МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

Федеральное государственное бюджетное образовательное учреждение

высшего образования

**«КУБАНСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ»**

**(ФГБОУ ВО «КубГУ»)**

**Факультет компьютерных технологий и прикладной математики**

**Кафедра вычислительных технологий**

**ЛАБОРАТОРНАЯ РАБОТА №5**

**Дисциплина: Платформо-независимое программирование**

Работу выполнил: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В.С. Таран

Направление подготовки: 02.03.02 Фундаментальная информатика и информационные технологии

Преподаватель: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ В. И. Шиян

**Тема**: Параметризация в Java.

**Цель**: Освоить коллекции и методы параметризации.

**Задание**:

Используя коллекции и приемы паратметризации ( а) с помощью интерфейсов-предикатов и б) с лямбда-функцией) решить следующие варианты заданий:

В магазине спортивных товаров имеется список различных товаров: мячи, ракетки, кроссовки, спортивная одежда и аксессуары. Каждый товар характеризуется следующими атрибутами: тип, бренд, размер (если применимо), цвет и цена. Необходимо разделить товары по типам, обеспечить возможность сортировки по цене и размеру (если применимо) внутри каждого типа. Реализовать функцию, позволяющую выбрать товары определенного бренда и цвета. Также требуется найти самый дорогой и самый дешевый товар в каждом типе.

**Ход работы**

1. Параметризация через интерфейсы-предикаты.

В классе ProductServive  метод filterByPredicate принимает  ProductPredicate , но метод filterByPredicate не имеет условий фильтрации – он делегирует проверку переданному предикату.

public static List<Product> filterByPredicate(List<Product> products, ProductPredicate predicate) {  
 return products.stream()  
 .filter(predicate::test)  
 .collect(Collectors.toList());  
}

1. Параметризация через лямбда-функции.

Фильтрация по цене:

public static List<Product> sortByPrice(List<Product> products) {

return products.stream()

.sorted(Comparator.comparing(Product::getPrice))

.collect(Collectors.toList());

}

Фильтр по размеру:

public static List<Product> sortBySize(List<Product> products) {

return products.stream()

.sorted(Comparator.comparing(

Product::getSize,

Comparator.nullsLast(Comparator.naturalOrder())

))

.collect(Collectors.toList());

}

1. Использование потоков.

Создаётся компаратор, который задаёт первичный и вторичный порядки сортировки. После чего запускается поток, который группирует по типу товаров. Внутри группы список преобразуется в поток, выполняет сортировку с помощью компаратора и возвращает отсортированные элементы в список.

public static Map<String, List<Product>> groupByType(List<Product> products) {

return products.stream()

.collect(Collectors.groupingBy(Product::getType));

}

for (Map.Entry<String, List<Product>> entry : groupedProducts.entrySet()) {  
 String type = entry.getKey();  
 List<Product> productList = entry.getValue();  
  
 List<Product> sortedByPrice = ProductService.sortByPrice(productList);  
 List<Product> sortedBySize = ProductService.sortBySize(productList);  
  
 System.out.println("\nТип: " + type);  
 System.out.println("Сортировка по цене: " + sortedByPrice);  
 System.out.println("Сортировка по размеру: " + sortedBySize);  
}

public static Optional<Product> findMaxPrice(List<Product> products) {  
 return products.stream()  
 .max(Comparator.comparing(Product::getPrice));  
}  
  
public static Optional<Product> findMinPrice(List<Product> products) {  
 return products.stream()  
 .min(Comparator.comparing(Product::getPrice));  
}

public static Map<String, List<Product>> groupByType(List<Product> products) {  
 return products.stream()  
 .collect(Collectors.groupingBy(Product::getType));  
}  
  
public static List<Product> sortByPrice(List<Product> products) {  
 return products.stream()  
 .sorted(Comparator.comparing(Product::getPrice))  
 .collect(Collectors.toList());  
}  
  
public static List<Product> sortBySize(List<Product> products) {  
 return products.stream()  
 .sorted(Comparator.comparing(  
 Product::getSize,  
 Comparator.nullsLast(Comparator.naturalOrder())  
 ))  
 .collect(Collectors.toList());  
}

