HarbNet - API dokumentasjon

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

Contents

Introduction

Welcome to HarbNet framework!

HarbNet is a simulation framework that lets you simulate activities connected to a container harbor.

The harbor contains of ships, trucks, AGVs, containers, storage space, two different dock types, anchorage and a logging system!

There are many parts a harbor is built up of, but HarbNet focuses on simplifying and getting you started as quickly as possible!

Getting started

There are only four objects you need to create from the ground up yourself: ships, storage rows, the harbor and the simulation itself! Everything else is internalized, with customiziation from you, the user.

Alternatively, you can use our simplified "plug-and-play" harbor-constructor, where you don't need to create ship or storage rows yourself!
Feel free to skip to and read more about this constructor under step 3.

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.

There are three types of ship sizes:

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"** of size **Small**.
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"** of size **Large**.
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 our 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));
}
```

3. 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 objects.
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 adding 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 area 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** AGVs that each can do **15** loads/moves per hour.

Alternatively, our "plug-and-play" setup:

This is our simplified harbor-constructor, that gets you up and running with only a few configurations! This **does not** need a list of ship or a list of storage rows.

```
Harbor harbor = new(3, 15, 10, 3, 5, 4, 10);
```

This constructor takes in the number of ships the harbor simulates (**3**), the number of rows the harbor container storage contains of (**15**) and the capacity of each of these rows (**10**).

It then takes how many loading docks the harbor has (**3**), how many cranes the loading dock area has (**5**) and how many cranes the storage area has (**10**).
Lastly it takes the number of AGVs the harbor has (**10**).

The constructor also has a set of optional parameters, that includes moves per crane per hour, trucks arriving per hour, percentage of containers directly loaded from ship to trucks, and so on.

See the documentation of `Harbor` constructors for more information about each of these parameters.

4. We can now make the `SimpleSimulation`!

Creating the simulation itself is a one-step-only process. We simply initialize a `SimpleSimulation` object and send in the harbor we want to simulate and the time period we like it to run for:

```
DateTime startTime = new(2024,04,15,10,0);
DateTime endTime = new(2024,06,15,10,0),

SimpleSimulation simulation = new(harbor, startTime, endTime);
```

To start the simulation, we simply use the Run() method that comes with the Simulation class:

```
simulation.Run();
```

And that is the basics of using the HarbNet framework to create a harbor simulation!

Continue to “**Advancing on**” to see how we can take advantage of the multiple Events the simulation is signalling!

Advancing on – Utilize and respond to activities in the simulation

Now that we have our simulation set up and runnable, we can continue on with how we can respond to what is taking place in the simulation!

The framework has multiple events that is “subscribable”. When our program has subscribed to an Event, the event will send a signal when certain activities or, “events”, have taken place in the simulation. The event has been *raised*. With these signals, it will also be included a set of *Args*, that contains relevant information regarding the event.

How we respond and use these signals and information we are sent, is completely up to us.

Here is an example - continuing on our setup from “**Getting Started**”:

```
simulation.Run();

simulation.ShipAnchored += ShipAnchoredHandler;
```

```
// Here we want to keep a list of all the dates and times ships
   Anchored in our harbor
List<DateTime> anchoredDates = new();

static void ShipAnchoredHandler(object sender,
ShipAnchoredEventArgs e)
{
  Console.WriteLine($"{e.Ship.name} has anchored at
                    {e.CurrentTime}")

  // The event has been raised, so we add the time to our list!
  anchoredDates.Add(e.CurrentTime);

  Console.WriteLine(countAnchored);
}
```

Here we are subscribing to the event `ShipAnchored`. In our Handler we have decided to print the statement *“{args.Ship.name} has anchored at {args.CurrentTime}”*.
The output would look something like this:

```
Brief Breeze has anchored at 2024-04-15 10:30:00
```

We’re also making use and including the args and events in our own internal code setup – in this example this is our `anchoredDates` list! We want this list to contain all the dates when a ship anchored to the anchorage.

We use the args that came with the event, to retrieve relevant and useful information.

This is just a small example and look into what can be done with the Events!
Refer to the **Events** documentation for all the currently available events.

Classes

Ship

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

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 `Ship`s 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 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.

Examples of use

The examples take the setup from the "Getting Started" section as a foundation. briefBreeze is the Ship object created in that section!

```
Guid shipID = briefBreeze.ID;
// 8962ff9a-41a5-43eb-9a3a-2c8880ea389d

ShipSize shipSize = briefBreeze.ShipSize; // ex. Small

String shipName = briefBreeze.Name; // ex. Brief Breeze

DateTime shipStart = briefBreeze.StartDate; // 15.04.2024 10:00:00

int roundTripInDays = briefBreeze.RoundTripInDays; // 5

Guid shipLocation = briefBreeze.CurrentLocation;
// ex. 6e9a20ed-3f91-463f-be60-78d61f7c04f1

ReadOnlyCollection<StatusLog> shipHistory = briefBreeze.History;
// Collection with StatusLog-objects.
// ex. Collection:
```

```
// [
    {
      "Date”: “07.03.2024 08:00:00”, “Subject ID”: “6232e63e-e82d-
      4aa1-b48d-0be3e1b8714e”, “Location”: “1ca8f8ff-2c62-48cb-
      b436-76da730e9261”, “Status”: “Anchoring”
    },
    {
      “Date”: '07.03.2024 08:00:00', “Subject ID”: “6232e63e-e82d-
      4aa1-b48d-0be3e1b8714e”, “Location”: “1ca8f8ff-2c62-48cb-
      b436-76da730e9261”, “Status”: “Anchored”
    },
    {
      “Date”: “10.03.2024 17:00:00”, “Subject ID”: “6232e63e-e82d-
      4aa1-b48d-0be3e1b8714e”, “Location”: “74adeec2-96e1-46de-
      83cb-8c57fa308426”, “Status”: “DockingToLoadingDock”
    },
   ]

