

CMPS242 – Object Oriented Programming

Course Project

A PROGRAMMER'S PERSPECTIVE

Port Management
System Manager

Presented By
Haifa Sidani (202400882)

4/23/2024

Abstract

The Port Management System project is about creating a software that helps manage ships and containers at different ports. It deals with different types of containers and ensures smooth loading/unloading operations between ships and ports. It also keeps tracks of how much fuel ships need to travel between ports, also includes many main features. This project helps improve how ports handle containers, making things more efficient and organized.

This program or system is written using Java Programming Language in Eclipse IDE. It covers object-oriented topics such as encapsulation, inheritance, and polymorphism to structure the code.

Introduction

General idea about the topic in the project.

Managing the transit of ships and containers between ports is the main idea of the project. Ships move several kinds of cargo or containers, and each one needs to be handled differently. Ships can be loaded with containers at ports and unloaded from ships back at ports. Ships require fuel in order to move between ports as well. Creating a system that effectively coordinates these activities is the main objective.

Idea related to the project and what was done in this project.

The general idea about the project was accomplished successfully. A Port Management System is made in which it is a program that allows admin to simulate the Ships and Ports operations. It also helps visualize the containers, ships, and ports present in the list using OOP concepts.

Related Work and Technologies

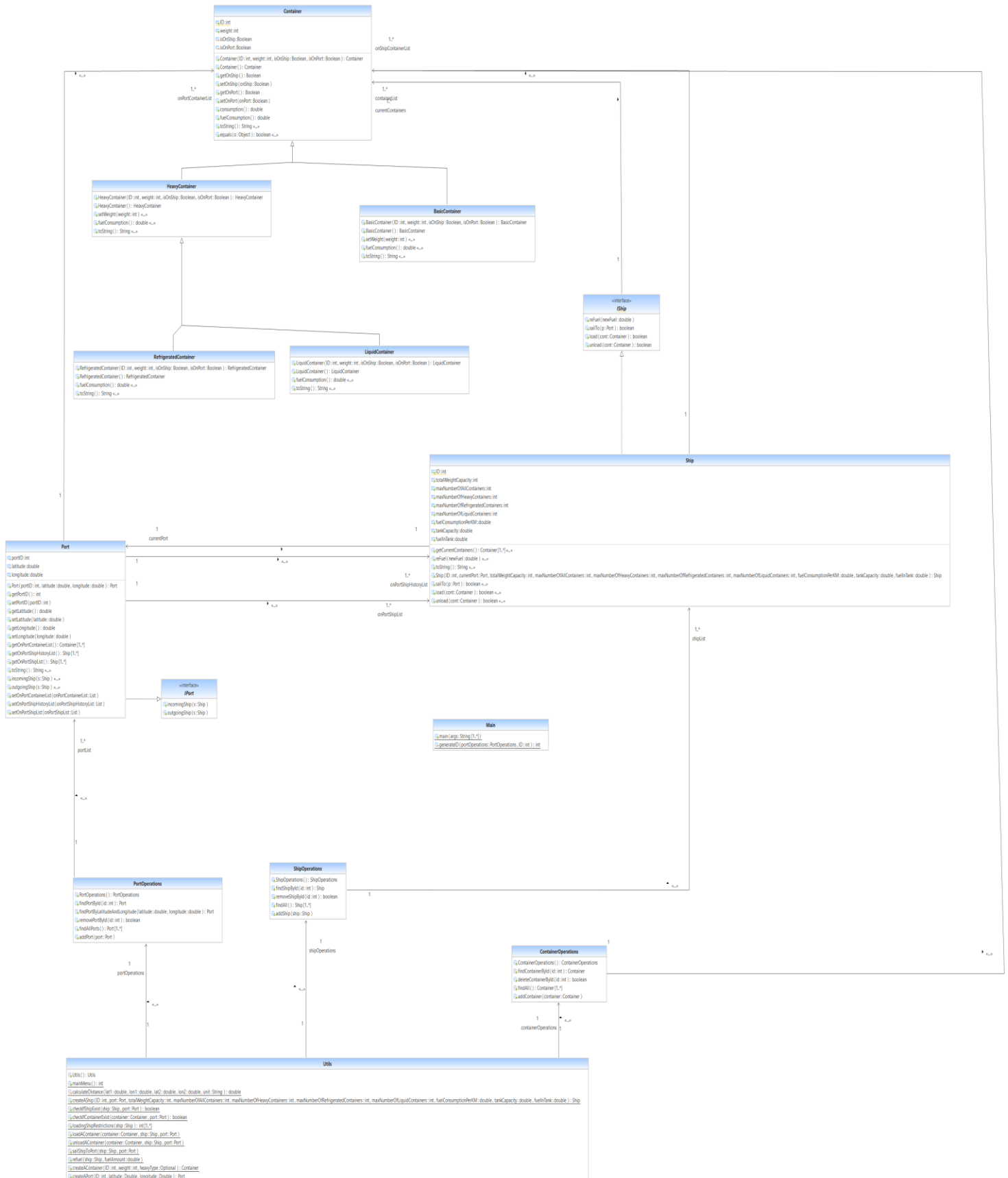
Program Used:

- Eclipse IDE for Java Developers
- UML Lab extension in Eclipse

How was it used?

Eclipse was used to edit the code and run it. Also, to draw the UML diagram by UML lab extension in Eclipse.

UML Diagram (You can find a .png file in the project folder)



Implementation

Implementation of the project

In the main class, basic OOP concept is used to initialize objects. The port, ship and containers control/operations were initialized.

```
package main;

import operations.ContainerOperations;

public class Main {

    public static void main(String[] args) {

        try {
            PortOperations portOperations = new PortOperations();
            ShipOperations shipOperations = new ShipOperations();
            ContainerOperations containerOperations = new ContainerOperations();
        }
    }
}
```

```
package container;

public class Container {
    private int ID;
    private int weight;
    private Boolean isOnShip;
    private Boolean isOnPort;

    public Container(int ID, int weight, boolean isOnShip, boolean isOnPort) {
        this.ID = ID;
        this.weight = weight;
        this.isOnShip = isOnShip;
        this.isOnPort = isOnPort;
    }

    public Container() {
    }
}
```

```
package container;

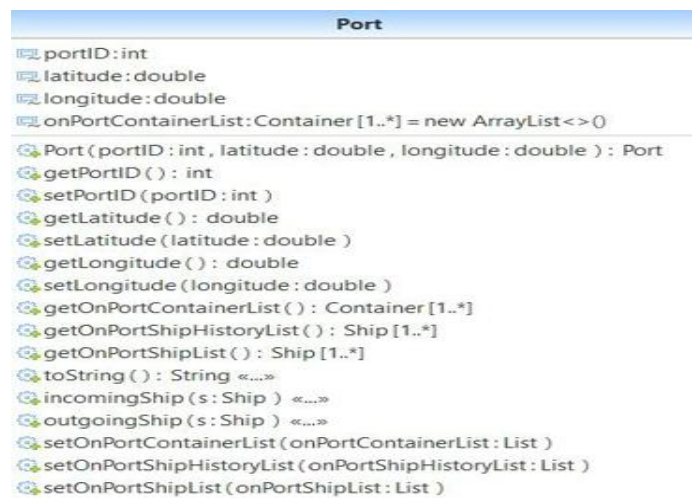
public class HeavyContainer extends Container {
    public HeavyContainer(int ID, int weight, Boolean isOnShip, Boolean isOnPort) {
        super(ID, weight, isOnShip, isOnPort);
    }

    public HeavyContainer() {
    }
}
```

Explaining The Classes

Port Class

- It implements the **IPort** interface and the methods.
- It represents a port in the system with attributes such as portID, latitude, and longitude.
- The constructor initializes a Port object with the provided attributes.
- The class has getters and setters for all the attributes.
- The class has three lists: onPortContainerList, onPortShipHistoryList, and onPortShipList.
- onPortContainerList: List of containers currently at the port.
- onPortShipHistoryList: List of all ships that have visited the port.
- onPortShipList: List of ships currently at the port.



```
Port
portID: int
latitude: double
longitude: double
onPortContainerList: Container [1..*] = new ArrayList<>()

Port (portID : int , latitude : double , longitude : double ) : Port
getPortID ( ) : int
setPortID ( portID : int )
getLatitude ( ) : double
setLatitude ( latitude : double )
getLongitude ( ) : double
setLongitude ( longitude : double )
getOnPortContainerList ( ) : Container [1..*]
getOnPortShipHistoryList ( ) : Ship [1..*]
getOnPortShipList ( ) : Ship [1..*]
toString ( ) : String «...»
incomingShip ( s : Ship ) «...»
outgoingShip ( s : Ship ) «...»
setOnPortContainerList ( onPortContainerList : List )
setOnPortShipHistoryList ( onPortShipHistoryList : List )
setOnPortShipList ( onPortShipList : List )
```

