TP8 : TimeSeries

Exercice 1 - TimeSeries

1.
```
public class TimeSeries<T> {
public record Data<E>(long timestamp, E element) {
    public Data {
        Objects.requireNonNull(element);
    }
}
```

2.
```
private final ArrayList<Data<T>> list = new ArrayList<>();

public void add(long timestamp, T element) {
    Objects.requireNonNull(element);

    if (!list.isEmpty() && list.get(list.size() - 1).timestamp > timestamp) {
        throw new IllegalStateException("new timestamp need to be >= to the
last timestamp");
    }
    list.add(new Data<>(timestamp, element));
}

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

public Data<T> get(int index) {
    return list.get(index);
}
```

3. public class Index implements Iterable<Data> { private final int[] array;

```
private Index(int[] array) {
    this.array = array;
}
```

}

4.
```
 public record Data<E>(long timestamp, E element) {
    public Data {
        Objects.requireNonNull(element);
    }

    @Override
    public String toString() {
        return timestamp + " | " + element;
    }
}
public class Index implements Iterable<Data<T>> {
    private final int[] array;
```

```
    private Index(int[] array) {
        this.array = array;
    }

    public int size() {
        return array.length;
    }

    @Override
    public String toString() {
        return Arrays.stream(array)
                .mapToObj(list::get)
                .map(Data::toString)
                .collect(Collectors.joining("\n"));
    }
}
```

5. Le type du paramètre de **index(lambda)** doit être un **Predicate<? super T>** car on veut accepter les types T ainsi qu'un super-type de T.

```
public Index index(Predicate<? super T> predicate) {
    var array = IntStream.range(0, list.size())
            .filter(index ->
predicate.test(list.get(index).element)).toArray();
    return new Index(array);
}
```

6. Le type du paramètre de **forEach(lambda)** doit être un **Consumer<? super Data>**

```
public void forEach(Consumer<? super Data<T>> consumer) {
        Arrays.stream(array).mapToObj(list::get).forEach(consumer);
    }
```

7. **Index** doit implémenter l'interface **Iterable** pour pouvoir être utilisé dans une boucle. Elle doit implémenter la méthode **iterator** qui renvoie un **Iterator<Data>**

```
@Override
public Iterator<Data<T>> iterator() {
    return new Iterator<Data<T>>() {
        private int i;
        @Override
        public boolean hasNext() {
            return i < array.length;
        }
        public Data<T> next() {
            if (!hasNext()) {
                throw new NoSuchElementException("it has no next");
            }
            return list.get(array[i++]);
        }
    };
}
```

8.
```
public Index or(Index index) {
   Objects.requireNonNull(index);
   if (!equals(index)) {
      throw new IllegalArgumentException("Both index need to be in the same
TimeSeries");
   }
   return new Index(IntStream.concat(Arrays.stream(index.array),
Arrays.stream(this.array)).distinct().sorted().toArray());
}

@Override
public boolean equals(Object other) {
   return other instanceof TimeSeries.Index index && index.hashCode() ==
hashCode();
}

@Override
public int hashCode() {
   return list.hashCode();
}
```

Steve Chen