List<Container> shipContainers = briefBreeze.ContainersOnBoard;
// IList with Container-objects.
// ex. IList:
// [
    {
      “ID”: “fa20dc8d-984c-4741-851b-e354ed8331ac”,
      “Size”: “Full”, “Weight”: “20 tonnes”
    },
    {
      “ID”: “98c66485-bfc3-41e8-a494-a9a15cfa632d”,
      “Size”: “Half”, “Weight”: “10 tonnes”
    }
   ]

int containerCapacity = briefBreeze.ContainerCapacity; // ex. 20

int maxWeight = briefBreeze.MaxWeightInTonn; // ex. 5600

int baseWeight = briefBreeze.BaseWeightIntonn; // ex. 5000

int currentWeight = briefBreeze.CurrentWeightInTonn; // ex. 5400

briefBreeze.PrintHistory();
```

```
// ex output:
// Ship name: Brief Breeze, Ship ID 8962ff9a-41a5-43eb-9a3a-
   2c8880ea389d
// ------------------------------------
// Date: 06.03.2024 08:00:00 Status: Anchoring|
// Date: 06.03.2024 08:00:00 Status: Anchored|
// Date: 10.03.2024 18:00:00 Status: DockingToLoadingDock|
// Date: 10.03.2024 19:00:00 Status: DockedToLoadingDock|
// Date: 10.03.2024 20:00:00 Status: Unloading|
// Date: 11.03.2024 05:00:00 Status: UnloadingDone|
// Date: 11.03.2024 05:00:00 Status: Loading|
// Date: 11.03.2024 11:00:00 Status: LoadingDone|
// Date: 11.03.2024 11:00:00 Status: Undocking|
// Date: 11.03.2024 12:00:00 Status: Transit|
// Date: 14.03.2024 12:00:00 Status: Anchoring|
// Date: 14.03.2024 13:00:00 Status: Anchored|

String briefBreezeHistoryString = briefBreeze.HistoryToString();
// Same format and output as PrintHistory, but as a String object!
```

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 containers 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 `ContainerStorageRows` 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.

Examples of use

The examples take the setup from the "Getting Started" section as a foundation. storageArea is the List containing ContainerStorageRows created in that section!

```
ContainerStorageRow storageRow = storageArea[0];
// We're getting the first storageRow object of our storage area.

Guid storageRowID = storageRow.ID;
// ex. 38de77e4-1c94-4097-ae95-69089175cdeb

int freeSpaces = storageRow.NumberOfFreeContainerSpaces();
// ex. returns 5, meaning there are 5 unoccupied spaces in the
   first container storage row of our storageArea.

ContainerSize storableSize = storageRow.SizeOfContainersStored();
// ex. returns ContainerSize.Half, meaning this row can store
   Half-sized containers.

List<Guid> storedContainers =
storageRow.GetIDOfAllStoredContainers();
// returns an Ilist of Guids that we are collecting to a List.

Guid firstStoredContainer = storedContainers[0]);
// Gets the first container in the harbor storage area.
// ex. : f47ac10b-58cc-4372-a567-0e02b2c3d479
```

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 numberOfAgv, int loadsPerAgvPerHour
)
```

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 numberOfAgv`: Number of AGVs in harbor. AGVs move containers between loading dock cranes and storage area cranes.	`Int loadsPerAgvPerHour`: Number of loads (containers) each AGV is capable of per hour.

Name

```
Harbor(
int numberOfShips,
int numberOfHarborContainerStorageRows,
int containerStorageCapacityInEachStorageRow,
int numberOfLoadingDocks,
int numberOfCranesNextToLoadingDocks,
int numberOfCranesOnHarborStorageArea,
int numberOfAgvs,
int loadsPerCranePerHour = 35,
int trucksArrivePerHour = 10,
int loadsPerAgvPerHour = 25,
int percentageOfContainersDirectlyLoadedFromShipToTrucks = 10,
int percentageOfContainersDirectlyLoadedFromHarborStorageToTrucks = 15
)
```

Description

Initializes a new Harbor-object with the specified details. Alternative constructor with simplified parameters and default values.

Parameter description

`Int numberOfShips:` Number of ships to be created. Creates an even number of small, medium and large ships. If uneven number, small and medium is prioritized.	`Int numberOfHarborContainerStorageRows`: Number of rows the harbor container storage area contains of.
`Int containerStorageCapacityInEachStorageRow`: The container capacity of each row in the harbor container storage area.	`Int numberOfLoadingDocks`: Number of loading docks in the harbor, where Ships will unload and load their cargo. Creates an even number of small, medium and large loading docks. If an uneven number, small and medium is prioritized.

`Int numberOfCranesNextToLoadingDocks`: Number of cranes located at the dock area. To be used for unloading and loading cargo from and to ships.	`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 numberOfAgvs`: Number of AGVs in harbor. AGVs move containers between loading dock cranes and storage area cranes.	
Optional parameters	
`Int loadsPerCranePerHour`: Number of loads (containers) each AGV is capable of per hour. Default value of 35.	`Int numberOfTrucksArriveToHarborPerHour`: Number of trucks to arrive and be placed in "truck queue" at the harbor each hour. Default value of 10.
`Int loadsPerAgvPerHour`: Number of loads (containers) each AGV is capable of per hour. Default value of 25.	`Int percentageOfContainersDirectlyLoadedFromShipToTrucks`: Percentage of containers loaded onto trucks from ship. The rest is stored in storage area. Default value of 10.
`Int percentageOfContainersDirectlyLoadedFromHarborStorageToTrucks`: Percentage of containers loaded onto trucks from storage area. The rest is loaded onto ships. Default value of 15.	

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.
`AgvCargoID`	`Guid`	Gets the unique ID of the location the cargo is, when placed on an AGV.
`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
`GetAvailabilityStatusForAllLoadingDocks()`	`IDictionary<Guid, bool>`	Returns a Dictionary containing the IDs of all the loading docks and their availability.
`GetAvailabilityStatusForAllShipDocks()`	`IDictionary<Guid, bool>`	Returns a Dictionary containing the IDs of all the ship docks and their availability.
`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.

