**C# OOP Retake Exam - 18 April 2019**

**Overview**

**Преглед**

PlayersAndMonsters is a battle game. It's all about battles between players with their cards. Each player has health and deck of cards. Each card gives bonus damage and bonus health. The players fight on the battle field with their cards.

**Setup**

* Upload **only the PlayersAndMonsters** 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
* Качете само проекта на PlayersAndMonsters при всеки проблем, с изключение на тестовете на единица
* Не променяйте интерфейсите или техните пространства от имена
* Използвайте силно сцепление и разхлабена връзка
* Използвайте наследяване и предоставените интерфейси, когато е възможно. Това включва конструктори, параметри на метода и видове връщане
* Не нарушавайте реализациите на вашия интерфейс, като добавите повече публични методи или свойства в конкретния клас, отколкото интерфейсът е определил
* Уверете се, че нямате публични полета никъде

**Task 1: Structure (50 points)**

You are given **7** interfaces, and you have to implement their functionality in the **correct classes**.

There are **3** types of entities in the application: **Player, Card and BattleField**:

Дават ви 7 интерфейса и трябва да внедрите тяхната функционалност в правилните класове.

В приложението има 3 вида субекти: Player, Card и BattleField:

**Player**

**Player** is a **base class** for any **type of player** and it **should not be able to be instantiated**.

Играчът е базов клас за всеки тип играчи и не трябва да може да бъде инстанциран.

**Data**

* **Username** – string (If the username **is null or empty,** throw an **ArgumentException** with message "**Player's username cannot be null or an empty string.** ")
* **Health** – the health of а player (if the health is below **0,** throw an **ArgumentException** with message "**Player's health bonus cannot be less than zero.** ")
* **CardRepository** – repository of all **user's** cards.
* **IsDead –** calculated property which returns **bool**.
* Потребителско име - низ (Ако потребителското име е нулево или празно, хвърлете ArgumentException със съобщение „Потребителското име на играча не може да бъде нулево или празен низ.“)
* Здраве - здравето на играч (ако здравето е под 0, хвърлете ArgumentException със съобщение "Бонусът за играч на здравето не може да бъде по-малък от нула.")
* CardRepository - хранилище на всички карти на потребителя.
* IsDead - изчислено свойство, което връща bool.

**Behavior**

**void TakeDamage(int damagePoints)**

The TakeDamage method decreases players' points.

* If the **damagePoints** are **below 0** throw an **ArgumentException** with message "**Damage points cannot be less than zero.**"
* Player’s health should not drop below zero
* Методът TakeDamage намалява точките на играчите.
* Ако точките за повреда са под 0, хвърлете ArgumentException със съобщение "Точките на щетите не могат да бъдат по-малки от нула."
* Здравето на играча не трябва да пада под нулата

**Constructor**

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

Играчът трябва да приеме следните стойности при инициализация:

ICardRepository cardRepository, string username, int health

**Child Classes**

There are several concrete types of **players**:

Има няколко конкретни типа играчи:

**Beginner**

Has 50 initial health points.

Constructorshould take the following values upon initialization:

ICardRepository cardRepository, string username

**Advanced**

Has 250 initial health points.

Constructorshould take the following values upon initialization:

ICardRepository cardRepository, string username

начинаещ

Има 50 начални здравни точки.

Конструктор трябва да вземе следните стойности при инициализация:

ICardRepository cardRepository, потребителско име за низ

напреднал

Има 250 начални здравни точки.

Конструктор трябва да вземе следните стойности при инициализация:

ICardRepository cardRepository, потребителско име за низ

**Card**

The **Card** is a base class for any type of card and it should not be able to be instantiated.

Картата е базов клас за всеки тип карта и не трябва да може да бъде инстанцирана.

**Data**

* **Name** – string (If the card name **is null or empty** throw an **ArgumentException** with message "**Card's name cannot be null or an empty string.**")
* **DamagePoints** – int (If the damage points are **below zero,** throw an **ArgumentException** with message "**Card's damage points cannot be less than zero.**") if
* **HealthPoints** - int (If the health points are **below zero,** throw an **ArgumentException** with message "**Card's HP cannot be less than zero.**")
* Име - низ (Ако името на картата е нулево или празно, хвърлете ArgumentException със съобщение „Името на картата не може да бъде нулево или празен низ.“)
* DamagePoints - int (Ако точките на щетите са под нулата, хвърлете ArgumentException със съобщение „Точките за повреда на картата не могат да бъдат по-малки от нула.“)
* HealthPoints - int (Ако точките за здраве са под нулата, хвърлете ArgumentException със съобщение "HP на картата не може да бъде по-малко от нула

**Constructor**

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

Картата трябва да приема следните стойности при инициализация:

string name, int damagePoints, int healthPoints

**Child Classes**

There are several concrete types of **cards**:

Има няколко конкретни вида карти:

**MagicCard**

Has 5 damage points and 80 health points.

