**Unit Testing with C# – Practical exam - 09 August 2016**

**IntergalacticTravel**

**Description**

You are given an already built software modules. Figure out how the different modules are working. Check out the test requirements, to know exactly what you must test. For a warm-up - start with the tests that look trivial to you (those that you know you can implement in seconds). Think of the different testing strategies you learned during the course - ***Extending classes, Mocking dependencies, Testing private members through the class public API, etc***. Follow the ***best practices*** for unit testing that we discussed in our ***lectures*** and ***workshops***.

**Requirements**

**Test cases for *IntergalacticTravel.UnitsFactory* (5 tests)**

1. **GetUnit** should return **new Procyon** unit, when a valid corresponding command is passed (i.e. "create unit Procyon Gosho 1");
2. **GetUnit** should return **new Luyten** unit, when a valid corresponding command is passed (i.e. "create unit Luyten Pesho 2");
3. **GetUnit** should return **new Lacaille** unit, when a valid corresponding command is passed (i.e. "create unit Lacaille Tosho 3");
4. **GetUnit** should **throw InvalidUnitCreationCommandException**, when the command passed is not in the valid format described above. **(Check for at least 2 different cases)**

**Test cases for *IntergalacticTravel.ResourcesFactory* (10 tests)**

1. **GetResources** should **return a newly created Resources** object with correctly set up properties(Gold, Bronze and Silver coins), no matter what the order of the parameters is, when the input string is in the correct format. **(Check with all possible cases)**:

***Example:*** The following lines should all create a **new Resources object with 40 Bronze Coins, 30 Silver Coins and 20 Gold Coins**.  
create resources **gold(20) silver(30) bronze(40)**  
create resources **gold(20) bronze(40) silver(30)**  
create resources **silver(30) bronze(40) gold(20)**  
create resources **silver(30) gold(20) bronze(40)**  
create resources **bronze(40) gold(20) silver(30)**  
create resources **bronze(40) silver(30) gold(20)**

1. **GetResources** should throw **InvalidOperationException**, which contains the string **"command"**, when the input string represents an **invalid** command. **(Check with at least 2 different cases)**.  
   Invalid commands are any commands that does not follow the pattern described above.

***Example:***  
create resources x y z  
tansta resources a b  
absolutelyRandomStringThatMustNotBeAValidCommand

1. **GetResources** should throw **OverflowException**, when the input string command is in the correct format, but any of the values that represent the resource amount is larger than **uint.MaxValue**. **(Check with at least 2 different cases)**

***Example:***  
create resources silver(10) gold(97853252356623523532) bronze(20)  
create resources silver(555555555555555555555555555555555) gold(97853252356623523532999999999) bronze(20)  
create resources silver(10) gold(20) bronze(4444444444444444444444444444444444444)

**Test cases for *IntergalacticTravel.TeleportStation* (15 tests)**

1. **Constructor** should set up all of the provided fields **(owner, galacticMap & location)**, when a **new TeleportStation** is created with valid parameters passed to the constructor.
2. **TeleportUnit** should throw **ArgumentNullException**, with a message that contains the string **"unitToTeleport"**, when **IUnit unitToTeleport** is null.
3. **TeleportUnit** should throw **ArgumentNullException**, with a message that contains the string **"destination"**, when **ILocation destination** is null.
4. **TeleportUnit** should throw **TeleportOutOfRangeException**, with a message that contains the string **"unitToTeleport.CurrentLocation"**, when а unit is trying to use the TeleportStation from a distant location (another planet for example).
5. **TeleportUnit** should throw **InvalidTeleportationLocationException**, with a message that contains the string **"units will overlap"** when trying to teleport a unit to a location which another unit has already taken.
6. **TeleportUnit** should throw **LocationNotFoundException**, with a message that contains the string **"Galaxy"**, when trying to teleport a unit to a Galaxy, which is not found in the locations list of the teleport station.
7. **TeleportUnit** should throw **LocationNotFoundException**, with a message that contains the string **"Planet"**, when trying to teleport a unit to a Planet, which is not found in the locations list of the teleport station.
8. **TeleportUnit** should throw **InsufficientResourcesException**, with a message that contains the string **"FREE LUNCH"**, when trying to teleport a unit to a given Location, but the service **costs** more than the unit's current available resources.
9. **TeleportUnit** should **require a payment** from the **unitToTeleport** for the provided services, when all of the validations pass successfully and the unit is ready for teleportation.
10. **TeleportUnit** should **obtain a payment** from the **unitToTeleport** for the provided services, when all of the validations pass successfully and the unit is ready for teleportation, and as a result - the amount of Resources of the **TeleportStation** must be increased by the amount of the payment.
11. **TeleportUnit** should **Set** the **unitToTeleport**'s previous location to **unitToTeleport**'s current location, when all of the validations pass successfully and the unit is being teleported.
12. **TeleportUnit** should **Set** the **unitToTeleport**'s current location to **targetLocation**, when all of the validations pass successfully and the unit is being teleported.
13. **TeleportUnit** should **Add** the **unitToTeleport** to the list of Units of the **targetLocation (Planet.Units)**, when all of the validations pass successfully and the unit is on its way to being teleported.
14. **TeleportUnit** should **Remove** the **unitToTeleport** from the list of Units of the **unit's current location (Planet.Units)**, when all of the validations pass successfully and the unit is on its way to being teleported.
15. **PayProfits** should **return the total amount of profits(Resources)** generated using the TeleportUnit service, when the argument passed represents the **actual owner** of the TeleportStation.

**Test cases for *IntergalacticTravel.BusinessOwner* (1 test)**

1. **CollectProfits** should **increase the owner Resources** by the total amount of Resources generated from the Teleport Stations that are in his possession.

**Test cases for *IntergalacticTravel.Unit* (3 tests)**

1. **Pay** should **throw NullReferenceException** if the object passed is null.
2. **Pay** should **decrease the owner's amount of Resources** by the amount of the cost.
3. **Pay** should **return Resource object** with the amount of resources in the cost object.