13-11-2021

ESINF Report

LAPR3 INTEGRATIVE PROJECT
SPRINT 1

Group 96:
Miguel Morais (1181032)
Rodrigo Rodrigues (1191008)
Tiago Ferreira (1200601)
Mariana Lages (1200902)
Eduardo Sousa (1200920)
Miguel Jordão (1201487)

Index

Class Diagram Explanation 2
US102 Complexity Analysis 5
US103 Complexity Analysis 5
US104 Complexity Analysis 6
US105 Complexity Analysis 7
US106 Complexity Analysis 8
US107 Complexity Analysis 8
Contributions 8

Class Diagram Explanation

Our team decided to build a program that uses UI classes to interact with the user and each User Story has its own UI. The program will direct the user to the corresponding UI, whenever the user wishes to use one of the program functionalities.

The Controller classes are responsible for sending and receiving information from other classes. Before they start working, these classes need to get the main information from the Company class, which is responsible for stocking all the Store classes. Each Store class stores a set of many objects of the same type.

In the Sprint 1, we only have one Store class, which is the ShipStore. This class stores all the objects that belong to the Ship class. These objects are stored and organized by an AVL Tree. The ShipStore class is one of the most important classes of our program because it is responsible for doing the required calculations and operations requested by the User Stories. Not only that but it can get access to any ship inputted by the user, either by its MMSI, IMO or Call Sign.

Our program can read files with a specific data format. To do that, the ImportShipsUI asks the user for the file path, which will be sent to the respective Controller that will ask the ShipImporter class to import the ships of that file. Then, it will send it to the CsvUtils, which will read all the data of that file. This class will start by analyzing if the path contains any readable file and, if it does, the class will read all the data and save it into our program.

The program has the functionality to search or generate the summary of a ship by its MMSI, IMO or Call Sign. The user inputs one of these object attributes and the UI will send it to the respective Controller, which will get the needed information from the Store and show it to the user.

It can also calculate the top-N ships with the most kilometers travelled and their average speed. The user inputs the number of ships he wishes to see, the ship's vessel type and the period of time (initial and final date) in the console. The inputted data will be sent to the TopNShipsController, which will ask the ShipStore to calculate and compare the kilometers of each ship, generating the Top-N ships requested by the user.

Our program can return ships with close departure/arrival coordinates (no more than 5 kilometers away) and with different travelled distances. When the user asks for this functionality, the UI will give a sign to the TopPairsController, which will get the ShipStore to return a list of pairs of ships for the user.

The Position class is stored in the PositionTree class, which is a Binary Search Tree that will organize and store the position. Each Ship class has a PositionTree to be able to store its positions. The program can show the user all the ship's positional messages. To do that, the user inputs the ship's MMSI in the console and that will be sent to the PositionalMessageController. The software will find the specific ship required by the user and, when it is found, the Ship Class will send to the UI all the positional messages of the requested ship.

Finally, our program can list all the ships sorted in descending order of travelled distance and ascending order of total number of movements. For that, the user only needs to select the respective option from

the menu. After that, the UI will send a sign to the ListShipController and, since the controller has an object of the ShipStore, it has access to all the stored ships and can easily sort them. After the sort, the controller calls the ShipMapper class, which will ask the ShipDTO to generate a ship list with certain ship attributes in it. The ShipMapper will save all the information and will send it back to the UI to print the ship list requested by the user.