Ship Class

- It implements the **IShip** interface and the methods.
- It represents a ship in the system with attributes such as ID, currentPort, totalWeightCapacity.
- The constructor initializes a Ship object with the provided attributes.
- The class has getters and setters for all the attributes.
- The class has a list onShipContainerList.
- onShipContainerList: List of containers currently on the ship.
- Methods such as sailTo, reFuel, load, and unload.
- sailTo: Method to move a ship to a port.
- refuel: Method to refuel the ship with fuel.
- load: Method for loading a container to the ship.
- unload: Method to unload a container from the ship.

| Ship |
|---|
| <pre> ID: int totalWeightCapacity: int maxNumberOfAllContainers: int maxNumberOfHeavyContainers: int maxNumberOfRefrigeratedContainers: int maxNumberOfLiquidContainers: int fuelConsumptionPerKM: double tankCapacity: double fuelInTank: double onShipContainerList: Container [1..*] = new ArrayList<>() getCurrentContainers(): Container [1..*] «...» refuel(newFuel: double) «...» toString(): String «...» Ship(ID: int, currentPort: Port, totalWeightCapacity: int, maxNumberOfAllContainers: int, maxNumberOfHeavyContainers: int, maxNumberOfRefrigeratedContainers: int, maxNumberOfLiquidContainers: int, fuelConsumptionPerKM: double, tankCapacity: double, fuelInTank: double): Ship sailTo(p: Port): boolean «...» load(cont: Container): boolean «...» unload(cont: Container): boolean «...» </pre> |

Container Class

- It represents a generic container used in the software.
- The class has constructor Container with attributes such as ID, weight, isOnPort, and isOnShip.
- isOnShip and isOnPort: Boolean to check if a container is on a ship or at a port.
- The class has getters and setters for all the attributes to return and set values.
- It represents as super class for more specific container types.
- Methods such as fuelConsumption which represents the fuel consumption of the container.

| Container |
|---|
| <pre> ID: int weight: int isOnShip: Boolean isOnPort: Boolean Container(ID: int, weight: int, isOnShip: Boolean, isOnPort: Boolean): Container Container(): Container getOnShip(): Boolean setOnShip(onShip: Boolean) getOnPort(): Boolean setOnPort(onPort: Boolean) consumption(): double fuelConsumption(): double toString(): String «...» equals(o: Object): boolean «...» </pre> |

BasicContainer Class

- It represents a basic container type and extends the Container class.
- The class has constructor BasicContainer with attributes such as ID, weight, isOnPort, and isOnShip.
- isOnShip and isOnPort: Boolean to check if a container is on a ship or at a port.
- The class has getters and setters for all the attributes to return and set values.
- fuelConsumption: Overrides the fuelConsumption method to calculate the fuel consumption of a basic container. It multiplies the weight of the container by (2.50) and returns the result.
- setWeight: Overrides the setWeight method to check if the weight is 5000, if it exceeds it will print that the container is a heavy container.

| BasicContainer |
|--|
| <pre> BasicContainer(ID: int, weight: int, isOnShip: Boolean, isOnPort: Boolean): BasicContainer BasicContainer(): BasicContainer setWeight(weight: int) «...» fuelConsumption(): double «...» toString(): String «...» </pre> |

