList.

Вопрос 3. Что такое итератор?

Итератор (объект, реализующий интерфейс Iterator) — это объект, который предоставляет стандартный способ последовательного доступа к элементам коллекции без раскрытия её внутренней структуры. Основные методы итератора: hasNext() (проверяет, есть ли следующий элемент), next() (возвращает следующий элемент и перемещает курсор), remove() (удаляет текущий элемент, полученный через last call to next()). Цикл for-each неявно использует итератор. Преимущество итератора — возможность безопасно удалять элементы из коллекции во время обхода, что невозможно в обычном цикле for по индексу для некоторых коллекций.

Вопрос 4. Как работают коллекции на основе интерфейса Map?

Коллекции Map (например, HashMap, TreeMap) хранят данные в виде пар «ключ-значение». Ключи уникальны (в рамках одной карты), в то время как значения могут повторяться. Основные операции: put(K key, V value) — добавляет или обновляет пару, get(Object key) — возвращает значение по ключу, remove(Object key) — удаляет пару. HashMap для быстрого доступа (в среднем за O(1)) использует хэш-код ключа. TreeMap хранит пары, отсортированные по ключам (в естественном порядке или заданному Comparator), обеспечивая доступ за O(log n). Map не является коллекцией в прямом смысле (не наследует Collection), но предоставляет методы keySet(), values() и entrySet() для получения представлений своих данных в виде коллекций.

Вопрос 5. Как работают коллекции на основе интерфейса List?
Коллекции List (списки) представляют собой упорядоченные последовательности элементов. Они допускают дубликаты и предоставляют доступ к элементам по целочисленному индексу (начинается с 0). Основные реализации: ArrayList и LinkedList. ArrayList — это обертка над динамическим массивом. Он обеспечивает очень быстрый доступ для чтения по индексу (O(1)), но вставка или удаление элементов в середине списка требует сдвига всех последующих элементов (O(n)). LinkedList реализован как двусвязный список. Доступ по индексу в нем медленный (O(n)), так как требует последовательного прохода от начала или конца. Однако вставка и удаление элементов в известной позиции (например, в начале или конце, или если у нас уже есть итератор на нужную позицию) выполняются быстро (O(1)).

Вопрос 6. Как работают коллекции на основе интерфейса Set?
Коллекции Set (множества) предназначены для хранения уникальных элементов, то есть они не допускают дубликатов. Равенство элементов определяется с помощью метода equals(). Основные реализации: HashSet, LinkedHashSet и TreeSet. HashSet — наиболее распространенная реализация. Для хранения элементов используется хэш-таблица (на основе HashMap). Это обеспечивает высокую производительность операций добавления, удаления и проверки наличия элемента (в среднем O(1)). Порядок элементов не гарантируется. LinkedHashSet наследует от HashSet, но дополнительно сохраняет порядок вставки элементов, используя связный список. TreeSet хранит элементы в отсортированном порядке (по естественному порядку или заданному Comparator), используя красно-черное дерево. Операции добавления, удаления и поиска выполняются за O(log n).

Вопрос 7. Как можно синхронизировать коллекции в Java?
Стандартные реализации коллекций (ArrayList, HashMap и т.д.) не являются потокобезопасными. Для работы в многопоточной среде можно использовать несколько подходов: 1) Использовать устаревшие синхронизированные классы Vector и Hashtable (не рекомендуется для нового кода). 2) Создавать синхронизированные обертки с помощью статических методов класса Collections: Collections.synchronizedList(new ArrayList<>()), Collections.synchronizedMap(new HashMap<>()) и т.д. Эти обертки делают все методы коллекции синхронизированными, что обеспечивает безопасность, но может снизить производительность из-за грубой блокировки всей коллекции. 3) Использовать специализированные потокобезопасные коллекции из пакета java.util.concurrent:

ConcurrentHashMap (высокопроизводительная потокобезопасная карта), CopyOnWriteArrayList (список, где все мутирующие операции создают новую копию внутреннего массива, отлично подходит для сценариев, где чтение сильно преобладает над записью), ConcurrentLinkedQueue, BlockingQueue и другие. Эти коллекции используют более сложные и эффективные алгоритмы (например, сегментированные блокировки), чем простые синхронизированные обертки.

Вопрос 8. Какие методы предоставляет интерфейс Collection?
Интерфейс Collection объявляет основные операции для работы с группой объектов. Ключевые методы можно разделить на группы: 1) Операции над элементами: add(E e), remove(Object o), contains(Object o), size(), isEmpty(), clear(). 2) Операции над группами (пакетные): addAll(Collection<? extends E> c), removeAll(Collection<?> c), retainAll(Collection<?> c) (оставляет только элементы, которые есть и в данной коллекции, и в указанной), containsAll(Collection<?> c). 3) Итерация: iterator() — возвращает итератор для обхода коллекции. 4) Преобразование в массив: toArray() и toArray(T[] a). 5) Stream API (добавлено в Java 8): stream() и parallelStream() для работы с потоками данных. Важно отметить, что не все реализации поддерживают все методы (например, некоторые неизменяемые коллекции выбросят UnsupportedOperationException при вызове add).

