**Project developers:** Miguel Gonzalez, Yeison Rodriguez, Esteban Gaviria.

**Requirements Analysis Table.**

| **Client** | Mercado Libre |
| --- | --- |
| **User** | Merchant |
| **Functional requirements** | **FR1 -** Register product  **FR2 -** Register order  **FR3 -** Increase quantity of registered product  **FR4 -** Search products  **FR5 -** Search orders  **FR6 -** Search by range  **FR7 -** Search by interval  **FR8 -** Choose the order of the data displayed in the search result.  **FR9 -** Data serialization.  **FR10 -** Load information. |
| **Context of the problem** | The company MercadoLibre has commissioned the development of an application that allows the online sale of its virtual store. The task is to create a program that facilitates the introduction of inventory data and allows the search and withdrawal of products. In addition, the program must have the functionality to record orders placed by customers on the store's platform. |
| **Non-functional requirements** | **NFR1 -** The program must handle exceptions to avoid unexpected states during product entry and search. |
| **Product requirements** | **PR1 - TDD:** The project must be developed using TDD (Test-Driven Development).  **PR2 - Commits:** Throughout the development there must be at least 10 commits distributed equally-temporally. In each of the 10 commits, report 3 simple quality indicators in the repository README. The idea is that the evolution of the indicators can be tracked. |

**Functional Requirements Analysis Tables.**

| **Name or identifier** | **FR1 - Register product** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the registration of a product based on its name, description, price, quantity available, category and number of times purchased. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| productName | String | The entered string must be different of null or empty. |
| description | String | The entered string must be different of null or empty. |
| price | double | The entered number must be a positive number. |
| quantity | int | The entered number must be greater than 0. |
| category | int | The entered number must be greater than 0. |
| **General activities necessary to obtain the results** | 1. Read the product name. 2. Check that the entered string is valid. 3. Read the description. 4. Check that the entered string is valid. 5. Read the price. 6. Check that the entered number is valid. 7. Read the quantity available. 8. Check that the entered number is valid. 9. Read the category chosen option. 10. Check that the chosen option is valid. 11. Register product. | | |
| **Result or post-condition** | The new product is registered and added to the database. The system generates a message notifying the success of the operation. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| message | String |  |

| **Name or identifier** | **FR2 - Register order** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the registration of an order based on the buyer's name, list of products, total price and date of purchase. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| buyerName | String | The entered string must be different of null or empty. |
| products | ArrayList<String> | The entered list of strings must be different of null. |
| productQuantity | int | The entered number must be greater than 0. |
|  | priceForm | int | Must be 1 or 2. 1 to enter the total price of the order to be calculated by the user, and 2 to have the price calculated by the system. |
|  | totalPrice | double | The user should have selected the option to enter the price on his own. |
| **General activities necessary to obtain the results** | 1. Read the buyer's name. 2. Check that the entered string is valid. 3. Read the product list. 4. Check that the products entered are registered. 5. Read the solicited products' quantity. 6. Checks that there is sufficient stock to meet demand. 7. Read the price form 8. Check that the priceForm integer is valid. 9. If the user selects option 1 for priceForm, read the total price. 10. If the user selects option 2 for priceForm, the total price is calculated from the products and their selected quantities. 11. Register order. | | |
| **Result or post-condition** | The order is registered in the system database and a success message is generated. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| message | String |  |

| **Name or identifier** | **FR3 - Increase quantity of registered product** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the user to increase the quantity of a product already registered. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| productName | String | The entered string must be different of null or empty. |
| increaseAmount | int | The entered number must be greater than 0. |
| **General activities necessary to obtain the results** | 1. Read the product name. 2. Check that the product is registered. 3. Read the increased amount. 4. Check that the entered number is valid. 5. Search the product 6. Updates the corresponding product information. | | |
| **Result or post-condition** | The system saves the product change and a success message is generated. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| message | String |  |

| **Name or identifier** | **FR4 - Search product** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the search of a product based on its name, price, category or number of times purchased. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| searchVariable | String | The entered string must be different from null or empty. |
|  | searchValue | double | The entered number must be greater than 0. The selected variable search is of ***int*** or ***double*** type. |
|  | searchValue | String. | The entered string must be different from null or empty.  The selected variable search is of ***String*** type. |
| **General activities necessary to obtain the results** | 1. Read the variable in which the search will be based on. 2. Check the entered value. 3. Read the value to search (its type will depend on the variable chosen). 4. Check the entered value. 5. Query the first coincidence that matches. 6. Display the search result. | | |
| **Result or post-condition** | The system displays a message with the search result. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| result | String |  |

