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 **attacked 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**.

Structure

You have to study the code provided it is split into different packages, so go and look what you have provided.

There are **3 core** interfaces and **4 entities** interfaces:

The **3** core interfaces represent the two main factories, those should be implemented in the following classes **"PilotFactoryImpl"** and **"MachineFactoryImpl".** Also you are given the **"MachineManagerImpl"** class which you have to study and use later on.

The **4** interfaces representing the behavior of the data model classes, should be implemented in the **correct classes, there should be 4 types in** the application: **"BaseMachine"**, **"FighterImpl", "TankImpl"** and **"PilotImpl"**:

Base Machine

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

- Implements the **`Machine`** interface
- It should have the following properties:
  - **name** – **String** (If the name is null or whitespace **throw an IllegalArgumentException** with message **"Machine name cannot be null or empty."**)
  - **`pilot`** – the machine pilot (if the pilot is null **throw NullPointerException** with message **"Pilot cannot be null."**)
  - **`attackPoints`** – **double**
  - **`defensePoints - double`**
  - **`healthPoints - double`**
  - **`targets – collection of Strings`**

A **`BaseMachine`** should take the following values upon initialization:

`String` name, `double` attackPoints, `double` defensePoints, `double` healthPoints

There are generally 2 types of **`BaseMachine`**:

Fighter

`FighterImpl` is initialized with a **name** (**String**), **`attackPoints`** (**double**) and **`DefensePoints`** (**double**).

- The **`FighterImpl`** is a type of machne that has **200** initial health points upon initialization
- implements **`Fighter`** interface
- It has several additional properties:
  - **`aggressiveMode – boolean`**
    - **`true`** by default
  - **`attackPointsModifier – double`**
    - constant value of **50.0**

- **`deffencePointsModifier – double`**
  - constant value of **25.0**

Tank

`TankImpl` is initialized with a **`name`** (**String**), **`attackPoints`** (**double**) and **`DefensePoints`** (**double**).

- The **`TankImpl`** is a type of machne that has **100** initial health points upon initialization
- implements **`Tank`** interface
- It has several additional properties:
  - **`defenseMode – boolean`**
    - **`true`** by default
  - **`attackPointsModifier – double`**
    - constant value of **40.0**
  - **`deffencePointsModifier – double`**
    - constant value of **30.0**

Pilot

`PilotImpl` is initialized with a **`name`** (**String**).

- Implements the **`Pilot`** interface
- It should have the following properties:
  - **`name`** – **String** (If the name is null or whitespace **throw an IllegalArgumentException** with message **" Pilot name cannot be null or empty string."**)
  - **`machines – collection of machines`**

Functionality

The BaseMachine Class

void attack(String name);

- Adds the provided target to the targets collection. If the target is null or whitespace throw **`IllegalArgumentException`** with message **"Attack target cannot be null or empty string."**

The Fighter Class

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.

The Tank Class

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.

The Pilot Class

void addMachine(Machine machine);

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

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 `MachinesManagerImpl` class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!

The first interface is **`MachinesManager`**. You must implement the following methods inside the **`MachinesManagerImpl`** class which are:

hirePilot(String name);

- Creates a pilot with the provided name and adds it to the collection of pilots.

manufactureTank(String name, double attackPoints, double defensePoints);

- Creates a tank with given name, attack and defense points. Duplicate names are not allowed.

manufactureFighter(String name, double attackPoints, double defensePoints);

- Creates a fighter with given name, attack and defense points. Duplicate names are not allowed.

engageMachine(String selectedPilotName, String selectedMachineName);

- Searches for a pilot and machine by given names, adds the machine to the pilot's list of machines and initializes the machine's pilot.

attackMachines(String attackingMachineName, String defendingMachineName);

- Searches for two machines by given names and the first one attacks the second one if both are manufactured. Otherwise return appropriate message.
- If first machine attack points are higher than the defense points of the other, then second machine's health points are decreased by the value of attack points of the attacking machine. If the health of the target become less than zero, set it to zero. Otherwise the attack is unsuccessful and the stats points of attacking and defending machines remains as they were before.
- Regardless the success of the attack when both machines are present the **attacker** adds the name of the **defending** machine to its collection of targets.

pilotReport(String pilotName);

- Searches for a hired pilot with given name and returns the **`pilot.report()`** method result.

toggleFighterAggressiveMode(String fighterName);

- 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.
- Searches for fighter with given name and toggles its aggressive mode.

toggleTankDefenseMode(String tankName);

- 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.
- Searches for tank with given name and toggles its defense mode.

