HarbNet - API dokumentasjon

Gruppe 8 – Andreas Berg, Mathilde Olesen, Michael Karlsrud, Tuva Marie Andersen

Contents

Getting started

To create and run a simulation, you need:

- Ship(s)
- Storage space on your harbor
- A harbor

1. First step will be to decide how many ships you want to simulate, and create these with the `Ship`-class.

Refer to the documentation further in the document as to what a Ship-object contains, and what is needed for the constructor.

There are two types of ships:

Single-trip ship

A *single-trip* ship completes one voyage, before returning to the Harbor, and after unloading the last cargo, permantely docks to a ship dock. If a ship dock is not available, the ship will anchor in the anchorage instead.

Continuous ship

A *continous* ship leaves the harbor on voyage, comes back to the harbor, unloads and loads new cargo and sails on voyage again – thus repeating the cycle. It will never dock to a ship dock, as it is on a repeatable schedule.

The different ship sizes decides the ships cargo capacity in both amount and weight.

<u>There are three types of ship sizes:</u>

Small	Capacity of 20 containers, and a base weight of 5000 tons
Medium	Capacity of 50 containers, and a base weight of 50 000 tons
Large	Capacity of 100 containers, and a base weight of 10 000 tons

We want to make two ships: One small *single-trip* ship and one large *continous* ship.

We would do something like this:

```
Ship briefBreeze = new("Brief Breeze", ShipSize.Small, new
DateTime(2024,04,15,10,0), true, 5, 4, 16);

Ship flyingDutchman = new("The Flying Dutchman", ShipSize.Large,
new DateTime(2024,04,16,10,0), false, 12, 15, 85);
```

```
List<Ship> listOfShips = new();

listOfShips.add(briefBreeze);
listOfShips.add(flyingDutchman);
```

Here we have our single-trip ship, **“Brief Breeze”**.
It will start its simulation on **April 15, 2024 at 10:00am**.
It takes **5** days for a roundtrip, and it has **4** half-size containers and **16** full-size containers onboard.

We also have our continous ship, **“The Flying Dutchman”**.
It starts its simulation one day later, on **April 16, 2024 at 10:00am**.
It takes **12** days for a roundtrip and has **15** half-size and **85** full-size containers for its first voyage.

We then create a list, **listOfShips**, to collect all our ships that is to be used in the simulation.

! See the `ContainerStorageRow` section for more information about half VS full size containers.

2. Next step is to decide the capacity for your container storage area.

The storage area is composed of rows, that each can hold a specified amount of containers.

If we want our storage area to fit 500 containers, distributed over **8** rows with a capacity of **50** each, and **4** rows with a capacity of **25** each, we could do something like this:

```
List<ContainerStorageRow> storageArea = new();

// Our 8 larger rows with space for 50 containers each
for (int i = 0; i < 8; i++)
{
  storageArea.add(new ContainerStorageRow(50));
}

// Our 4 smaller rows with space for 25 containers each
for (int i = 0; i < 4; i++)
{
  storageArea.add(new ContainerStorageRow(25));
}
```

Then we need to create our Harbor – and send in our ships and storage rows.

Both the ships and storage rows is sent in as a list containing the object. You can imagine that the list of rows is your “storage area”.

```
Harbor harbor = new(listOfShips, storageArea, 1, 2, 1, 5, 20, 2 ,
1, 1, 0, 10, 15, 20, 10, 15);
```

Here we have created a new harbor with the necessary information.
Here is a brief explaination of each parameter. *See further documentation for a wider understanding.*
We are sending with our **listOfShips** from step 1 and list of storage rows, that we called **storageArea** in step 2.
We have **1** small loading dock, **2** medium loading docks and **1** large loading dock.
We have **5** cranes located next to our loading dock and each crane can do **20** loads/lifts per hour.

We also have **2** additional cranes located next to our storage area.

Continuing on, we have **1** small ship dock, **1** medium ship dock and **0** large ship docks.
We have specified that **10** trucks will arrive at the harbor each hour.
15% of containers on a ship is loaded directly onto a truck - leaving the harbor by truck.
Additionally **20**% of containers stored in our storage area is also picked up by truck.
We have **10** ADVs that each can do **15** loads/moves per hour.

