Exercises: Unit Testing

Problems for exercises and homework for the "C# OOP" course @ SoftUni".

Problem 1. Database

You are provided with a simple class - **Database**. It should **store integers**. **The initial integers should be set by constructor**. They are stored **in array**. **Database** have a functionality to **add**, **remove** and **fetch all stored items**. Your task is to **test the class**. In other words **write tests**, so you are sure its methods are working as intended.

Constraints

- Storing array's **capacity** must be **exactly 16 integers**
  - If the size of the array is not 16 integers long, **InvalidOperationException** is thrown.
- **Add** operation, should **add an element at the next free cell** (just like a stack)
  - If there are 16 elements in the Database and try to add 17th, **InvalidOperationException** is thrown.
- **Remove** operation, should support only removing an element **at the last index** (just like a stack)
  - If you try to remove element from empty Database, **InvalidOperationException** is thrown.
- **Constructors** should take integers only, and store them in **array**.
- **Fetch method** should return the elements as **array**.

Hint

Do not forget to **test the constructor(s)**. They are methods too!

Problem 2. Extended Database

You already have a class - **Database**. We have **modified it** and added some **more functionality** to it. It supports **adding, removing and finding People**. In other words, it **stores People**. There are two types of finding methods - first: **FindById (long id)** and the second one: **FindByUsername (string username)**. As you may guess, each person has its own **unique id**, and **unique username**. Your task is to test **the provided project**.

Constraints

Database should have methods:

- Add
  - If there are already users with this username, **InvalidOperationException** is thrown.
  - If there are already users with this id, **InvalidOperationException** is thrown.
- Remove
- FindByUsername
  - If no user is present by this username, **InvalidOperationException** is thrown.
  - If username parameter is null, **ArgumentNullException** is thrown.
  - Arguments are all **CaseSensitive**.
- FindById
  - If no user is present by this id, **InvalidOperationException** is thrown.
  - If negative ids are found, **ArgumentOutOfRangeException** is thrown.

Hint

Do not forget to test the constructor(s). They are methods too! Also keep in mind that all the functionality from the previous task still exists and you need to test it again!

Problem 3. Car Manager

You are provided with a simple project **containing only one class** - "**Car**". The provided class is simple - its **main point is to represent some of the functionality of a car**. **Each car contains information** about its **Make**, **Model**, **Fuel Consumption**, **Fuel Amount** and **Fuel Capacity**. Also **each car can add some fuel to its tank by refueling** and **can travel distance by driving**. **In order to be driven**, **our car needs to have enough fuel**. **Everything in the provided skeleton is working perfectly fine** and **you mustn't change it**.

In the skeleton you are provided **Test Project** named "**CarManager.Tests**". There you **should place all the unit tests** you write. The **Test Project** have only **one class** inside:

- "**CarTests**" - here you should place **all code** testing the "**Car**" and **it's functionality**.

Your job now is to **write unit tests on the provided project** and **it's functionality**. You should test exactly **every part** of code inside the "**Car**" class:

- You should test **all the constructors**.
- You should test **all properties** (**getters** and **setters**).
- You should test **all the methods** and **validations inside the class**.

Before you submit your solution to Judge, you should **remove all the references and namespaces referencing the other project**. You should **upload only** the "**CarManager.Tests**" project **holding the class with your tests**. **Remove** the "**bin**" and "**obj**" folders **before** submission.

Constraints

- **Everything in the provided skeleton is working perfectly fine**.
- **You mustn't change anything in the project structure**.
- **You can test both constructors together**.
- **You shouldn't test the auto properties**.
- **Any part of validation should be tested**.
- **There is no limit on the tests you will write but keep your attention on the main functionality**.