Constructorshould take the following values upon initialization:

Има 5 точки за щети и 80 точки за здраве.

Конструктор трябва да вземе следните стойности при инициализация:

string name

**TrapCard**

Has 120 damage points and 5 health points.

Constructorshould take the following values upon initialization:

TrapCard

Има 120 точки за щети и 5 точки за здраве.

Конструктор трябва да вземе следните стойности при инициализация:

string name

**BattleField**

The battle field is the place where the fight happens.

Бойното поле е мястото, където се случва битката.

**Behavior**

**void Fight(IPlayer attacker, IPlayer enemy)**

That's the most interesting method.

* If one of the users **is dead**, throw new **ArgumentException** with message "**Player is dead!**"
* If the player is a **beginner,** increase his **health** with **40** points and **increase** all damage **points** of all **cards** for the user with **30**.
* Before the fight, both players get bonus health points from their deck.
* Attacker attacks **first** and after that the enemy attacks. If **one of the players** is dead you should **stop** the fight.
* Това е най-интересният метод.
* Ако някой от потребителите е мъртъв, хвърлете нов ArgumentException със съобщение "Играчът е мъртъв!"
* Ако играчът е начинаещ, увеличете здравето си с 40 точки и увеличете всички точки за щети от всички карти за потребителя с 30.
* Преди двубоя и двамата играчи получават бонус здравни точки от тестето си.
* Нападателят първо атакува и след това врага атакува. Ако някой от играчите е мъртъв, трябва да спрете двубоя.

**PlayerRepository**

The player repository holds information for all users.

Репозиторият на плейъра съдържа информация за всички потребители.

**Data**

* **Count** – int – the count of players
* **Players** – **collection of players (unmodifiable)**

**Behavior**

**void Add(IPlayer player)**

Adds a player in the collection.

* If the player **is null**, throw an **ArgumentException** with message "**Player cannot be null**".
* If a player exists with a name equal to the name of the given player, throw an **ArgumentException** with message "**Player {username} already exists!**".
* Добавя играч в колекцията.
* Ако играчът е нулев, хвърлете ArgumentException със съобщение "Играчът не може да бъде нулев".
* Ако играч съществува с име, равно на името на дадения играч, хвърлете ArgumentException със съобщение "Играч {потребителско име} вече съществува!".

**bool Remove(IPlayer player)**

Removes a player from the collection.

* If the player **is null**, throw an **ArgumentException** with message "**Player cannot be null**".
* Премахва плейър от колекцията.
* Ако играчът е нулев, хвърлете ArgumentException със съобщение "Играчът не може да бъде нулев".

**IPlayer Find(string username)**

Returns a player with that username.

**CardRepository**

The card repository holds information for all cards.

Картата съхранява информация за всички карти.

**Data**

* **Count** – int – the count of cards
* **Cards** – **collection of cards (unmodifiable)**

**Behavior**

**void Add(ICard card)**

Adds a card in the collection.

* If the card **is null**, throw an **ArgumentException** with message "**Card cannot be null!**".
* If a card exists with a name equal to the name of the given card, throw an **ArgumentException** with message "**Card {name} already exists!**".
* Добавя карта в колекцията.
* Ако картата е нулева, хвърлете ArgumentException със съобщение "Картата не може да бъде нула!".
* Ако съществува карта с име, равно на името на дадената карта, хвърлете ArgumentException със съобщение "Карта {име} вече съществува!".

**bool Remove(ICard card)**

Removes a card from the collection.

* If the **card is null**, throw an **ArgumentException** with message "**Card cannot be null!**".
* Премахва карта от колекцията.
* Ако картата е нулева, хвърлете ArgumentException със съобщение "Картата не може да бъде нула!".

**ICard Find(string name)**

Returns a card with that 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 ManagerController class SHOULD NOT handle exceptions! The tests are designed to expect exceptions, not messages!**

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

Бизнес логиката на програмата трябва да се концентрира около няколко команди. Дават ви интерфейси, които трябва да внедрите в правилните класове.

