**C# OOP Retake Exam - 20 December 2021**

**Overview**

In a naval base factory, there are two types of **vessels**: **submarines** and **battleships**. Each **vessel** has a **name**, **captain**, **armor thickness**, **main weapon caliber**, **speed,** and **targets**. Each captain has a **full name, combat experience,** and the **vessels** he **commands**. Captains make status **reports** on all vessels they were assigned to. One vessel can be **commanded** by one captain at a time. **Submarines** have **submerged mode** which can be turned **on** and **off**. **Battleships** have **sonar mode** which can be turned **on** and **off**.

**Setup**

* Upload **only the NavalVessels** project for the first and second tasks.
* **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: **Vessel**, **Submarine, Battleships,** and **Captain**:

**Vessel**

The **Vessel** is a **base class** for any **type of vessel,** and it **should not be able to be instantiated**.

**Data**

* **Name** - string**,** if the name is null or whitespace throws an **ArgumentNullException** with a message"Vessel name cannot be null or empty."
* **Captain** – the vessel’s captain, if it is null throw **NullReferenceException** with a message "Captain cannot be null."
* **ArmorThinkness - double**
* **MainWeaponCaliber** - **double**
* **Speed - double**
* **Targets - a collection of strings**

**Behavior**

**void Attack(IVessel target)**

If the target (defending vessel) is null throw **NullReferenceException** with a message "Target cannot be null."

When the **attacking vessel** attacks the **target vessel**, the **target's armor thickness points** are **reduced** by the **attacking vessel's main weapon caliber points**. Keep in mind that the **target's armor thickness** **points** can **not** go below **zero**. If the **target's armor thickness points become a negative number**, set it to **zero**. Add the name of the target vessel to the **attacker's list of targets**.

**void RepairVessel()**

Set the vessel’s initial armor thickness to the default value based on the vessel type.

**string ToString()**

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

"- {vessel name}

\*Type: {vessel type name}

\*Armor thickness: {vessel armor thickness points}

\*Main weapon caliber: {vessel main weapon caliber points}

\*Speed: {vessel speed points} knots

\*Targets: " – if there are no targets "None" Otherwise print "{target1}, {target2}, {target3}, {targetN}"

**NOTE: Do not use** **"\r\n"**.

**Constructor**

The constructor of the **Vessel** class should accept the following parameters:

string name, double mainWeaponCaliber, double speed, double armorThickness

**Child Classes**

There are two concrete types of vessels:

**Battleship**

Has 300 initial armor thickness.

**Data**

* **SonarMode** - **bool**
* "false"by default

**Behavior**

**void ToggleSonarMode()**

Flips **SonarMode** (false -> true or true -> false).

When **SonarMode** is activated (false -> true):

* **The main weapon caliber** is **increased** by 40 points
* **Speed** is **decreased** by 5 points

When **SonarMode** is deactivated (true -> false):

* **The main weapon caliber** is **decreased** by 40 points
* **Speed** is **increased** by 5 points

**void RepairVessel()**

If the battleship was **attacked** (its initial armor thickness is **less than 300**), set the battleship’s armor thickness back to the **initial one**.

**string ToString()**

Returns the same info as the **Vessel class**, but at the end depending on the **SonarMode** mode writes the message on a new line:

" \*Sonar mode: {ON/OFF}"

**Submarine**

Has 200 initial armor thickness.

**Data**

* **SubmergeMode** - **bool**
* "false"by default

**Behavior**

**void ToggleSubmergeMode()**

Flips **SubmergeMode** (false -> true or true -> false).

When **SubmergeMode** is activated (false -> true):

* **The main weapon caliber** is **increased** by 40 points
* **Speed** is **decreased** by 4 points

When **SonarMode** is deactivated (true -> false):

* **The main weapon caliber** is **decreased** by 40 points
* **Speed** is **increased** by 4 points

**void RepairVessel()**

If the submarine was **attacked** (its initial armor thickness is **less than 200**), set the submarine’s armor thickness back to the **initial one**.

**string ToString()**

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

**"** \*Submerge mode: {ON/OFF}"

**Captain**

**Data**

* **FullName** – **string,** if the captain’s name is null or whitespace throw **ArgumentNullException** with a message **"**Captain full name cannot be null or empty string.**"**
* **CombatExperience – int**, with the initial value of 0, could be increased by 10.
* **Vessels – a collection of IVessels**

**Behavior**

**void AddVessel(IVessel vessel)**

Adds the provided vessel to the captain’s vessels. If the provided vessel is null throw **NullReferenceException** with a message: **"**Null vessel cannot be added to the captain.**"**

**void IncreaseCombatExperience()**

Increase captain’s **combat experience** by **10** when a vessel that he commands **attack or defend**. There will be no case where **the attacking vessel and defend vessel will have the same captain**.