Classes

Ship

The `Ship` class is creating and representing the individual ships that is to be simulated in the Simulation.

Each ship creation needs to specifiy if the ship is a ship to be used for one voyage only (*single-trip*) or an unlimited voyages throughout the simulation time (*continous*).

A list of `Ships` are needed in the creation of a `Harbor`-object.

Constructor

Name

```
public Ship(
string shipName,
ShipSize shipSize,
DateTime startDate,
bool isForASingleTrip,
int roundTripInDays,
int numberOfHalfContainersOnBoard,
int numberOfFullContainersOnBoard
)
```

Description

Initializes a new instance of the Ship class with the specified specifications:

Parameters

`String shipName`: The ship name.	`ShipSize shipSize:` The ships size, represented as a ShipSize enum: Small, Medium or Large.
`DateTime startDate`: The date and time the ship is starting its simulation.	`Bool IsForASingleTrip`: True if the ship should only does one trip, false otherwise.
`Int RoundTripInDays`: Number of days the ship uses to complete a roundtrip at sea before retuning to harbor.	`Int NumberOfHalfContainersOnBoard`: How many half containers will be in the ships storage when it enters the harbor for the first time.
`Int NumberOfHalfContainersOnBoard`: How many full containers will be in the ships storage when it enters the harbor for the first time.	

Fields

Name	Type	Description
`ID`	`Guid`	Gets the unique ID of the ship, which is assigned internally when a ship is constructed.
`ShipSize`	`ShipSize`	Gets the ship size, Small, Medium or Large.
`Name`	`String`	Gets the given name of the ship.
`StartDate`	`DateTime`	Gets the scheduled date and time the ship is to start its activities in the simulation.
`RoundTripInDays`	`Int`	Gets the number of days the ship uses to complete a roundtrip at sea, before returning to the harbor.
`CurrentLocation`	`Guid`	Gets the ID of the current location of the ship.
`History`	`ReadOnly Collection`	Gets a Collection of all the activities the ship has gone through throughout the simulation, as `StatusLog` objects.
`ContainersOnBoard`	`IList`	Gets all the containers in the ship's storage.
`ContainerCapacity`	`Int`	Gets the max number of containers the ship can hold at any given time
`MaxWeightInTonn`	`Int`	Gets the max number of tons the ship can weigh, including cargo.
`BaseWeightInTonn`	`Int`	Gets the base weight of the ship alone, without cargo.
`CurrentWeightInTonn`	`Int`	Gets the current weight of the ship, including the cargo's weight, in ton.

Methods

Name	Type	Description
`PrintHistory()`	`void`	Prints `History` to console.
`HistoryToString()`	`String`	Returns `History` as a String.
`ToString()`	`String`	Returns a String containing information about the ship.

ContainerStorageRow

The `ContainerStorageRow` class is creating and representing the individual rows that together define the Harbor storage area. Each row contains a set amount of spaces, that is specified in the constructor.

Full and half size containers

A row can be assigned for either full size cotnainers or half size containers, but never both.

Which size that is allowed in the row is determined by the first container that is stored in the row. If the row is emptied out, it will again be determined by the first container it then recieves.

The number of *spaces* is the same number for both full and half size.
The difference in capacity of the row lays in the *spaces* and it’s set capacity. A space can store one(1) full size or two(2) half size containers.

A list of `ContainerStorageRow`s are needed in the creation of a `Harbor`-object.

Constructors

Name

`ContainerStorageRow(`
`int numberOfContainerStorageSpaces`
`)`

Description
Initializes a new instance of the ContainerStorageRow with a capacity of the specified number of container storage spaces.
Parameters
`Int numberOfContainerStorageSpaces:` The number of container spaces the row contains of, that can fit either one full size or two half size container, each. Example: 10 container storage spaces gives the row a capacity of 10 full containers or a maximum of 20 half-size containers.

Fields

Name	Type	Description
`ID`	`Guid`	Gets the unique ID of the storage row, which is assigned internally when a row is constructed.

Methods

Name	Type	Description
`numberOfFreeContainerSpaces(ContainerSize size)`	`Int`	Gets the number of free, unoccupied container spaces.
`SizeOfContainersStored()`	`ContainerSize`	Gets the size of the containers stored in the row.
`GetIDOfAllStoredContainers()`	`IList <Guid>`	Gets a list containing all the IDs of the stored containers
`ToString()`	`String`	Returns a String containing information about the storage row.