HeavyContainer Class

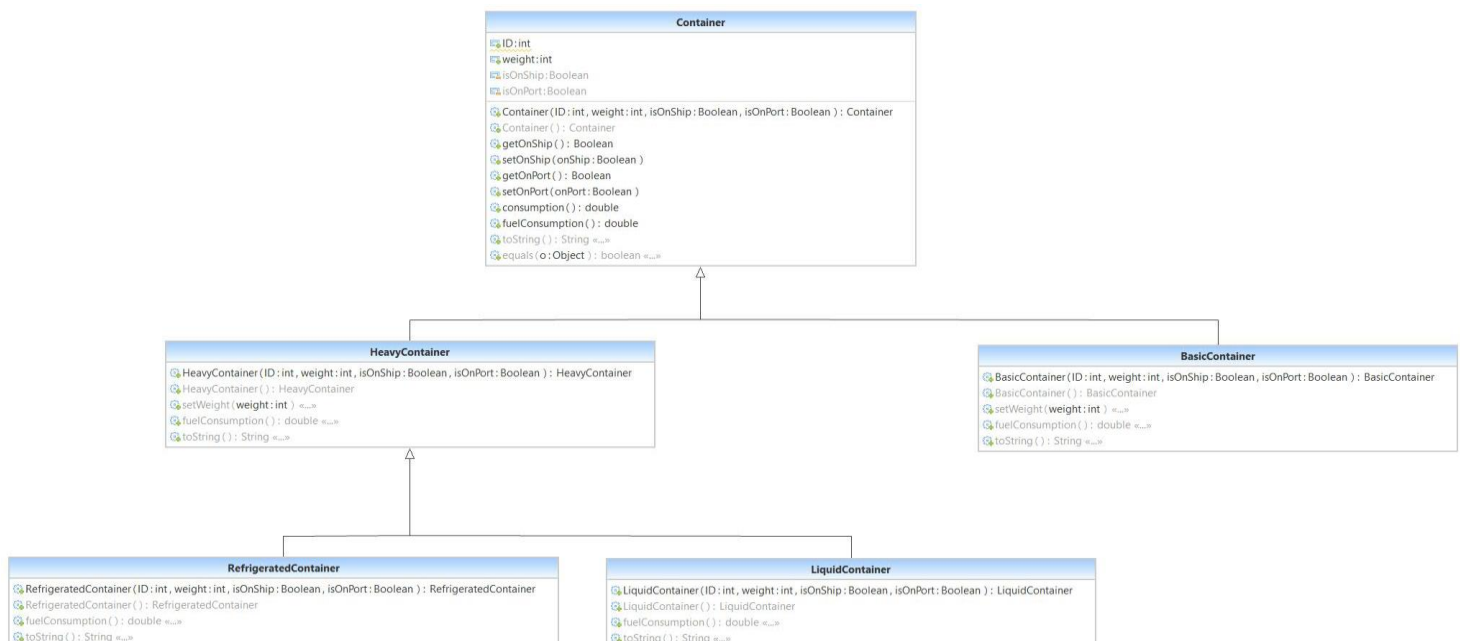
- It represents a heavy container type and extends the Container class.
- The class has constructor HeavyContainer with attributes such as ID, weight, isOnPort, and isOnShip.
- isOnShip and isOnPort: Boolean to check if a container is on a ship or at a port.
- The class has getters and setters for all the attributes to return and set values.
- fuelConsumption: Overrides the fuelConsumption method to calculate the fuel consumption of a basic container. It multiplies the weight of the container by (3.00) and returns the result.
- setWeight: Overrides the setWeight method to check if the weight is more than 5000, if it is less than that it will print that the container is a basic container.

LiquidContainer Class

- It represents a heavy container type specifically designed for carrying liquids, and extends the HeavyContainer class.
- The class has constructor LiquidContainer with attributes such as ID, weight, isOnPort, and isOnShip.
- isOnShip and isOnPort: Boolean to check if a container is on a ship or at a port.
- The class has getters and setters for all the attributes to return and set values.
- It inherits characteristics and behaviors from its parent class HeavyContainer.
- fuelConsumption: Overrides the fuelConsumption method to calculate the fuel consumption of a basic container. It multiplies the weight of the container by (4.00) and returns the result.

RefrigeratedContainer Class

- It represents a heavy container type specifically designed for refrigerated container, and extends the HeavyContainer class.
- The class has constructor RefrigeratedContainer with attributes such as ID, weight, isOnPort, and isOnShip.
- isOnShip and isOnPort: Boolean to check if a container is on a ship or at a port.
- The class has getters and setters for all the attributes to return and set values.
- It inherits characteristics and behaviors from its parent class HeavyContainer.
- fuelConsumption: Overrides the fuelConsumption method to calculate the fuel consumption of a basic container. It multiplies the weight of the container by (5.00) and returns the result.