Examples of use

The examples take the setup from the “Getting Started” section as a foundation. harbor is the Harbor object and briefBreeze is the Ship object created in that section!

```
Guid harborID = harbor.ID;
// ex. c81746c2-5214-4c8c-850f-8496bd8aaa17
List<Container> harbor.ArrivedAtDestination;
// IList with Container-objects.
// ex. IList:
// [
    {
      “ID”: “a1f39a12-b741-4e90-8f63-5b65b13e5c42”,
      “Size”: “Full”, “Weight”: “20 tonnes”
    },
    {
      “ID”: “f50e2b64-d3e4-4e7f-a7bc-2e2b9e0958ff”,
      “Size”: “Half”, “Weight”: “10 tonnes”
    },
   ]
Guid transitID = harbor.TransitLocationID;
// ex. bd684b1e-8c13-4f4d-99b3-8704f9b8f5a4
```

```
Guid anchorageID = harbor.AnchorageID;
// ex. c65a1a1a-62e5-488b-927f-8b58c63a8cb6
Guid agvLocationID = harbor.AgvCargoID;
// ex. 5e1d3436-5c3c-4a9c-bb55-7ef89f312e71
Guid truckInTransitID = harbor.TruckTransitLocationID;
// ex. d6e2a5a7-0c45-4190-bf3c-f7c1fcdb6b19
Guid truckQueueLocationID = harbor.TruckQueueLocationID;
// ex. d13d6e9f-5d57-4a59-85fc-c2d91f69d6d5
Guid storageAreaID = harbor.HarborStorageAreaID;
// ex. 5f4993d7-8f5e-42f4-8a7a-6e789ff1a6d7
Guid dockAreaID = harbor.HarborDockAreaID;
// ex. 759843b3-f023-4c76-9e91-1a567d49b2e2
Guid destinationID = harbor.DestinationID;
// ex. 4cbe1f6e-7606-46d6-89d2-31b1d8d98a9a

Dictionary<Guid, bool> loadingDockStatuses =
harbor.GetAvailabilityStatusForAllLoadingDocks();
// Collecting a dictionary containing all the Guids of the harbors
   loading dock, paired with their availability status.
// ex. Dictionary<Guid, bool>:
// {
     “a1f39a12-b741-4e90-8f63-5b65b13e5c42”: true,
     “f50e2b64-d3e4-4e7f-a7bc-2e2b9e0958ff”: false
   }

List<Guid> availableLoadingDocks = new();

foreach (var pair in loadingDockStatuses)
{
  if (pair.Value == true)
  availableLoadingDocks.Add(pair.Key);
}

int availableCount = availableLoadingDocks.Count;
// Counting the number of available loading docks.

// Same logic for GetAvailabilityStatusForAllShipDocks() but only
   for ship docks!

Guid shipID = briefBreeze.ID;
```

```
Status shipStatus = harbor.GetShipStatus(shipID);
// ex. Unloading

Dictionary shipStatuses = harbor.GetStatusAllShips();
// ex. Dictionary<Guid, bool>:
// {
     “8962ff9a-41a5-43eb-9a3a-2c8880ea389d”: “Anchored”,
     “1eac7dd0-df85-49f8-8f51-8736f9ca7da8”: “Unloading”
   }

Status currentShipStatus = shipStatuses[briefBreeze];
// ex. Transit
```

SimpleSimulation

The `SimpleSimulation` 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

```
SimpleSimulation(
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.

Examples of use

The examples take the setup from the "Getting Started" section as a foundation. simulation is the SimpleSimulation object and briefBreeze is the Ship object created in that section!

```
ReadOnlyCollection<DailyLog> history = simulation.History;
// Collection of DailyLog-objects.
// ex. Collection:
// [
    {
      "Time": "01.03.2024 08:00:00", "Ships in anchorage": 0,
      "Ships in loading docks": 0, "Ships in ship dock": 0,
      "Ships in transit": 0, "Containers in harbor": 0,
      "Containers arrived at their destination": 0
    },
    {
      "Time": "02.03.2024 00:00:00", "Ships in anchorage": 3,
      "Ships in loading docks": 1, "Ships in ship dock": 0,
      "Ships in transit": 12, "Containers in harbor": 1,
      "Containers arrived at their destination": 16
    },
    {
      "Time": "03.03.2024 00:00:00", "Ships in anchorage": 11,
      "Ships in loading docks": 3, "Ships in ship dock": 0,
      "Ships in transit": 18, "Containers in harbor": 52,
      "Containers arrived at their destination": 119
    }
   ]

simulation.Run() // Starts and runs the simulation
simulation.PrintShipHistory();
```

```
// ex output:
// ---------------------------------
// DATE:01.03.2024 08:00:00
// ---------------------------------
//
// NO SHIPS IN ANCHORAGE
//
// NO SHIPS IN TRANSIT
//
// NO SHIPS DOCKED IN LOADING DOCKS
//
// NO SHIPS DOCKED IN SHIP DOCKS
//
// ---------------------------------
// DATE:02.03.2024 00:00:00
// ---------------------------------
// SHIPS IN ANCHORAGE:
// NAME: The Flying Dutchman, SIZE: Large, STATUS: Anchored, MAX
   WEIGHT: 103600tonns , CURRENT WEIGHT: 100500 tonns, CONTAINER
   CAPACITY: 100, CONTAINERS ONBOARD: 34, ID: 1eac7dd0-df85-49f8-
   8f51-8736f9ca7da8
//
// NO SHIPS IN TRANSIT
//
// SHIPS IN LOADING DOCK:
// NAME: Brief Breeze, SIZE: Small, STATUS: Unloading, MAX WEIGHT:
   5600tonns, CURRENT WEIGHT: 5010 tonns, CONTAINER CAPACITY: 20,
   CONTAINERS ONBOARD: 1, ID: 8962ff9a-41a5-43eb-9a3a-2c8880ea389d
//
// SHIPS IN SHIP DOCK:
//

simulation.PrintShipHistory(briefBreeze);
// ex output:
// Ship name: Brief Breeze, Ship ID 8962ff9a-41a5-43eb-9a3a-
2c8880ea389d
// ------------------------------------
// Date: 06.03.2024 08:00:00 Status: Anchoring|
//
// Date: 06.03.2024 08:00:00 Status: Anchored|
//
```

