AEPE: TAE Challenge

Objective
Evaluate the candidate's knowledge of Java (preferably Java 11 or higher), including lambdas, streams, API calls, and classes based on OOP.

1. API

The candidate must perform a series of tasks using the Pokémon API.

Requirements

Java: Knowledge of Java 8 or higher (Java 11 preferred).
Maven: Use Maven as a project management tool.
Libraries: You may use any library to make API calls, such as RestAssured or RestTemplate.
Testing: The code must include tests using JUnit or a similar framework.
API: `https://pokeapi.co/api/v2/`

Tasks

1. Project Setup

    Create a Maven project.
    Ensure to include the necessary dependencies in the pom.xml file, especially for JUnit and the library you use for API calls. For example:

```java
<dependencies>
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter</artifactId>
        <version>${junit.version}</version>
        <scope>test</scope>
    </dependency>
    <dependency>
```

```
        <groupId>io.rest-assured</groupId>
        <artifactId>rest-assured</artifactId>
        <version>${restassured.version}</version>
    </dependency>
</dependencies>
```

**The dependencies shown above serve as an example; you can use any dependency you choose.

2. Class Creation

Create a ServiceEndPoint class that will handle API calls.

```java
public class ServiceEndPoint {

    private final String BASE_URL = "https://pokeapi.co/api/v2/";

    public List<Pokemon> getPokemones() {
        // Implement logic to call the Pokémon API and return the list of
Pokémon.
    }

    public Pokemon getPokemonDetails(String pokemonName) {
        // Implement logic to call the Pokémon API with the given name and
return the details.
    }

    public List<Ability> getAbilities() {
        // Implement logic to call the abilities API.
    }
}
```

Create the necessary model classes, such as Pokemon and Ability.

3. Logic Implementation

Implement the logic in ServiceEndPoint to perform the following calls:

1. Retrieve the list of Pokémon from the endpoint **`https://pokeapi.co/api/v2/pokemon`**/.
2. Extract the name of a Pokémon from the list (for example, "raichu").

3. Use the name of the Pokémon to call the endpoint `https://pokeapi.co/api/v2/pokemon/{name}` and retrieve its details, especially the abilities.

4. Test Creation

Create a test class ApiTest with the following empty methods that the candidate must implement:

```java
public class ApiTest {

    private final ServiceEndPoint serviceEndPoint = new ServiceEndPoint();

    @Test
    public void getPokemonListTest() {
        // Implement the logic to retrieve the list of Pokémon and perform an assertion.
    }

    @Test
    public void getPokemonDetailsTest() {
        // Implement the logic to retrieve details of a specific Pokémon and perform assertions.
    }

    @Test
    public void getPokemonAbilitiesAndValidateInAbilityEndpointTest() {
        // Implement the logic to extract abilities from the Pokémon and validate in the abilities endpoint. Encourage use lambdas
    }
}
```

Tips and Recommendations

- Make sure to handle exceptions properly in your API calls.
- We encourage the use of Streams and Lambdas.
- Keep your code clean and well-commented.
- Conduct thorough tests to validate your implementation.

2. DATABASE

Evaluate the candidate's ability to connect to the MySQL database, handle exceptions, and work with query results in Java.

Requirements

Java: Knowledge of Java 8 or higher (Java 11 preferred).
Maven: Use Maven as a project management tool.
Libraries: You may use any library to make database connections, we recommend choosing one of the following:

- https://www.baeldung.com/java-in-memory-databases
- https://www.baeldung.com/java-connect-mysql

Tasks

1. Ensure adding necessary dependencies to create a database connection
2. Create a class named **`DatabaseConnection`** that handles a connection
3. Create a class named **`DatabaseUtils`** that contains an `executeQuery(String query)` method to execute queries, and return results as `(List<Map<String, Object>>)`.
4. Add a query as an example where you need to include multiple parameters, and try to use more than one.

Tips and Recommendations

- Make sure to handle exceptions properly in your database connection
- Keep your code clean and well-commented.

3. WEB TESTING

Requirements

Java: Knowledge of Java 8 or higher (Java 11 preferred).
Maven: Use Maven as a project management tool.
Libraries: Use necessary libraries such as Selenium, AssertJ, and JUnit.
WebPage: The Internet web page navigating to the Download section:

- The Internet

Tasks

1. Project setup

   Create a Maven project and set up the pom.xml file with the necessary dependencies for Selenium, AssertJ, and JUnit.

```Java
 <dependency>
        <groupId>org.seleniumhq.selenium</groupId>
        <artifactId>selenium-java</artifactId>
        <version>${version}</version>
    </dependency>

    <!-- AssertJ -->
    <dependency>
        <groupId>org.assertj</groupId>
        <artifactId>assertj-core</artifactId>
        <version>${version}</version>
    </dependency>

    <!-- JUnit -->
    <dependency>
        <groupId>org.junit.jupiter</groupId>
        <artifactId>junit-jupiter-engine</artifactId>
        <version>${version}</version>
        <scope>test</scope>
    </dependency>
```

2. Create the Page Object Model (POM):

   - Create a class called **`DownloadPage`** that represents the download page on the website.
   - In this class, define a method called **`downloadFile()`** that performs actions to download a .txt file by name.
   - Create a class to read the downloaded file.

3. Write the Test Method:
   Implement a test that navigates to the Internet page, chooses a file and waits for a moment to ensure it has been downloaded. (Consider using `WebDriverWait` for a production scenario).

   - Verifies that the downloaded file exists in the default downloads directory.

- Reads the contents of the downloaded file and performs assertions (using AssertJ optional) to check that the content is not empty

Tips and Recommendations

- Include a Before component to setup and open the page
- Include an After component to quit the driver.

Submission

Submit the project as a compressed file or in a GitHub repository with instructions on how to compile and run the tests.