**Результат работы программы**

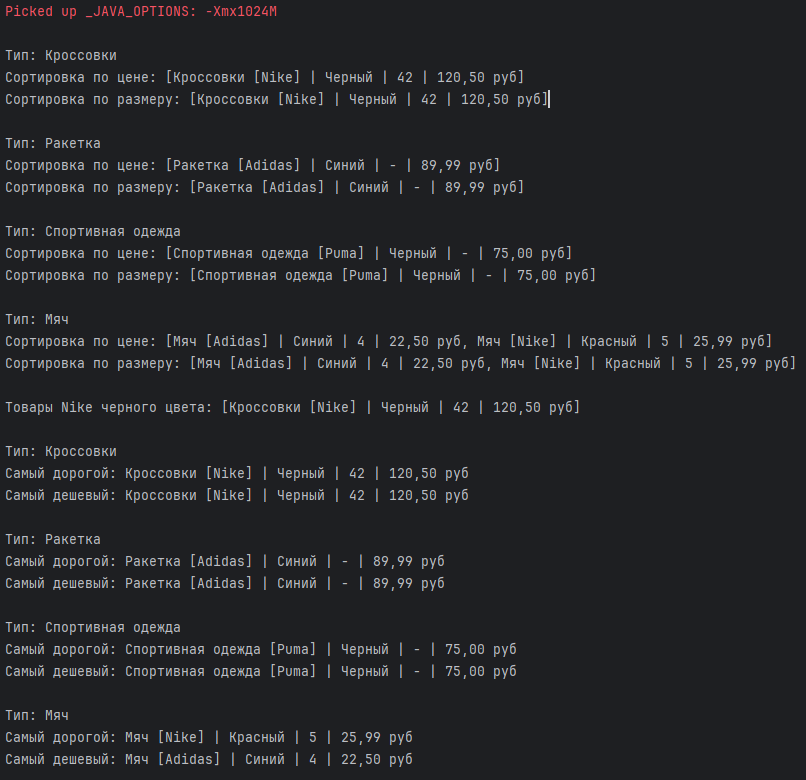
****

Рисунок 1 – Результат работы программы.

**Вывод:** в ходе выполнения работы были освоены основы работы с интерфейсами-предикатами и лямбда-функциями.

**Код программы**

**SportStore**

package ru.practicum;  
  
import java.util.\*;  
  
public class SportStore {  
 public static void main(String[] args) {  
 List<Product> products = Arrays.*asList*(  
 new Product("Мяч", "Nike", 5, "Красный", 25.99),  
 new Product("Ракетка", "Adidas", null, "Синий", 89.99),  
 new Product("Кроссовки", "Nike", 42, "Черный", 120.50),  
 new Product("Мяч", "Adidas", 4, "Синий", 22.50),  
 new Product("Спортивная одежда", "Puma", null, "Черный", 75.00)  
 );  
  
 Map<String, List<Product>> groupedProducts = ProductService.*groupByType*(products);  
  
 for (Map.Entry<String, List<Product>> entry : groupedProducts.entrySet()) {  
 String type = entry.getKey();  
 List<Product> productList = entry.getValue();  
  
 List<Product> sortedByPrice = ProductService.*sortByPrice*(productList);  
 List<Product> sortedBySize = ProductService.*sortBySize*(productList);  
  
 System.*out*.println("\nТип: " + type);  
 System.*out*.println("Сортировка по цене: " + sortedByPrice);  
 System.*out*.println("Сортировка по размеру: " + sortedBySize);  
 }  
  
 List<Product> nikeBlackProducts = ProductService.*filterByPredicate*(  
 products,  
 p -> p.getBrand().equals("Nike") && p.getColor().equals("Черный")  
 );  
 System.*out*.println("\nТовары Nike черного цвета: " + nikeBlackProducts);  
  
  
 for (Map.Entry<String, List<Product>> entry : groupedProducts.entrySet()) {  
 String type = entry.getKey();  
 Optional<Product> maxProduct = ProductService.*findMaxPrice*(entry.getValue());  
 Optional<Product> minProduct = ProductService.*findMinPrice*(entry.getValue());  
  
 System.*out*.printf("\nТип: %s%nСамый дорогой: %s%nСамый дешевый: %s%n",  
 type,  
 maxProduct.map(Object::toString).orElse("null"),  
 minProduct.map(Object::toString).orElse("null"));  
 }  
 }  
}

**ProductServive**

package ru.practicum;  
  
import java.util.\*;  
import java.util.stream.Collectors;  
  
public class ProductService {  
  
 public static Map<String, List<Product>> groupByType(List<Product> products) {  
 return products.stream()  
 .collect(Collectors.*groupingBy*(Product::getType));  
 }  
  
 public static List<Product> sortByPrice(List<Product> products) {  
 return products.stream()  
 .sorted(Comparator.*comparing*(Product::getPrice))  
 .collect(Collectors.*toList*());  
 }  
  
 public static List<Product> sortBySize(List<Product> products) {  
 return products.stream()  
 .sorted(Comparator.*comparing*(  
 Product::getSize,  
 Comparator.*nullsLast*(Comparator.*naturalOrder*())  
 ))  
 .collect(Collectors.*toList*());  
 }  
  
 public static List<Product> filterByPredicate(List<Product> products, ProductPredicate predicate) {  
 return products.stream()  
 .filter(predicate::test)  
 .collect(Collectors.*toList*());  
 }  
  
 public static Optional<Product> findMaxPrice(List<Product> products) {  
 return products.stream()  
 .max(Comparator.*comparing*(Product::getPrice));  
 }  
  
 public static Optional<Product> findMinPrice(List<Product> products) {  
 return products.stream()  
 .min(Comparator.*comparing*(Product::getPrice));  
 }  
}

**ProductPredicate**

package ru.practicum;  
  
public interface ProductPredicate {  
 boolean test(Product product);  
}

**Product**

package ru.practicum;  
  
public class Product {  
 private String type;  
 private String brand;  
 private Integer size;  
 private String color;  
 private double price;  
  
 public Product(String type, String brand, Integer size, String color, double price) {  
 this.type = type;  
 this.brand = brand;  
 this.size = size;  
 this.color = color;  
 this.price = price;  
 }  
  
 public String getType() {  
 return type;  
 }  
  
 public String getBrand() {  
 return brand;  
 }  
  
 public Integer getSize() {  
 return size;  
 }  
  
 public String getColor() {  
 return color;  
 }  
  
 public double getPrice() {  
 return price;  
 }  
  
 @Override  
 public String toString() {  
 return String.*format*("%s [%s] | %s | %s | %.2f руб", type, brand, color, size != null ? size : "-", price);  
 }  
}