| **Name or identifier** | **FR5 - Search order** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the search of an order based on the buyer's name, total price or date of purchase. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| searchVariable | String | The entered string must be different from null or empty. |
|  | searchValue | int | The entered number must be greater than 0. The selected variable is int type |
|  | searchValue | String | The entered string must be different from null or empty. The selected variable is string type. |
| **General activities necessary to obtain the results** | 1. Read the variable in which the search will be based on. 2. Check the entered value. 3. Read the value to search (its type will depend on the variable chosen). 4. Check the entered value. 5. Query the first coincidence that matches. 6. Display the search result. | | |
| **Result or post-condition** | The system displays a message with the search result. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| result | String |  |

| **Name or identifier** | **FR6 - Search by range** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the user to search products by range for the numerical attributes. The user must define the minimum and maximum value to search products in that range. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| searchVariable | String | The entered string must be different from null or empty. |
| minValue | int | The entered number must be greater than 0. |
|  | maxValue | int | The entered number must be greater than 0. |
| **General activities necessary to obtain the results** | 1. Read the variable in wich the search will be based on 2. Validate the entered minimum value. 3. Read the minimum value. 4. Validate the entered minimum value. 5. Read the maximum value. 6. Validate the entered maximum value. 7. Query the products that fall within the specified range 8. Choose the order in which the results will be displayed (see FR8). 9. Display the ordered search results | | |
| **Result or post-condition** | The system displays a message with the search results. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| result | String |  |

| **Name or identifier** | **FR7 - Search by interval** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the user to search products by interval for String attributes. The user must define a start letter or prefix and an end letter or suffix to search for products in that alphabetical interval. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| searchVariable | int | The entered number must be greater than 0. |
| startPrefix | String | The entered string must be different from null or empty. |
| finalPrefix | String | The entered string must be different from null or empty. |
| **General activities necessary to obtain the results** | 1. Read de sort variable 2. Validate the entered sort variable value. 3. Read the search startprefix 4. Validate the search prefix entered. 5. Read the search finalPrefix 6. Validate the entered search finalPrefix 7. Query the products that fall within the specified range. 8. Choose the order in which the results will be displayed (see FR8). 9. Display the ordered search results. | | |
| **Result or post-condition** | The system displays a message with the search results. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| results | String |  |

| **Name or identifier** | **FR8 - Choose the order of the data displayed in the search result.** | | |
| --- | --- | --- | --- |
| **Summary** | The system must allow the user to choose how to display the search results (ascending or descending) as well as the variable order. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| senseOrdering | int | The entered number must be greater than 0. |
| sortVariable | String | The entered string must be different from null or empty. |
| **General activities necessary to obtain the results** | 1. Read the sense ordering 2. Validate the selected sense ordering 3. Read the sort variable. 4. Validate the selected sort variable 5. Obtain the results of the query 6. Sort the obtained data | | |
| **Result or post-condition** | The results obtained in the search are sorted as follows. | | |

| **Name or identifier** | **FR9- Data serialization.** | | |
| --- | --- | --- | --- |
| **Summary** | The system must be able to save the data information stored in the program of the products and orders information to be loaded in future executions. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| none | none | none |
| **General activities necessary to obtain the results** | 1. Package the information and the changes made in the program (when the user chooses the option to exit the program). 2. Save the packaged information of products in a JSON file format. 3. Save the packaged information of orders in a JSON file format. | | |
| **Result or post-condition** | The products and orders information is stored in two JSON files. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| none | none | none |

| **Name or identifier** | **FR10- Load information.** | | |
| --- | --- | --- | --- |
| **Summary** | The system must be able to load the products and orders information from JSON’s format files. | | |
| **Inputs** | **Input name** | **Datatype** | **Selection or**  **repetition condition** |
| none | none | none |
| **General activities necessary to obtain the results** | 1. Create the folder in which the program information will be stored (only if this is the first time the program is run). 2. Load the information of products stored in the corresponding JSON file 3. Load the information of orders stored in the corresponding JSON file | | |
| **Result or post-condition** | The program information is loaded in the program. | | |
| **Outputs** | **Output name** | **Datatype** | **Selection or repetition condition** |
| none | none | none |

