|  |  |  |  |
| --- | --- | --- | --- |
| ***SPRAWOZDANIE NR 2*** | | | |
| Nazwa ćwiczenia | Strumienie API | | Obraz zawierający tekst  Opis wygenerowany automatycznie |
| Przedmiot | Zaawansowane programowanie obiektowe | |
| Student  grupa | Paweł Jońca gr 7 | |
| Data ćwiczeń | 09.04 | 09.04 | Data oddania sprawozdania |

1. **Czym różnią się operacje pośrednie (z ang. intermediate) od kończących**

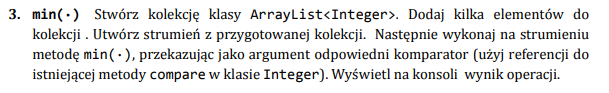
**(z ang. terminal)? Wyjaśnij krótko zasadnicze różnice i sposób ich wykorzystania.**

Operacje pośrednie (intermediate): przekształcają strumień na inny strumień, np. filter(), map(), sorted() – nie wykonują się od razu.

Operacje kończące (terminal): inicjują rzeczywiste wykonanie operacji i kończą strumień, np. forEach(), collect(), min().

1. **Co oznacza, że operacje pośrednie są „leniwe”? Odpowiedź zawrzyj w sprawozdaniu.**

Operacje pośrednie są leniwe, czyli nie są wykonywane natychmiast – dopóki nie zostanie użyta operacja kończąca, nie dzieje się nic. Dzięki temu przetwarzanie jest bardziej efektywne.

****

****

import java.util.ArrayList;  
import java.util.Optional;  
public class Main {  
 public static void main(String[] args) {  
 // Tworzenie kolekcji klasy ArrayList<Integer>  
 ArrayList<Integer> numbers = new ArrayList<>();  
 // Dodanie kilku elementów do kolekcji  
 numbers.add(5);  
 numbers.add(12);  
 numbers.add(3);  
 numbers.add(7);  
 numbers.add(9);  
 // Utworzenie strumienia z przygotowanej kolekcji i wykonanie metody min()  
 Optional<Integer> minValue = numbers.stream()  
 .min(Integer::compare); // Użycie referencji do metody compare w klasie Integer  
 minValue.ifPresent(value -> System.out.println("Minimalna wartość: " + value));  
 }  
}

**Obraz zawierający tekst, Czcionka, linia, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

import java.util.ArrayList;  
import java.util.stream.Stream;  
public class filter {  
 public static void main(String[] args) {  
 ArrayList<Integer> liczby = new ArrayList<>();  
 liczby.add(1);  
 liczby.add(2);  
 liczby.add(3);  
 liczby.add(4);  
 liczby.add(5);  
 liczby.add(6);  
 liczby.add(7);  
 liczby.add(8);  
 liczby.add(9);  
 liczby.add(10);  
 // Utwórz strumień z kolekcji i odfiltruj elementy parzyste  
 Stream<Integer> parzysteLiczby = liczby.stream()  
 .filter(liczba -> liczba % 2 == 0);  
 parzysteLiczby.forEach(System.out::println);  
 }  
}

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

**Obraz zawierający tekst, Czcionka, zrzut ekranu, design

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

import java.util.ArrayList;  
import java.util.List;  
import java.util.Comparator;  
  
public class sorted {  
 public static class Person{  
 private String nick;  
 private int age;  
  
 public Person(String nick, int age) {  
 this.nick = nick;  
 this.age = age;  
 }  
 public String getNick() {  
  
 return nick;  
 }  
  
 public int getAge() {  
 return age;  
 }  
 @Override  
 public String toString() {  
 return "Person{" + "nick=" + nick + ", age=" + age + '}';  
 }  
 }  
 public static void main(String[] args) {  
 List<Person> people = new ArrayList<>();  
 people.add(new Person("Daniel", 18));  
 people.add(new Person("Ewa", 101));  
 people.add(new Person("Anna", 13));  
 people.add(new Person("Daniel", 10));  
  
 people.stream().sorted(Comparator.comparing(Person::getNick).thenComparing(Person::getAge))  
 .forEach(System.out::println);  
 }  
  