IPort Interface

- It outlines the essential behaviors that port classes must implement.
- Method `incomingShip(Ship s)`: This method is responsible for handling the arrival of a ship at the port. It takes a `Ship` object as a parameter, indicating the ship that has arrived at the port.
- Method `outgoingShip(Ship s)`: This method deals with the departure of a ship from the port. It takes a `Ship` object as a parameter, indicating the ship that is departing from the port.

IShip Interface

- It outlines the essential behaviors that ship classes must implement.
- Method `getCurrentContainers`: This method is responsible for checking what containers are currently on the ship.
- Method `sailTo(Port p)`: Checks if the ship can go to a specific port and checks if it managed to get there.
- Method `reFuel(double newFuel)`: Adds more fuel to the ship's tank.
- Method `load(Container cont)`: Load a container onto the ship and checks if it succeeded.
- Method `unload(Container cont)`: Unload a container off the ship and checks if it succeeded.

MainTest Class

A test class was used during the development of this project to test every feature that was worked on.

- It starts by initializing operations for ports, ships, and containers.
- Present a menu of options to the user until choose to exit.
- Within the loop, it reads the user's choice and performs actions accordingly.
- Options include creating containers, ships, ports, loading/unloading containers from ships, sailing ships to other ports, refueling ships, and exiting the system.
- Each option is a separate case in a switch statement.
- Exception handling is implemented to catch and handle errors.
- The `generateID` method is used to ensure unique IDs for ports.
- `generateID`: Recursively generates a unique ID for a port.

| ContainerOperations |
|--|
| <code>containerList: Container [1..*] = new ArrayList<>()</code> |
| <code>ContainerOperations() : ContainerOperations</code> |
| <code>findContainerById (id : int) : Container</code> |
| <code>deleteContainerById (id : int) : boolean</code> |
| <code>findAll () : Container [1..*]</code> |
| <code>addContainer (container : Container)</code> |

| PortOperations |
|--|
| <code>portList: Port [1..*] = new ArrayList<>()</code> |
| <code>PortOperations() : PortOperations</code> |
| <code>findPortById (id : int) : Port</code> |
| <code>findPortByLatitudeAndLongitude (latitude : double , longitude : double) : Port</code> |
| <code>removePortById (id : int) : boolean</code> |
| <code>findAllPorts () : Port [1..*]</code> |
| <code>addPort (port : Port)</code> |

| ShipOperations |
|--|
| <code>shipList: Ship [1..*] = new ArrayList<>()</code> |
| <code>ShipOperations() : ShipOperations</code> |
| <code>findShipById (id : int) : Ship</code> |
| <code>removeShipById (id : int) : boolean</code> |
| <code>findAll () : Ship [1..*]</code> |
| <code>addShip (ship : Ship)</code> |

Results

Results and Techniques

First, a main class was developed with numerous test methods that tested every single feature that was implemented in the project, and everything passed properly after fixing couple of bugs.

After that, the main method was developed, linked everything up, and ran the program for the first time, which worked flawlessly.

Visual Results

Main Menu

```
#####
***** Welcome to Ships And Port Management System *****
#####

Choose an option
[1] Create a container
[2] Create a ship
[3] Create a port
[4] Load a container to a ship
[5] Unload a container from a ship
[6] Sail ship to another port
[7] Refuel ship
[8] Exit

Enter Option:
```

Create Container

```
#####
***** Welcome to Ships And Port Management System *****
#####

Choose an option
[1] Create a container
[2] Create a ship
[3] Create a port
[4] Load a container to a ship
[5] Unload a container from a ship
[6] Sail ship to another port
[7] Refuel ship
[8] Exit

Enter Option:
1
:::::::::::::::::CREATE CONTAINER:::::::::::::::::
Enter ID:
1
Enter weight:
6000
Enter type of heavy container
[L] for Liquid [R] for Refrigerated [N] is not specified:
L
Container created successfully
{
ID: 1
Weight: 6000
isOnShip: false
isOnPort: true
Type: Heavy Container
}
Category: Liquid Container
}
```

Conclusion

In the end of the project, we can say that the project was a success, and that it helped us develop the skills that we learned from this course. It taught how to use Object Oriented topics by developing a program to serve a certain purpose as a real-world scenario.