**string Report()**

Returns the message in format:

**"**{FullName} has {CombatExperience} combat experience and commands {vessels count} vessels.**"**

**If the captain commands any vessel**, return:

"- {vessel name}

\*Type: {vessel type name}

\*Armor thickness: {vessel armor thickness points}

\*Main weapon caliber: {vessel main weapon caliber points}

\*Speed: {vessel speed points} knots

\*Targets: None/{targets}

\*Sonar/Submerge mode: ON/OFF"

**Otherwise do not do not return enything about a vessel.**

**Constructor**

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

string fullName

**VesselRepository**

The vessel repository is a repository for all created vessels.

**Data**

* **Models** – **a** **collection of vessels (unmodifiable)**

**Behavior**

**void Add(IVessel vessel)**

* Adds a vessel in the **vessel’s collection**.
* Every vessel is unique and it is guaranteed that there will not be a vessel with the same name.

**bool Remove(IVessel vessel)**

* Removes a vessel from the collection. Returns true if the deletion was successful.

**IVessel FindByName(string name)**

* Returns a vessel with that name if he exists. If he doesn't, returns null.

**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 Controller class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

The first interface is **IController**. You must create a **Controller** class, which implements the interface and implements all of its methods. The constructor of **Controller** does not take any arguments. The given methods should have the logic described for each in the Commands section. When you create the **Controller** class, go into the **Engine** class constructor and uncomment the "this.controller = new Controller();" line.

**Data**

You need to keep track of some things, this is why you need some private fields in your controller class:

* **vessels** - **VesselRepository**
* **captains** - **a** **collection of ICaptain**

**Commands**

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

**HireCaptain Command**

**Parameters**

* **fullName** – **string**

**Functionality**

Creates a captain with the **provided full name** and **adds** him/her to the **collection** of captains. The method should return one of the following **messages**:

* If the captain is hired **successfully** return: **"**Captain {fullName} is hired.**"** and add him/her to the **collection** of captains.
* If a captain with the given name **already exists** return: **"**Captain {fullName} is already hired.**"**, and the given captain should **not be hired**.

**ProduceVessel Command**

**Parameters**

* **name – string**
* **vesselType - string**
* **mainWeaponCaliber - double**
* **speed - double**

**Functionality**

Creates a **Vessel** of the given type (**Submarine** or **Battleship**) with a given **name**, **main weapon caliber**, and **speed** points.The method should return one of the following messages:

* If the **vessel** with the given name exists return: **"**{typeVessel} vessel {name} is already manufactured.**"**
* If the **vesselType** is invalid return**: "**Invalid vessel type.**"**
* If the **vessel** is successfully produced return: **"**{typeVessel} {name} is manufactured with the main weapon caliber of {mainWeapon} inches and a maximum speed of {speed} knots." and adds the vessel to the **VesselRepository**.

**AssignCaptain Command**

**Parameters**

* **selectedCaptainName – string**
* **selectedVesselName - string**

**Functionality**

Searches for a **captain** and **vessel** by given names.

As a result, the command returns one of the following **messages**:

* **If the captain does not exist** return**: "**Captain {selectedCaptainName} could not be found.**"**
* **If the vessel does not exist** return**: "**Vessel {selectedVesselName} could not be found.**"**
* **If the vessel has a captain** return**: "**Vessel {selectedVesselName} is already occupied.**"**
* **If the captain is successfully assigned to the vessel** return**: "**Captain {selectedCaptainName} command vessel {selectedVesselName}.**" and** add the vessel to the captain's list of vessels and set the vessel's captain to the **selectedCaptainFullName**

**NOTE: Follow the exact order of messages.**

**CaptainReport Command**

**Parameters**

* **Name** – **string**

**Functionality**

Searches for an assigned captain with a given name and returns the **ICaptain.Report()** method result.