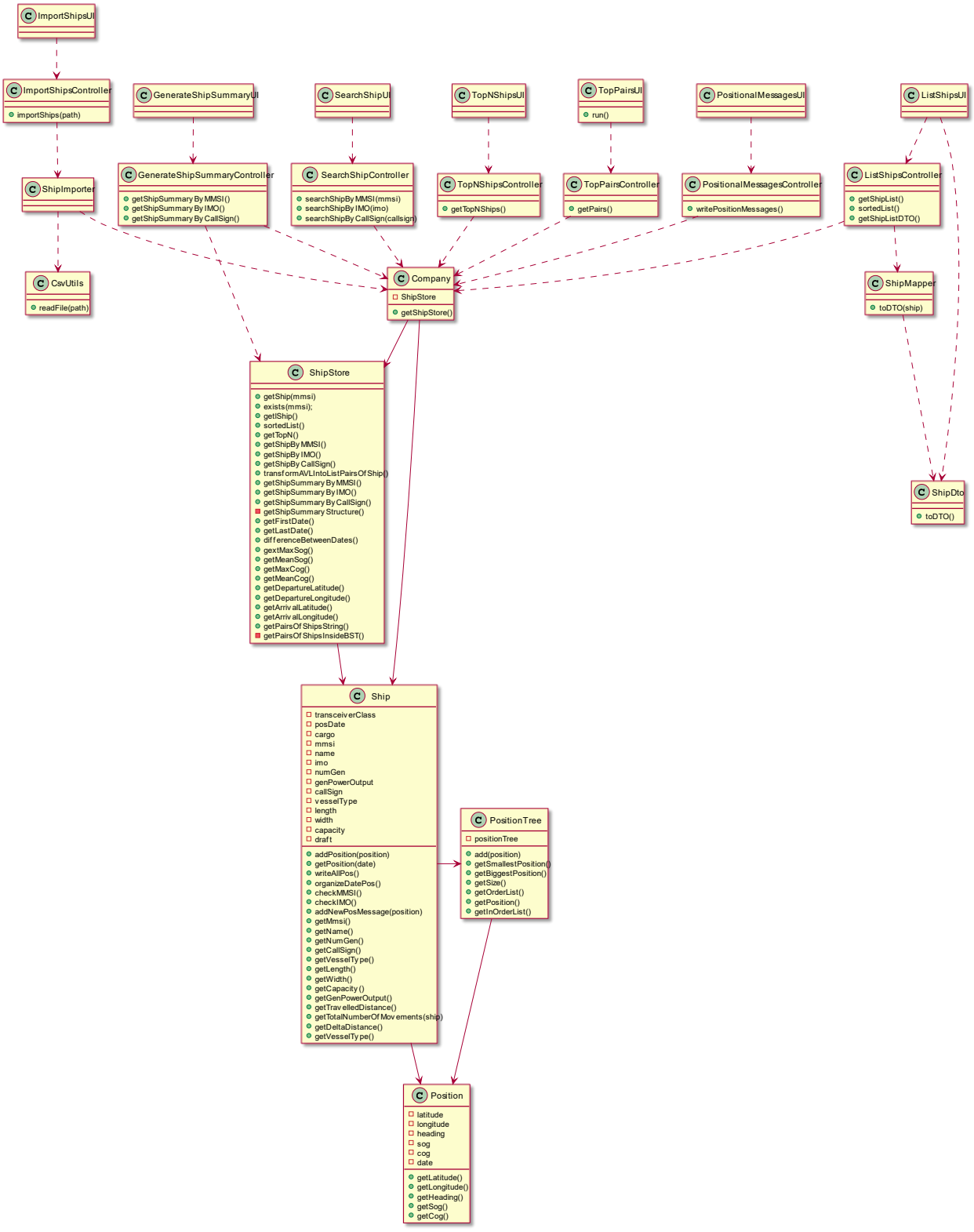

FIGURE 1 GLOBAL CLASS DIAGRAM

US102 Complexity Analysis

The methods implemented in this User Story (in the ShipStore class) were insertIntoMMSIAVl() , insertIntoIMOAVL() , insertIntoCallSignAVL() , getShipByMMSI(), getShipByIMOcode(), getShipByCallSign(). Since insertIntoMMSIAVl() , insertIntoIMOAVL() and insertIntoCallSignAVL() methods call the same methods of the AVL class we will refers to them as the same, complexity wise, the same thing happens with the getShipByMMSI(), getShipByIMOcode(), getShipByCallSign() methods.

To have efficient searching of ships using their MMSI, IMO code and call sign, we've created three different classes of ships which are subclasses of the class ship and we created three different AVL's to store the tree "types" of ships. Therefore, we have optimised search for all the attributes of the ship.

These methods have the following temporal complexity:

- insertInto…AVL(), since this is a non-deterministic method we have to do the following analysis:
  - Worst Case - when there are several elements in the AVL the method needs to traverse all the elements of a branch since the AVL is balance the complexity of this operation will be the height off the tree which is log(n) being n the number of the elements in the tree. $O(\log(n))$
  - Best Case – the best case is when the AVL is empty, once it is empty there will only be one attribution
- getShipBy…AVL(), since this is a non-deterministic method we have to do the following analysis:
  - Worst Case - in the worst-case scenario the element that we are looking for is in one of the leaves of the tree being that we need to traverse one branch in order to find that element leading to the complexity of this method being the height after three which is log(n) being n the number of elements in the tree.
  - Best Case – best case is when the root of the AVL is the element that we are looking for in this case we only have one operation so the complexity of the method will be O(1).

