UseCase Java Lambda Warehouse

Ausgangslage

Ein Warenhaus führt diverse Artikel im Sortiment. Das Warehouse bietet die Liste einerseits ungefiltert aber auch gefiltert für die Abfrage an.Weiter sollen die Artikel von den Preisen und Quantitäten her via Consumer angepasst werden können.Das folgende Listing zeigt die Klasse ch.std.jpf2.lambda.stream.warehouse.Article:package ch.std.jpf2.lambda.stream.warehouse;

public class Article {

 private Integer id;
 private String name;
 private Integer quantity;
 private Double price;

 public Article() {
  this(null, null, null, null);
 }

 public Article(Integer id, String name, Integer quantity, Double price) {
  this.id = id;
  this.name = name;
  this.quantity = quantity;
  this.price = price;
 }

 public Integer getId() {
  return id;
 }

 public void setId(Integer id) {
  this.id = id;
 }

 public String getName() {
  return name;
 }

 public void setName(String name) {
  this.name = name;
 }

 public Integer getQuantity() {
  return quantity;
 }

 public void setQuantity(Integer quantity) {
  this.quantity = quantity;
 }

 public Double getPrice() {
  return price;
 }

 public void setPrice(Double price) {
  this.price = price;
 }

 @Override
 public String toString() {
  return "Article [id=" + id + ", name=" + name + ", quantity=" + quantity + ", price=" + price + "]";
 }

}Die Klasse ch.std.jpf2.lambda.stream.warehouse.Warehouse verwaltet die Artikel:package ch.std.jpf2.lambda.stream.warehouse;

import java.util.ArrayList;
import java.util.List;
import java.util.function.Consumer;
import java.util.function.Predicate;

public class Warehouse {

 private List<Article> articles = createInitialStock();

 public Warehouse() {
 }

 public List<Article> getArticles() {
  return this.articles;
 }

 private static List<Article> createInitialStock() {
  List<Article> result = new ArrayList<Article>();
  int id = 1;

  result.add(new Article(id++, "Bleistift", 75, 0.50));
  result.add(new Article(id++, "Ordner", 120, 3.70));
  result.add(new Article(id++, "Hefter", 45, 5.00));
  result.add(new Article(id++, "Kugelschreiber", 115, 1.50));
  result.add(new Article(id++, "Markierstift", 50, 2.50));
  result.add(new Article(id++, "Block A4", 200, 1.75));
  result.add(new Article(id++, "Block A5", 150, 1.25));

  return result;
 }

}

Article Filter

Die aktuelle Warehouse Klasse bietet nur die gesamte Article Liste an. Der folgende Unit Test arbeitet mit dieser Liste und filtert solche via Collection Streams:package ch.std.jpf2.lambda.stream.warehouse.test;

import java.util.List;
import java.util.stream.Collectors;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import ch.std.jpf2.lambda.stream.warehouse.Article;
import ch.std.jpf2.lambda.stream.warehouse.Warehouse;

public class ArticleFilterTest {

 private Warehouse warehouse;

 @BeforeEach
 public void setup() {
  this.warehouse = new Warehouse();
 }

 @Test
 public void test_getAllArticles() {
  List<Article> articleList = this.warehouse.getArticles();
  Assertions.assertEquals(7, articleList.size());
 }

 @Test
 public void test_getArticlesWithQuantityLess100() {
  List<Article> articleListLessQuantity = this.warehouse.getArticles().stream().filter(a -> a.getQuantity() < 100).collect(Collectors.toList());
  Assertions.assertEquals(3, articleListLessQuantity.size());
  articleListLessQuantity.forEach(a -> Assertions.assertTrue(a.getQuantity() < 100));
 }

 @Test
 public void test_getArticlesWithQuantityGreaterOrEqual100() {
  List<Article> articleListLessQuantity = this.warehouse.getArticles().stream().filter(a -> a.getQuantity() >= 100).collect(Collectors.toList());
  Assertions.assertEquals(4, articleListLessQuantity.size());
  articleListLessQuantity.forEach(a -> Assertions.assertTrue(a.getQuantity() >= 100));
 }

}Die Stream filter Methode arbeitet mit dem Predicate FunctionalInterface. Um ganz sicher zu sein verifizieren wir ob die gefilterten Artikel wirklich der gesuchten Regel entsprechen.Die Stream forEach Methode

arbeitet hier mit dem Consumer FunctionalInterface.

