TPK4186 - Advanced Tools for Performance Engineering Spring 2023

Assignment 1: Container Ships

Done by Christian G Kartveit

Introduction

This project is done with previous knowledge in python from ITGK (TDT4109), and done solo. I had to make many assumptions throughout the project, and therefore some of my functions are advanced, which made the whole project a lot harder to finish.

Assumptions

1. The ship and its containers needed to be represented in a 3D list, which can be printed out with one of the following methods below. This made the later tasks harder to do. Found out too late that this was not the task, and too late for me to change all my code. If I were to start over, I would create stacks, and then use methods like pop, and push to change the order of containers.
2. The sum of 2 small containers will not be superior to one big. The highest total weight per container is used when stacked.
3. To remove a container at the bottom, all the containers above need to be removed, ref Task 5 explanation.
4. The csv files mentioned in the documentation is supposed to be .tsv , but not discovered before writing the documentation.
5. Removing containers needs to happen in the reversed order of placement, or remove all containers at dock. To hard to implement otherwise with my approach
6. Task 11/12 only work with specified ship setup
7. Calculating the weight of the ship is only relevant if the ship is partly filled up.

Tasks

Task 1

This is the Container class. It has the appropriate attributes s.a length, weight, freight, id, big/small, and a position. When a container is placed on the ship, it gets a position.

```python
class Container:
    #Container Constructor
    def __init__(self, length, weight, freight, id):
        self.length = length
        self.weight = weight
        self.freight = freight
        self.id = id
        self.big = True if length == 40 else False   # Boolean
        self.position = [0, 0, 0]

    def setContainer(self, length, weight, freight):
        self.length = length
        self.weight = weight
        self.freight = freight

    def setPosition(self, position):
        self.position = position

    def getPosition(self):
        return self.position

    def getLength(self):
        return self.length

    def getId(self):
        return self.id

    def getWeight(self) -> int:
        return int(self.weight)

    def getFreight(self) -> int:
        return int(self.freight)

    def setFreight(self, newFreight):
        self.freight = newFreight

    def getContainer(self):
        return self

    def isBigContainer(self):
        return self.big

    def getTotalWeight(self) -> int:
        return self.getWeight() + self.getFreight()

    def __str__(self):
        return str(self.id) + "\t" + str(self.length) + "\t" + str(self.weight) + "\t" + str(self.freight)
```

Task 2

This is more appropriate to do in the Ship class, or specify that you need a manager of containers/dock to use these methods. Therefore was this implemented in the Ship, and explained later.

Task 3

First method, randomContainer() returns a new Container object, that is randomly created. randomContainers() returns a set of x unique containers.

```
# code = id
def randomContainer() -> (Container):
    #Random length of container, 20 or 40
    length = random.sample((20, 40), 1)[0]
    if length > 0:
        if length <= 20:
            weight = 2
            #Uses python builtin id function to generate a unique id with datetime now as seed, for a unique id
            code = id(datetime.datetime.now())
            load = random.sample(range(0, 20), 1)[0]
        elif length <= 40:
            weight = 4
            code = id(datetime.datetime.now())
            load = random.sample(range(0, 22), 1)[0]
    return Container(length, weight, load, code)


def randomContainers():
    containers = []
    # 6600 containers in this method for typical ship in this task.
    # Can be changed to any number of containers
    for i in range(0, 6600):
        containers.append(randomContainer())
    return set(containers)
```

Task 4

writeContainersToFile(containers) takes in a list, and writes the list to file, by iterating through the list, and using getters to get information needed. This is saved in a separated .csv file

```
def writeContainersToFile(containers):
    file = open("containers.csv", "w")
    for container in containers:
        file.write(str(container.getId()) + "\t" + str(container.getLength()) +
                   "\t" + str(container.getWeight()) + "\t" + str(container.getFreight()) +"\t" + str(container.getTotalWeight()) + "\n")
```

```
def readContainersFromFile():
    containers = []
    file = open("containers.csv", "r")
    for line in file:
        line = line.split("\t")
        containers.append(Container(int(line[1]), int(
            line[2]), int(line[3]), int(line[0])))
    return containers
```