}

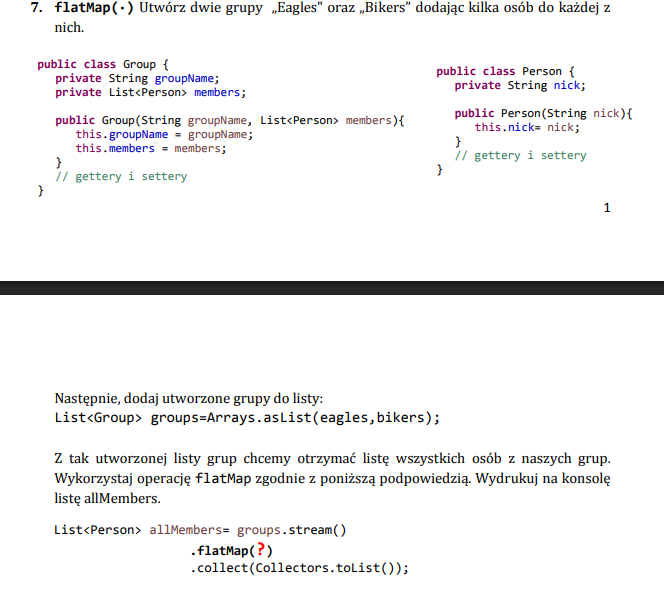
**Obraz zawierający tekst, Czcionka, zrzut ekranu, informacja

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

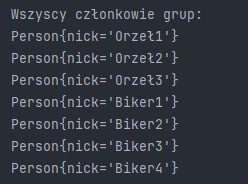
import java.util.ArrayList;  
import java.util.List;  
import java.util.stream.Collectors;  
public class map {  
 public static class PunktXY {  
 private double x;  
 private double y;  
 public PunktXY(double x, double y) {  
 this.x = x;  
 this.y = y;  
 }  
 public double getX() {  
 return x;  
 }  
 public double getY() {  
 return y;  
 }  
 @Override  
 public String toString() {  
 return "PunktXY{" + "x=" + x + ", y=" + y + '}';  
 }  
 }  
 public static class PunktXYZ {  
 private double x;  
 private double y;  
 private double z;  
  
 public PunktXYZ(double x, double y, double z) {  
 this.x = x;  
 this.y = y;  
 this.z = z;  
 }  
  
 public double getX() {  
 return x;  
 }  
  
 public double getY() {  
 return y;  
 }  
  
 public double getZ() {  
 return z;  
 }  
  
 @Override  
 public String toString() {  
 return "PunktXYZ{" + "x=" + x + ", y=" + y + ", z=" + z + '}';  
 }  
 }  
  
 public static void main(String[] args) {  
 List<PunktXYZ> punktyXYZ = new ArrayList<>();  
 punktyXYZ.add(new PunktXYZ(1.0, 2.0, 3.0));  
 punktyXYZ.add(new PunktXYZ(4.5, 5.5, 6.5));  
 punktyXYZ.add(new PunktXYZ(7.0, 8.0, 9.0));  
 punktyXYZ.add(new PunktXYZ(10.1, 11.1, 12.1));  
  
 List<PunktXY> punktyXY = punktyXYZ.stream()  
 .map(punktXYZ -> new PunktXY(punktXYZ.getX(), punktXYZ.getY()))  
 .collect(Collectors.toList());  
  
 System.out.println("Współrzędne punktów XY:");  
 for (PunktXY punkt : punktyXY) {  
 System.out.println("x: " + punkt.getX() + ", y: " + punkt.getY());  
 }  
 }  
}

**Obraz zawierający tekst, Czcionka, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

****

import java.util.Arrays;  
import java.util.List;  
import java.util.stream.Collectors;  
public class flatmap {  
  
 public static class Person {  
 private String nick;  
  
 public Person(String nick) {  
 this.nick = nick;  
 }  
  
 public String getNick() {  
 return nick;  
 }  
 @Override  
 public String toString() {  
 return "Person{" +  
 "nick='" + nick + '\'' +  
 '}';  
 }  
 }  
 public static class Group {  
 private String groupName;  
 private List<Person> members;  
  
 public Group(String groupName, List<Person> members) {  
 this.groupName = groupName;  
 this.members = members;  
 }  
  
 public List<Person> getMembers() {  
 return members;  
 }  
  
 public String getGroupName() {  
 return groupName;  
 }  
  
 @Override  
 public String toString() {  
 return "Group{" +  
 "groupName='" + groupName + '\'' +  
 ", members=" + members +  
 '}';  
 }  
 }  
 public static void main(String[] args) {  
 List<Person> eaglesMembers = Arrays.asList(  
 new Person("Orzeł1"),  
 new Person("Orzeł2"),  
 new Person("Orzeł3")  
 );  
 Group eagles = new Group("Eagles", eaglesMembers);  
  
 List<Person> bikersMembers = Arrays.asList(  
 new Person("Biker1"),  
 new Person("Biker2"),  
 new Person("Biker3"),  
 new Person("Biker4")  
 );  
 Group bikers = new Group("Bikers", bikersMembers);  
 List<Group> groups = Arrays.asList(eagles, bikers);  
 List<Person> allMembers = groups.stream()  
 .flatMap(group -> group.getMembers().stream())  
 .collect(Collectors.toList());  
 System.out.println("Wszyscy członkowie grup:");  
 allMembers.forEach(System.out::println);  
 }  
}