```
// Date: 10.03.2024 19:00:00 Status: DockedToLoadingDock|
//
// Date: 10.03.2024 20:00:00 Status: Unloading|
//
// Date: 11.03.2024 05:00:00 Status: UnloadingDone|
//
// Date: 11.03.2024 05:00:00 Status: Loading|
//
// Date: 11.03.2024 11:00:00 Status: LoadingDone|
//
// Date: 11.03.2024 11:00:00 Status: Undocking|
//
// Date: 11.03.2024 12:00:00 Status: Transit|
//
// Date: 14.03.2024 12:00:00 Status: Anchoring|
//
// Date: 14.03.2024 13:00:00 Status: Anchored|

simulation.PrintShipHistory(Guid.Parse(“8962ff9a-41a5-43eb-9a3a-
                                        2c8880ea389d”));
// ex output:
// Ship name: Brief Breeze, Ship ID 8962ff9a-41a5-43eb-9a3a-
   2c8880ea389d
// ------------------------------------
// Date: 06.03.2024 08:00:00 Status: Anchoring|
// Date: 06.03.2024 08:00:00 Status: Anchored|
// Date: 10.03.2024 19:00:00 Status: DockedToLoadingDock|
// Date: 10.03.2024 20:00:00 Status: Unloading|
// Date: 11.03.2024 05:00:00 Status: UnloadingDone|
// Date: 11.03.2024 05:00:00 Status: Loading|
// Date: 11.03.2024 11:00:00 Status: LoadingDone|
// Date: 11.03.2024 11:00:00 Status: Undocking|
// Date: 11.03.2024 12:00:00 Status: Transit|
// Date: 14.03.2024 12:00:00 Status: Anchoring|
// Date: 14.03.2024 13:00:00 Status: Anchored|

simulation.PrintContainerHistory();
// ex. output:
// ---------------------------------
// DATE:01.03.2024 00:00:00
// ---------------------------------
// NO CONTAINERS ONBOARD SHIPS IN ANCHORAGE
//
// NO CONTAINERS ONBOARD SHIPS IN TRANSIT
//
// NO CONTAINERS ONBOARD SHIPS IN LOADING DOCKS
//
// MO CONTAINERS ARRIVED TO THEIR DESTINATION

// ---------------------------------
// DATE:02.03.2024 00:00:00
// ---------------------------------
// NO CONTAINERS ONBOARD SHIPS IN ANCHORAGE
//
// CONTAINERS ONBOARD SHIPS IN TRANSIT:
// SHIP NAME: The Flying Dutchman, SHIP ID: e01969f2-7646-488e-
   8cae-bf521a941a16
// CONTAINER SIZE: HALF, WEIGHT: 10tonns, STATUS: Transit, ID:
   858f1f77-7dd6-4fcb-95fe-7ef458359b94
//
// CONTAINERS ONBOARD SHIPS IN LOADING DOCKS:
// SHIP NAME: Brief Breeze, SHIP ID: 8962ff9a-41a5-43eb-9a3a-
   2c8880ea389d
// CONTAINER SIZE: HALF, WEIGHT: 10tonns, STATUS: Transit, ID:
   8cf03438-66e4-4934-a797-78c8792fa289
//
// CONTAINERS IN HARBOR STORAGE:
// CONTAINER SIZE: Full, WEIGHT: 20tonns, STATUS: InStorage, ID:
   dbd50443-e1e7-4ac8-bb68-254c59d49d34
//
// CONTAINERS ARRIVED AT THEIR DESTINATION:
// CONTAINER SIZE: Full, WEIGHT: 20tonns, STATUS:
   ArrivedAtDestination, ID: 758k6f2c-7h4f-kl6f-84cx-24f458sf9nh4

String completeHistoryString = simulation.HistoryToString();
// Alternatively HistoryToString(“ships”);

String completeContainerHistoryString =
simulation.HistorytoString(“containers”);
// Same format and output as PrintContainerHistory, but as a
   String object!
String completeShipHistoryString =
simulation.HistoryToString(briefBreeze);
// Alternatively HistoryToString(Guid.Parse(“8962ff9a-41a5-43eb-
                                                     9a3a-2c8880ea389d”));
```

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 AGVs. 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.
`CurrentLocation`	`Guid`	Gets the unique ID of the containers current location.

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.

Examples of use

The examples take the setup from the “Getting Started” section as a foundation. briefBreeze is the Ship object created in that section!

```
Guid containerID = container.ID;
// ex. aa367649-99f0-45d7-ba02-e1741b5435ec
ReadOnlyCollection<StatusLog> history = container.History;
// Collection of StatusLog-objects.
// ex. Collection:
// [
    {
      "Date”: “06.03.2024 11:00:00”, “Subject ID”: “82388567-0982-
       4fd7-9a43-5db6c08118dc”, “Location”: “70b791c8-7262-4a2f-
       9980-7d290d286a11”, “Status”: “Transit”
    },
    {
      "Date”: “20.03.2024 09:00:00”, “Subject ID”: “82388567-0982-
       4fd7-9a43-5db6c08118dc”, “Location”: “0fa01228-3235-47c3-
       b3b3-98162453dffc”, “Status”: “LoadingToCrane”
    },
    {
      "Date”: “20.03.2024 09:00:00”, “Subject ID”: “82388567-0982-
       4fd7-9a43-5db6c08118dc”, “Location”: “67bd722f-7509-4fa4-
       bc31-69fe684a4154”, “Status”: “LoadingToAgv”
    }
    {
      "Date”: “20.03.2024 09:00:00”, “Subject ID”: “82388567-0982-
       4fd7-9a43-5db6c08118dc”, “Location”: “67bd722f-7509-4fa4-
       bc31-69fe684a4154”, “Status”: “LoadingToCrane”
    }
    {
      "Date”: “20.03.2024 09:00:00”, “Subject ID”: “82388567-0982-
       4fd7-9a43-5db6c08118dc”, “Location”: “67bd722f-7509-4fa4-
       bc31-69fe684a4154”, “Status”: “InStorage”
    }
   ]

ContainerSize containerSize = container.Size; // ex. Half

int containerWeight = container.WeightInTonn; // ex. 10

Guid currentLocation = container.CurrentLocation;
// ex. 5f4993d7-8f5e-42f4-8a7a-6e789ff1a6d7

Container container = briefBreeze.ContainersOnBoard[0];
// Getting the first container onboard the ship Brief Breeze.