This is a screenshot of the .csv file.
readContainersFromFile() will iterate line to line through the file, and create container objects in the returned list, containers.

```
1   140222434123824 20  2   14  16
2   140222435172400 20  2   2   4
3   140222435696688 20  2   18  20
4   140222434123872 40  4   15  19
5   140222435172448 20  2   13  15
6   140222435696736 40  4   21  25
7   140222434123920 40  4   16  20
8   140222435172496 20  2   2   4
9   140222435696784 40  4   21  25
10  140222434123968 40  4   6   10
11  140222435172544 40  4   2   6
12  140222435696832 20  2   13  15
13  140222434124016 40  4   3   7
14  140222435172592 40  4   9   13
15  140222435696880 40  4   10  14
16  140222434124064 20  2   11  13
17  140222435172640 40  4   21  25
```

Task 5

This is the Ship constructor. It has a length, width and height.
It also contains a containers dictionary to more easily find container objects.
containerBay is a 3D representation of the loading zone in the Ship. I created methods to fulfill the different restrictions, ref a 40 foot container cant be loaded on a single 20 foot.

```python
class Ship:
    def __init__(self, length, width, height):
        self.length = length
        self.width = width
        self.height = height
        self.containers = dict()  # For å finne container objekter lettere
        self.containerBay = [
            [[0 for j in range(width)] for i in range(length)] for k in range(height)]
        # width, lenght, height
        # containerBay[0] = height, containerBay[0][0] = length, containerBay[0][0][0] = width
```

In a Ship that is 23 Long, 22 Width and 18 height, this is the representation of One layer in the ship, unloaded:

```
Layer 0

 Row 1: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 2: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 3: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 4: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 5: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 6: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 7: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 8: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 9: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 10: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 11: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 12: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 13: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 14: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 15: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 16: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 17: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 18: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 19: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 20: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 21: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 22: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 Row 23: 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

```
def printNice(self):
    for i in range(0, self.height):
        line = ""
        print("Layer " + str(i))
        for j in range(0, self.length):
            line += "\n Row " + str(j+1) + ": "
            for k in range(0, self.width):
                if isinstance(self.containerBay[i][j][k], Container):
                    line += str(self.containerBay[i]
                                [j][k].getLength()) + " "
                else:
                    line += "0 "
        print(line)
```

Every 0 represents a free space in the ship.
printNice is the method that prints out the ship

To find a container on the ship, we used the dictionary for fast lookup times, and returned the Container object, with the id given, else returns None if it is not in the Ship. We can also use this to get the position of the container on the Ship with the getPosition() method of the returned container.

```
def getContainerDict(self, id):
    return self.getContainersDict().get(id)
```

The Ship also has appropriate getters, but it will not be shown in this documentation.

The function placeFirstAvailableSpot, takes in a container, and number of cranes, and places the container on the first available spot. It will first look for a spot that fill these requirements:

- The ship is not full, checked with the findLastSpot function
- It has places for a 40 foot or 20 foot

- If 40 foot, check if there are no holes below, given we are not on the first layer
    - Checks if there is place on the side, since it takes up 2 x 20 spots
    - Places the container in the 3D map, and then tells the method, that is has finished with setting Continue to False
    - It also sets the position of the container to the exact position in the ship
    - Index error appears when we try to check the spot next to a wall, so it exceeds the wall. E.g try to place a 40 foot to the far right wall, and there is not space enough for 2 * 20 slots. Then it will go back up and choose another row, or layer.

The algorithm will find the first available spot to place a container in the ship, starting from the left corner on the ship. It will maximize the amount of containers that can be placed on the ship, regarding if its 40 foot, or 20 foot.

