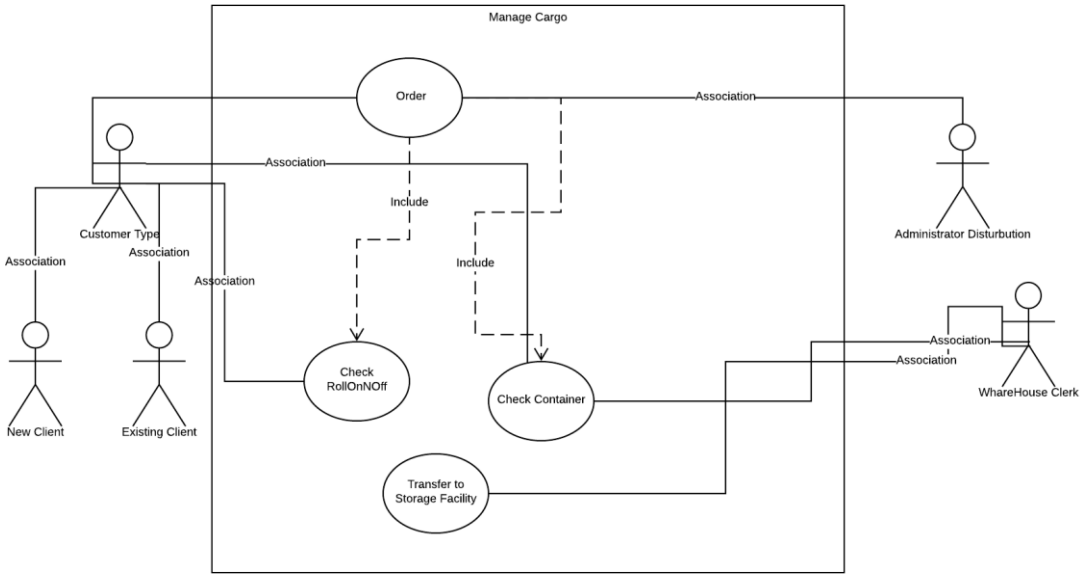
Use Case:



Implementation of the Terminal Operating System:

Manage General Cargo:

We decided to put Manage General cargo as the main class of this project, as the main class for this project the core managing operation for moving cargo from in and out of the company. All types are located within the class including containers, **RollOnAndOff** involving all types of cars from Tanks to Trucks etc., and delivery order having to record and keeping track of products to and from. Specifying the type of cargo by user such as **rollOnOff** or **Containers** this give individual ID and store them in the system called the collection class all of this process is for collecting valuable information. Categorising the type of cargo the user needs to specify the location to put in terms of storage capacity (storage facility).

For User can have the option to change the outcome when in terms of adding or editing a specific item by using a form, such as under the Main Menu form, having to adding an instance of a vehicle on **addRollOnOff** including its **brandtype** model Name “skyline”, its Year of Model “2000”, it weight limit “2000” its **heightLimit** “4”, Moreover car **numberID** would automatically be assigned when the XML file is loaded into the system.

Container:

On Task at hand packing cargo containers need to know how much quantity needed to make up a cargo, the count of products also the unpacking in terms of location wise.

We decided that a container class requires stuffing and unstuffing of products in container and used to transport them to safe locations which is one of its attributes and it is an inheritance to the superclass **Mange General cargo**.

Other attributes involve:

containerID - which specifies the identification of the container for easy sorting.

Destination - The area or location of the storage section area within the storage facility.

Type of Container - Including the quantity required to be packed in type size Forty ft. and Twenty ft.

The **create** method involves creating a **containiner ID** for every container involve and having that container get the ID which is the **getContainerID**, we set its properties to create its own features of how the object behaviours, this method involves **setPropsOnCreate**. The **setPropsUpdate** if there were any properties of container that needs to be changed.

The Total cost of container so far is the total cost of both of the specific type of container including **quantity Twenty Required** or **Forty**.

RollOnOff:

The **RollOnOff** class is designed to classify vehicles coming in and out of the port it is associated with the delivery order because it products that are coming in and going out are being delivered and order to or by customer, It has an attribute type listed below:

Obtaining the **weight** and **height limits** to allocate total amount of vehicles to be transported on ships or on trucks**.** Getting the number of total vehicle would maximize the capacity of its ship size or Truck.

Brand Type: including type of vehicle type e.g. skyline

Year of Model: The year the car was made in Integer form

However the method **create** is used to declare the **nextCarNumber** of the vehicle from the collection type **nextRollOffIDNum** in the collection Type class, in relation to **getnextCarID** method is used for the object to get that ID number in return, The method **getBrandName** is set for getting the brand Type.