Status currentStatus = container.GetCurrentStatus();
// ex. Transit
```

```
container.PrintHistory();
// ex. output:
// Container ID: aa367649-99f0-45d7-ba02-e1741b5435ec
// Date: 08.03.2024 09:00:00 Status: InStorage
```

Truck

The `Truck` class creates and represents individual trucks that is to be simulated in the Simulation.

Trucks are used by the `Harbor` to transport `Container`s out by land, either directly from the incoming ships, or picked up from the storage area. The percentage of containers that is to be picked up by trucks, both from ships and from storage, is to be specified in the
`Harbor` constructor.

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 Truck.
`Location`	`Guid`	Gets the ID of the trucks location.
`Status`	`Status`	Gets the current status of the truck.
`Container`	`Container`	Gets the container in the vehicles storage.

Methods

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

Examples of use

The examples take the setup from the "Getting Started" section as a starting point. simulation is the Simulation object created in that section!

```
// We can for example get a Truck object through one of the
events:

simulation.TruckLoadingFromStorage += GetTruckLoadingFromStorage;
```

```
Truck? truck = null;
void GetTruckLoadingFromStorage(object? sender,
TruckLoadingFromStorageEventArgs e)
{
  truck = e.Truck;
}

Guid truckID = truck.ID;
// ex. 4014d5cb-2b70-4ba2-b4b3-68f4ba1a2fb3

Guid truckLocation = truck.Location;
// ex. c81746c2-5214-4c8c-850f-8496bd8aaa17

Status truckStatus = truck.Status; // ex. Queuing

Container container = truck.Container; // Container object
```

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.
`Timestamp`	`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.

Examples of use

The examples take the setup from the "Getting Started" section as a foundation. briefBreeze is the Ship object created in that section!

```
StatusLog shipFirstLog = briefBreeze.History[1];
// Gets the first registered StatusLog-object in the ships History

Guid logID = shipFirstLog.Subject;
// ex. c66e67e2-19d1-4325-a0de-05be7f8d2398

Guid logLocation = shipFirstLog.SubjectLocation;
// ex. 1ca8f8ff-2c62-48cb-b436-76da730e9261

DateTime logTime = shipFirstLog.Timestamp;
// ex. 07.03.2024 08:00:00

Status logStatus = shipFirstLog.Status;
// ex. Anchored
```

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
`Timestamp`	`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.

Examples of use

The examples take the setup from the "Getting Started" section as a foundation. simulation is the SimpleSimulation object created in the earlier section!

```
DailyLog intitialLog = simulation.History[0];
// Gets the initial DailyLog-object of the simulation

DailyLog dayOneLog = simulation.History[1];
// Gets the DailyLog-object of the first day of the simulation

DateTime timeOfLog = dayOne.Timestamp; // ex. 02.03.2024 00:00:00
ReadOnlyColleciton<Ship> shipsInAnchorage =
dayOne.ShipsInAnchorage;
// Collection of Ship-objects
// ex. Collection:
```

```
// [
     {
       “ID”: “1eac7dd0-df85-49f8-8f51-8736f9ca7da8”,
       “Name”: “The Flying Dutchman”, “Size”: “Large”,
       “Start date”: “15.04.2024 10:00:00”,
       “Round trip time”: “5 days”,
       “Containers on board”: “0 small, 2 large”, ,
       “Base weight”: “100000 tonnes”,
       “Current weight”: 100500 tonns,
       “Max weight”: “103600 tonns”,
    }
   ]

ReadOnlyCollection<Ship> shipsInTransit = dayOne.ShipsInTransit;
// Collection of Ship-objects.

ReadOnlyCollection<Ship> shipsDockedToLoadingDock =
dayOne.ShipsDockedToLoadingDocks;
// Collection of Ship-objects.

ReadOnlyCollection<Ship> shipsDockedToShipDocks =
dayOne.ShipsDockedToShipDocks;
// Collection of Ship-objects.

ReadOnlyCollection<Container> containersInHarbor =
dayOne.ContainersInHarbour;
// Collection of Ship-objects.

ReadOnlyCollection<Container> containersArrivedAtDestination =
dayOne.ContainersArrivedAtDestination;
// Collection of Ship-objects.

initialLog.PrintInfoForAllShips();
// ex. output:
// ---------------------------------
// DATE:01.03.2024 08:00:00
// ---------------------------------
//
// NO SHIPS IN ANCHORAGE
//
// NO SHIPS IN TRANSIT
```

