ALOM - TP - Modern Java

Table of Contents

Dans ce TP, nous allons un peu jouer avec les nouveautés du langage Java :

- records
- streams & lambdas
- multi-line strings

Nous avons bien besoin de Java 17 (minimum) pour ce TP !

1. Initialisation du projet

Initialisez votre projet avec ce lien : https://gitlab-classrooms.cleverapps.io/assignments/9a50d05f-7a5b-4d26-8756-0ee8e82fac12/accept

Initialisez un `pom.xml`, ainsi que les répertoires `src/main/java` et `src/test/java`:

```
<project>
    <modelVersion>4.0.0</modelVersion>
    <groupId>com.miage.alom.tp</groupId>
    <artifactId>w02-modern-java</artifactId>
    <version>0.1.0</version>
    <packaging>jar</packaging>

    <properties>
        <maven.compiler.source>17</maven.compiler.source> ①
        <maven.compiler.target>17</maven.compiler.target> ②
    </properties>

    <dependencies>
    </dependencies>

</project>
```

① On indique à maven quelle version de Java utiliser pour les sources !

② On indique à maven quelle version de JVM on cible !

Maven considère par défaut que le code est écrit en Java 7 !

Ajoutez les dépendances JUnit :

pom.xml

```
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-api</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.10.0</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.24.2</version>
    <scope>test</scope>
</dependency>
```

Il vous faut également surcharger la version du `maven-surefire-plugin` (qui est le plugin maven qui implémente la phase d'exécution des tests).

pom.xml

```
<build>
  <pluginManagement>
    <plugins>
        <plugin>
            <artifactId>maven-surefire-plugin</artifactId>
            <version>3.1.2</version> ①
        </plugin>
    </plugins>
  </pluginManagement>
</build>
```

① On a besoin de la version 2.22.0 minimum pour JUnit 5 comme indiqué dans la documentation junit

Pour que votre TP soit évalué automatiquement, ajoutez également le fichier `.gitlab-ci.yml` suivant à la racine de votre projet :

.gitlab-ci.yml

```
include:
```

```
  project: gitlab-classrooms/autograding
  file: maven-junit.yml
```

2. Records

Dans un premier temps, nous allons manipuler des records.

Créez les records suivants :

- PokemonType
  - id: int
  - name: String
  - types: List<String>
- Pokemon
  - type: PokemonType
  - nickname: String
  - level: int

Le record `PokemonType` représente les types de Pokemon (Pikachu, Rattata, ...).

Le record `Pokemon` représente un Pokemon particulier (Le Pikachu de Sasha, le Rattata de Julien...).

Importez dans votre code les tests unitaires suivants :

```java
import org.junit.jupiter.api.Test;

import java.util.List;

import static org.assertj.core.api.Assertions.assertThat;

public class RecordsTest {

    @Test
    public void testPokemonTypeCreation(){
        var pikachu = new PokemonType(25, "Pikachu", List.of("electric"));

        assertThat(pikachu.id()).isEqualTo(25);
        assertThat(pikachu.name()).isEqualTo("Pikachu");
        assertThat(pikachu.types()).containsOnly("electric");
    }

    @Test
    public void testPokemonCreation(){
        var geodude = new PokemonType(74, "Racaillou", List.of("rock", "ground"));
        var petersGeodude = new Pokemon(geodude, "Racaillou de Pierre", 12);

```

```
23          assertThat(petersGeodude.nickname()).isEqualTo("Racaillou de Pierre");
24          assertThat(petersGeodude.type()).isEqualTo(geodude);
25          assertThat(petersGeodude.level()).isEqualTo(12);
26      }
27
28  }
```

3. Streams et Lambdas

Nous allons maintenant charger une liste de types de Pokemons, et la manipuler avec des Streams.

Récupérez le fichier pokemons.json, et placez-le dans le répertoire `src/main/resources` de votre projet.

