# Spring Data Regular Exam

# Solar Moons

*Moons, or natural satellites, are celestial bodies that orbit planets or dwarf planets. In our Solar System, there are over 200 known moons, each with unique characteristics. Moons vary widely in size, composition, and behavior, with some potentially harboring conditions for life. Let's take a look at the bigger and more significant ones.*

## Functionality Overview

The application should be able to easily **import** hard-formatted data and **support functionalities** for also **exporting** the imported data. The application is called – **Solar Moons**.

Look at the pictures below to see what must happen:

* The home page before importing anything:

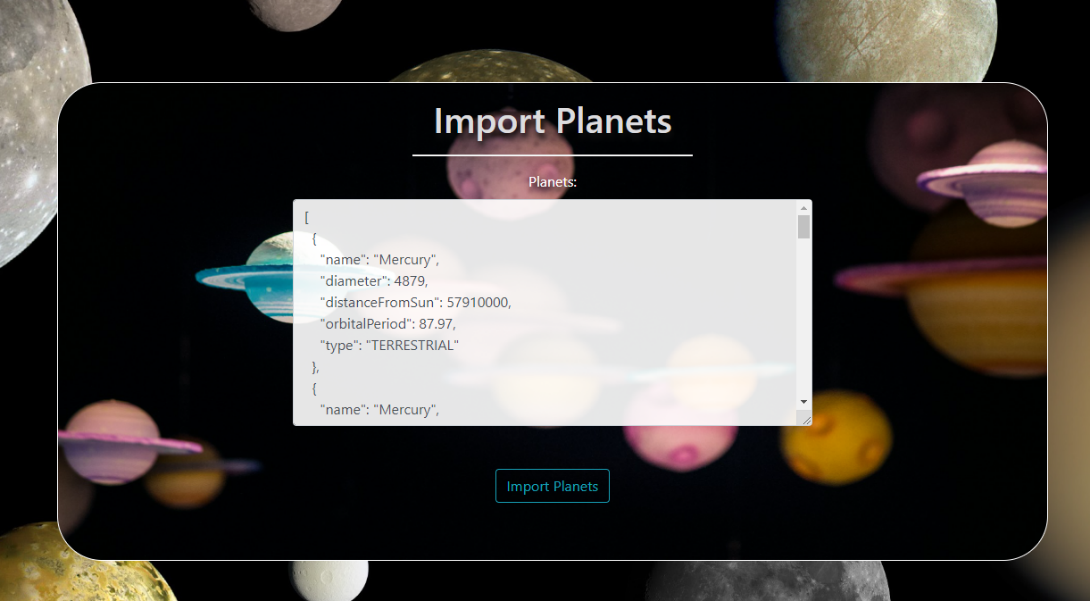


* The import JSON page before importing anything:

A screenshot of a computer screen

Description automatically generated

* Import the **planets** first:



* Import the **discoverers** second:

A screenshot of a computer

Description automatically generated

* The import JSON page after importing both files:

A screenshot of a computer screen

Description automatically generated

* The import XML page before importing the given data:

A screenshot of a computer screen

Description automatically generated

* Import the **moon's** data:

A screenshot of a computer

Description automatically generated

* The import XML page after importing the data:A screenshot of a computer

  Description automatically generated
* The home page after the data is imported:

A screenshot of a computer screen

Description automatically generated

* Export moons that are satellites of planets of type **GAS\_GIANT** and have radius in the range from 700km. to 2000km.

A screenshot of a computer

Description automatically generated

## Project Skeleton Overview

You will be given a **skeleton**, containing a **certain architecture (MVC)** with **several classes**, some of which are completely empty. The **Skeleton** will include the **files** with which you will **seed** the **database**.

You must use **SDK(JDK) 21** for this project. Make sure you use **jakarta** for the package import statement instead of **javax**. **Don't change the skeleton structure.**

When submitting your project in Judge please make a **.zip** file only with **src** folder and **pom.xml** file.  
There are four problems in **Judge**:

**1. Database** will test the structure of your database.

**2. Service** will test the two methods **areImported()** and **readClassFromFile()** in every service class.  
**3. Import JSON** will test the logic of importing **JSON** files.

**4. Import XML** will test the logic of importing **XML** files.

**5. Export** will test the logic of the final **export** button which is in the **ExportController**.

Please note that the automated system Judge may take up to **5 minutes** to show results for each problem. We kindly ask you to be patient.