```
//
// NO SHIPS DOCKED IN LOADING DOCKS
//
// NO SHIPS DOCKED IN SHIP DOCKS

dayOneLog.PrintInfoForAllShips();
// ex. output:
// ---------------------------------
// DATE:02.03.2024 00:00:00
// ---------------------------------
// SHIPS IN ANCHORAGE:
// NAME: The Flying Dutchman, SIZE: Large, STATUS: Anchored, MAX
   WEIGHT: 103600tonns , CURRENT WEIGHT: 100500 tonns, CONTAINER
   CAPACITY: 100, CONTAINERS ONBOARD: 34, ID: 1eac7dd0-df85-49f8-
   8f51-8736f9ca7da8
//
// NO SHIPS IN TRANSIT
//
// SHIPS IN LOADING DOCKS:
// NAME: Brief Breeze, SIZE: Small, STATUS: Unloading, MAX WEIGHT:
   5600tonns , CURRENT WEIGHT: 5010 tonns, CONTAINER CAPACITY: 20,
   CONTAINERS ONBOARD: 1, ID: 8962ff9a-41a5-43eb-9a3a-2c8880ea389d
//
// NO SHIPS DOCKED IN SHIP DOCKS

initialLog.PrintInfoForAllContainers();
// ex. output:
// ---------------------------------
// DATE:01.03.2024 00:00:00
// ---------------------------------
// NO CONTAINERS ONBOARD SHIPS IN ANCHORAGE
//
// NO CONTAINERS ONBOARD SHIPS IN TRANSIT
//
// NO CONTAINERS ONBOARD SHIPS IN LOADING DOCKS
//
// NO CONTAINERS ARRIVED TO THEIR DESTINATION

dayOneLog.PrintInforForAllContainers();
// ex. output:
// ---------------------------------
// DATE:02.03.2024 00:00:00
// ---------------------------------
// CONTAINERS ONBOARD SHIPS IN ANCHORAGE:
// NONE
//
// CONTAINERS ONBOARD SHIPS IN TRANSIT:
// SHIP NAME: The Flying Dutchman, SHIP ID: e01969f2-7646-488e-
   8cae-bf521a941a16
// CONTAINER SIZE: HALF, WEIGHT: 10tonns, STATUS: Transit, ID:
   858f1f77-7dd6-4fcb-95fe-7ef458359b94
//
// CONTAINERS ONBOARD SHIPS IN LOADING DOCKS:
// SHIP NAME: Brief Breeze, SHIP ID: 8962ff9a-41a5-43eb-9a3a-
   2c8880ea389d
// CONTAINER SIZE: HALF, WEIGHT: 10tonns, STATUS: Transit, ID:
   8cf03438-66e4-4934-a797-78c8792fa289
//
// CONTAINERS IN HARBOR STORAGE:
// CONTAINER SIZE: Full, WEIGHT: 20tonns, STATUS: InStorage, ID:
   dbd50443-e1e7-4ac8-bb68-254c59d49d34
//
// CONTAINERS ARRIVED AT THEIR DESTINATION:
// CONTAINER SIZE: Full, WEIGHT: 20tonns, STATUS:
   ArrivedAtDestination, ID: 758k6f2c-7h4f-kl6f-84cx-24f458sf9nh4

String dayOneLogString = dayOneLog.HistoryToString();
// Alternatively HistoryToString(“ships”);

String dayOneLogContainersString =
dayOneLog.HistoryToString(“containers”);
// Same format and output as PrintInfoForAllContainers, but as a
   String!
```

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 `Truck` -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.
`LoadingToAgv`	Container is being loaded onto AGV.
`Queuing`	Truck is queuing.
`ArrivedAtDestination`	Container has arrived at destination.

Events & EventArgs

Each event has a `Description` field. This field contains a short description of what the event is raised by. This is a helpful field, if there is unclearity of why the event is raised, past the name of the event and the Args set themselves.

Examples of use

The events can be used in a variety of ways.
Here follows a few quick and easy examples – let your creativity loose!

The examples take the setup from the "Getting Started" section as a foundation. simulation is the SimpleSimulation object created in that section!

Console printing, including arguments from the EventArgs class:

```
simulation. ShipAnchored += OnShipAnchored;

static void OnShipAnchored(object? sender,
                           ShipAnchoredEventArgs e)
{
  Console.WriteLine($"{e.Ship.Name} anchored at {e.CurrentTime}");
}
```

Local counter – keeping local track of how many times the event has been raised:

```
int numberOfAnchorings = 0;

simulation.ShipAnchoring += OnShipAnchoring;

void OnShipAnchoring(object? sender, ShipAnchoringEventArgs e)
{
  numberOfAnchorings++;
}
```

List of received objects – Keeping a local list for further use:

```
List<Container> containersOutForDeliveryByShip = new();

simulation.ShipLoadedContainer += OnShipLoadedContainer;