Harbor

The `Harbor` class is creating and representing the Harbor being simulated in the Simulation.

It needs a list of `Ship`-objects (to define which ships to simulate activities) and a list of `ContainerStorageRow`-objects (to define the size of the storage area on the Harbor).

Constructors

Name

```
Harbor(
IList<Ship> listOfShips,
IList<ContainerStorageRow> listOfContainerStorageRows,
int numberOfSmallLoadingDocks, int numberOfMediumLoadingDocks,
int numberOfLargeLoadingDocks, int numberOfCranesNextToLoadingDocks,
int LoadsPerCranePerHour, int numberOfCranesOnHarborStorageArea,
int numberOfSmallShipDocks, int numberOfMediumShipDocks,
int numberOfLargeShipDocks, int numberOfTrucksArriveToHarborPerHour,
int percentageOfContainersDirectlyLoadedFromShipToTrucks,
int percentageOfContainersDirectlyLoadedFromHarborStorageToTrucks,
int numberOfAdv, int loadsPerAdvPerHour
)
```

Description

Initializes a new Harbor-object with the specified details.

Parameter description

`IList<Ship> listOfShips`: A list of Ship-objects to be simulated in the harbor.	`IList<ContainerStorageRow> listOfContainerStorageRows`:

	A list of ContainerStorageRows, that the storage area of the harbor consists of.
`Int numberOfSmallLoadingDocks`: Number of small loading docks, where Ships will unload and load their cargo.	`Int numberOfMediumLoadingDocks`: Number of medium loading docks, where Ships will unload and load their cargo.
`Int numberOfLargeLoadingDocks`: Number of large loading docks, where Ships will unload and load their cargo.	`Int numberOfCranesNextToLoadingDocks`: Number of cranes located at the dock area. To be used for unloading and loading cargo from and to ships.
`Int LoadsPerCranePerHour`: Number of loads (containers) each crane iscapable of per hour.	`Int numberOfCranesOnHarborStorageArea`: Number of cranes located at the storage area. To be used to load and unload containers at the harbor's storage area.
`Int numberOfSmallShipDocks`: Number of small ship docks. A separate dock where ships are docked indefinintely and "stored".	`Int numberOfMediumShipDocks`: Number of medium ship docks. A separate dock where ships are docked indefinintely and "stored".
`Int numberOfLargeShipDocks`: Number of large ship docks. A separate dock where ships are docked indefinintely and "stored".	`Int numberOfTrucksArriveToHarborPerHour`: Number of trucks to arrive and be placed in "truck queue" at the harbor each hour.
`Int percentageOfContainersDirectlyLoadedFromShipToTrucks`: Percentage of containers loaded onto trucks from ship. The rest is stored in storage area.	`Int percentageOfContainersDirectlyLoadedFromHarborStorageToTrucks`: Percentage of containers loaded onto trucks from storage area. The rest is loaded onto ships.
`Int numberOfAdv`: Number of ADVs in harbor. ADVs move containers between loading dock cranes and storage area cranes.	`Int loadsPerAdvPerHour`: Number of loads (containers) each ADV is capable of per hour.

Fields

Name	Type	Description
`ID`	`Guid`	Gets the unique ID of the harbor, which is assigned internally when a harbor is constructed.
`ArrivedAtDestination`	`IList <Container>`	Gets all containers that have left the harbor and arrived at their destination.
`TransitLocationID`	`Guid`	Gets the unique ID representing the "on sea" transit location.
`AnchorageID`	`Guid`	Gets the unique ID of the harbor's anchorage.
`AdvCargoID`	`Guid`	Gets the unique ID of the ???
`TruckTransitLocationID`	`Guid`	Gets the unique ID representing the trucks' "in transit" location.
`TruckQueueLocationID`	`Guid`	Gets the unique ID representing the queue location for trucks.

`HarborStorageAreaID`	`Guid`	Gets the unique ID representing the storage area location.
`HarborDockAreaID`	`Guid`	Gets the unique ID representing the area containing all docks.
`DestinationID`	`Guid`	Gets the unique ID representing a containers destination.