**Be aware that the Judge might not recognize the var keyword, leading to compilation errors.**

To help you ensure the correctness of your solutions and **gain immediate feedback**, we have included a set of **commented-out tests** within the provided skeleton project. These tests are designed to verify the functionality of the tasks you are required to implement.

## Model Definition

There are 3 main models that the **Solar Moons database** application should contain in its functionality.

We have provided you with an Entity-Relationship (ER) diagram that represents the database schema. The ER diagram displays the tables and their relationships, along with the field types for each table.

A screenshot of a computer

Description automatically generated

Ensure the **Java** code **validates** data against the specified constraints before persisting it into the database. Handle any validation errors gracefully and provide meaningful feedback to the users.

Design them most appropriately, considering the following **data constraints**:

### Planet

* id – accepts **integer** values, a **primary identification field, and an auto incremented field**.
* diameter – accepts positive **number** values. It **cannot be nullable**.
* **distance from sun** - accepts **large** positive number values. It **cannot be nullable**.
* name – accepts **char sequence** (between **3** to **20** both inclusive). The values are **unique in the database**.  
  It **cannot be nullable**.
* orbital period - accepts positive **double** values. It **cannot be nullable.**
* type – categorization of the planets. **String enumeration**, one of the following – **DWARF\_PLANET, GAS\_GIANT, ICE\_GIANT, TERRESTRIAL.** It **cannot be nullable**.
* Constraint: The **planets** table has a relation with the **moon's** table. It **cannot be nullable**.

### Discoverer

* id – accepts **integer** values, a **primary identification field, an auto incremented field**.
* first name - accepts **char sequence** (between **2** to **20** both inclusive). It **cannot be nullable**.
* last name - accepts **char sequence** (between **2** to **20** both inclusive). It **cannot be nullable**
* **nationality** – accepts **char sequence** (between **5** to **20** both inclusive). It **cannot be nullable**.
* occupation – accepts **char sequence** (between **5** to **20** both inclusive). It can be nullable.
* Constraint: The **discoverers** table has a relation with the **moon's** table. It **cannot be nullable**.

### Moon

* id - accepts **integer** values, a **primary identification field, an auto incremented field**.
* **discovered** - indicates the date the moon was discovered. It **cannot be nullable**.
* distance from planet - accepts positive number values. It can be nullable.
* name – accepts **char sequence** (between **2** to **10** both inclusive). The values are **unique in the database**. It **cannot be nullable**.
* radius - accepts positive **double** values. It **cannot be nullable**.
* Constraint: The **moons** table has a relation with **planets** and **discoverers** tables. They **cannot be nullable**.

### Relationships

Your partners gave you a little hint about the more complex relationships in the database so that you can implement it correctly.

One **Planet** may have many **Moons** orbiting it, but one **Moon** can only orbit one **Planet**.

One **Moon** was discovered by only one **Discoverer**, but one **Discoverer** may have discovered many **Moons**.

#### Constraint

* Name the entities and their class members **exactly** in the **format stated** above.
* All fields are **NOT NULL** unless explicitly stated to be nullable.

## Data Import

Use the provided files to populate the database with data. Import all the information from those files into the database.

**You are not allowed to modify the provided files.**

**ANY INCORRECT** data should be **ignored** and a message should be printed:

**"Invalid {planet/discoverer/moon}"**

**When the import is finished:**

**"Successfully imported {planet/discoverer/moon} {planetName}/{discovererFirstName} {discovererLastName}/{moonName)"**

**Judge** will only accept **file paths** in a specific format. When dealing with file paths for files, please adhere to the following format: "src/main/resources/files/xml/format-example.xml"

### JSON Import

Your new colleagues have prepared some JSON data for you to import.