Pour charger le fichier JSON, nous allons devoir utiliser la librairie `jackson-databind`:

pom.xml

```
1 <dependency>
2     <groupId>com.fasterxml.jackson.core</groupId>
3     <artifactId>jackson-databind</artifactId>
4     <version>2.15.2</version>
5 </dependency>
```

Importez et complétez la classe suivante :

```
1  public class PokemonStreams {
2
3      private Collection<PokemonType> pokemonsTypes;
4
5      public void loadPokemonTypes() throws IOException {
6          var objectMapper = new ObjectMapper();
7          objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES,
   false);
8
9          this.pokemonsTypes = objectMapper.readValue(new FileInputStream
   ("src/main/resources/pokemons.json"), new TypeReference<Collection<PokemonType>>()
   {});
10     }
11
12     public List<PokemonType> sortById(){
13         // TODO
14     }
15
16     public Set<PokemonType> findByType(String type) {
17         // TODO
18     }
19
```

```
    public Optional<PokemonType> findFirstByTypes(String... types) {
        // TODO
    }
}
```

Vous pouvez valider vos développements avec la classe de test suivante :

```
public class PokemonStreamsTest {

    PokemonStreams pokemonStreams;

    @BeforeEach
    void setUp() throws IOException {
        pokemonStreams = new PokemonStreams();
        pokemonStreams.loadPokemonTypes();
    }

    @Test
    public void testSortById(){
        assertThat(pokemonStreams.sortById())
                .extracting(it -> it.id())
                .isSorted();
    }

    @Test
    public void testFindElectricPokemons(){
        assertThat(pokemonStreams.findByType("electric"))
                .hasSize(9);
    }

    @Test
    public void testFindFirePokemons(){
        assertThat(pokemonStreams.findByType("fire"))
                .hasSize(12);
    }

    @Test
    public void testFindFirstPsychicPokemon(){
        assertThat(pokemonStreams.findFirstByTypes("psychic"))
                .isNotEmpty()
                .get()
                .extracting("name")
                .isEqualTo("abra");
    }

    @Test
    public void testFindFirstUnknownPokemon(){
        assertThat(pokemonStreams.findFirstByTypes("unknown"))
                .isEmpty();
    }
```

```
}
```

4. Multi-Line Strings

Importez le test suivant, et faites ce qu'il faut pour qu'il passe !

```
public class TextBlockTest {

    ObjectMapper objectMapper;

    @BeforeEach
    void setUp() {
        objectMapper = new ObjectMapper();
        objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);
    }

    @Test
    void pokemonAsJsonString() throws JsonProcessingException {
        var jsonStringOldFashioned = "{\n" +
                "    \"id\": 47,\n" +
                "    \"name\": \"parasect\",\n" +
                "    \"baseExperience\": 142,\n" +
                "    \"weight\": 295,\n" +
                "    \"height\": 10,\n" +
                "    \"types\": [\n" +
                "      \"grass\",\n" +
                "      \"bug\"\n" +
                "    ],\n" +
                "    \"stats\": {\n" +
                "      \"speed\": 30,\n" +
                "      \"attack\": 95,\n" +
                "      \"defense\": 80,\n" +
                "      \"hp\": 60\n" +
                "    },\n" +
                "    \"sprites\": {\n" +
                "      \"front_default\": \"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/47.png\",\n" +
                "      \"back_default\": \"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/47.png\"\n" +
                "    }\n" +
                "  }";
        var pokemonTypeFromJsonString = objectMapper.readValue(jsonStringOldFashioned, PokemonType.class);
        System.out.println("pokemonType.toString() = " + pokemonTypeFromJsonString.toString());

```

```
37
38          // TODO : écrivez un text block ici !
39          String textBlockString = null;
40          var pokemonTypeFromTextBlock = objectMapper.readValue(textBlockString,
    PokemonType.class);
41
42          assertThat(pokemonTypeFromJsonString).isEqualTo(pokemonTypeFromTextBlock);
43      }
44  }
```