Commands

There are several commands which are given from the user input, in order to control the program.
Here you can see how they are formed.

The **parameters** will be given in the **EXACT ORDER**, as the one **specified below**.
You can see the exact input format in the **Input section**.

Each command will **generate an output result**, which you must **print**.
You can see the exact output format in the **Output section**.

Hire

Parameters –**name** (String).

Hires a **Pilot** with the **given name**.

Report

Parameters – **pilotName** (String)

Invoke **pilotReport** method from the controller class.

ManufactureTank

Parameters – **name** (String), **attackPoints** (double), **defensePoints** (double)

Invoke the **manufactureTank** method from the controller class.

ManufactureFighter

Parameters – **name** (String), **attackPoints** (double), **defensePoints** (double)

Invoke the **manufactureFighter** method from the controller class.

Engage

Parameters – **pilotName** (String), **machineName** (String)

Invoke **engageMachine** method from the controller class.

Attack

Parameters – **attackMachineName** (String), **defendingMachineName** (String)

Invoke **attackMachines** method from the controller class.

AggressiveMode

Parameters – **machineName** (string)

Invoke **toggleFighterAggressiveMode** method from the controller class.

DefenseMode

Parameters – **machineName** (string)

Invoke **toggleTankDefenseMode** method from the controller class.

Over

Exits the program.

Skeleton

In this section you will be given information about the Skeleton, or the code that has been given to you.

You are allowed to change the **internal** and **private logic** of the **classes** that have been given to you.
In other words, you can change the **body code** and the **definitions** of the **private members** in whatever way you like.

However. . .

You are **NOT ALLOWED** to **CHANGE** the **Interfaces** that have been provided by the **skeleton** in **ANY way**.
You are **NOT ALLOWED** to **ADD** more **PUBLIC LOGIC**, than the **one**, **provided** by the **Interfaces**, **ASIDE FROM** the **`toString()`** method and **`compareTo()`** method.

Interfaces & Others

You will be given the **interfaces** for the entities. You should use them when you are implementing your entities.

Input

The input consists of several commands which will be given in the format, specified below:

- **`Hire {pilotName}`**
- **`Report {pilotName}`**
- **`ManufactureTank {tankName} {attackPoints} {DefensePoints}`**
- **`ManufactureFighter {fighterName} {attackPoints} {DefensePoints}`**
- **`Engage {pilotName} {machineName}`**
- **`Attack {attackingMachineName} {defendingMachineName}`**
- **`DefenseMode {machineName}`**
- **`AggressiveMode {machineName}`**

Output

Each of the commands generates **output**. Here are the **output formats** of each command:

- **Hire Command** - create a **Pilot** with the **given name**. Prints the following result:
  "Pilot {name} hired"
- **ManufactureTank Command** – create a **Tank**, with the **given properties**. Prints the following result:
  "Tank {tankName} manufactured - attack: {attack}; defense: {defense}"
  - If **tank** is already manufactured, print:
    "Machine {tankName} is manufactured already"
- **ManufactureFighter Command** create a **Fighter**, with the **given properties**. Prints the following result:
  "Fighter (name) manufactured - attack: {attack}; defense: {defense}; aggressive: (ON/OFF)"
  - If **fighter** is already manufactured, print:
    "Machine {fighterName} is manufactured already"
- **Engage Command** – engage machine with the given pilot. Prints the following result:
  "Pilot {pilot-name} engaged machine {machine-name}"
- **Attack Command –** first machine attacks second.
- If attack is successful you should print:
  "Machine {defending-machine-name} was attacked by machine {attacking-machine-name} - current health: {defending-machine-health}"
- If one of the machines doesn't exist, print:
  "Machine {attackingMachineName/defendingMachineName} could not be found"
- **AggressiveMode Command –** toggle aggressive mode of the given Fighter. Print the following result:
  "Fighter {name} toggled aggressive mode"
- **DefenseMode Command –** toggle defense mode of the given Fighter. Print the following result:
  "Tank {name} toggled defense mode"
- **Report command** – provides **detailed information** about the **Pilot** and **machines** that he has engaged with, in following format:

Pilot
{pilotName} – {machines count} machines - {machine1.toString()} - {machine2.toString()} ... *Aggressive/Defense Mode(ON/OFF)
BaseMachine
*Type: {machineTypeName} *Health: {machineHealth} *Attack: {attackPoints} *Defense: {defensePoints} *Targets: {machineTargets}
Fighter
*Type: {machineTypeName} // Fighter *Health: {machineHealth} *Attack: {attackPoints} *Defense: {defensePoints}