SetsPropsCreate is used on how the class is displayed and behaviour, it includes the attributes that were mentioned above and are stored in the reference called **myRollOnandOff**.

setsUpdateProps is in a similar way but the user is able to make changes to the above information given of all the attributes shown within the method.

DeliveryOrder:

We included the Delivery Order as one of our class because it tracks cargo and other products to be delivered and is used to store valuable information for later use. The **date order placed** is used to track and record cargo products to be shipped, it has its own **company Name** attribute, **contract Phone** attribute, the **delivery Address** attribute, it has a distance given which provide an estimation of how far the distance to reach its destination,

the **invoice** is automatically provided and issued to the client for cargo activities not only its delivery fee but the Yard storage and the calculated Price method is used to be calculated. The **Create** method is used to obtain the **order Id** to be stored in the **myManageCargo** reference. **SetPropsOnCreate** are properties being set from attributes mentioned above and is used on how the information is entered. The **setPropsUpdate** is used in a similar way but the exiting information can be changed by the user when an object needs to be updated or changed, It is Associated with **RollOnandOff** because it manage the delivery phase and it is an inheritance toward the superclass **Manage Cargo**.

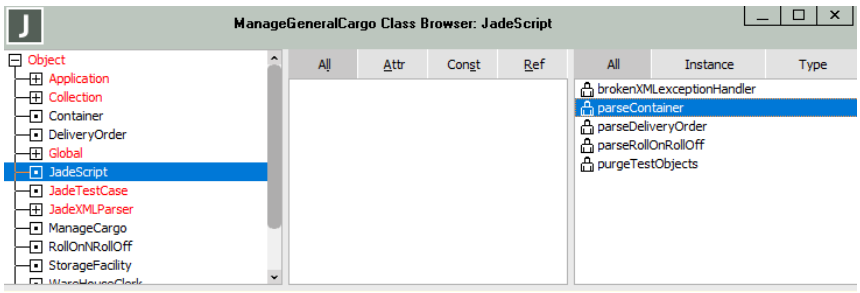
Storage Facility:

This is one of the task at hand for Managing Cargo, The storage area must be effectively monitored when it comes to storage capacity, Using some sort of handheld devices with integrated software giving users the opportunity to view the area capacity, the available cargo in storage, able to look up on **cargoID** in areas of where the products is stored, and providing a **phone** number of customers that own the product cargo that were shipped in and out of the port and importantly the **customer Name** attribute for the verified product stored. The method for **getCapacityLimit** is used for seeing if there is any space available for other cargo products to be imported. And the **getCustomerName** method is provided to obtain the name of the customer. It is an inheritance to the superclass of Manage Cargo and it has many storage area located differently in the warehouse

WareHouse Clerk:

The warehouse Clerk is used to register all different cargos coming in and out of the warehouse and updating information within the storage facility.

Exception Handling/Logging Functionality:



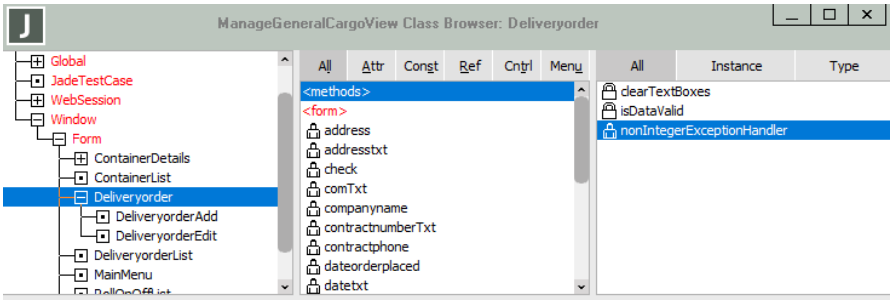
You can run the **parseContiner**, **parseDeliveryOrder** and **parseRollOnRollOff** method to import “container.xml”, “deliveryOrderList.xml” and “rollOnRollOf.xml” (both of them are correct xml file) in the **JadeScript**.

In the Container class, we have **stringTooLongExceptionHandlermethod** method to catch the error (error code 1035), which the length of the destination or location longer than 25 characters. This exception handler will provide the error to the user and ask user to reduce length of characters and be logged to a file called “ManageCargo_errors.log”. In this case, you can use the “ContainerStringTooLong.xml” file to test this exception handler and in the “ManageCargo_errors.log”, you can see the error item is “destination” and error type is

“Container” in the “ManageCargo_errors.log”. We also have this exception handler in **DeliveryOrder** and **RollOnNRollOff** classes, you can load “deliveryOderStringTooLong.xml” and “rollOnRollOfStringTooLong.xml” to test.