**List of tests by functionality.**

**Scenario Configuration**

| **Name** | **Class** | **Scenery** |
| --- | --- | --- |
| setupStage1 | ControllerTest | An empty object of the Store class. |
| setupStage2 | ControllerTest | The objects of the Searcher class are initialized. Also, an ArrayList called ‘productsList’ with three objects of the Product class is created: {[  "name": "Miguel in Wonderland",  "description": "An unexpected adventure awaits Miguel in wonderland. Join him on his journey.",  "price": 100000,  "quantity": 7,  "category": "BOOKS",  "timesPurchased": 0]  "name": "Cboc Two",  "description": "An incredible video game console.",  "price": 2000000,  "quantity": 9,  "category": "ELECTRONIC",  "timesPurchased": 0[]  ["name": "HD laptop",  "description": "Intel Core, 2 Ram, 500GB HDD",  "price": 1200000,  "quantity": 7,  "category": "ELECTRONIC",  "timesPurchased": 0]  } |
| setupStage1 | StoreTest | An Store object called “store” is initializated. Also, an arrayList with 5 objects of class Product is initialized: productsList = [( “Gamer headphones”, ...), (“Olympic bar”, ...), (“Diapers”, ...), (“Barbie”, ...), (“Lipstick”, …)]. It is assumed that the objects contained in this arrayList were given as a result of a search. |
| setupStage1 | SearcherTest | The objects of the Searcher class are initialized. Also, an ArrayList called ‘productsList’ with three objects of the Product class is created: {  ["name": "Cboc Two",  "description": "An incredible video game console.",  "price": 120000.0,  "quantity": 2,  "category": "ELECTRONIC",  "timesPurchased": 6  ] [  "name": "Ball",  "description": "Soccer ball",  "price": 200000.0,  "quantity": 7,  "category": "SPORTS",  "timesPurchased": 4  ] [  "name": "Miguel in Wonderland",  "description": "An unexpected adventure awaits Miguel in wonderland. Join him on his journey.",  "price": 300000.0,  "quantity": 2,  "category": "BOOKS",  "timesPurchased": 5  ] [  "name": "apple watch",  "description": "Smartwatch with various features.",  "price": 500000.0,  "quantity": 3,  "category": "ELECTRONIC",  "timesPurchased": 0  ] [  "name": "play state 3",  "description": "Video game console with advanced graphics.",  "price": 2500000.0,  "quantity": 1,  "category": "ELECTRONIC",  "timesPurchased": 0  ] [  "name": "wood chair",  "description": "Comfortable chair made of solid wood.",  "price": 100000,  "quantity": 8,  "category": "BEAUTY",  "timesPurchased": 0  ] [  "name": "wood table",  "description": "Sturdy table made of high-quality wood.",  "price": 250000,  "quantity": 3,  "category": "BEAUTY",  "timesPurchased": 0  ] [  "name": "HD laptop",  "description": "Intel Core, 2 Ram, 500GB HDD",  "price": 1200000,  "quantity": 2,  "category": "ELECTRONIC",  "timesPurchased": 0  } |
| setupStage2 | SearcherTest | The objects of the Searcher class are initialized. Also, an ArrayList called ‘productsList’ is inicializated |
| setupStage3 | SearcherTest | Some products of the Searcher class are initialized. After, ordered ArrayList called ‘ordersList’ with three objects of the Order class is created :  { [ buyerName = “Camilo“,  productsList = [ ( “wood chair”...), ( “wood table”... ) ],  totalPrice = 1’550.000,  datePurchase = *currentDate* ],  [ buyerName = “Esteban”,  productList = [(“Play state 3”…)],  totalPrice = 2500000,  datePurchase = *currentDate* ],  [ buyerName = “Sara”,  productList = [ (“HP laptop”, …), (“ apple watch “, …) ],  totalPrice = 3’900.000 ,  datePurchase = *currentDate* ] }  ***Clarification:*** *Differences between order registration dates are seconds.* |
| setupStage1 | WriterTest | An object of Gson class and other of Writer class are initialized. In addition, a list of products and a list of orders is also initialized. The product list contains the following three objects of class Product: {  ["name": "Cboc Two",  "description": "An incredible video game console.",  "price": 120000.0,  "quantity": 2,  "category": "ELECTRONIC",  "timesPurchased": 6  ] [  "name": "Ball",  "description": "Soccer ball",  "price": 200000.0,  "quantity": 7,  "category": "SPORTS",  "timesPurchased": 4  ] [  "name": "Miguel in Wonderland",  "description": "An unexpected adventure awaits Miguel in wonderland. Join him on his journey.",  "price": 300000.0,  "quantity": 2,  "category": "BOOKS",  "timesPurchased": 5  ] } |
| setupStage1 | ReaderTest | An object of class Reader is initialized. In addition, a list of products and a list of orders is also initialized. |