Warehouse Filter

Jetzt möchten wir diesen Artikel Filter direkt über die Klasse Warehouse anbieten. Hierzu definieren wir eine 2. getArticles Methode mit dem Predicate FunctionalInterface in der Klasse Warehouse:public List<Article> getArticles(Predicate<Article> predicate) {
 List<Article> result = new ArrayList<Article>();
 for (Article article : this.articles) {
  if (predicate.test(article)) {
  result.add(article);
 }
 }
 return result;
}Der analoge Unit Test zur Warehouse Filterung:package ch.std.jpf2.lambda.stream.warehouse.test;

import java.util.List;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import ch.std.jpf2.lambda.stream.warehouse.Article;
import ch.std.jpf2.lambda.stream.warehouse.Warehouse;

public class WarehouseFilterTest {

 private Warehouse warehouse;

 @BeforeEach
 public void setup() {
  this.warehouse = new Warehouse();
 }

 @Test
 public void test_getArticlesWithQuantityLess100() {
  List<Article> articleListLessQuantity = this.warehouse.getArticles(a -> a.getQuantity() < 100);
 Assertions.assertEquals(3, articleListLessQuantity.size());
  articleListLessQuantity.forEach(a -> Assertions.assertTrue(a.getQuantity() < 100));
 }

 @Test
 public void test_getArticlesWithQuantityGreaterOrEqual100() {
  List<Article> articleListLessQuantity = this.warehouse.getArticles(a -> a.getQuantity() >= 100);
  Assertions.assertEquals(4, articleListLessQuantity.size());
 articleListLessQuantity.forEach(a -> Assertions.assertTrue(a.getQuantity() >= 100));
 }

}Die Komplexität der Filterung selbst wird ins Warehouse verschoben, die Filter Regel wird durch den Anwender definiert. Die Methode getArticles(Predicate) kann auch selbst mit Streams Filtern arbeiten:public List<Article> getArticlesStream(Predicate<Article> predicate) {
 return this.articles.stream().filter(predicate).collect(Collectors.toList());
}

Warehouse Article Consumer

Jetzt erweitern wir das Warehouse so dass wir eine Menge von Artikeln aktualisieren können. Hierzu definieren wir die folgende Methode in der Warehouse Klasse:public void changeArticles(Predicate<Article> predicate, Consumer<Article> consumer) {
  for (Article article : this.articles) {
  if (predicate.test(article)) {
 consumer.accept(article);
  }
  }
 }Der folgende Unit Test erhöht die Anzahl aller Artikel um 2 Einheiten. Dann Mappen wir die Artikel auf die Liste von Quantities und prüfen das Resultat:package ch.std.jpf2.lambda.stream.warehouse.test;

import org.junit.jupiter.api.Assertions;
import org.junit.jupiter.api.BeforeEach;
import org.junit.jupiter.api.Test;

import ch.std.jpf2.lambda.stream.warehouse.Warehouse;

public class WarehouseConsumerTest {

 private Warehouse warehouse;

 @BeforeEach
 public void setup() {
  this.warehouse = new Warehouse();
 }

 @Test
 public void test_incrementQuantityBy2ForAllArticles() {
 this.warehouse.changeArticles(a -> true, a -> a.setQuantity(a.getQuantity()+2));
  Integer[] expected = {77, 122, 47, 117, 52, 202, 152};
  Integer[] quantities = this.warehouse.getArticles().stream().map(a -> a.getQuantity()).toArray(Integer[]::new);
  Assertions.assertArrayEquals(expected, quantities);
 }

}

Fazit

Der UseCase zeigt den möglichen Einsatz von Functional Interface angewendet auf eigene Klassen. Wir benötigen die Funktionalität eigentlich nicht in dieser Form und könnte solche auch rein über die Artikelliste anwenden. Mit der Methode changeArticles(...) manipulieren wir eine bestehende Instanz. Dies ist in der Praxis eher nicht gern gesehen. Solche Manipulationen auf Mengen werden in der Regel über die Persistenz (z.B. SQL) gelöst.

Beispiel Code

Sie finden das komplette Beispiel hier

Kontakt

Simtech AG
Finkenweg 23
3110 Münsingen
Schweiz

Impressum

Das Copyright für sämtliche Inhalte dieser Website liegt bei Simtech AG, Schweiz.
Beachten Sie auch unsere Hinweise zum Urheberrecht, Datenschutz und Haftungsauschluss.
Jeder Hinweis auf Fehler nehmen wir gerne entgegen.

Copyright

2024 Simtech AG, All rights reserved, Powered by stack.ch written in Golang by Daniel Schmutz

://www.simtech-ag.ch/kurs-java-advanced-ressourcen-übungen-lambda-&-stream-api-usecase-wareh