```
# Testet
def placeFirstAvailableSpot(self, container,cranes):
    sizeNeeded = container.getLength()
    Continue = True
    if self.findLastSpot() == 0:
        for i in range(0, self.height):
            if Continue:
                for j in range(cranes, self.length):
                    if Continue:
                        for k in range(0, self.width):
                            if Continue:
                                if self.containerBay[i][j][k] == 0:
                                    if sizeNeeded == 40:
                                        try:
                                            if self.containerBay[i][j][k+1] == 0:
                                                # Check if container below is not empty
                                                try:
                                                    if i > 0:
                                                        if self.containerBay[i-1][j][k] != 0 and self.containerBay[i-1][j][k+1] != 0:
                                                            self.containerBay[i][j][k] = container
                                                            self.containerBay[i][j][k +
                                                                                    1] = container
                                                            container.setPosition(
                                                                [i, j, k])
                                                            Continue = False
                                                            return True
                                                        else:
                                                            print(
                                                                "Container below is not empty")
                                                    else:
                                                        self.containerBay[i][j][k] = container
                                                        self.containerBay[i][j][k +
                                                                                1] = container
                                                        container.setPosition(
                                                            [i, j, k])
                                                        Continue = False
                                                        return True
                                                except IndexError:
                                                    continue
                                        except IndexError:
                                            continue
                                    elif sizeNeeded == 20:
                                        self.containerBay[i][j][k] = container
                                        container.setPosition([i, j, k])
                                        Continue = False
                                        return True
    else:
        pass
```

It will check the last layer and the last position, to check if it's full.

```
def findLastSpot(self):
    return (self.containerBay[-1][-1][-1] != 0) and self.checkLastLayer()

def checkLastLayer(self):
    for i in range(0, self.length):
        for j in range(0, self.width):
            if self.containerBay[-1][i][j] != 0:
                return False
    return True
```

If we are able to place the container, addContainerToDict will run, and add the container to the Dictionary.

```
def addContainer(self, container):
    if (self.placeFirstAvailableSpot(container,0)):
        self.addContainerToDict(container)
```

These methods will show that the methods over will work.

```
newShip = Ship(23, 22, 18)
containers = randomContainers()
newShip.loadContainersIncreasingWeight(containers)
newShip.printNice()
```

Since it places the heaviest containers low, it will mostly only contain 40 foot long containers.

```
Layer 0

 Row 1: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 2: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 3: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 4: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 5: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 6: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 7: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 8: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 9: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 10: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 11: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 12: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 13: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 14: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 15: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 16: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 17: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 18: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 19: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 20: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 21: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 22: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
 Row 23: 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40 40
```

To remove a container from the ship, we need to use removeContainerFromShip, and it needs the ID to the container.
If it exist in the ship, it will remove it from the dictionary, and set the position to zero in the map

```
def removeContainerFromShip(self, id):
    container = self.getContainerDict(id)
    if container != None:
        # del self.containers[id]
        position = container.getPosition()
        del self.containers[id]
        if container.getLength() == 40:
            self.containerBay[position[0]][position[1]][position[2]] = 0
            self.containerBay[position[0]][position[1]][position[2]+1] = 0
        else:
            self.containerBay[position[0]][position[1]][position[2]] = 0
        self.loadContainersIncreasingWeight("")
```

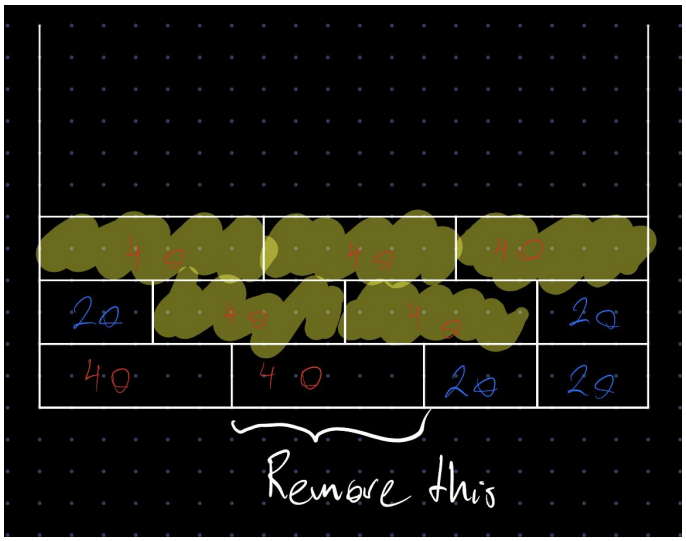
Tried to make a method to remove the containers above, but it was too complicated to do with this implementation of the Ship. The containers can still be removed in correct order, but it needs to use the method unloadContainersToSet, and use that to get the correct order to remove them.