**Test Case Design**

**FR1 - Register product**

| **Test objective:** Verify that the method addProduct works correctly and add the indicated product. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Controller | addProduct | setupStage1 | name = ”Miguel in Wonderland”, description = ”An unexpected adventure awaits Miguel in wonderland. Join him on his journey.”, price = 100000, quantity = 7, category = 1 (BOOK) | The product is registered in the system with the given information |
| Controller | addProduct | setupStage1 | name: "HD laptop",  description: "Intel Core, 2 Ram, 500GB HDD",  price: 1200000,  quantity: 7,  category: 2 (ELECTRONIC),  timesPurchased: 0 | The product is registered in the system with the given information |

| **Test objective:** Verify that the registerProduct method throws an exception when erroneous information is entered. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Controller | registerProduct | setupStage1 | name = ””, description = ”An unexpected adventure awaits Miguel in wonderland. Join him on his journey.”, price = 100000, quantity = 7, category = BOOK | An exception is thrown because the name of the product you are trying to add is empty. |
| Controller | registerProduct | setupStage1 | name = ””, description = ”An unexpected adventure awaits Miguel in wonderland. Join him on his journey.”, price = 100000, quantity = -3, category = BOOK | An exception is thrown because the amount of product you are trying to add is a negative number. |
| Controller | registerProduct | setupStage1 | name = ””, description = ”An unexpected adventure awaits Miguel in wonderland. Join him on his journey.”, price = 100000, quantity = 7, category = 10 | An exception is expected because the category don't exist. |

**FR2 - Register order**

| **Test objective:** Verifies that the method addOrder works correctly, adding an order. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Controller | addOrder | setupStage2 | buyerName = “Miguel”  productName = “Miguel in Wonderland”  productQuan = 3  productName = “Cboc Two”  productQuan = 5 | The order is saved with the products and quantities indicated, the success message is displayed. Stocks of products “Miguel in wonderland” and “Cboc Two” are updated to 4. |
| Controller | addOrder | setupStage2 | buyerName = “Esteban”  productName = “Miguel in wonderland”  productQuan = 3  productName = “Cboc Two”  productQuan = 5  productName = “HD laptop”  productQuan = 2 | The order is saved with the products and quantities indicated, the success message is displayed. Stocks of products “Miguel in wonderland” and “Cboc Two” are updated to 4. The stock of “HD laptop” is updated to 5. |

| **Test objective:** Verifies that the methos addOrder throws an exception in case of invalid information. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Controller | addOrder | setupStage2 | buyerName = “”  productName = “Miguel in wonderland”  productQuan = 3  productName = “Cboc Two”  productQuan = 5  productName = “HD laptop”  productQuan = 2 | An exception is thrown, because the buyer’s name is empty. |
| Controller | addOrder | setupStage2 | buyerName = “Miguel”  productName = “Miguel in Wonderland”  productQuan = 50  productName = “Cboc Two”  productQuan = 5 | An exception is thrown because the specified quantity of product “Miguel in wonderland” exceeds the stock of the product |
| Controller | addOrder | setupStage2 | none | An exception is expected as the product list is empty. |

**FR3 - Increase quantity of registered product**

| **Test objective:** Verify that the method increaseQuantity works correctly for a product. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Controller | increaseQuantity | setupStage2 | productName = “HP laptop” & increaseAmount = 3 | The quantity of the product in stock is updated to 10. |
| Controller | increaseQuantity | setupStage2 | productName = “HP laptop” & increaseAmount = 0 | The available quantity of the product remains at 7. |

| **Test objective:** Verify that the method increaseQuantity throws correctly an exception when invalid data is entered | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Controller | increaseQuantity | setupStage2 | productName = “HP laptop” & increaseAmount = -2 | The available quantity of the product remains at 7. Throws an exception because the increase amount is negative. |
| Controller | increaseQuantity | setupStage2 | productName = “” & increaseAmount = 5 | Throws an exception because the product name is empty. |

