# Spring Data Exam

# Mobiles

*Mobile smart devices are portable electronic gadgets equipped with advanced features such as internet connectivity, touchscreen interfaces, and downloadable applications. These devices include smartphones, tablets and so on, which allow users to make phone calls, send messages, browse the internet, access social media, play games, and perform various tasks on the go. With powerful processors, high-resolution displays, and integrated sensors, mobile smart devices offer an immersive and interactive user experience. They have become essential tools for communication, productivity, entertainment, and information access in today's digital age.*

## 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 – **Mobiles**.

Look at the pictures below to see what must happen:

* The home page before importing anything:

A screenshot of a computer

Description automatically generated

* The import JSON page before importing anything:

A close up of a blue device

Description automatically generated

* Import the **sellers** first:

A screenshot of a computer

Description automatically generated

* Import the **sales** second:

A close up of a cell phone

Description automatically generated

* The import JSON page after importing both files:

A screenshot of a computer

Description automatically generated

* The import XML page before importing the given data:

* Import the **device's** data:

A close up of a cell phone

Description automatically generated

* The import XML page after importing the data:

A screenshot of a cell phone

Description automatically generated

* The home page after the data is imported:

A screenshot of a computer

Description automatically generated

* Export devices from type **SMART\_PHONE** with a price less than 1000 and storage equal to or greater than 128:

A close up of a camera

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**.

**Don't change the skeleton.**

When submitting your project in [Judge](https://alpha.judge.softuni.org/contests/java-db-spring-data-retake-exam-14-april-2024/4534/practice) please make a **.zip** file only with **src** folder and **pom.xml** file.  
There are five problems in [Judge](https://alpha.judge.softuni.org/contests/java-db-spring-data-retake-exam-14-april-2024/4534/practice):

**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 **Mobiles 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**:

### Seller

* id – accepts **integer** values, a **primary identification field, and an auto incremented field**.
* first name – accepts **char sequence** (between **2** to **30** both inclusive). It **cannot be nullable**.
* last name – accepts **char sequence** (between **2** to **30** both inclusive). The values are **unique in the database**. It **cannot be nullable**.
* personal number – accepts **char sequence** (between **3** to **6** both inclusive). The values are **unique in the database**. It **cannot be nullable**.
* Constraint: The **sellers** table has a relation with the **sales** table. It can be nullable.

### Sale

* id – accepts **integer** values, a **primary identification field, an auto incremented field**.
* discounted - accepts a true or false, representing whether the sale is at a lower price. It can be nullable.
* number – Every sale has an individual purchase number - . **char sequence** with **exact 7(seven)** characters. The values are **unique in the database**. It **cannot be nullable**.
* **sale date** – indicates the date and time the sale happened. It cannot be nullable.
* Constraint: The **sales** table has a relation with the **devices** table. It can be nullable.
* **Constraint**: The **sales** table has a relation with the **sellers** table. It can be nullable.

### Device

* id - accepts **integer** values, a **primary identification field, an auto incremented field**.
* brand - accepts **char** **sequence** (between **2** to **20** both inclusive). It **cannot be nullable**.
* device type - categorization of the devices. String enumeration, one of the following – **SMART\_PHONE, TABLET, SMART\_WATCH, LAPTOP**. It can be nullable.
* model - accepts **char sequence** (between **1** to **20** both inclusive). The values are **unique in the database**. It **cannot be nullable**.
* price - accepts positive **double** values. It can be nullable.
* storage - accepts positive number values. It can be nullable.
* Constraint: The **devices** table has a relation with **sales** table. It can 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 **Device** may appear in only one **Sale**, but one **Sale** may have many **Devices**.

One **Seller** can have many **Sales**, but one **Sale** has only one **Seller**.

#### 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 {seller/sale/device}"**

**When the import is finished:**

**"Successfully imported {seller/sale/device} {sellerFirstName sellerLastlName/with number {saleNumber}/of type {deviceType} with brand {deviceBrand}"**

**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.

#### Sellers (sellers.json)

##### Constraint

**If a seller with the same last name already exists in the DB or the first/last name does not meet size constraints return "Invalid seller".**