void OnShipLoadedContainer (object? sender,
                            ShipLoadedContainerEventArgs e)
{
  containersOutForDeliveryByShip.Add(e.Container);
}
```

Refer to "**Advancing on**" for a more in-depth example of how event-handling with EventArgs works!

SimulationStarting

This event is raised when the `SimpleSimulation` is starting.

SimulationStartingEventArgs

Fields

`Harbor HarborToBeSimulated` The harbor object that is being simulated.	`DateTime StartDate` The time the simulation is starting from.
`String Description` A description of the event.	

SimulationEnded

This event is raised when the `SimpleSimulation` has ended.

SimulationEndedEventArgs

Fields

`ReadOnlyCollection<DailyLog> SimulationHistory` A collection of DailyLog objects that together represent the history of the simulation.	`String Description` A description of the event.

OneHourHasPassed

This event is raised when one hour has passed in the `SimpleSimulation`.

OneHourHasPassedEventArgs

Fields

`DateTime CurrentTime` The time in the simulation the event was raised.	`String Description` A description of the event.

DayEnded

This event is raised when a day has ended in the `SimpleSimulation`.

DayOverEventArgs

Fields

`DailyLog TodaysLog` A DailyLog object containing information about the previous day in the simulation.	`DateTime CurrentTime` The time in the simulation the event was raised.
`Dictionary<Ship, List<StatusLog>> DayReviewAllShipLogs` A dictionary with Ship-List pairs, where List is all the StatusLog-objects for the ship the previous day	`String Description` A description of the event.

ShipAnchoring

This event is raised when a `Ship` is starting anchoring to the `Harbor`s anchorage.

ShipAnchoringEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid AnchorageID` The unique ID of the anchorage.

ShipAnchored

This event is raised when a `Ship` has anchored to the `Harbor`s anchorage.

ShipAnchoredEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid AnchorageID` The unique ID of the anchorage.

ShipDockingToShipDock

This event is raised when a `Ship` is starting docking to a ship dock in the `Harbor`.

ShipDockingToShipDockEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship is docking to.

ShipDockedToShipDock

This event is raised when a `Ship` has docked to a ship dock in the `Harbor`.

ShipDockedToShipDock

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship docked to.

ShipDockingToLoadingDock

This event is raised when a `Ship` is starting docking to a loading dock in the `Harbor`.

ShipDockingToLoadingDockEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship is docking to.

ShipDockedToLoadingDock

This event is raised when a `Ship` has docked to a loading dock in the `Harbor`.

ShipDockedToLoadingDockEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship docked to.

ShipStartingLoading

This event is raised when a `Ship` is starting the loading process of `Containers`.

ShipStartingLoadingEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship is located at and loading from.

ShipLoadedContainer

This event is raised when a `Ship` has loaded one (1) `Container`.

ShipLoadedContainerEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Container Container` The container loaded onboard the ship.

ShipDoneLoading

This event is raised when a `Ship` is done with the loading process of `Container`s.

ShipDoneLoadingEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship is located and loaded at.

ShipStartingUnloading

This event is raised when a `Ship` is starting the unloading process of `Container`s.

ShipStartingUnloadingEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.

`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship is located and unloading at.

ShipUnloadedContainer

This event is raised when a `Ship` has unloaded one (1) `Container`.

ShipUnloadedContainerEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation
`String Description` A description of the event.	`Container Container` The container unloaded from the ship.

ShipDoneUnloading

This event is raised when a `Ship` is done with the unloading process of `Container`s.

ShipDoneUnloadingEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation
`String Description` A description of the event.	`Guid DockID` The unique ID of the dock the ship is located and unloaded at.

ShipUndocking

This event is raised when a `Ship` is starting undocking from one of the loading docks in the
`Harbor`.

ShipUndockingEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid LocationID` The unique ID of the location the ship undocked from, anchorage or Dock.

ShipInTransit

This event is raised when a `Ship` has finished undocking and has entered transit.

ShipInTransitEventArgs

Fields

`Ship Ship` The ship involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	`Guid TransitLocationID` The unique ID of the transit location the ship is located at.

TruckLoadingFromHarborStorage

This event is raised when a `Truck` is loading a `Container` from the `Harbor` container storage area. One truck can only fit one container - no matter the size.

TruckLoadingFromHarborEventArgs

Fields

`Truck Truck` The truck involved in the event.	`DateTime CurrentTime` The current time in the simulation.
`String Description` A description of the event.	

Exceptions

The HarbNet framework has multiple safety exceptions throughout the framwork, to ensure and guide you, the user, through a proper usage and let you know if something goes wrong.

Some exceptions are user-input related, and can be handled and fixed by the user themselves.

Other exceptions are internal-logic related, and can be triggered if for instance bugs where to occur. Even though the user can't always handle or deal with these exceptions directly, they are useful and necessary for further maintance of the framework!
We appreciate any reports back regarding these exceptions!

User-input handling

HarbNet uses the standardized exception-classes `ArgumentException`, `ArgumentOutOfRangeException` and `NullReferenceException` to notify you when the provided arguments are invalid, out of the expected range, or when an attempt is made to access an object that is null.

Some examples of this:

`ArgumentException` :

In the `Ship`-constructor, an `ArgumentException` is thrown if the given ship size is invalid.

Message: "Invalid ship size given. Valid ship sizes: ShipSize.Small, ShipSize.Medium, ShipSize.Large”

`ArgumentOutOfRangeException`

In the `Harbor` constructor, an exception is thrown if the ‘percentageOfContainersDirectlyLoadedFromShipToTrucks’ parameter is outside the allowable range of 0 to 100. This is to make sure the number that is provided is in fact a valid percentage.

Message:
"percentageOfContainersDirectlyLoadedFromShipToTrucks must be a number between 0 and 100”

Message:
"percentageOfContainersDirectlyLoadedFromHarborStorageToTrucks must be a number between 0 and 100"

The `Ship`-constructor throws an `ArgumentOutOfRangeException` if the ship size given is invalid, and not of ShipSize.Small, ShipSize.Medium or ShipSize.Large:

Message:
"Invalid ship size given. Valid ship sizes: ShipSize.Small, ShipSize.Medium, ShipSize.Large"

In addition, `ArgumentOutOfRangeException`s is thrown if the weight of the ship becomes too heavy or has too many containers onboard. This can mainly be triggered by using the overload constructor for `Ship`, and the user provides a list with too many containers on onboard (or they are in total too heavy).

Message:
"The ships current weight is to heavy. Max overall container weight for small ships is 600 tonns (about 55 containers), for medium ships: 1320 tonns (about 55 containers), for large ships: 5600 tonns (about 150 containers)"

Message:
"The ship has too many containers on board. The container capacity for small ships is max 20 containers"
"The ship has too many containers on board. The container capacity for medium ships is max 50 containers"
"The ship has too many containers on board. The container capacity for large ships is max 100 containers"

Exceptions on internal logic

In addition to these user-specific exceptions, the internal logic of the framework includes exceptions to handle unexpected issues, such as attempts to load or unload containers from objects and locations that either can’t be found or are unavailable.

This is handled with `NullReferenceException`s, `ArgumentOutOfRangeException`s and `ArgumentException`s.
In addition, the HarbNet framework has three custom Exceptions-classes: `CraneCantBeLoadedException`, `AgvCantBeLoadedException` and `TruckCantBeLoadedException`.