Забележка: Класът ManagerController НЕ трябва да обработва изключения! Тестовете са създадени да очакват изключения, а не съобщения!

Първият интерфейс е IManagerController. Трябва да създадете клас ManagerController, който реализира интерфейса и реализира всичките му методи. Дадените методи трябва да имат следната логика:

**Commands**

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

Има няколко команди, които контролират бизнес логиката на приложението. Те са посочени по-долу.

**AddPlayer Command**

**Parameters**

* **Type - string**
* **Username** – **string**

**Functionality**

Creates a **player** with the provided **type** and **name**. The method should **return** the following **message**:

Създава играч с предоставения тип и име. Методът трябва да върне следното съобщение:

"**Successfully added player of type {type} with username: {username}**"

**AddCard Command**

**Parameters**

* **Type - string**
* **Name - string**

**Functionality**

Creates a **card** with the provided **type** and **name**. The method should **return** the following message:

Създава карта с предоставения тип и име. Методът трябва да върне следното съобщение:

"**Successfully added card of type {type}Card with name: {name}**"

**AddPlayerCard Command**

**Parameters**

* **Username - string**
* **CardName - string**

**Functionality**

Adds the given **card** to the **user** card **repository**. The method should **return** the following message:

Добавя дадената карта в хранилището на потребителските карти. Методът трябва да върне следното съобщение:

"**Successfully added card: {cardName} to user: {userName}**"

**Fight Command**

**Parameters**

* **AttackPlayer - string**
* **EnemyPlayer – string**

**Functionality**

Sends the **attacker** player and **enemy** player to the **battle** **field**. The method should return the following message:

Изпраща играча на нападателя и противника на бойното поле. Методът трябва да върне следното съобщение:

"**Attack user health {attack player} - Enemy user health {enemy player}**"

**Report Command**

**Functionality**

Returns a report message in format:

**"Username: {username} - Health: {health} – Cards {cards count}"  
"Card: {name} - Damage: {card damage}"  
"###"**

**Input / Output**

You are provided with one interface, which will helps with the correct execution process of your program. The interface is **IEngine** and the class implementing this interface should read the input and when the program finishes, this class should print the output.

Осигурен ви е един интерфейс, който ще помогне за правилния процес на изпълнение на вашата програма. Интерфейсът е IEngine и класът, реализиращ този интерфейс, трябва да чете входа и когато програмата завърши, този клас трябва да отпечата резултата.

**Input**

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

По-долу можете да видите формата, в който всяка команда ще бъде дадена във входа:

* **AddPlayer** {player type} {player username}
* **AddCard** {card type} {card name}
* **AddPlayerCard** {username} {card name}
* **Fight** {attack user} {enemy user}
* **Report**

**Output**

Print the output from each command when issued. If an exception is thrown during any of the commands’ execution, print the exception message.

Отпечатайте изхода от всяка команда при издаването. Ако по време на изпълнение на която и да е от командите е хвърлено изключение, отпечатайте съобщението за изключение.

