

Design Patterns and Software Development Process

Exercise 1 – CustomQueue – Generics

1. Introduction

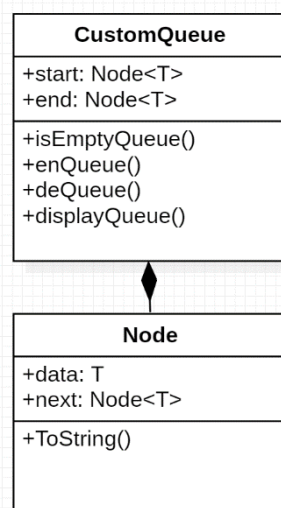
In this exercise we need to construct a custom queue. That's why we're going to create 2 generic classes **CustomQueue** and **Node**. Finally we will write basic methods in order to **Enqueue** and **Dequeue** elements from the Custom Queue.

2. Design Hypotheses

We will create 2 classes. Firstly **Node** class. A **Node** is composed of **data (Type T)** and of a **next Node (Type Node<T>)** in order to link the Queue. Then we will create a **CustomQueue** class composed of an **end** and a **start Node (Type Node<T>)** which allows us to construct our queue.

3. UML diagrams

a. Class diagram of the solution



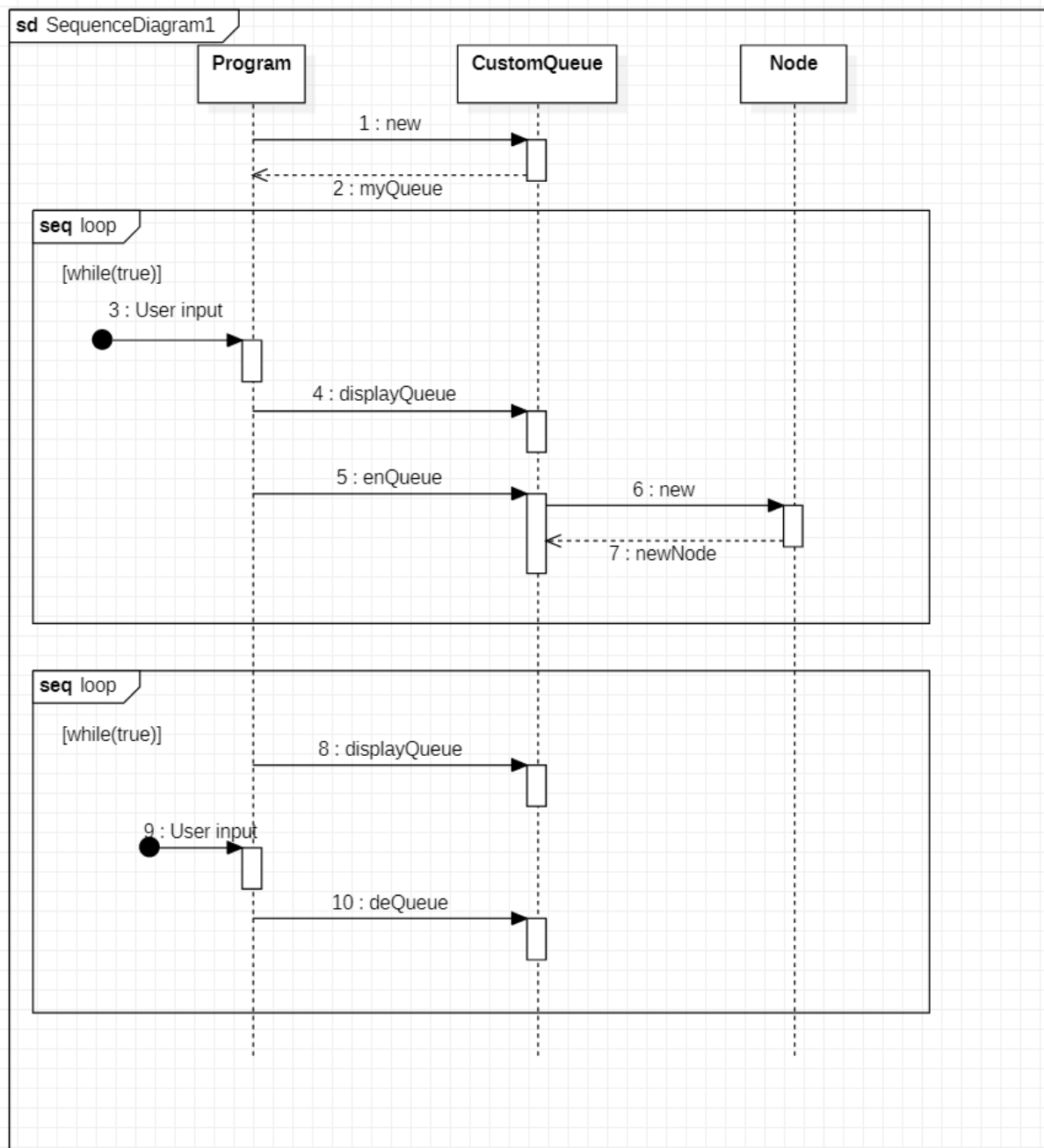
b. Sequence diagrams

Version of our Implementation:

Step 1 : creation of the **CustomQueue<string>**

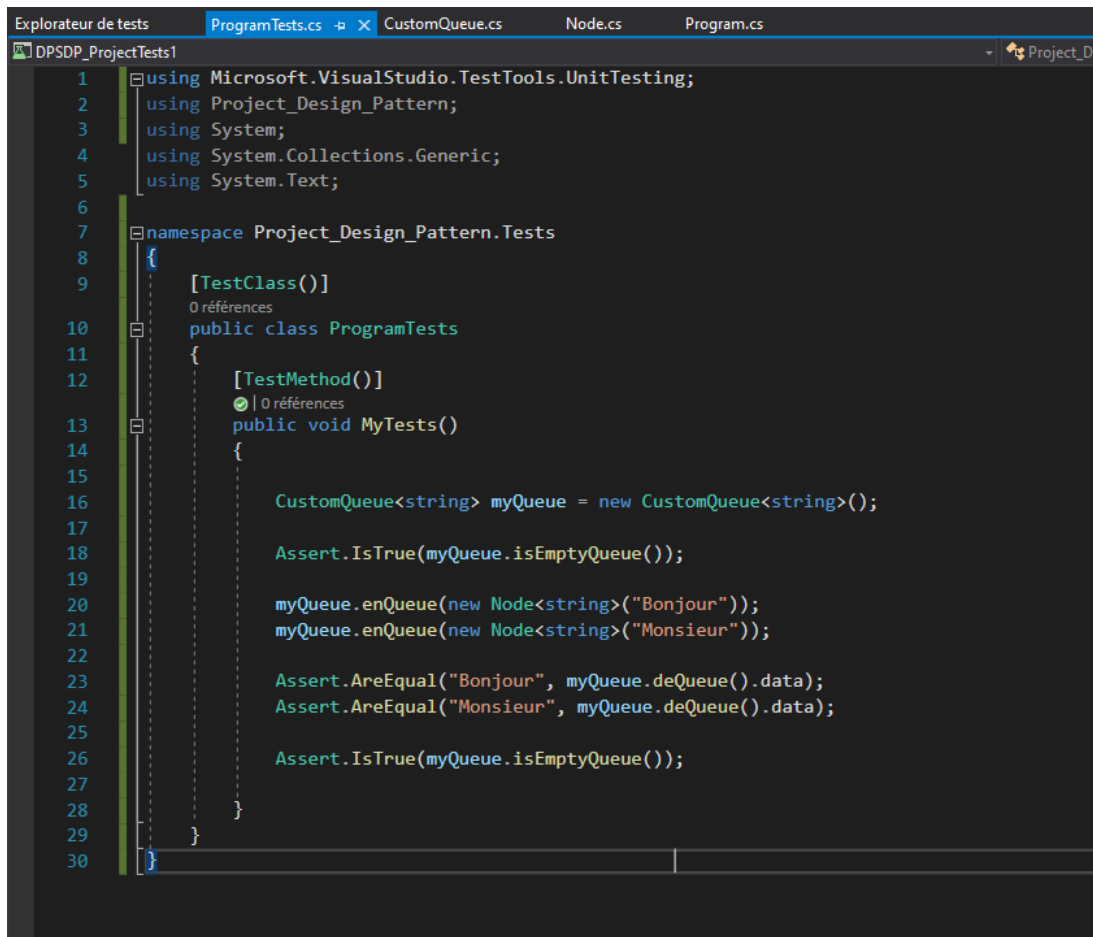
Step 2: ask to user to enqueue elements while true

