TP - Collections en Java

Emmanuel BRUNO
Université de Toulon, LIS UMR CNRS 7020
emmanuel.bruno@univ-tln.fr

2026-10-31

Table des matières

1 Objectif 1
1.1 Partie 1 : Manipulation basique des Sets 1
1.2 Partie 2 : Parcours et Conversion 2
1.3 Partie 3 : Collections spécialisées 2
2 Simulateur d'un Réseau de Capteurs avec Génération de Mesures Basée sur des Fonctions 3
2.1 Objectif 3
2.2 Étapes de l'exercice 3
2.3 Une illustration du diagramme de classe et du programmes principal possible est donné ci-dessous. 4

1 Objectif

L'objectif de ce TP est de présenter l'utilisation des collections.

1.1 Partie 1 : Manipulation basique des Sets

1. Créez un `HashSet` d'animaux avec les attributs suivants :

```java
public class Animal {
    private String nom;
    private int age;
    private String espece;
    private double poids;
    private LocalDate dernierExamen;

    // Constructeur, getters, setters
    // Implémenter equals() et hashCode() basés sur le nom
}
```

2. Implémentez `equals()` et `hashCode()` pour définir l'égalité basée sur le nom.
3. Créez un HashSet et ajoutez 5 animaux

4. Démontrez le fonctionnement de l'unicité dans un test unitaire avec Junit5:

```
@Test
public void testUniciteAnimaux() {
    Set<Animal> animaux = new HashSet<>();
    Animal lion1 = new Animal("Leo", 5, "Lion");
    Animal lion2 = new Animal("Leo", 3, "Lion");
    assertTrue(animaux.add(lion1));
    assertFalse(animaux.add(lion2));
}
```

1.2 Partie 2 : Parcours et Conversion

1. Implantez deux méthodes de parcours différentes :

- Avec un Iterator classique
- Avec un forEach (Java 8+)

2. Implémentez Comparable pour trier par nom.
3. Convertissez votre HashSet en TreeSet en implémentant `Comparable<Animal>` :

- Démontrez la différence entre List et Set :
  - Créez une ArrayList à partir de votre Set
  - Ajoutez des doublons

4. Créez différents Comparator :

- `Comparator<Animal> parPoids = ...`
- `Comparator<Animal> parDernierExamen = ...`
- Trier et afficher les animaux selon ces critères

1.3 Partie 3 : Collections spécialisées

1. Créez une classe `Tatouage` immuable

```
public final class Tatouage {
    private final String espece;
    private final int numero;
    // Constructeur, equals, hashCode
}
```

2. Ajoutez un attribut `tatouage` à la classe `Animal`
3. Implémentez un système de gestion d'animaux à partir de :

```
public class Clinique {
    private Map<Tatouage, Animal> registre;
    private Queue<Animal> salleAttente;
```

```
    public void ajouterAnimal(Animal animal) {
        // À implémenter
    }

    public Animal chercherAnimal(Tatouage tatouage) {
        // À implémenter
    }

    public void ajouterSalleAttente(Tatouage tatouage) {
        // Vérifier si l'animal est dans le registre
        // À implémenter
    }

    public Animal prochaineConsultation() {
        // Retirer et retourner le prochain animal de la salle d'attente
        // À implémenter
    }
}
```

2 Simulateur d'un Réseau de Capteurs avec Génération de Mesures Basée sur des Fonctions

2.1 Objectif

L'objectif de cet exercice est de créer un réseau de capteurs capable de générer des mesures basées sur des fonctions mathématiques. Vous allez implémenter plusieurs classes pour modéliser les capteurs, les mesures, et le réseau de capteurs, ainsi qu'une classe pour générer des mesures en fonction du temps.

Cet exercice reprend tous les concepts vus depuis le début y compris les collections avec une illustration du typage fort avec l'utilisation d'un standard et d'une librairie pour gérer les unités dans les mesures https://belief-driven-design.com/java-measurement-jsr-385-210f2/

Cette partie sera traitée avec Git AVEC LE LIEN GITHUBCLASSROOM « TP-Collection » fourni

2.2 Étapes de l'exercice

1. Création de l'interface `Function`

Créez une interface `Function` qui représente une fonction mathématique prenant le temps en secondes comme entrée et retournant un double comme sortie. Ajoutez la méthode `apply(long x)` pour évaluer la fonction à un instant donné.