**Examples**

|  |
| --- |
| **Input** |
| AddPlayer Beginner handyUser33  AddPlayer Advanced cool11  AddPlayer Beginner testUser  AddPlayer Advanced goro5  AddPlayer Beginner ivan12  AddPlayer Advanced goerge00  AddPlayer Advanced userUser  AddPlayer Beginner fakeAccount123  AddCard Trap Cyber  AddCard Magic Sorcerer  AddCard Trap Iris  AddCard Trap Jar  AddCard Magic Blaster  AddCard Trap Scientist  AddCard Magic Plushfire  AddCard Magic Substitoad  AddCard Trap Neptune  AddPlayerCard handyUser33 Cyber  AddPlayerCard handyUser33 Blaster  AddPlayerCard handyUser33 Neptune  AddPlayerCard ivan12 Iris  AddPlayerCard ivan12 Scientist  AddPlayerCard ivan12 Plushfire  AddPlayerCard goro5 Plushfire  AddPlayerCard userUser Neptune  Fight handyUser33 ivan12  Fight goro5 userUser  Report  Exit |
| **Output** |
| Successfully added player of type Beginner with username: handyUser33  Successfully added player of type Advanced with username: cool11  Successfully added player of type Beginner with username: testUser  Successfully added player of type Advanced with username: goro5  Successfully added player of type Beginner with username: ivan12  Successfully added player of type Advanced with username: goerge00  Successfully added player of type Advanced with username: userUser  Successfully added player of type Beginner with username: fakeAccount123  Successfully added card of type TrapCard with name: Cyber  Successfully added card of type MagicCard with name: Sorcerer  Successfully added card of type TrapCard with name: Iris  Successfully added card of type TrapCard with name: Jar  Successfully added card of type MagicCard with name: Blaster  Successfully added card of type TrapCard with name: Scientist  Successfully added card of type MagicCard with name: Plushfire  Successfully added card of type MagicCard with name: Substitoad  Successfully added card of type TrapCard with name: Neptune  Successfully added card: Cyber to user: handyUser33  Successfully added card: Blaster to user: handyUser33  Successfully added card: Neptune to user: handyUser33  Successfully added card: Iris to user: ivan12  Successfully added card: Scientist to user: ivan12  Successfully added card: Plushfire to user: ivan12  Successfully added card: Plushfire to user: goro5  Successfully added card: Neptune to user: userUser  Attack user health 180 - Enemy user health 0  Attack user health 0 - Enemy user health 150  Username: handyUser33 - Health: 180 - Cards 3  Card: Cyber - Damage: 150  Card: Blaster - Damage: 35  Card: Neptune - Damage: 150  ###  Username: cool11 - Health: 250 - Cards 0  ###  Username: testUser - Health: 50 - Cards 0  ###  Username: goro5 - Health: 0 - Cards 1  Card: Plushfire - Damage: 35  ###  Username: ivan12 - Health: 0 - Cards 3  Card: Iris - Damage: 150  Card: Scientist - Damage: 150  Card: Plushfire - Damage: 35  ###  Username: goerge00 - Health: 250 - Cards 0  ###  Username: userUser - Health: 150 - Cards 1  Card: Neptune - Damage: 150  ###  Username: fakeAccount123 - Health: 50 - Cards 0  ### |

|  |
| --- |
| **Input** |
| AddPlayer Beginner handyUser33  AddPlayer Advanced handyUser33  AddPlayer Advanced cool11  AddPlayer Beginner testUser  AddCard Trap Cyber  AddCard Magic Sorcerer  AddCard Trap Iris  AddCard Trap Iris  AddCard Trap Jar  AddPlayerCard handyUser33 Cyber  AddPlayerCard handyUser33 Blaster  AddPlayerCard cool11 Neptune  AddPlayerCard testUser Neptune  Fight handyUser33 testUser  Fight handyUser33 testUser  Fight handyUser33 testUser  Fight cool11 testUser  Report  Exit |
| **Output** |
| Successfully added player of type Beginner with username: handyUser33  Player handyUser33 already exists!  Successfully added player of type Advanced with username: cool11  Successfully added player of type Beginner with username: testUser  Successfully added card of type TrapCard with name: Cyber  Successfully added card of type MagicCard with name: Sorcerer  Successfully added card of type TrapCard with name: Iris  Card Iris already exists!  Successfully added card of type TrapCard with name: Jar  Successfully added card: Cyber to user: handyUser33  Card cannot be null!  Card cannot be null!  Card cannot be null!  Attack user health 95 - Enemy user health 0  Player is dead!  Player is dead!  Player is dead!  Username: handyUser33 - Health: 95 - Cards 1  Card: Cyber - Damage: 150  ###  Username: cool11 - Health: 250 - Cards 0  ###  Username: testUser - Health: 0 - Cards 0  ### |

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

You will receive a skeleton with **SoftPark** and **Car** classes inside. The class will have some methods, properties, fields and one constructor, which are working properly. You are **NOT ALLOWED** to change any 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 **SoftPark** you need to test is in the **global namespace**, so **remove any using statements,** pointing towards the namespace **ParkingSystem**.

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

Ще получите скелет с класове SoftPark и Car вътре. Класът ще има някои методи, свойства, полета и един конструктор, които работят правилно. НЕ СЕ ИЗПРАТЕН да променяте някой клас. Покрийте целия клас с единични тестове, за да сте сигурни, че класът работи по предназначение.

Осигурен ви е тестов проект в скелета на проекта. НЕ променяйте NuGet пакетите си.

Забележка: SoftPark, който трябва да тествате, е в глобалното пространство на имена, така че премахнете всички използващи оператори, насочени към пространството на имена ParkingSystem.

НЕ използвайте подигравки в тестовете на вашето устройство!