**FR4 - Search product**

| **Test objective:** Verify that the searchProduct method of the Search class works correctly, finding the first coincidence that matches the search value. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **input values** | **Result** |
| Search | searchProduct | setupStage1 | products = productsList & searchVariable = “name” & searchValue = “Harry Potter and the sorcerer's stone” | A null is expected. This indicates that the product wasn’t found. |
| Search | searchProduct | setUpStage1 | products = productsList & searchVariable = “price” & searchValue = 100000.0 | The product [ name = ”Miguel in Wonderland”, description = ”An unexpected adventure awaits Miguel in wonderland. Join him on his journey.”, price = 300000.0  , quantityAvailable = 2, category = BOOKS, timesPurchased = 5] is expected. |
| Search | searchProduct | setupStage1 | products = productsList & searchVariable = “Category” & searchValue = ELECTRONIC | The product [ name=”Cboc Two”, description = “An incredible video game console.” , price = 2000000, quantity = 2, category = ELECTRONIC ] is expected. |
| Search | searchProduct | setupStage2 | products = new ArrayList<>() & searchVariable = “name” & searchValue = “andres”. | Since the product list is empty, a null is expected to be returned. |

| **Test objective:** Verify that the searchProduct method of the Search class throws correctly an exception when the searchValue is invalid and when the list of products is null. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Search | searchProduct | setupStage1 | products = productsList & searchVariable = “pricess” & searchValue = “100” | An exception indicating that the searchVariable is not valid is thrown. |
| Search | searchProduct | setUpStage1 | products = productsList & searchVariable = “name” & searchValue = 100 | An exception indicating that the searchValue is not valid is thrown. |

**FR5 - registerProduct();registerProduct();registerProduct();registerProduct();registerProduct();registerProduct();**

| **Test objective:** Verify that the searchOrder method of the Search class works correctly, finding the first coincidence that matches the search value. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **input values** | **Result** |
| Search | searchOrder | setUpStage3 | orders = ordersList & searchVariable = “buyerName” & searchValue = “Andres” | Null, indicating that no orders were found is displayed. |
| Search | searchOrder | setUpStage3 | orders = orderList & searchVariable = “buyerName” & searchValue = “Esteban” | Object type order with ( [ buyerName = “Esteban“, ProductsList = [(“Play state 3”…)], totalPrice = 2500000, datePurchase = march 16 2023 ] ). |
| Search | searchOrder | setUpStage3 | orders = orderList & searchVariable = “totalPrice” & searchValue = 2500000 | Object type order with ( [ buyerName = “Esteban“, ProductsList = [(“Play state 3”…)], totalPrice = 2500000, datePurchase = march 16 2023 ] ). |
| Search | searchOrder | setUpStage3 | orders = orderList & searchVariable = “date” & searchValue = \_*currentDate*\_ | The object "order2" is expected to be returned. |
| Search | searchOrder | setUpStage2 | orders = null & searchVariable = “totalPrice” & searchValue = 100000 | Null, indicating that no orders were found because the order list is empty. |

| **Test objective:** Verify that the searchOrder method of the Search class throws correctly an exception when the searchValue is invalid and when the list of orders is null. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Search | searchOrder | setUpStage3 | orders = OrdersList & searchVariable = “buyer’sName” & searchValue = “Sara” | An exception indicating that the searchVariable is not valid is thrown. |
| Search | searchOrder | setUpStage2 | orders = OrdersList & searchVariable = “buyerName” & searchValue = 10 | An exception indicating that the searchValue is not valid is thrown. |

**FR6 - Search by range**

| **Test objective:** Verify that the searchByRange method works correctly, returning the matches found with the search variable, min and max value entered by the user. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Search | searchByrange | setupStage1 | products = productsList &  searchVariable = “price”  min = 0 &  max = 2000. | An empty list is returned. |
| Search | searchByRange | setupStage1 | products = productsList &  searchVariable = “price”  min = 0 &  max = 5000000. | A list with all the products added to the productsList is returned. |
| Search | searchByRange | setupStage1 | products = productsList &  searchVariable = “price”  min = 200000.0&  max = 300000.0 | A list with three products is returned. It contains the products with name “Ball”, “wood table” and "Miguel in Wonderland". |

| **Test objective:** Verify that the searchByRange method throws an exception when the entered values are not valid. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Search | searchByRange | setupStage1 | products = null & searchValue = 4(times purchased) & min = 0 & max = 10. | An exception indicating that the list is null is thrown. |
| Search | searchByRange | setupStage1 | products = productsList. & searchValue = 2(price) & min = 10& max = 0. | An exception indicating that the min value is greater than the max value is thrown. |