```
def unloadContainersToSet(self):
    # Stack
    orderedList = []
    containers = list(self.getContainers())
    for container in containers:
        self.removeContainerFromShip(container.getId())
        orderedList.append(container)
    return orderedList
```

This unimplemented method does not work optimal because of the placement of containers. (part of removeContainerFromShip method)

```
# while positionz < self.height:
#     container = self.getContainerDict(id)
#     try:
#         if container.getLength() == 40:
#                 if self.containerBay[positionz+1][position[1]][position[2]] != 0:
#                     id = self.containerBay[positionz+1][position[1]][position[2]].getId()
#                     if id != None:
#                         print("Fjerner ", id)
#                         del self.containers[id]
#                         self.containerBay[positionz+1][position[1]][position[2]] = 0
#                         self.containerBay[positionz+1][position[1]][position[2]+1] = 0
#                     else:
#                         id = self.containerBay[positionz+1][position[1]][position[2]-1]
#                         print("Fjerner ", self.containerBay[id].getPosition() )
#                         del self.containers[id]
#                         self.containerBay[positionz+1][position[1]+1][position[2]-1] = 0
#                         self.containerBay[positionz+1][position[1]+1][position[2]] = 0
#                     positionz +=1
#         else:
#                 if self.containerBay[positionz][position[1]][position[2]] != 0:
#                     id = self.containerBay[positionz][position[1]][position[2]].getId()
#                     print("Fjerner ", self.containerBay[id].getPosition())
#                     del self.containers[id]
#                     self.containerBay[positionz][position[1]][position[2]] = 0
#                     positionz +=1
# # kommer av at vi er på feil side av containeren når vi prøver å fjerne den
#     except AttributeError or IndexError:
#         print("IndexError")
#         self.containerBay[positionz+1][position[1]+1][position[2]-1] = 0
#         self.containerBay[positionz+1][position[1]+1][position[2]] = 0
#         positionz+=1
#         continue

# print("Removed container from " + str(position))
```



The image to the right describes the problem of unloading a container. Almost all containers above need to be removed, and not smart in real life. But since this is a simulation, it is okay. One of my assumptions is that you need to remove all containers, or remove them in reversed placed order.

Task 6

Very familiar to the previous read/write, but it only writes containers loaded on the Ship, from the getContainers method.

```
def writeLoadedContainersToFile(self):
    file = open("loadedContainers.csv", "w")
    for container in self.getContainers():
        file.write(str(container.getId()) + "\t" + str(container.getLength()) +
                   "\t" + str(container.getWeight()) + "\t" + str(container.getFreight()) + "\n")

def readLoadedContainersFromFile(self):
    file = open("loadedContainers.csv", "r")
    for line in file:
        line = line.split("\t")
        container = Container(int(line[1]), int(
            line[2]), int(line[3]), int(line[0]))
        self.addContainer(container)
```

loadedContainers.csv

```
140171102131472 40  4   21
140171105278256 40  4   21
140171102132912 40  4   21
140171102133296 40  4   21
140171104759856 40  4   21
140171104236336 40  4   21
140171102159904 40  4   21
140171102162160 40  4   21
140171104787168 40  4   21
140171104269584 40  4   21
140171104809200 40  4   21
140171102188048 40  4   21
140171102189200 40  4   21
140171104811360 40  4   21
140171104290304 40  4   21
140171104291024 40  4   21
140171104291552 40  4   21
140171104292464 40  4   21
140171102212288 40  4   21
140171102215840 40  4   21
140171104842352 40  4   21
140171104842832 40  4   21
140171104845136 40  4   21
140171104845376 40  4   21
140171102241344 40  4   21
140171102243792 40  4   21
```

Task 7

loadContainersFromSet will add containers in the ship, and return an orderedList by appending each container to the list.

unloadContainersToSet will return an orderedList.
They will follow all the constraints by using addContainer method, and removeContainer.

