OOP Exam – Mortal Engines

Overview

In a war machine virtual factory there are two types of **machines**: **tanks** and **fighters**. Each machine has **name**, **pilot**, **health points**, **attack points**, **defense points** and **attack targets**. Each pilot has **name** and **machines** he **engages**. Pilots make status **reports** on all machines they engage. One machine can be **engaged** by one pilot at a time. **Tanks** have **defense mode** which can be turned **on** and **off**. **Fighters** have aggressive mode which can be turned **on** and **off**.

Setup

- Upload **only the `MortalEngines`** project in every problem **except Unit Tests**.
- **Do not modify the interfaces or their namespaces!**
- Use **strong cohesion** and **loose coupling**.
- **Use inheritance and the provided interfaces wherever possible**. This includes **constructors**, **method parameters** and **return types**!
- **Do not** violate your **interface implementations** by adding **more public methods** or **properties** in the concrete class than the interface has defined!
- Make sure you have **no public fields** anywhere.

Task 1: Structure (50 points)

You are given **4** interfaces, and you have to implement their functionality in the **correct classes**.

There are **4** types of entities in the application: **BaseMachine**, **Fighter, Tank** and **Pilot**:

BaseMachine

The **`BaseMachine`** is a **base class** for any **type of machines** and it **should not be able to be instantiated**.

Data

- **`Name`** – **string (If the name is null or whitespace throw an ArgumentNullException with message "Machine name cannot be null or empty.")**
- **`Pilot`** – **the machine pilot (if the pilot is null throw NullReferenceException with message "Pilot cannot be null.")**
- **`AttackPoints`** – **double**
- **`DefensePoints - double`**
- **`HealthPoints - double`**
- **`Targets – collection of strings`**

Behavior

`void Attack(IMachine target)`

If the target is null throw **`NullReferenceException`** with message **`"Target cannot be null"`**.

The machine attacks the target and decreases the target's health by the difference between the attacker's attack points and target's defense points. If the health of the target become less than zero, set it to zero.

Add the name of the target to the attacker's list of targets.

string ToString()

Returns a **string** with information about **each machine**. The returned string must be in the following format:

"- {machine name}"

" *Type: {machine type name}"

" *Health: {machine health points}"

" *Attack: {machine attack points}"

" *Defense: {machine defense points}"

" *Targets: " – if there are no targets "None". Otherwise {target1},{target2}…{targetN}

Constructor

A **base machine** should take the following values upon initialization:

`string` name, `double` attackPoints, `double` defensePoints, `double` healthPoints

Child Classes

There are several concrete types of **base machine**:

Fighter

Has 200 initial health points.

Data

- **AggressiveMode – bool**
  - **true** by default

Behavior

void ToggleAggressiveMode()

Flips **AggressiveMode** (true -> false or false -> true). When **AggressiveMode** is activated, attack points are increased with 50 and defense points are decreased with 25, otherwise (**AggressiveMode** is deactivated) attack points are decreased with 50 and defense points are increased with 25.

string ToString()

Returns the same info as the base machine class, but at the end depending on the aggressive mode writes the message:

" *Aggressive: {ON/OFF}"

Tank

Has 100 initial health points.

Data

- **DefenseMode – bool**
  - **true** by default

Behavior

Void ToggleDefenseMode()

Flips **DefenseMode**(true -> false or false -> true). When **DefenseMode** is activated, attack points are decreased with 40 and defense points are increased with 30, otherwise (**DefenseMode** is deactivated) attack points are increased with 40 and defense points are decreased with 30.

string ToString()

Returns the same info as the base machine class, but at the end depending on the defense mode writes the message:

" *Defense: {ON/OFF}"

Pilot

Data

- **Name – string (If the pilot name is null or whitespace throw ArgumentNullException with message "Pilot name cannot be null or empty string.")**
- **Machines – collection of machines**

Behavior

void AddMachine(machine)

Adds the provided machine to the pilot's machines. If the provided machine is null throw **NullReferenceException** with message **"Null machine cannot be added to the pilot."**

string Report()

Returns the message in format:

"{pilot name} - {machines count} machines"

And for each machine:

"- {machine name}"

" *Type: {machine type name}"

" *Health: {machine health}"

" *Attack: {machine attack)"

" *Defense: {machine defense}"

" *Targets: {machine targets}"

Constructor

A **pilot** should take the following values upon initialization:

`string` name

Task 2: Business Logic (150 points)

The Controller Class

The business logic of the program should be concentrated around several **commands**. You are given interfaces which you have to implement in the correct classes.

Note: The MachinesManager class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!

The first interface is **IMachinesManager**. You must create a **MachinesManager** class which implements the interface and implements all of its methods. The methods given should have the following logic:

Commands

There are several commands, which control the business logic of the application. They are stated below.

HirePilot Command

Parameters

- **Name** – **string**

Functionality

Creates a pilot with the provided name and adds him/her to the collection of pilots. The method should return one of the following messages:

- **"Pilot {name} hired"**
- **"Pilot {name} is hired already"** - if the pilot with the given name already exists and you should not create a pilot.

PilotReport Command

Parameters

- **Name** – **string**

Functionality

Searches for a hired pilot with given name and returns the **IPilot.Report()** method result.