Every time when we import XML file, if the file is broken and we have a **brokenXMLexceptionHandler** method in the **JadeScript** to catch this error (error code 8901). This exception handler will tell the user to check the XML is complete or not and the error will be logged to the “ManageCargo_errors.log”. In this case, you can use the “rollOnRollOfStringTooLong.xml” file to test this exception handler and in the “ManageCargo-_errors.log” you can see the type is “XML parser error” and the extended text is “mismatched tag”.

Under the **ManageGeneralCargoView** schema, we create **nonIntegerExceptionHandler** in the class.



This exception handler will catch the error like the user enter the distance of delivery order is not an integer type (e.g. “abc”) and send this message to user and ask user to change to the right one. In the subclass of **Deliveryorder** (**DeliveryorderAdd** and **DeliveryorderEdit**), we have **intergerAndCharacterException** and **mixStringAndNumberExceptionHandler** to catch the error like the user enter the distance of delivery order is mixed with integer and string (e.g.“123abc”) and be stored in the "DeliveryOrder_Error.log".

You can open the “Exception Log file” file to see the example of "DeliveryOrder_Error.log" and “ManageCargo_errors.log”.

Unit tests Implementation and Functionality:

In this Manage General Cargo project, three classes were implemented in test cases are (Container class, Delivery Order class and **RollOnNRollOff** class). Those three classes are the fundamental of the system. Because, three of the classes were used to create GUI. There are essential to ensure all of functionality of the class is working. The Unit test covers all of the attribute and behaviours of each class with variation of test cases.

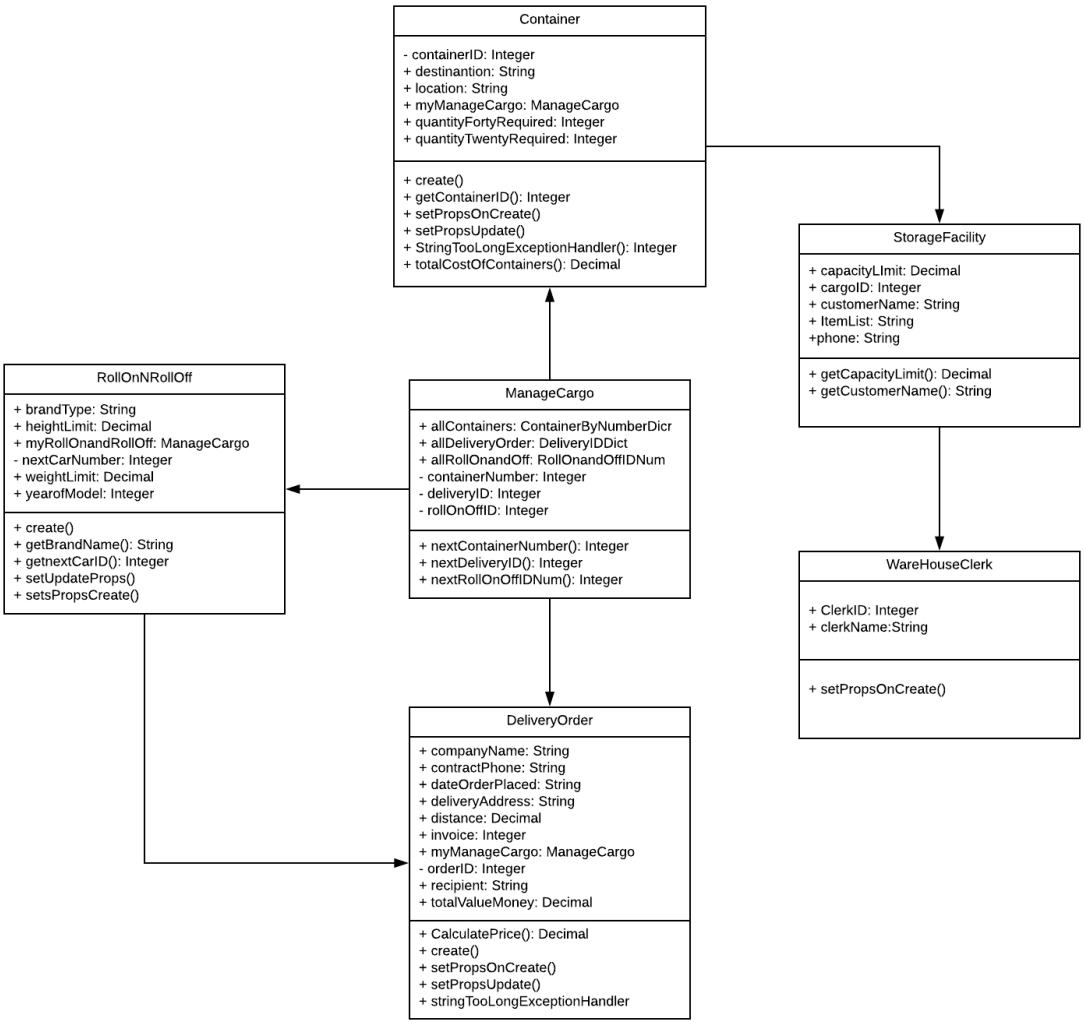
First before any unit test there are necessities to have a function that delete all of the instances in the program (‘initialize’). It will make the test obvious and it will not be affect to anything related to the system.