US103 Complexity Analysis

In order to complete this user story, we implemented the following methods: writeAllPos() and compareTo() from the Position class. To find the specific ship that the user wants, we reused a previous method: getShipByMmsi() and getInOrderList(). The writeAllPos() method writes all the positional messages of a certain ship, the method runs a period of time selected by the user, it uses the Calendar object to run the initial date to the final date, if it find a positional messages with that exact date time, it will add to a list which will store all the positional messages of that ship.

- WriteAllPos():
  - Worst case: O(n^2), writes all the positions of a certain ship.
  - Best case: O(1), writes all the positions of a certain ship.
- CompareTo()
  - Only Case: O(1), compares the date between two positions.

US104 Complexity Analysis

In order to complete this user story, we reused some methods previously implemented (specified in US102). But new methods were also introduced such as: transformAVLintoListMMSI(), transformAVLintoListIMO(), transformAVLintoListCallSign(), getShipSummaryByMMSI(int mmsi), getShipSummaryByIMO(String imo), getShipSummaryByCallSign(String callSign), getShipSummaryStructure(Ship s), getFirstDate(Ship s), getLastDate(Ship s), differenceBetweenDates(LocalDateTime first, LocalDateTime second), getMaxSOG(Ship s), getMeanSOG(Ship s), getMaxCOG(Ship s), getMeanCog(Ship s), getDepartureLongitude(Ship s), getDepartureLatitude(Ship s), getArrivalLongitude(Ship s), getArrivalLatitude(Ship s) and calculateTravelledDistanceOfAllShips().

These methods have the following temporal complexity:

- transformAVLintoListMMSI():
  - Only case – O(n), gets all the elements of the AVL ordered by MMSI (ascending) and adds them to an arrayList.
- transformAVLintoListIMO():
  - Only case – O(n), gets all the elements of the AVL ordered by IMO code (ascending) and adds them to an arrayList.
- transformAVLintoListCallSign():
  - Only case – O(n), gets all the elements of the AVL ordered by CallSign (ascending) and adds them to an arrayList.
- getShipSummaryByMMSI():
  - Worst case – O(log(n)), gets the summary of a ship by its MMSI.
  - Best case – O(1), gets the summary of a ship by its MMSI.
- getShipSummaryByIMO():
  - Worst case – O(log(n)), gets the summary of a ship by its IMO code.
  - Best case – O(1), gets the summary of a ship by its IMO code.
- getShipSummaryCallSign():
  - Worst case – log(n), gets the summary of a ship by its call sign.
  - Best case – O(1), gets the summary of a ship by its call sign.
- getShipSummaryStructure():
  - Worst case – O(log(n)), gets the ship's summary structured.
  - Best case – O(1), gets the ship's summary structured.
- getFirstDate(Ship s):
  - Worst case – O(log(n)), gets ship's first date.
- getLastDate(Ship s):
  - Worst case – O(log(n)), gets ship's last date.
- differenceBetweenDates(LocalDateTime first , LocalDateTime second):
  - Only Case – O (1), calculates the date between two dates.
- getMaxSOG(Ship s):

    - Only Case – O(n), gets the biggest SOG inside the AVL (first we must convert the AVL into an arrayList).
- getMeanSOG(Ship s):
    - Only case – O(n), gets the mean SOG inside the AVL (first we must convert the AVL into an arrayList).
- getMaxCOG(Ship s):
    - Only Case – O(n), gets the biggest SOG inside the AVL (first we must convert the AVL into an arrayList).
- getMeanCOG(Ship s):
    - Only case – O(n), gets the mean SOG inside the AVL (first we must convert the AVL into an arrayList).
- getShipByMMSIAVL():
    - Only case – O(1), gets the Ship by MMSI AVL
- getDepartureLatitude(Ship s):
    - Worst case – O(log(n)), gets the departure latitude from a specific ship from the Position AVL.
- getDepartureLongitude(Ship s):
    - Worst case – O(log(n)), gets the departure longitude from a specific ship from the Position AVL.
- getArrivalLatitude(Ship s):
    - Worst case – O(log(n)), gets the arrival latitude from a specific ship from the Position AVL.
- getArrivalLongitude(Ship s):
    - Worst case – O(log(n)), gets the arrival longitude from a specific ship from the Position AVL.
- calculateTravelledDistanceOfAllShips():
    - Worst case – $O(n^2 \log(n))$, calculates the current travelled distance, biggest position, smallest position and the Position AVL size of all ships inside the ship by MMSI AVL, ship by IMO code AVL and ship by call sign AVL.