Вопрос 9. Какие реализации интерфейса List вы знаете?
Основные реализации интерфейса List в стандартной библиотеке Java: 1) ArrayList: Реализация на основе динамически изменяемого массива. Обеспечивает быстрый произвольный доступ по индексу (get/set), но вставка/удаление в середину может быть медленным. 2) LinkedList: Реализация на основе двусвязного списка. Быстрая вставка/удаление в начале, конце или через итератор, но медленный доступ по индексу. Также реализует интерфейсы Deque и Queue. 3) Vector: Устаревшая синхронизированная версия, похожая на ArrayList. Не рекомендуется для использования в новом коде. 4) Stack: Класс, наследующий Vector, реализующий структуру данных «стек» (LIFO). Также считается устаревшим. Вместо него лучше использовать интерфейс Deque и его реализацию ArrayDeque. 5) CopyOnWriteArrayList: Потокобезопасная реализация из пакета java.util.concurrent. Все мутирующие операции создают новую копию внутреннего массива, что делает её идеальной для сценариев с редкой записью и частым чтением.

Вопрос 10. Какие реализации интерфейса Set вы знаете?
Основные реализации интерфейса Set: 1) HashSet: Самая распространенная реализация.

Хранит элементы в хэш-таблице, обеспечивая высокую производительность. Не гарантирует порядок итерации. 2) LinkedHashSet: Наследник HashSet. Сохраняет порядок вставки элементов, используя связный список в дополнение к хэш-таблице. Производительность чуть ниже, чем у HashSet. 3) TreeSet: Реализация на основе красно-черного дерева (TreeMap). Хранит элементы в отсортированном порядке (естественном или заданном Comparator). Обеспечивает логарифмическое время для основных операций. 4) EnumSet: Высокопроизводительная специализированная реализация для работы с элементами перечисления (enum). 5) CopyOnWriteArraySet: Потокобезопасная реализация, основанная на CopyOnWriteArrayList. Подходит для очень маленьких множеств с редкой записью.

Вопрос 11. Что такое Comparable и Comparator?
Это два интерфейса в Java, предназначенные для сравнения объектов и определения их естественного или альтернативного порядка сортировки. Comparable (интерфейс с одним методом `compareTo(T o)`) определяет *естественный порядок* сортировки объектов класса. Класс, реализующий Comparable, сам «знает», как его экземпляры должны сравниваться между собой (например, String, Integer). Comparator (интерфейс с методом `compare(T o1, T o2)`) представляет собой *внешнюю стратегию сравнения*. Он используется, когда нужно отсортировать объекты по какому-то другому, не естественному для них признаку, или если класс не реализует Comparable. Comparator можно передавать в методы сортировки (Collections.sort, Arrays.sort) или в конструкторы сортированных коллекций (TreeSet, TreeMap). Основное различие: Comparable определяет порядок «по умолчанию» внутри самого класса, а Comparator — гибкий, внешний механизм для задания различных порядков сортировки.

Вопрос 12. Что такое параметр типа?
Параметр типа (type parameter) — это переменная-заполнитель, используемая в дженериках (Generics) для указания обобщенного типа. Он указывается в угловых скобках <> при объявлении класса, интерфейса или метода. Например, в объявлении `class Box<T>` буква `T` является параметром типа. Параметры типа принято обозначать одной заглавной буквой: `T` (Type), `E` (Element), `K` (Key), `V` (Value), `N` (Number). При создании экземпляра обобщенного класса или вызове обобщенного метода параметр типа заменяется на конкретный тип (например, `Box<Integer> integerBox = new Box<>();`). Это позволяет создавать типобезопасные, повторно используемые компоненты.

Вопрос 13. Как параметризовать метод, класс?

- Параметризация класса: Параметры типа указываются после имени класса. `public class Container<T> { private T item; ... }` Теперь класс Container может работать с любым типом `T`.
- Параметризация интерфейса: Аналогично классу. `public interface Repository<E> { void save(E entity); }`
- Параметризация метода: Параметры типа указываются перед возвращаемым типом метода. Их область видимости ограничена этим методом. `public <T> T getFirstElement(List<T> list) { return list.get(0); }` Этот метод объявляет собственный параметр типа `T`, который используется для аргумента и возвращаемого значения. При вызове компилятор автоматически выводит конкретный тип `T` на основе переданного аргумента.