In **TestContainer** Class, there are three test cases which are implemented in corresponding to methods of the class. First test case, the unit test is tested its functionality of the constructor of the class (create method). In this test case, the test will create ten instances of the Container class. After Instance is created, the class automatically call the constructor that would setup the ID for the instance. By creating ten instances, the ID expect will be incremented from one to ten. This test also test "**getContainerID**" method of the class. The second test of the class that is "**testSetPropsOnCreate**". This test is used for testing the getter and setter of the class. Because the "**testSetPropsOnCreate** "method in class is using for setting all of the attributes. Therefore, This Unit test will create five instances and setup appropriate value for the method. After that, the test will test all attributes that are assigned in the method. And the last method, that we are testing is the total cost of the containers which 2600 for each 20 ft. containers and 5000 for each 40 ft. containers. The test creates number of instances then calculate all the total cost that customer have to pay.

In "**TestDeliveryOrder**" class, there are also three test cases provided to test the constructors, "**SetPropsOnCreate**" and calculate the price. In testing constructor the way to test is quite similar to the pervious test, it also creates a number of instances and test the ID of each instance when they create to define the function and see if it is working or not. Then in **calculate price**, the price is depended on the distance. And the last test case is testing setup properties of the method. It is using to test the method whether it works by setting up the new instance with appropriate properties and testing all the attribute of the class("**companyName**", "**contractPhone**", "**deliveryAddress**", "**dateOrderPlaced**", "recipient", "invoice", "distance", "**totalValueMoney**"). By testing to ensure the class works probably.

In final test case, it also tests all the methods in class "**RollOnNRollOff**". Base on all method in **RollOnNRollOff**, This Unit test also have three tests case. Which also has test on constructor, setup properties and the brand name. Using the same kind of testing as above, but it has number of test cases that are appropriate with the behaviours of the function.

UML Diagram:


Container
- containerID: Integer
+ destinantion: String
+ location: String
+ myManageCargo: ManageCargo
+ quantityFortyRequired: Integer
+ quantityTwentyRequired: Integer
+ create()
+ getContainerID(): Integer
+ setPropsOnCreate()
+ setPropsUpdate()
+ StringTooLongExceptionHandler(): Integer
+ totalCostOfContainers(): Decimal
StorageFacility
+ capacityLImit: Decimal
+ cargoID: Integer
+ customerName: String
+ ItemList: String
+phone: String
+ getCapacityLimit(): Decimal
+ getCustomerName(): String
RollOnNRollOff
+ brandType: String
+ heightLimit: Decimal
+ myRollOnandRollOff: ManageCargo
- nextCarNumber: Integer
+ weightLimit: Decimal
+ yearofModel: Integer
+ create()
+ getBrandName(): String
+ getnextCarID(): Integer
+ setUpdateProps()
+ setsPropsCreate()
ManageCargo
+ allContainers: ContainerByNumberDicr
+ allDeliveryOrder: DeliveryIDDict
+ allRollOnandOff: RollOnandOffIDNum
- containerNumber: Integer
- deliveryID: Integer
- rollOnOffID: Integer
+ nextContainerNumber(): Integer
+ nextDeliveryID(): Integer
+ nextRollOnOffIDNum(): Integer
WareHouseClerk
+ ClerkID: Integer
+ clerkName:String
+ setPropsOnCreate()
DeliveryOrder
+ companyName: String
+ contractPhone: String
+ dateOrderPlaced: String
+ deliveryAddress: String
+ distance: Decimal
+ invoice: Integer
+ myManageCargo: ManageCargo
- orderID: Integer
+ recipient: String
+ totalValueMoney: Decimal
+ CalculatePrice(): Decimal
+ create()
+ setPropsOnCreate()
+ setPropsUpdate()
+ stringTooLongExceptionHandler