**When the import is finished:**

**"Successfully imported seller {sellerFirstName} {sellerLastName}"**

|  |
| --- |
| **Sellers (sellers.json)** |
| [  {  "firstName": "John",  "lastName": "Harrison",  "personalNumber": "123123" }, {  "firstName": "Alice",  "lastName": "Smith",  "personalNumber": "456456" },  {  "firstName": "John",  "lastName": "Smith",  "personalNumber": "456456" }, {  "firstName": "Michael",  "lastName": "Johnson",  "personalNumber": "789789" }, {  "firstName": "Emily",  "lastName": "Williams",  "personalNumber": "101010" },  {  "firstName": "Daniel",  "lastName": "Brown",  "personalNumber": "111222" },  ... |
| Successfully imported seller John Harrison  Successfully imported seller Alice Smith  Invalid seller  Successfully imported seller Michael Johnson  Successfully imported seller Emily Williams  Successfully imported seller Daniel Brown… |

#### Sales (sales.json)

##### Constraint

* **If a sale with the same number already exists in the DB or the size of the number is not exactly 7(seven) characters return "Invalid sale".**
* **When the import is finished:**

**"Successfully imported sale with number {saleNumber}"**

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

|  |
| --- |
| **Sales (sales.json)** |
| [  {  "discounted": true,  "number": "1000123",  "saleDate": "2022-02-02 12:43:00",  "seller": 1 }, {  "discounted": true,  "number": "1000123",  "saleDate": "2023-12-02 12:10:00",  "seller": 1 }, {  "discounted": true,  "number": "1000124",  "saleDate": "2023-12-02 12:10:00",  "seller": 1 }, {  "discounted": true,  "number": "3320987",  "saleDate": "2022-09-15 08:23:00",  "seller": 12 }, {  "discounted": false,  "number": "8756321",  "saleDate": "2021-11-30 17:50:00",  "seller": 5 }, {  "discounted": false,  "number": "8756",  "saleDate": "2021-11-30 17:50:00",  "seller": 5 },  ... |
| Successfully imported sale with number 1000123  Invalid sale  Successfully imported sale with number 1000124  Successfully imported sale with number 3320987  Successfully imported sale with number 8756321  Invalid sale… |

### XML Import

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

#### Devices (devices.xml)

##### Constraint

* **If a device with the same brand and model already exists in the DB return "Invalid device".**
* **If a device appears in a sale that doesn't exist in the DB return "Invalid device".**
* **When the import is finished:**

**"Successfully imported device of type {deviceType} with brand {deviceBrand}"**

|  |
| --- |
| **Devices (devices.xml)** |
| *<?*xml version='1.0' encoding='UTF-8'*?>*  <devices>  <device>  <brand>HTC</brand>  <device\_type>SMART\_PHONE</device\_type>  <model>Ultra23+</model>  <price>999.00</price>  <storage>128</storage>  <sale\_id>1</sale\_id>  </device>  <device>  <brand>Samsung</brand>  <device\_type>SMART\_PHONE</device\_type>  <model>S23</model>  <price>1899.00</price>  <storage>256</storage>  <sale\_id>2</sale\_id>  </device>  <brand>Samsung</brand>  <device\_type>SMART\_PHONE</device\_type>  <model>S23</model>  <price>899.99</price>  <storage>256</storage>  <sale\_id>2</sale\_id>  </device>  <device>  <brand>iPhone</brand>  <device\_type>SMART\_PHONE</device\_type>  <model>15</model>  <price>999.00</price>  <storage>128</storage>  <sale\_id>1</sale\_id>  </device>  <device> |
| Successfully imported device of type SMART\_PHONE with brand HTC  Successfully imported device of type SMART\_PHONE with brand Samsung  Invalid device  Successfully imported device of type SMART\_PHONE with brand iPhone  … |

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 devices of type SMART\_PHONE with a price less than 1000 and storage equal to or more than 128 from the Database

* Extract from the database, the **device brand, model, storage and price**.
* **Filter only smartphones whose price is less than 1000 and have storage equal to or greater than 128. Order the results ascending by device brand.** *Keep in mind that in many systems, by default,* ***uppercase*** *letters are considered "less than"* ***lowercase*** *letters when sorted alphabetically because they come first in the ASCII and Unicode tables. For example, 'S' (ASCII 83) is less than 'i' (ASCII 105). Think about how to do proper sorting.*
* You have to round the value of the **price** to the **second** decimal digit.
* Return the information in this format:

**"Device brand: {brand}**

**\*Model: {model}**

**\*\*Storage: {storage}**

**\*\*\*Price: {price}**

**. . ."**

A screenshot of a phone

Description automatically generated