Ćwiczenie: Strumienie API 8 (java.util.stream)

(2 godziny zajęciowe)

- **1.** Czym różnią się operacje pośrednie (z ang. *intermediate*) od kończących (z ang. *terminal*)? Wyjaśnij krótko zasadnicze różnice i sposób ich wykorzystania.
- 2. Co oznacza, że operacje pośrednie są "leniwe"? Odpowiedź zawrzyj w sprawozdaniu.
- 3. min(·) Stwórz kolekcję klasy ArrayList<Integer>. Dodaj kilka elementów do kolekcji. Utwórz strumień z przygotowanej kolekcji. Następnie wykonaj na strumieniu metodę min(·), przekazując jako argument odpowiedni komparator (użyj referencji do istniejącej metody compare w klasie Integer). Wyświetl na konsoli wynik operacji.
- **4. filter(·)** Utwórz strumień z kolekcji ArrayList<Integer> i za pomocą metody filter(·) oraz odpowiedniego wyrażenia lambda jako argument, zwróć strumień, zawierający tylko elementy parzyste. Następnie, zastosuj operację kończącą forEach(·) w celu wyświetlenia odfiltrowanych elementów na konsoli.
- 5. sorted(·) Utwórz prostą klasę Person (zgodnie z kodem poniżej). Stwórz kilka obiektów tej klasy i dodaj je do kolekcji. Przejdź na strumień i posortuj osoby względem pola nick, a następnie pola age. Wykorzystaj konstrukcję: Comparator.comparing(Person::getNick).thenComparing(Person::getAge) Za pomocą operacji kończącej forEach(·) zaprezentuj wyniki sortowania.

```
public class Person {
   private String nick;
   private int age;

   // konstruktor, gettery i settery
}
```

- 6. map(·) Mamy dwie klasy PunktXY oraz PunktXYZ reprezentujące odpowiednio punkty na płaszczyźnie oraz w przestrzeni. Stwórz listę z kilkoma punktami przestrzennymi. Następnie, przechodząc na strumień odwzoruj obiekty klasy PunktXYZ na obiekty PunktXY (zmienną z odrzucamy). Na zakończenie utwórz ze strumienia kolekcję (metoda collect(·)) i w pętli for wypisz współrzędne x oraz y, korzystając obiektów klasy PunktXY.
- 7. **flatMap(·)** Utwórz dwie grupy "Eagles" oraz "Bikers" dodając kilka osób do każdej z nich.

```
public class Group {
   private String groupName;
   private List<Person> members;

public Group(String groupName, List<Person> members){
    this.groupName = groupName;
    this.members = members;
}

// gettery i settery

public class Person {
   private String nick;

public Person(String nick){
    this.nick= nick;
   }
   // gettery i settery
}
```

```
Następnie, dodaj utworzone grupy do listy:
List<Group> groups=Arrays.asList(eagles,bikers);
```

Z tak utworzonej listy grup chcemy otrzymać listę wszystkich osób z naszych grup. Wykorzystaj operację flatMap zgodnie z poniższą podpowiedzią. Wydrukuj na konsolę listę allMembers.

8. reduce(·) Stwórz kolekcję zawierającą kilka elementów typu Integer. Następnie, zsumuj elementy kolekcji wykorzystując metodę z punktu A poniżej. Używając metody z punktu B dokonaj mnożenia wszystkich elementów znajdujących się w kolekcji.

Ogólny mechanizm pozwalający obliczyć wartość ze strumienia.

- **A)** Optional<T> **reduce**(BinaryOperator<T> *accumulator*)
- **B)** T reduce(T identityVal, BinaryOperator<T> accumulator)

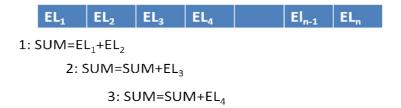


Operacje na akumulatorze muszą spełniać następujące kryteria:

- ✓ Bezstanowość (każdy element jest przetwarzany niezależnie od innych)
- ✓ Brak ingerencji (źródło danych nie jest modyfikowane przez operację)
- ✓ Łączność (a*b)*c=a*(b*c)

Jak to działa operacja redukcji...?

Strumień n-elementowy, operacja ((x,y)->x+y)



N-1: SUM=SUM+EL_n

9. parallelStream(⋅) Strumienie równoległe.

Wykorzystaj klasę java.util.UUID do wygenerowania miliona identyfikatorów UUID (z ang. universally unique identifier). Umieść je w kolekcji ArrayList.

Następnie pomierz czasy wykonania (STOP - START) następujących poleceń:

... // czas STOP

A) Przetwarzanie sekwencyjne
... // czas START
list.stream().sorted().collect(Collectors.toList());
... // czas STOP

B) Przetwarzanie równoległe
... // czas START

list.parallelStream().sorted().collect(Collectors.toList());

Porównaj otrzymane wyniki. Jak je wytłumaczyć? Zawrzyj odpowiedź w sprawozdaniu.