*Targets: {machineTargets} *Aggressive Mode(ON/OFF)
Tank
*Type: {machineTypeName} // Fighter *Health: {machineHealth} *Attack: {attackPoints} *Defense: {defensePoints} *Targets: {machineTargets} *Defense Mode(ON/OFF)

- If the **targets** collection is emtpy print **"None"**
- Otherwise print all **targets** separated by ", "
- If the **aggressiveMode/defenseMode is true** print **(ON)**, if it is false print **(OFF)**

- **Over command** – Terminates the program

Note: All output **floating-point numbers** must be formatted to the **2nd digit** after the **decimal point**.

Constrains

- All **input lines** will be **absolutely valid**.

Examples

Input
Hire John Hire Nelson Report Bender ManufactureTank T-72 100 100 ManufactureFighter Kingcobra 150 90 Report John Engage John T-72 Engage John Kingcobra Report John Report Nelson Engage Nelson T-72 Engage Nelson Kingcobra ManufactureFighter Boeing 180 90 Engage Nelson Boeing Attack T-72 Kingcobra Attack T-72 Boeing DefenseMode T-72 DefenseMode Kingcobra DefenseMode Boeing Attack T-72 Kingcobra Attack T-72 Boeing AggressiveMode Kingcobra AggressiveMode Boeing AggressiveMode T-72 Attack Kingcobra T-72 Attack Boeing T-72 Report Nelson Report John Over

Output

```
Pilot John hired
Pilot Nelson hired
Pilot Bender could not be found
Tank T-72 manufactured - attack: 100.00; defense: 100.00
Fighter Kingcobra manufactured - attack: 150.00; defense: 90.00
John - 0 machines
Pilot John engaged machine T-72
Pilot John engaged machine Kingcobra
John - 2 machines
- T-72
 *Type: Tank
 *Health: 100.00
 *Attack: 60.00
 *Defense: 130.00
 *Targets: None
 *Defense Mode(ON)
- Kingcobra
 *Type: Fighter
 *Health: 200.00
 *Attack: 200.00
 *Defense: 65.00
 *Targets: None
 *Aggressive Mode(ON)
Nelson - 0 machines
Machine T-72 is already occupied
Machine Kingcobra is already occupied
Fighter Boeing manufactured - attack: 180.00; defense: 90.00
Pilot Nelson engaged machine Boeing
Machine Kingcobra was attacked by machine T-72 - current health: 200.00
Machine Boeing was attacked by machine T-72 - current health: 200.00
Tank T-72 toggled defense mode
Machine Kingcobra does not support this operation
Machine Boeing does not support this operation
Machine Kingcobra was attacked by machine T-72 - current health: 165.00
Machine Boeing was attacked by machine T-72 - current health: 165.00
Fighter Kingcobra toggled aggressive mode
Fighter Boeing toggled aggressive mode
Machine T-72 does not support this operation
Machine T-72 was attacked by machine Kingcobra - current health: 50.00
Machine T-72 was attacked by machine Boeing - current health: 0.00
Nelson - 1 machines
- Boeing
 *Type: Fighter
 *Health: 165.00
 *Attack: 180.00
 *Defense: 90.00
 *Targets: T-72
 *Aggressive Mode(OFF)
```

```
John - 2 machines
- T-72
 *Type: Tank
 *Health: 0.00
 *Attack: 100.00
 *Defense: 100.00
 *Targets: Kingcobra, Boeing, Kingcobra, Boeing
 *Defense Mode(OFF)
- Kingcobra
 *Type: Fighter
 *Health: 165.00
 *Attack: 150.00
 *Defense: 90.00
 *Targets: T-72
 *Aggressive Mode(OFF)
```

Tasks

Task 1: High Quality Structure

High Quality Code.

Achieve good separation of concerns using abstractions and interfaces to decouple classes, while reusing code through inheritance and polymorphism. Your classes should have strong cohesion - have single responsibility and loose coupling - know about as few other classes as possible.

Note: For this task,zip the content in your **`src`** pakage and submit it.

Task 2: Correct business logic.

The given code provides some functionality, but it does not cover the entire task. Implement the rest of the business logic, using the given code, and implement everything following the requirements specification. Check your solutions in the Judge system.

Note: For this task, submit the whole "**`src`**" folder.