MachineReport

Parameters

- **Name – string**

Functionality

Searches for an existing machine with given name and returns **ToString()** method result.

ManufactureTank Command

Parameters

- **Name - string**
- **Attack – double**
- **Defense - double**

Functionality

Creates a tank with given name, attack and defense points. The method should return one of the following messages:

- **"Tank {name} manufactured - attack: {attack}; defense: {defense}"**
- **"Machine {name} is manufactured already"** – if tank with the given name exists and you should not create a tank.

ToggleTankDefenseMode Command

Parameters

- **Name – string**

Functionality

Searches for tank with given name and toggles its defense mode. As a result, the command returns one of the following messages:

- **`"Tank {name} toggled defense mode"`**
- **`"Machine {name} could not be found"`** – if tank with the given name doesn't exist

ManufactureFighter Command

Parameters

- **`Name - string`**
- **`Attack – double`**
- **`Defense - double`**

Functionality

Creates a fighter with given name, attack and defense points. Duplicate names are not allowed. As a result, the command returns one of the following messages:

- **`"Fighter {name} manufactured - attack: {attack}; defense: {defense}; aggressive: ON"`**
- **`"Machine {name} is manufactured already"`** – if fighter with the given name exists and you should not create a figher.

ToggleFighterAggressiveMode Command

Parameters

- **`Name - string`**

Functionality

Searches for fighter with given name and toggles its aggressive mode. As a result, the command returns one of the following messages:

- **`"Fighter {name} toggled aggressive mode"`**
- **`"Machine {name} could not be found"`** – if fighter with the given name doesn't exist

EngageMachine Command

Parameters

- **`PilotName – string`**
- **`MachineName - string`**

Functionality

Searches for a pilot and machine by given names. If both exist and the machine is not occupied, add the machine to the pilot's list of machines and set the machine's pilot. As a result, the command returns one of the following messages:

- **`"Pilot {pilot name} could not be found"`**
- **`"Machine {machine name} could not be found"`**
- **`"Machine {machine name} is already occupied"`**
- **`"Pilot {pilot name} engaged machine {machine name}"`**

NOTE: Follow the exact order of messages.

AttackMachines Command

Parameters

- **AttackingMachineName – string**
- **DefendingMachineName - string**

Functionality

Searches for two machines by given names and the first one attacks the second one if both machines have health higher than zero. As a result, the command returns one of the following messages:

- **"Machine {name} could not be found"** – if one of the machines doesn't exist, the attacking machine is with priority
- **"Dead machine {name} cannot attack or be attacked"** – if one of the machines has health equal to zero, the attacking machine is with priority
- **"Machine {defending machine name} was attacked by machine {attacking machine name} - current health: {defending machine health}"**

Input / Output

Input

- You will receive commands **until you receive "Quit"** as a command.

Below, you can see the **format** in which **each command** will be given in the input:

- **HirePilot {name}**
- **PilotReport {name}**
- **ManufactureTank {name} {attack} {defense}**
- **ManufactureFighter {name} {attack} {defense}**
- **MachineReport {name}**
- **AggressiveMode {name}**
- **DefenseMode {name}**
- **Engage {pilot name} {machine name}**
- **Attack {attacking machine name} {defending machine name}**
- **Quit**

Output

Print the output from each command when issued.

If an exception is thrown during any of the commands' execution, print:

- **"Error:"** plus the message of the exception

Constraints

- The commands will always be in the provided format.

Examples

Input
HirePilot John HirePilot Nelson ManufactureFighter Boeing 180 90 Engage John Boeing AggressiveMode Boeing Quit
Output
Pilot John hired Pilot Nelson hired Fighter Boeing manufactured - attack: 230.00; defense: 65.00; aggressive: ON Pilot John engaged machine Boeing Fighter Boeing toggled aggressive mode
Input
HirePilot Smith HirePilot Stones ManufactureFighter Boeing 180 90 ManufactureTank T-72 100 100 Engage Stones T-72 Engage Smith Boeing Attack Boeing T-72 MachineReport Boeing MachineReport T-72 Quit
Output
Pilot Smith hired Pilot Stones hired Fighter Boeing manufactured - attack: 230.00; defense: 65.00; aggressive: ON Tank T-72 manufactured - attack: 60.00; defense: 130.00 Pilot Stones engaged machine T-72 Pilot Smith engaged machine Boeing Machine T-72 was attacked by machine Boeing - current health: 0.00 - Boeing *Type: Fighter *Health: 200.00 *Attack: 230.00 *Defense: 65.00 *Targets: T-72 *Aggressive: ON - T-72 *Type: Tank *Health: 0.00 *Attack: 60.00 *Defense: 130.00 *Targets: None *Defense: ON

Task 3: Unit Tests (100 points)

You will receive a skeleton with **Phone** class inside. The class will have some methods, properties, fields and one constructor which are working properly. You are **NOT ALLOWED** to change class. Cover the whole class with unit tests to make sure that the class is working as intended.

You are provided with a **unit test project** in the **project skeleton**. **DO NOT modify its NuGet packages**.

Note: The **Phone** you need to test is in the **global namespace**, so **remove any using statements** pointing towards the namespace **`Telecom`**.

Do **NOT** use **Mocking** in your unit tests!