```
def loadContainersFromSet(self, containers):
    # Stack
    orderedList = []
    for container in containers:
        self.addContainer(container)
        orderedList.append(container)
    return orderedList

def unloadContainersToSet(self):
    # Stack
    orderedList = []
    containers = list(self.getContainers())
    for container in containers:
        self.removeContainerFromShip(container.getId())
        orderedList.append(container)
    return orderedList
```

Task 8

Loading containers from either the list or the dictionary, then sort them after weight before adding them to the list

```
def loadContainersIncreasingWeight(self, containers):
    if len(containers) == 0:
        liste = list(self.getContainers())
        self.resetLoadedContainers()
    else:
        liste = list(containers)
    liste.sort(key=lambda x: x.getTotalWeight(), reverse=True)
    for container in liste:
        self.addContainer(container)
```

This method will do the same, but takes longer to load, and not needed when python has a built-in function to sort after weight.

```
def findHeaviestContainers(self, containers):
    popped = []
    containers = list(containers)
    readyToPlace = []
    readyToPlace.append(containers.pop())

    for i in range(len(containers)):
        w = containers[i].getTotalWeight()
        try:
            while w > readyToPlace[-1].getTotalWeight():
                popped.append(readyToPlace.pop())
            readyToPlace.append(containers[i])
            for j in range(len(popped)):
                readyToPlace.append(popped.pop())
        except IndexError:
            readyToPlace.append(containers[i])
    return readyToPlace
```

Task 9

Total weight of containers on ship:

```
def calculateTotalContainerWeightLoaded(self):
    totalWeight = 0
    for container in self.getContainers():
        totalWeight += container.getTotalWeight()
    return totalWeight
```

Weight starboard and portside:

```
def calculateTotalContainerWeightStarboard(self):
    totalWeight = 0
    for container in self.getContainers():
        if container.getPosition()[2] > self.getWidth()/2:
            totalWeight += container.getTotalWeight()
    return totalWeight

def calculateTotalContainerWeightPortside(self):
    totalWeight = 0
    for container in self.getContainers():
        if container.getPosition()[2] < self.getWidth()/2:
            totalWeight += container.getTotalWeight()
    return totalWeight
```

Weight front, middle and back:

```
def calculateTotalContainerWeightForward(self):
    totalWeight = 0
    for container in self.getContainers():
        # fra 16 til 22 (7) plasser pos 16 til 22
        if container.getPosition()[1] > self.getLength()*2/3:
            totalWeight += container.getTotalWeight()
    return totalWeight

def calculateTotalContainerWeightMiddle(self):
    totalWeight = 0
    for container in self.getContainers():
        # 15<=x>=7 pos 8 til 15 (8)
        if container.getPosition()[1] <= self.getLength()*2/3 and container.getPosition()[1] > self.getLength()/3:
            totalWeight += container.getTotalWeight()
    return totalWeight

def calculateTotalContainerWeightBack(self):
    totalWeight = 0
    for container in self.getContainers():
        # fra 0 -> <8 8plasser pos 0 til 7 (8)
        if container.getPosition()[1] < round(self.getLength()/3):
            totalWeight += container.getTotalWeight()
    return totalWeight
```

Calculate percentage weights:

```
def calculatePercentageStarPort(self):
    try:
        res = round((abs(self.calculateTotalContainerWeightStarboard(
        )-self.calculateTotalContainerWeightPortside())/self.calculateTotalContainerWeightPortside())*100.0, 3)
        print(
            "Percent difference in weight from Starboard to Portside is: " + str(res) + "%")
        return res
    except ZeroDivisionError:
        print("Ship is not loaded properly")
```