****

**Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

**Obraz zawierający tekst, Czcionka, zrzut ekranu

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

import java.util.Arrays;  
import java.util.List;  
import java.util.Optional;  
public class reduce {  
 public static void main(String[] args) {  
 List<Integer> liczby = Arrays.asList(1, 2, 3, 4, 5);  
 //Zsumuj elementy kolekcji używając Optional<Integer> reduce(BinaryOperator<Integer> accumulator)  
 Optional<Integer> sumaOptional = liczby.stream()  
 .reduce((a, b) -> a + b);  
 sumaOptional.ifPresent(suma -> System.out.println("Suma elementów (Optional): " + suma));  
 //Pomnóż wszystkie elementy kolekcji używając T reduce(T identityVal, BinaryOperator<T> accumulator)  
 int iloczyn = liczby.stream()  
 .reduce(1, (a, b) -> a \* b);  
 System.out.println("Iloczyn elementów: " + iloczyn);  
 }  
}

**Obraz zawierający tekst, Czcionka, zrzut ekranu, algebra

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

import java.util.ArrayList;  
import java.util.List;  
import java.util.UUID;  
import java.util.stream.Collectors;  
  
public class ParallelStream {  
 public static void main(String[] args) {  
 // Wygenerowanie miliona identyfikatorów UUID i umieszczenie ich w ArrayList  
 List<String> uuidList = new ArrayList<>();  
 for (int i = 0; i < 1000000; i++) {  
 uuidList.add(UUID.randomUUID().toString());  
 }  
 //Przetwarzanie sekwencyjne  
 long startTimeSequential = System.nanoTime();  
 List<String> sortedListSequential = uuidList.stream()  
 .sorted()  
 .collect(Collectors.toList());  
 long endTimeSequential = System.nanoTime();  
 long durationSequential = endTimeSequential - startTimeSequential;  
  
 System.out.println("Czas sortowania sekwencyjnego: " + durationSequential / 1\_000\_000 + " ms");  
 //Przetwarzanie równoległe  
 long startTimeParallel = System.nanoTime();  
 List<String> sortedListParallel = uuidList.parallelStream()  
 .sorted()  
 .collect(Collectors.toList());  
 long endTimeParallel = System.nanoTime();  
 long durationParallel = endTimeParallel - startTimeParallel;  
  
 System.out.println("Czas sortowania równoległego: " + durationParallel / 1\_000\_000 + " ms");  
  
 //Porównanie wyników  
 System.out.println("\nPorównanie wyników:");  
 System.out.println("Czas sortowania sekwencyjnego: " + durationSequential / 1\_000\_000 + " ms");  
 System.out.println("Czas sortowania równoległego: " + durationParallel / 1\_000\_000 + " ms");  
  
 System.out.println("\nWnioski:");  
 if (durationParallel < durationSequential) {  
 System.out.println("Sortowanie równoległe było szybsze od sortowania sekwencyjnego.");  
 } else if (durationParallel > durationSequential) {  
 System.out.println("Sortowanie sekwencyjne było szybsze od sortowania równoległego.");  
 } else {  
 System.out.println("Czasy sortowania sekwencyjnego i równoległego były zbliżone.");  
 }  
 }  
}

**Obraz zawierający tekst, zrzut ekranu, Czcionka

Zawartość wygenerowana przez sztuczną inteligencję może być niepoprawna.**

Przetwarzanie równoległe wykorzystuje wiele wątków do wykonania operacji na różnych częściach kolekcji jednocześnie, co potencjalnie może znacznie skrócić czas wykonania dla dużych zbiorów danych i złożonych operacji. Jednak narzut związany z zarządzaniem wątkami i łączeniem wyników może czasami sprawić, że dla mniejszych zbiorów danych lub prostych operacji, przetwarzanie sekwencyjne będzie wydajniejsze W konkretnym przypadku sortowania miliona ciągów znaków UUID, można oczekiwać, że przetwarzanie równoległe będzie szybsze ze względu na relatywnie duży rozmiar danych i złożoność operacji porównywania ciągów znaków W konkretnym przypadku sortowania miliona ciągów znaków UUID, można oczekiwać, że przetwarzanie równoległe będzie szybsze ze względu na relatywnie duży rozmiar danych i złożoność operacji porównywania ciągów znaków

Wnioski:

Wykorzystanie strumieni w Javie znacząco upraszcza operacje na kolekcjach. Dzięki temu bardziej zwięźle możemy wykonywać zadania takie jak wykonane w ćwiczeniu np. filtrowanie, mapowanie sortowanie… Przetwarzanie równoległe strumieni może znacząco poprawić wydajność operacji na dużych zbiorach danych, aczkolwiek jego efektywność zależy od zadania i zasobów systemowych.