2. Implémentation de la classe `Sensor`

Créez une classe Sensor générique qui peut stocker des mesures de différents types de quantités (par exemple, température, pression). Ajoutez des méthodes pour ajouter des mesures et itérer sur celles-ci.

3. Implémentation de la classe `Measurement`

Créez une classe Measurement pour représenter une mesure prise par un capteur à un instant donné. Incluez des attributs pour le capteur, le timestamp, et la valeur de la mesure.

4. Implémentation de la classe `SensorNetwork`

Créez une classe SensorNetwork pour gérer un ensemble de capteurs. Ajoutez des méthodes pour ajouter des capteurs de température et de pression, et pour itérer sur les capteurs du réseau.

5. Implémentation de la classe `FunctionBasedMeasureGenerator`

Créez une classe FunctionBasedMeasureGenerator pour générer des mesures basées sur une fonction mathématique. Ajouter une méthode `generate` qui génère des mesures pour un capteur donné sur une période de temps spécifiée.

6. Création de l'application principale `App`

Créez une classe `App` avec une méthode main pour initialiser un réseau de capteurs, ajouter des capteurs, et générer des mesures pour ces capteurs. Utilisez la classe `FunctionBasedMeasureGenerator` pour générer des mesures de température basées sur une fonction linéaire simple.

7. Question bonus
    a. Ajoutez des méthodes à la classe `Sensor` pour calculer la moyenne des mesures, le min, et le max. (Après Java 8 voir `DoubleSummaryStatistics`)
    b. Déléguer cela dans la classe `SensorNetwork` pour tous les capteurs du réseau.

8. Super Bonus.

Proposer une solution pour détecter les anomalies dans les mesures par exemple en utilisant le z-score.

2.3 Une illustration du diagramme de classe et du programmes principal possible est donné ci-dessous.

```
        // Create a sensor network
        SensorNetwork sensorNetwork1 = SensorNetwork.of("Sensor network 1");

        // Add temperature and pressure sensors to the network
```

```
      Sensor<Temperature> sensor1 = sensorNetwork1.addTemperatureSensor("T_1");

        Sensor<Pressure> sensor2 = sensorNetwork1.addPressureSensor("P_1");

        // Add initial measurements to the sensors
             sensor1.addMeasurement(Instant.now(),  Quantities.getQuantity(25,
Units.CELSIUS))
                                  .addMeasurement(Instant.now().plusSeconds(15),
Quantities.getQuantity(300.15, Units.KELVIN));

          sensor2.addMeasurement(Instant.now(), Quantities.getQuantity(1013.25,
Units.PASCAL));

        // Create a function-based measure generator for temperature
            FunctionBasedMeasureGenerator<Temperature>  temperatureGenerator  =
FunctionBasedMeasureGenerator.<Temperature>builder()
                .function(
                        new Function() {
                            @Override
                            public double apply(long x) {
                               // Simple linear function: temperature increases
by 1 degree per second
                                return 20 + 1 * x;
                            }
                        })
                .startTime(Instant.now())
                .endTime(Instant.now().plusSeconds(10))
               .timeStep(Duration.of(1, java.time.temporal.ChronoUnit.SECONDS))
                .variance(2)
                .outlierProbability(0.1)
                .build();

        // Generate measurements for the temperature sensor
        temperatureGenerator.generate(sensor1, Units.CELSIUS);

        // Log the sensor network details
        log.info("{}", sensorNetwork1);

        // Iterate over the sensors in the network
        Iterator<Sensor> sensorIterator = sensorNetwork1.sensorIterator();
        while (sensorIterator.hasNext()) {
            Sensor sensor = sensorIterator.next();
            log.info("{}", sensor);
```

```
            // Iterate over the measurements of each sensor
                                Iterator<Measurement>   measurementIterator   =
sensor.measurementIterator();
            while (measurementIterator.hasNext()) {
                Measurement measurement = measurementIterator.next();
                log.info("{}", measurement);
            }
        }
```