Methods

Name	Type	Description
`LoadingDockIsFreeForAllDocks()`	`IDictionary <Guid, bool>`	Returns a Dictionary containing the IDs of all the loading docks ???
`ShipDockIsFreeForAllDocks()`	`IDictionary <Guid, bool>`	Returns a Dictionary containing the IDs of all the ship docks ???
`GetShipStatus(Guid shipID)`	`Status`	Returns the last registered status of the specified ship.
`GetStatusAllShips()`	`IDictionary <Ship, Status>`	Returns a Dictionary containing all ship-objects and their last registered Status.
`ToString()`	`String`	Returns a String containing information about the harbor.

Simulation

The `Simulation` class instanciates the simulation itself.
It is responsible for the simulation as a whole, and starts and runs the simulation with its `run()`-method.

Constructors

Name

`Simulation(`
`Harbor harbor,`
`DateTime simulationStartTime,`
`DateTime simulationEndTime`
`)`

Description
Initializes a new Simulation of the specified harbor, with the given start and end time.

Parameters	
`Harbor harbor:` The harbor-object to be simulated.	`DateTime simulationStartTime:` The date and time the simulation will start and simulate from.
`DateTime simulationEndTime:` The date and time the simulation will end at.	

Fields

Name	Type	Description
`History`	`ReadOnly Collection<DailyLog>`	Gets a Collection of all the daily logs througout the simulation, represented as `DailyLog` objects.

Methods

Name	Type	Description
`Run()`	`IList <DailyLog>`	Starts the simulation, from the given start time and until the end time given when constructed.
`PrintShipHistory()`	`void`	Prints the history for each ship in the simulation, to console.
`PrintShipHistory(Ship shipToBePrinted)`	`void`	Prints the history of the given ship-object, to console.
`PrintShipHistory(Guid shipID)`	`void`	Prints the history of he ship based of the specified ID., to console.

`PrintContainerHistory()`	`void`	Prints the history of each container in the simulation, to console.
`ToString()`	`String`	Returns a String containing information about the simulation.
`HistoryToString()`	`String`	Returns a String of the History collection, containing the daily logs throughout the simulation.
`HistoryToString(String ShipsOrContainers)`	`String`	Returns a String of the History of either all ships or all containers.
`HistoryToString(Ship ship)`	`String`	Returns a String of the History of the specified ship.
`HistoryToString(Guid shipID)`	`String`	Returns a String of the History of the ship based of the specified ID.

Container

The `Container` class is creating and representing the containers that is used as cargo in the simulation. They are moved between other instances.

Containers are mainly carried by `Ship`. They are stored in `ContainerStorageRows` on the `Harbor`, and moved around to different location on the `Harbor` through `Cranes` and `ADVs`. They are also moved to their final destination out of the `Harbor` with `Trucks`.

Does not provide a constructor, only fields and methods for already existing internal objects.

Fields

Name	Type	Description
`ID`	`Guid`	Gets the unique ID of the container.
`History`	`ReadOnly Collection`	Gets a Collection of all the activities the container has gone through throughout the simulation, as `StatusLog` objects.
`Size`	`Container Size`	Gets the size of the container, Half or Full.
`WeightInTonn`	`Int`	Gets the weight of the container, in tons.
`CurrentPosition`	`Guid`	Gets the unique ID of the containers current position.

Methods

Name	Type	Description
`GetCurrentStatus()`	`Status`	Returns the current Status of the container
`PrintHistory()`	`void`	Prints the History of the container, to console.
`HistoryToString()`	`String`	Returns a String of the History collection, containing all logs throughout the simulation.
`ToString()`	`String`	Returns a String containing information about the container.

StatusLog

The `StatusLog` creates log-objects to the `Ship` and `Container` instances(subjects). It collects information about the instances activities and is saved in their `History` collection.

Does not provide a constructor, only fields and methods for already existing internal objects.

Fields

Name	Type	Description
`Subject`	`Guid`	Gets the unique ID of the StatusLog – the subject.
`SubjectLocation`	`Guid`	Gets the unique ID of the location where the logging took place.
`PointInTime`	`DateTime`	Gets the date and time the status change occured.
`Status`	`Status`	Gets the current Status of the subject.