Step 3: ask to user to dequeue elements while true



4. Test cases

We use `UnitTesting` package of Microsoft VS. Below we will test `EnQueue()`, `DeQueue()`, `isEmptyQueue()` Methods.



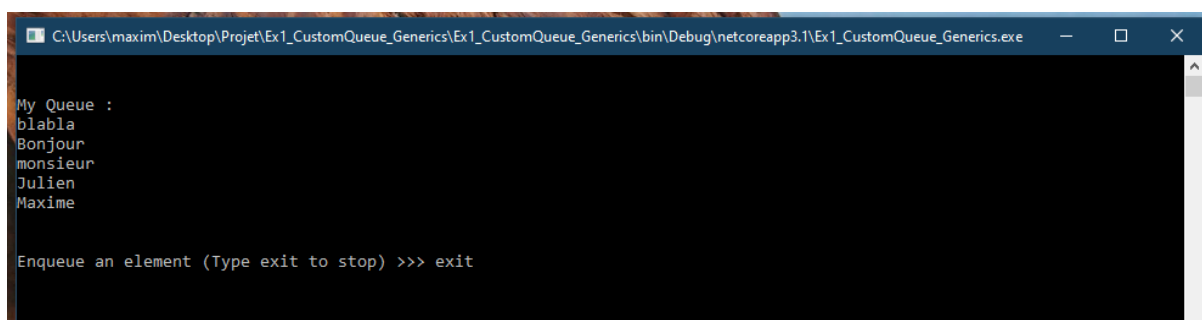
```
1 using Microsoft.VisualStudio.TestTools.UnitTesting;
2 using Project_Design_Pattern;
3 using System;
4 using System.Collections.Generic;
5 using System.Text;
6
7 namespace Project_Design_Pattern.Tests
8 {
9     [TestClass]
10    public class ProgramTests
11    {
12        [TestMethod]
13        public void MyTests()
14        {
15            CustomQueue<string> myQueue = new CustomQueue<string>();
16
17            Assert.IsTrue(myQueue.isEmptyQueue());
18
19            myQueue.enQueue(new Node<string>("Bonjour"));
20            myQueue.enQueue(new Node<string>("Monsieur"));
21
22            Assert.AreEqual("Bonjour", myQueue.deQueue().data);
23            Assert.AreEqual("Monsieur", myQueue.deQueue().data);
24
25            Assert.IsTrue(myQueue.isEmptyQueue());
26        }
27    }
28 }
29
30
```

Test	Durée	Caractéris...	Message d'erreur
✓ DPSPD_ProjectTests1 (1)	9 ms		
✓ Project_Design_Pattern.Tests (1)	9 ms		
✓ ProgramTests (1)	9 ms		
✓ MyTests	9 ms		

All tests succeed !