```
def calculatePercentageForwardMiddle(self):
    try:
        res = round((abs(self.calculateTotalContainerWeightMiddle(
        )-self.calculateTotalContainerWeightForward())/self.calculateTotalContainerWeightForward())*100.0, 3)
        print(
            "Percent difference in weight from Forward to Middle is: " + str(res) + "%")
        return res
    except ZeroDivisionError:
        print("Ship is not loaded properly")

def calculatePercentageForwardBack(self):
    try:
        res = round((abs(self.calculateTotalContainerWeightForward(
        )-self.calculateTotalContainerWeightBack())/self.calculateTotalContainerWeightBack())*100.0, 3)
        print(
            "Percent difference in weight from Forward to Back is: " + str(res) + "%")
        return res
    except ZeroDivisionError:
        print("Ship is not loaded properly")
```

```
def calculatePercentageMiddleBack(self):
    try:
        res = round((abs(self.calculateTotalContainerWeightMiddle(
        )-self.calculateTotalContainerWeightBack())/self.calculateTotalContainerWeightBack())*100.0, 3)
        print(
            "Percent difference in weight from Middle to Back is: " + str(res) + "%")
        return res
    except ZeroDivisionError:
        print("Ship is not loaded properly")
```

Task 10

This function will split the list of containers in x amount, S.T the heaviest containers will be placed differently, and split. This is not an optimal solution, but hard to balance the ship with my approach.

```
def reBalanceContainers(self,containers, randval):
    self.resetLoadedContainers()
    # list(containers).sort(key=lambda x: x.getTotalWeight())
    # random.randint(1, 40)
    for i in range(randval):
        totW = 0
        exec(f'container{i} = list(containers)[i::{randval}]')
        # for j in (eval(f'container{i}')):
        #     totW += j.getTotalWeight()
        # print(totW)
        self.loadContainersIncreasingWeight(eval(f'container{i}'))
    print(randval)

def reBalanceUntilStable(self):
    containers = self.getContainers()
    randval = 1  # 54 nice?
    while self.calculatePercentageForwardMiddle() >= 10 or self.calculatePercentageForwardBack() >= 10 or self.calculatePercentageMiddleBack() >= 10 or self.calculatePercentageStarPort() >= 5:
        print("Rebalancing")
        randval += 1
        self.reBalanceContainers(containers,randval)
```

Task 11

4 cranes only work with the current ship, else use 1 crane. The algorithm to split the sections is not perfect, therefore it will not work with all ship types.
Loading takes :

```
 Row 23: 20 20 40 40 40 40 40 40 20 20 20 20 40 40 20 40 40
83586 Total weight
Time taken to load ship with 4 cranes is 5971.0 minutes
Time taken to load ship with 1 cranes is 23884.0 minutes
```

- Do not take into consideration that you need to sort heaviest containers first.
- Unloading takes an equal amount of time, because you can only remove all containers, or the upper ones first. (Assumption)

```
def loadWithCranes(self, numberOfCranes):
    time = 0
    costInTime = 4 # per container
    self.containerBay = [
        [[0 for j in range(self.width)] for i in range(self.length)] for k in range(self.height)]
    # place containers in bay in different sections of the ship according to numberOfCranes
    # 1 crane = 1 section
    #section is the length of the ship divided by the number of cranes

    #This function works with 4 cranes, if 1 crane, replace with 0 in the place function
    start = []
    for i in range(self.length):
        start.append(((i % 6) * 4 + (i // 6)))
    if numberOfCranes == 1:
        start = [0]*self.length
    j = 0
    for cont in self.getContainers():
        self.placeFirstAvailableSpot(cont,start[j])
        time += costInTime
        j += 1
        if j == 22:
            j = 0
    time = time/numberOfCranes
    print("Time taken to load ship with " + str(numberOfCranes) + " cranes is " + str(time) + " minutes")
```

Task 12

The ship is divided up into four sections, one crane in each section. Total time to unload is a quarter of the time with one crane