Advanced: Abstract classes

The namespace Advanced defines classes that are not meant for concrete implementations in a harbor simulation, but instead intended for advanced senarios where the user would like to define their own classes to be used in a harbor simulation. The abstract classes are also well suited to be used in testing of the of the API. Regular use of the framework does not require the use of these classes although many of the commonly used classes in the API will inherit from these abstract classes.

CargoVessel

The CargoVessel abstract class defines the contract for the public API of CargoVessels such as Ships used in the HarbNet framework. It contains several abstract members that must be implemented in classes representing publicly available cargo vessels in the framework. The CargoVessel abstract class should not be directly used in a harbor simulation, instead use a class that inherits from the CargoVessel class such as the Ship class. The CargoVessel class can hovever be used to create fakes used to test the API of the HarbNet framework.

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

Fields

Name	Type	Description
`ID`	`Guid`	Gets the unique ID of the CargoVessel
`ShipSize`	`ShipSize`	Gets the ship size, Small, Medium or Large.
`Name`	`String`	Gets the given name of the CargoVessel.
`StartDate`	`DateTime`	Gets the scheduled date and time the CargoVessel is to start its activities in the simulation.

`RoundTripInDays`	`Int`	Gets the number of days the CargoVessel uses to complete a roundtrip at sea, before returning to the harbor.
`CurrentLocation`	`Guid`	Gets the ID of the current location of the CargoVessel.
`History`	`ReadOnly Collection`	Gets a Collection of all the activities the CargoVessel has gone through throughout the simulation, as `StatusLog` objects.
`ContainersOnBoard`	`IList`	Gets all the containers in the CargoVessel 's storage.
`ContainerCapacity`	`Int`	Gets the max number of containers the CargoVessel can hold at any given time
`MaxWeightInTonn`	`Int`	Gets the max number of tons the CargoVessel can weigh, including cargo.
`BaseWeightInTonn`	`Int`	Gets the base weight of the ship alone, without cargo.
`CurrentWeightInTonn`	`Int`	Gets the current weight of the CargoVessel, 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.

HistoryRecord

The HistoryRecord abstract class defines the contract for the public API of History Records such as the DailyLog class. HistoryRecords contain information about the state of a harbor on a given date and time of a simulation. The class contains several abstract members that must be implemented in classes representing publicly available history records in the framework. The HistoryRecord abstract class should not be directly used in a harbor simulation, instead use a class that inherits from the History Record class such as the DailyLog class. The HistoryRecord class can hovever be used to create fakes used to test the API of the HarbNet framework.

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

Fields

Name	Type	Description
`Timestamp`	`DateTime`	Gets the date and time the record occured.
`ShipsInAnchorage`	`ReadOnly Collection <Ship>`	Gets a Collection containing all the ship objects located in the anchorage at the time of the record.

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

Methods

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

Port

The Port abstract class defines the contract for the public API of ports such as the Harbor class. Ports defines the location where all the activety in a harbor simulation takes place. The class contains several abstract members that must be implemented in classes representing publicly available ports in the framework. The port abstract class should not be directly used in a harbor simulation, instead use a class that inherits from the Port class such as the Harbor class. The Port class can hovever be used to create fakes used to test the API of the HarbNet framework.

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

Fields

Name	Type	Description
`ID`	`Guid`	Gets the unique ID of the port.
`ArrivedAtDestination`	`IList <Container>`	Gets all containers that have left the port 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 port's anchorage.
`AgvCargoID`	`Guid`	Gets the unique ID of the location the cargo is, when placed on an AGV.
`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
`GetAvailabilityStatusForAllLoadingDocks()`	`IDictionary <Guid, bool>`	Returns a Dictionary containing the IDs of all the loading docks and their availability.
`GetAvailabilityStatusForAllShipDocks()`	`IDictionary <Guid, bool>`	Returns a Dictionary containing the IDs of all the ship docks and their availability.
`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 abstract class defines the contract for the public API of simulations such as the SimpleSimulation class. Simulations can be used to run a simulation on a harbor object. The class contains several abstract members that must be implemented in classes representing publicly available simulations in the framework. The simulation abstract class should not be directly used to simulate a harbor, instead use a class that inherits from the Simulation class such as the SimpleSimulation class. The Simulation class can hovever be used to create fakes used to test the API of the HarbNet framework.

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

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.

StorageArea

The StorageArea abstract class defines the contract for the public API of Storage Areas such as the ContainerStorageRow class. StorageAreas defines an area used to store containers in the harbor. The class contains several abstract members that must be implemented in classes representing publicly available storage areas in the framework. The StorageArea abstract class should not be directly used in a harbor simulation, instead use a class that inherits from the StorageArea class such as the ContainerStorageRow class. The StorageArea class can hovever be used to create fakes used to test the API of the HarbNet framework.

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

Fields

Name	Type	Description
`ID`	`Guid`	Gets the unique ID of the storage area.

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 storage area.
`GetIDOfAllStoredContainers()`	`IList <Guid>`	Gets a list containing all the IDs of the stored containers
`ToString()`	`String`	Returns a String containing information about the storage area.

StatusRecord

The StatusRecord abstract class defines the contract for the public API of Status Records such as the StatusLog class. Status Records contain information about the status, location and time of a subject that has gone trough a statuschange in a simulation. The class contains several abstract members that must be implemented in classes representing publicly available StatusRecords in the framework. The Status Record abstract class should not be directly used in a harbor simulation, instead use a class that inherits from the Status Record class such as the StatusLog class. The Status Record class can hovever be used to create fakes used to test the API of the HarbNet framework.

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

Fields

Name	Type	Description
`Subject`	`Guid`	Gets the unique ID of the StatusRecord – the subject.
`SubjectLocation`	`Guid`	Gets the unique ID of the location where the recording took place.
`Timestamp`	`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 StatusRecord.

StorageUnit

The StorageUnit abstract class defines the contract for the public API of StorageUnits such as the Container class. StorageUnits are units used to store gods that can be transported on trucks and ships. The class contains several abstract members that must be implemented in classes representing publicly available StorageUnits in the framework. The StorageUnit abstract class should not be directly used in a harbor simulation, instead use a class that inherits from the StorageUnit class such as the Container class. The StorageUnit class can hovever be used to create fakes used to test the API of the HarbNet framework.

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

Fields

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

Methods

Name	Type	Description
`GetCurrentStatus()`	`Status`	Returns the current Status of the StorageUnit
`PrintHistory()`	`void`	Prints the History of the StorageUnit, 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 StorageUnit.