Methods

Name	Type	Description
`ToString()`	`String`	Returns a String containing information about the statuslog.

DailyLog

The `DailyLog` creates log-objects for the `Simulation` class. It collects information about the current `Status` and situation of the `Simulation`, including every `Ship` and `Containers` location at the given time. It is used in the `Simulation` for logging the end of each day.

Does not provide a constructor, only fields and methods for already existing objects.

Fields

Name	Type	Description
`Time`	`DateTime`	Gets the date and time the log occured.
`ShipsInAnchorage`	`ReadOnly Collection <Ship>`	Gets a Collection containing all the ship objects located in the anchorage at the time of the log.
`ShipsInTransit`	`ReadOnly Collection <Ship>`	Gets a Collection containing all the ship objects in transit at the time of the log.
`ShipsDockedToLoadingDocks`	`ReadOnly Collection <Ship>`	Gets a Collection containing all the ship objects in loading docks at the time of the log.

`ShipsDockedToShipDocks`	`ReadOnly Collection <Ship>`	Gets a Collection containing all the ship objects in ship docks at the time of the log.
`ContainersInHarbour`	`ReadOnly Collection <Container>`	Gets a Collection containing all the Container objects stored in the Harbor's storage area, at the time of the log.
`ContainersArrivedAtDestination`	`ReadOnly Collection <Container>`	Gets a Collection containing all the Container objects that have arrived to their destination, at the time of the log.

Methods

Name	Type	Description
`PrintInfoForAllShips()`	`void`	Prints the whereabouts and information for all the ships in the daily log, to the console.
`PrintInfoForAllContainers()`	`void`	Prints the whereabouts and information for all the containers in the daily log, to the console.
`HistoryToString()`	`String`	Returns a String containing information about all ships and their whereabouts, in the daily log.
`HistoryToString(String ShipsOrContainers)`	`String`	Returns a String of the containing information about either all ships or all containers and their whereabouts, in the daily log.
`ToString()`	`String`	Returns a String containing information about the daily log.

Enums

ContainerSize

The `ContainerSize` enum is used to assign each `Container` apon creation with a specific size and weight.

Enum Constants

Name	Weight(ton)	Description
`None`	`0`	No size given. Default value when no value selected.
`Half`	`10`	Half-size container, weighing 10 tons.
`Full`	`20`	Full-size container, weighing 20 tons.

ShipSize

The `ShipSize` enum is used to assign each `Ship` apon creation with a specific size, that will determine the correct base information tio be set for the ship and which docks it can use.

Enum Constants

Name	Description
`None`	No size given. Default value when no value selected.
`Small`	Small ship. Base weight: 5000 tons, container capacity: 20
`Medium`	Medium ship. Base weight: 50 000 tons, container capacity: 50
`Large`	Large ship. Base weight: 10 000 tons, container capacity: 100

Status

The `Status` enum is used to set and update the current `Status` for `Ship`, `Container` and `Truck` objects. The status is stored in `StatusLog`-objects that is then saved in their respective `History` collections.

Enum Constants

Name	Description
`None`	No size given. Default value when no value selected.
`Anchoring`	Ship is anchoring.
`Anchored`	Ship is in anchorage.
`DockingToLoadingDock`	Ship is docking to a free loading dock
`DockingToShipDock`	Ship is docking to a free ship dock.
`DockedToLoadingDock`	Ship is docked to a loading dock.
`DockedToShipDock`	Ship is docked to a ship dock.
`Unloading`	Ship is unloading containers.

`UnloadingDone`	Ship is done unloading containers.
`Loading`	Ship is loading containers.
`LoadingDone`	Ship is done loading containers.
`Undocking`	Ship is undocking from harbor.
`Transit`	Ship: is in transit on sea. Container: is in transit on ship or truck.
`InStorage`	Container is stored in harbor storage.
`LoadingToCrane`	Container is being unloaded by ship and loaded onto crane.
`UnloadingFromCraneToShip`	Container is being unloaded by crane and loaded on ship
`LoadingTruck`	Container is being loaded onto trcuk.
`LoadingToAdv`	Container is being loaded onto ADV.
`Queuing`	Truck is queuing.
`ArrivedAtDestination`	Container has arrived at destination.