US105 Complexity Analysis

In order to complete this user story, we used a method previously implemented, namely the transformAVLintoListMMSI() method, from US104. Furthermore, a new method was added to the ShipStore (sortedList) and the DTO classes.

This method has the following temporal complexity:

- sortedList():
    - Sorts the ship list by travelled distance (descending) and total number of movements (ascending) using TimSort algorithm, which is already implemented in Java. TimSort is implemented in the Arrays.sort() method. TimSort is an ordering algorithm that has a complexity, in the worst-case scenario, of $O(nlog(n)$ as stated in the documentation of Java. Therefore, the complexity of this method is also $O(nlog(n))$.

US106 Complexity Analysis

To complete this user story, we used:

- GetTopN():
    - Gets the top-N ships with the most kilometers travelled and their average speed (MeanSOG), the program creates a list ShipVessel that contains all the ships with the same vessel type, then the method calculates the distances by using the shared method traveledDistanceBaseDateTime of the ships and compares each one of them and return a list with the Top N ships with the most kilometers traveled, this method has a complexity of O(n) as best case and O(n^4) as worst case.

US107 Complexity Analysis

In order to complete this user story, we used some methods previously implemented and only had the need to add two methods to the ShipStore (getPairOfShipsInsideAVL and getPairsOfShipsString).

These methods have the following temporal complexity:

- getPairOfShipsInsideAVL():
    - This method travels the list of ships one time and, each time it travels again the same list, in order to grab 2 instance of ship inside these two for's, we call the methods getSmallPosition() and getBiggestPosition() that have complexity of O(log(n)), making the complexity of this method be O($n^2\ (\log n)^2$).
    - This method is non-deterministic. If the result of second is false, then the getBiggestPosition() method is never called, leading to complexity of the method being O($n^2\ (\log n)$). If the first is never true, then the complexity of the method becomes O(n^2).

- getPairsOfShipsString()
    - This method calls getPairOfShipsInsideAVL() in order to get the list of the pairs of ships then it traverse this list in order to add it to the string that will be returned as a result. The complexity of getPairOfShipsInsideAVL() is O($n^2\ (\log n)^2$), the for loop has complexity of O(n), then the complexity of this method is O($n^2\ (\log n)^2$).

Contributions

- US101
    - Código: Eduardo Sousa (1200920)
    - Teste: Eduardo Sousa (1200920)

- US102
    - Código: Rodrigo Rodrigues (1191008) and Eduardo Sousa (1200920)
    - Teste: Rodrigo Rodrigues (1191008) and Eduardo Sousa (1200920)
    - Análise de complexidade: Eduardo Sousa (1200920) and Miguel Jordão (1201487)

- US103
    - Código: Tiago Ferreira (1200601)
    - Teste: Tiago Ferreira (1200601)
    - Análise de complexidade: Tiago Ferreira (1200601)

- US104
    - Código: Miguel Jordão (1201487)
    - Teste: Miguel Jordão (1201487)
    - Análise de complexidade: Miguel Jordão (1201487)

- US105
    - Código: Mariana Lages (1200902)
    - Teste: Mariana Lages (1200902)
    - Análise de complexidade: Mariana Lages (1200902)

- US106
    - Código: Tiago Ferreira (1200601)
    - Teste: Mariana Lages (1200902)
    - Análise de complexidade: Tiago Ferreira (1200601)

- US107
    - Código: Eduardo Sousa (1200920) and Miguel Jordão (1201487)
    - Teste: Eduardo Sousa (1200920) and Miguel Jordão (1201487)
    - Análise de complexidade: Eduardo Sousa (1200920) and Miguel Jordão (1201487)