"Brum…Brum…Brum-suuuutututu…"

Problem 4. Fighting Arena

You are provided with a project named "**FightingArena**" containing **two classes** - "**Warrior**" and "**Arena**". **Your task** here is simple - **you need to write tests** on the project **covering the whole functionality**. But before start writing tests, you need to **get know** with the project's **structure** and **bussiness logic**. Each **Arena** has a **collection of Warriors** enrolled for the fights. On the **Arena**, **Warriors** should be able to **Enroll for the fights** and **fight each other**. Each Warrior has **unique name**, **damage** and **HP**. **Warriors** can **attack** other **Warriors**. Of course there is some kind of validations:

- **Name** cannot be **null**, **empty** or **whitespace**.
- **Damage** cannot be **zero or negative**.
- **HP** cannot be **negative**.
- **Warrior** cannot **attack** if his **HP** are **below 30**.
- **Warrior** cannot **attack Warriors** which **HP** are **below 30**.
- **Warrior** cannot **attack stronger enemies**.

On the **Arena** there should be performed **some validations** too:

- **Already enrolled Warriors** should not be able to **enroll again**.
- **There cannot be fight** if **one of the Warriors** is not **enrolled** for the fights.

In the skeleton you are provided **Test Project** named "**FightingArena.Tests**". There you **should place all the unit tests** you write. The **Test Project** have **two classes** inside:

- "**WarriorTests**" - here you should place **all code** testing the "**Warrior**" and **it's functionality**.
- "**ArenaTests**" - here you should place **all code** testing the "**Arena**" and **it's functionality**.

Your job now is to **write unit tests on the provided project** and **it's functionality**. You should test exactly **every part** of code inside the "**Warrior**" and "**Arena**" classes:

- You should test **all the constructors**.
- You should test **all properties** (**getters** and **setters**).
- You should test **all the methods** and **validations inside the class**.

Before you submit your solution to Judge, you should **remove all the references and namespaces referencing the other project**. You should **upload only** the "**FightingArena.Tests**" project **holding the two classes with your tests**. **Remove** the "**bin**" and "**obj**" folders **before** submission.

Constraints

- **Everything in the provided skeleton is working perfectly fine**.
- **You mustn't change anything in the project structure**.
- **You shouldn't test the auto properties**.
- **Any part of validation should be tested**.
- **There is no limit on the tests you will write but keep your attention on the main functionality**.

Problem 5. *Service

You are provided with **part of the structure** of a big IT project. **The main point** of the project is to **easy up the work in our service**. Before continue to the main task of today's exercise, **first let's get know** with the provided project's structure. In the "**Models**" folder you have three nested folders:

- "**Contracts**" - it holds all the **interfaces** of the project.
- "**Parts**" - it holds a base class representing **part's main functionality** and **different kind of parts**.
- "**Devices**" - it holds a base class representing **device's main functionality** and **different kind of devices**.

So you have **different kind of parts** which can be placed **in different kind of devices**. In **every device** there **can be placed only appropriate type of parts**. For example in **PC** there **can be placed only PC parts**, in **Laptop** - **Laptop parts** and so on. Everything in the provided skeleton is **working perfectly fine**. **You shouldn't change the structure of the project under any circumstances**.

In the skeleton you are provided **Test Project** named "**Service.Tests**" which will **hold all the unit tests you will write**. Test Project have two classes as well:

- "**PartTests**" - here you should place **all code** testing the "**Part**" and **it's functionality**.
- "**DeviceTests**" – here you should place **all code** testing the "**Device**" and **it's functionality**.

So as you are a good tester, **your job here is to write unit tests on the provided project**. You should **test** exactly **every part** of the structure provided:

- You should test **all the constructors**.
- You should test **all properties** (**getters** and **setters**).
- You should test **all the methods** and **validations** inside the classes.

Before you submit your solution to Judge, you should **remove all the references and namespaces referencing the other project**. You should **upload only** the "**Service.Tests**" project **holding the two classes with your tests**. **Remove** the "**bin**" and "**obj**" folders **before** submission.

Constraints

- **Everything in the provided skeleton is working perfectly fine**.
- **You mustn't change anything in the project structure.**
- **You shouldn't test the auto properties**.
- **Any part of validation should be tested**.
- **There is no limit on the tests you will write but keep your attention on the main functionality**.