Вопрос 14. Что такое стирание типов?
Стирание типов (type erasure) — это процесс на этапе компиляции в Java, при котором вся информация о параметрах универсальных типов (дженериках) удаляется (стирается). Компилятор заменяет все параметры типов их *границами* (если они заданы, например, `T extends Number`) или `Object` (если границы не заданы), а также вставляет необходимые приведения типов (касты) для обеспечения типобезопасности. На этапе выполнения (runtime) объекты обобщенных классов не содержат информации о своих параметрах типа. Например, после компиляции `List<String>` и `List<Integer>` становятся просто сырыми `List`. Это делается для обеспечения обратной совместимости с кодом, написанным до Java 5 (когда дженериков не было).

Вопрос 15. Как можно обойти ограничения стирания типов?
Ограничения стирания типов не позволяют, например, создать массив обобщенного типа (`new T[]`) или проверить тип во время выполнения (`instanceof List<String>`). Обходные пути: 1) Для создания массивов: Использовать «сырой» тип с последующим приведением: `T[] array = (T[]) new Object[size];` (выдает предупреждение unchecked cast). 2) Для проверки типа во время выполнения: Проверять стираемый тип (например, `list instanceof List`), а для уточнения типа элемента можно передавать объект `Class<T>` как параметр: `public <T> void method(Class<T> type, List<T> list) { if (type == String.class) { ... } }`. 3) Для создания экземпляра T: Требуется передавать `Class<T>` или поставщика

(Supplier): public <T> T createInstance(Class<T> clazz) throws Exception { return clazz.newInstance(); }.

Вопрос 16. Как работают дженерики с массивами?
Дженерики и массивы в Java плохо совместимы. Нельзя создавать массивы параметризованных типов. Код new List<String>[10] приведет к ошибке компиляции. Причина в том, что массивы в Java *ковариантны* (если Integer — подтип Number, то Integer[] — подтип Number[]) и проверяют типы своих элементов во время выполнения (Runtime type checking). Дженерики, напротив, *инвариантны* (List<Integer> не является подтипом List<Number>) и из-за стирания типов не могут выполнять проверки на этапе выполнения. Разрешение таких массивов привело бы к возможности нарушить безопасность типов. Вместо массива параметризованных коллекций следует использовать коллекцию коллекций, например, ArrayList<ArrayList<String>>.

Вопрос 17. Можно ли создать массив дженериков?
Нет, нельзя напрямую создать массив с компонентами параметризованного типа (например, new T[] или new List<String>[]). Это вызывает ошибку компиляции «generic array creation». Однако можно создать массив «сырого» типа или массива с границей ? (wildcard) и затем привести его к нужному типу, что порождает предупреждение «unchecked cast». Более безопасной и предпочтительной альтернативой является использование коллекций (например, ArrayList<T>) вместо массивов.

Вопрос 18. Что такое wildcard тип?
Wildcard (подстановочный знак) в дженериках обозначается знаком вопроса ?. Он используется, когда тип является неизвестным или не важен. Существует три вида wildcard: 1) Неограниченная подстановка (?): List<?> — список элементов неизвестного типа. Можно читать элементы как Object, но нельзя добавлять (кроме null). 2) Верхне-ограниченная подстановка (? extends T): List<? extends Number> — список элементов, тип которых является Number или его подтипом (например, Integer, Double). Можно безопасно читать элементы как Number, но нельзя добавлять (так как неизвестен точный подтип). 3) Нижне-ограниченная подстановка (? super T): List<? super Integer> — список элементов, тип которых является Integer или его супертипом (например, Number, Object). Можно безопасно добавлять объекты типа Integer (и его подтипы) в такой список, но при чтении можно получить только Object.

Вопрос 19. В чем разница между `<? extends T>` и `<? super T>`?

Разница заключается в принципе *PECS (Producer Extends, Consumer Super)*, который определяет, для чего используется коллекция. `<? extends T>` («producer») означает, что коллекция *производит* (предоставляет) объекты типа `T`. Из такой коллекции можно безопасно *читать* (получать `T`), но нельзя в неё *записывать* (добавлять), потому что компилятор не знает точный подтип `T` внутри. `<? super T>` («consumer») означает, что коллекция *потребляет* (принимает) объекты типа `T`. В такую коллекцию можно безопасно *записывать* (добавлять `T` и его подтипы), но при *чтении* можно получить только `Object`, потому что компилятор не знает точный супертип. Пример: `void copy(List<? super T> dest, List<? extends T> src)` — копируем *из* источника (producer) *в* приемник (consumer).

UитХаб с файлами кода: [NT-005-TN/ITiP_LabWorks_Trukhina]