**FR7 - Search by interval**

| **Test objective:** Verify that the searchByInterval method works correctly, returning the matches found with the ordering variable, prefix and suffix entered by the user. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Seacrh | searchByInterval | setupStage1 | products = productsList,  searchVariable = “name”  startPrefix = “Aa”;  finalPrefix = “Bb”; | Return an empty list of matches |
| Seacrh | searchByInterval | setupStage1 | products = productsList,  searchVariable = “name”  prefix = “C”;  suffix = “H”; | A list with one product is returned. It only contains the “Cboc two” product. |
| Seacrh | searchByInterval | setupStage1 | products = productsList,  searchVariable = “name”  prefix = “C”;  suffix = “P”; | A list with three products is returned. It contains the products with name “Cboc two”, “HD laptop” and "Miguel in Wonderland". |

| **Test objective:** Check that the searchByInterval method throws an exception when the user enters an endPrefix that is lexicographically less than the startPrefix. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Seacrh | searchByInterval | setupStage1 | products = productsList,  searchVariable = “name”  prefix = “Zz”;  suffix = “Aa”; | Throws an exception saying that the startPrefix cannot be greater than the finalPrefix |
| Seacrh | searchByInterval | setupStage1 | products = productsList,  searchVariable = “totalPrice”  prefix = “Bb”;  suffix = “Aa”; | Throws an exception saying that the startPrefix cannot be greater than the finalPrefix |

**FR8 - Choose the order of the data displayed in the search result.**

| **Test objective:** Verify that the sortSearchResults method of the Store class works correctly, sorting the search results according to the ascendant sort direction and the sort variable selected by the user. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Search | sortSearchResult | setupStage1 | products = productsList,  senseOrdering = 1(ascendant),  sortVariable = “name” | The list of products sorted in ascending order by name is displayed. |
| Search | sortSearchResult | setupStage1 | products = productsList,  senseOrdering = 1(ascendant),  sortVariable = “price” | The list of products sorted in ascending order by price is displayed. |

| **Test objective:** Verify that the sortSearchResult method of the Search class works correctly, sorting the search results according to the descendant sort direction and the sort variable selected by the user. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **Input values** | **Result** |
| Search | sortSearchResult | setupStage1 | products = productsList,  senseOrdering = 2(descendant),  sortVariable = “name” | A message with the list of products sorted in descending order by name is displayed. |
| Search | sortSearchResult | setupStage1 | products = productsList,  senseOrdering = 2(descendant),  sortVariable = “price” | A message with the list of products sorted in descending order by price is displayed. |

**FR9- Data serialization.**

| **Test objective:** Verify that the save's methods of the Writer class correctly saves the information of the products and orders in JSON’s format files. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **input values** | **Result** |
| Writer | saveProducts && saveOrders | setupStage1 | The relative path where the Json test files are stored, a product and an order arraylist | Two files must be created in JSON format, one to store products and one for orders. The product file must store all 3 products, and the order file must not have any orders stored. |

| **Test objective:** Verify that the saveInformation method of the Store class does not damage previously created files when nothing is modified. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **input values** | **Result** |
| Writer | saveProducts && saveOrders | setupStage1. | The relative path where the Json test files are stored, a product and an order arraylist | The previously created file must remain the same as it was before after executing the method. An order is also added to the Json order file |

**FR10- Load information.**

| **Test objective:** Verify that the loadInformation method of the Store class creates correctly the project data folder. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **input values** | **Result** |
| Reader | readGson | setupStage1 | The relative path where the Json test files are stored, a product and an order arraylist | The Reader class must not have any objects created, since there is no JSON file, but the testData folder must have been created. |

| **Test objective:** Verify that the readGson method of the Reader class correctly reads a file in JSON format. | | | | |
| --- | --- | --- | --- | --- |
| **Class** | **Method** | **Scenery** | **input values** | **Result** |
| Reader | readGson | setupStage1 | The relative path where the Json test files are stored, a product and an order arraylist | The arrayLists of products and orders stores the objects loaded from the JSON's files (Three of Product class and one of Order class) |