**VesselReport**

**Parameters**

* **name – string**

**Functionality**

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

**ToggleSpecialMode Command**

**Parameters**

* **Name - string**

**Functionality**

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

* If the vessel is **battleship** and does **exist**, execute **ToggleSonarMode()** and return: "Battleship {name} toggled sonar mode."
* If the vessel is submarine and does exist, execute **ToggleSubmergeMode()** and return: "Submarine {name} toggled submerge mode."
* If the vessel **does not exist return: "**Vessel {name} could not be found.**"**

**ServiceVessel Command**

**Parameters**

* **vesselName - string**

**Functionality**

Search for a vessel with the given name and invoke its **RepairVessel()** method**.** As a result, the command returns one of the following messages:

* If the vessel is **successfully** **repaired** return: **"**Vessel {name} was repaired."
* If the vessel **does not exist** return**: "**Vessel {name} could not be found.**"**

**AttackVessels Command**

**Parameters**

* **attackingVesselName - string**
* **defendingVesselName - string**

**Functionality**

Searches for **two vessels** by given **names** and the **first one** attacks the **second one**. As a result, the command returns one of the following **messages**:

* If one of the **vessels doesn't exist**, the **attacking vessel** is with priority return: **"**Vessel {name} could not be found.**"**
* If one of the vessels **has armor thickness equal to zero**, the **attacking vessel** is with priority return: **"**Unarmored vessel {name} cannot attack or be attacked.**"**
* If all the criteria are matched **invoke** the attacking vessel **Attack()** method, increase combat experience of both vessel’s captains and return:

**"**Vessel {defendingVessleName} was attacked by vessel {attackVessleName} - current armor thickness: {defenderArmorThinckness}.**"**

**NOTE: Both the attacking vessel and the defending vessel will always have captains.**

**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:

* **HireCaptain {fullName}**
* **ProduceVessel {name} {vesselType} {mainWeaponCaliber} {speed}**
* **AssignCaptain {selectedCaptainName} {selectedVesselName}**
* **CaptainReport {captainFullName}**
* **VesselReport {vesselName}**
* **ToggleSpecialMode {vesselName}**
* **ServiceVessel {vesselName}**
* **AttackVessels {attackingVesselName} {defendingVesselName}**
* **Quit**

**Output**

Print the output from each command when issued.

If an exception is thrown during any of the commands' execution, print the corresponding error message.

**Constraints**

* The commands will always be in the provided format.

**Examples**

|  |
| --- |
| **Input** |
| HireCaptain Chester\_Nimitz  HireCaptain Karl\_Donitz  ProduceVessel USS\_Colorado Battleship 16 21  AssignCaptain Chester\_Nimitz USS\_Colorado  ToggleSpecialMode USS\_Colorado  VesselReport USS\_Colorado  Quit |
| **Output** |
| Captain Chester\_Nimitz is hired.  Captain Karl\_Donitz is hired.  Battleship USS\_Colorado is manufactured with the main weapon caliber of 16 inches and a maximum speed of 21 knots.  Captain Chester\_Nimitz command vessel USS\_Colorado.  Battleship USS\_Colorado toggled sonar mode.  - USS\_Colorado  \*Type: Battleship  \*Armor thickness: 300  \*Main weapon caliber: 56  \*Speed: 16 knots  \*Targets: None  \*Sonar mode: ON |
| **Input** |
| HireCaptain Chester\_Nimitz  HireCaptain Harald\_Lange  ProduceVessel USS\_Colorado Battleship 16 21  ProduceVessel U-505 Submarine 21.1 18.2  AssignCaptain Chester\_Nimitz USS\_Colorado  AssignCaptain Harald\_Lange U-505  AttackVessels USS\_Colorado U-505  VesselReport USS\_Colorado  VesselReport U-505  CaptainReport Chester\_Nimitz  Quit |
| **Output** |
| Captain Chester\_Nimitz is hired.  Captain Harald\_Lange is hired.  Battleship USS\_Colorado is manufactured with the main weapon caliber of 16 inches and a maximum speed of 21 knots.  Submarine U-505 is manufactured with the main weapon caliber of 21.1 inches and a maximum speed of 18.2 knots.  Captain Chester\_Nimitz command vessel USS\_Colorado.  Captain Harald\_Lange command vessel U-505.  Vessel U-505 was attacked by vessel USS\_Colorado - current armor thickness: 184.  - USS\_Colorado  \*Type: Battleship  \*Armor thickness: 300  \*Main weapon caliber: 16  \*Speed: 21 knots  \*Targets: U-505  \*Sonar mode: OFF  - U-505  \*Type: Submarine  \*Armor thickness: 184  \*Main weapon caliber: 21.1  \*Speed: 18.2 knots  \*Targets: None  \*Submerge mode: OFF  Chester\_Nimitz has 10 combat experience and commands 1 vessels.  - USS\_Colorado  \*Type: Battleship  \*Armor thickness: 300  \*Main weapon caliber: 16  \*Speed: 21 knots  \*Targets: U-505  \*Sonar mode: OFF |

**Task 3: Unit Tests (100 points)**

You will receive a skeleton with a **Book** class inside. The **Book** class has some methods, fields, and one constructor, which are working properly. You are **NOT ALLOWED** to change any class. Cover the whole **Book** 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** use **Mocking** in your unit tests!