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 21 (minimum) pour ce TP !

1. Initialisation du projet

Initialisez votre projet avec ce lien : https://univ-lille.gitlab-classrooms.org/assignments/db2396e1-9790-46ab-bf21-dbe6f195ec1e/accept

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

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

    <properties>
        <maven.compiler.source>21</maven.compiler.source> ①
        <maven.compiler.target>21</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>5.13.4</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.junit.jupiter</groupId>
    <artifactId>junit-jupiter-engine</artifactId>
    <version>5.13.4</version>
    <scope>test</scope>
</dependency>
<dependency>
    <groupId>org.assertj</groupId>
    <artifactId>assertj-core</artifactId>
    <version>3.27.4</version>
    <scope>test</scope>
</dependency>
```

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);

        assertThat(petersGeodude.nickname()).isEqualTo("Racaillou de Pierre");
        assertThat(petersGeodude.type()).isEqualTo(geodude);
        assertThat(petersGeodude.level()).isEqualTo(12);
    }

}
```

3. Streams et Lambdas

Nous allons maintenant charger une liste de types de Pokemon, 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

```xml
<dependency>
    <groupId>com.fasterxml.jackson.core</groupId>
    <artifactId>jackson-databind</artifactId>
    <version>2.20.0</version>
</dependency>
```

Importez et complétez la classe suivante :

```java
public class PokemonStreams {

    private Collection<PokemonType> pokemonsTypes;

    public void loadPokemonTypes() throws IOException {
        var objectMapper = new ObjectMapper();
        objectMapper.configure(DeserializationFeature.FAIL_ON_UNKNOWN_PROPERTIES, false);

        this.pokemonsTypes = objectMapper.readValue(new FileInputStream("src/main/resources/pokemons.json"), new TypeReference<Collection<PokemonType>>() {});
    }

    public List<PokemonType> sortById(){
        // TODO
    }

    public Set<PokemonType> findByType(String type) {
        // TODO
    }

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

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

```java
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);
    }
```

```
10
11      @Test
12      void pokemonAsJsonString() throws JsonProcessingException {
13          var jsonStringOldFashioned = "{\n" +
14                  "    \"id\": 47,\n" +
15                  "    \"name\": \"parasect\",\n" +
16                  "    \"baseExperience\": 142,\n" +
17                  "    \"weight\": 295,\n" +
18                  "    \"height\": 10,\n" +
19                  "    \"types\": [\n" +
20                  "      \"grass\",\n" +
21                  "      \"bug\"\n" +
22                  "    ],\n" +
23                  "    \"stats\": {\n" +
24                  "      \"speed\": 30,\n" +
25                  "      \"attack\": 95,\n" +
26                  "      \"defense\": 80,\n" +
27                  "      \"hp\": 60\n" +
28                  "    },\n" +
29                  "    \"sprites\": {\n" +
30                  "      \"front_default\":
    \"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/47.png\"
    ,\n" +
31                  "      \"back_default\":
    \"https://raw.githubusercontent.com/PokeAPI/sprites/master/sprites/pokemon/back/47.
    png\"\n" +
32                  "    }\n" +
33                  "  }";
34          var pokemonTypeFromJsonString = objectMapper.readValue
    (jsonStringOldFashioned, PokemonType.class);
35          System.out.println("pokemonType.toString() = " + pokemonTypeFromJsonString
    .toString());
36
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  }
```