#### Planets (planets.json)

##### Constraint

* **If a planet with the same name already exists in the DB or name does not meet size constraints return "Invalid planet".**
* **When the import is finished:**

**"Successfully imported planet {planetName}"**

|  |
| --- |
| **Planets (planets.json)** |
| [  {  "name": "Mercury",  "diameter": 4879,  "distanceFromSun": 57910000,  "orbitalPeriod": 87.97,  "type": "TERRESTRIAL"  },  {  "name": "Mercury",  "diameter": 4879,  "distanceFromSun": 57910000,  "orbitalPeriod": 87.97,  "type": "TERRESTRIAL"  },  {  "name": "Me",  "diameter": 4879,  "distanceFromSun": 57910000,  "orbitalPeriod": 87.97,  "type": "TERRESTRIAL"  },  {  "name": "Venus",  "diameter": 12104,  "distanceFromSun": 108200000,  "orbitalPeriod": 224.7,  "type": "TERRESTRIAL"  },  … |
| Successfully imported planet Mercury  Invalid planet  Invalid planet  Successfully imported planet Venus |

#### Discoverers (discoverers.json)

##### Constraint

* **If a discoverer with the combination of first and last name already exists in the DB return "Invalid discoverer".**
* **When the import is finished:**

**"Successfully imported discoverer {discovererFirstName} {discovererLastName}"**

* **The provided discoverer ids will always be valid.**

|  |
| --- |
| **Discoverers (discoverers.json)** |
| [  {  "firstName": "Galileo",  "lastName": "Galilei",  "nationality": "Italian",  "occupation": "astronomer",  },  {  "firstName": "Galileo",  "lastName": "Galilei",  "nationality": "Italian",  "occupation": "astronomer",  },  {  "firstName": "C",  "lastName": "Huygens",  "nationality": "Dutch",  "occupation": "mathematician",  },  {  "firstName": "Christiaan",  "lastName": "Huygens",  "nationality": "Dutch",  "occupation": "mathematician",  },  … |
| Successfully imported discoverer Galileo Galilei  Invalid discoverer  Invalid discoverer  Successfully imported discoverer Christiaan Huygens |

### XML Import

Your new colleagues have prepared some XML data for you to import.

#### Moons (moons.xml)

##### Constraint

* **If a moon with the same name already exists in the DB return "Invalid moon".**
* **When the import is finished:**

**"Successfully imported moon {moonName}"**

* **The provided discoverer and planet ids will always be valid.**

|  |
| --- |
| **Moons (moons.xml)** |
| *<?*xml version='1.0' encoding='UTF-8'*?>* <moons>  <moon>  <name>Moon</name>  <discovered>1609-11-30</discovered>  <distance\_from\_planet>384400</distance\_from\_planet>  <radius>1737.4</radius>  <discoverer\_id>1</discoverer\_id>  <planet\_id>3</planet\_id>  </moon>  <moon>  <name>Moon</name>  <discovered>1609-11-30</discovered>  <distance\_from\_planet>384400</distance\_from\_planet>  <radius>1737.4</radius>  <discoverer\_id>1</discoverer\_id>  <planet\_id>3</planet\_id>  </moon>  <moon>  <name>Phobos</name>  <discovered>1877-08-18</discovered>  <distance\_from\_planet>9377</distance\_from\_planet>  <radius>11.27</radius>  <discoverer\_id>6</discoverer\_id>  <planet\_id>4</planet\_id>  </moon>  <moon>  <name>P</name>  <discovered>1877-08-18</discovered>  <distance\_from\_planet>9377</distance\_from\_planet>  <radius>11.27</radius>  <discoverer\_id>6</discoverer\_id>  <planet\_id>4</planet\_id>  </moon>  … |
| Successfully imported moon Moon  Invalid moon  Successfully imported moon Phobos  Invalid moon |

Please be aware that due to variations in local settings on different computers, the representation of decimal numbers (Double) may differ. In some regions, the decimal separator is a comma (,), while in others, it is a dot (.).

Judge local settings represent all decimal numbers using a dot (.) as the decimal separator. *Locale.US*

## Data Export

Get ready to export the data you have imported in the previous task. Here you will have some complex database querying. Export the data in the formats specified below.

### Export all moons that orbits planets of type GAS\_GIANT and have radius between 700km and 2000km. from the Database

* Extract from the database, the **moon name, the name of the planet it orbits, moon radius and moon discoverer**.
* **Filter only moons whose radius is more than or equal to 700km and equal to or less than 2000km.** Order the results by **moon name** in **ascending order.**
* You have to round the value of the **radius** to the **second** decimal digit.
* Return the information in this format:

**"\*\*\*Moon {moonName} is a natural satellite of {planetName} and has a radius of {moonRadius} km.**

**\*\*\*\*Discovered by {discovererFirstName} {discoverereLastName}**

**. . ."**

A screenshot of a computer screen

Description automatically generated