5. Additional / Final remarks

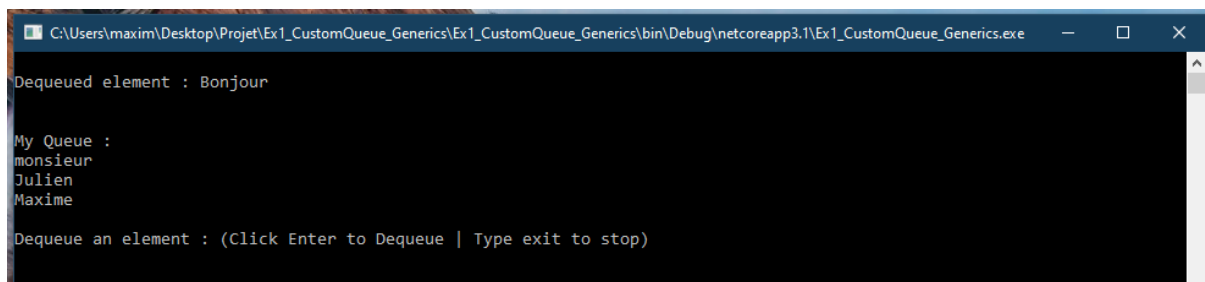
Capture of the execution.



```
C:\Users\maxim\Desktop\Projet\Ex1_CustomQueue_Generics\Ex1_CustomQueue_Generics\bin\Debug\netcoreapp3.1\Ex1_CustomQueue_Generics.exe

My Queue :
blabla
Bonjour
monsieur
Julien
Maxime

Enqueue an element (Type exit to stop) >>> exit
```



```
C:\Users\maxim\Desktop\Projet\Ex1_CustomQueue_Generics\Ex1_CustomQueue_Generics\bin\Debug\netcoreapp3.1\Ex1_CustomQueue_Generics.exe
Dequeued element : Bonjour

My Queue :
monsieur
Julien
Maxime

Dequeue an element : (Click Enter to Dequeue | Type exit to stop)
```

Exercise 2 – MapReduce – Design patterns, Threads & IPC

1. Introduction

MapReduce is a programming model and an associated implementation for processing and generating big data sets with a parallel, distributed algorithm on a cluster.

A MapReduce program is composed of a map procedure, which performs filtering and sorting and a reduce method, which performs a summary operation.

That's what we did.

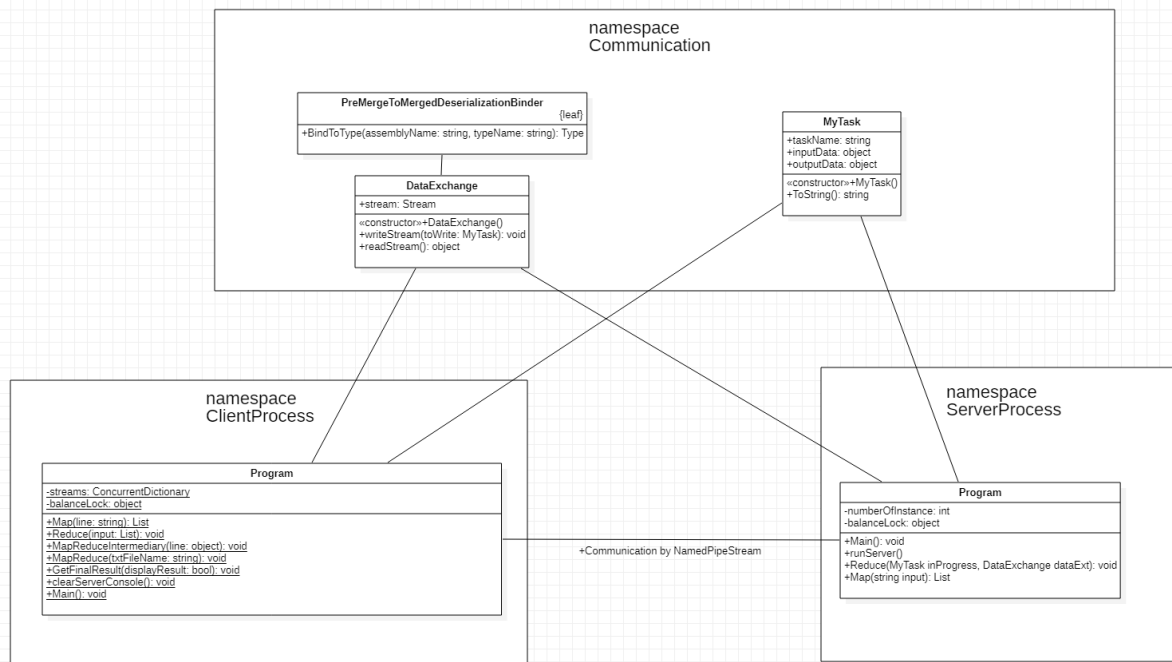
2. Design Hypotheses

In order to do this exercise:

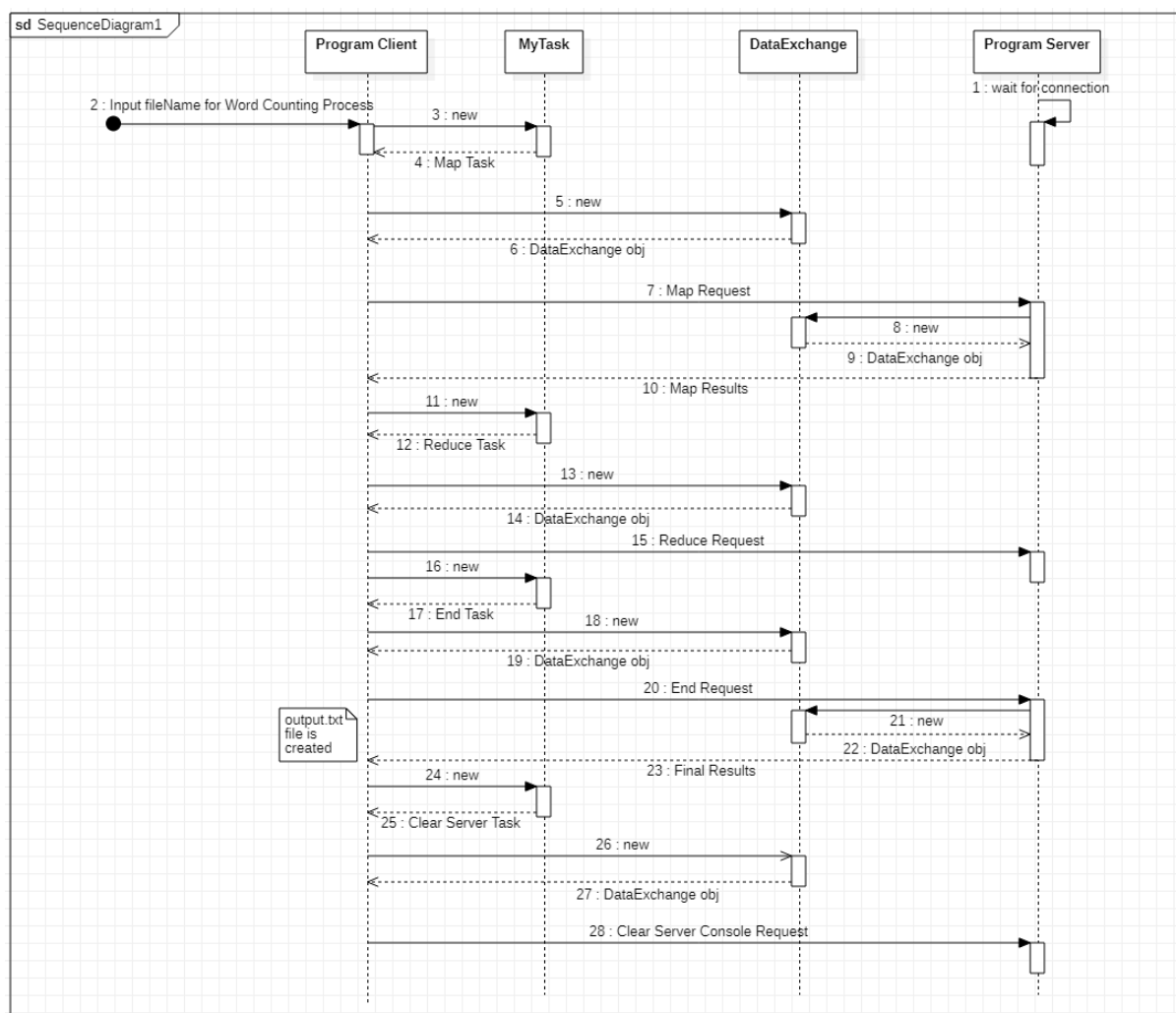
- We created a simulated network of nodes, this is our **ServerProcess**. This one will create 254 computers in cluster (254 instances allowed by **NamedPipeServer Stream**) using **Threads**.
- We created 2 useful classes **MyTask** and **DataExchange**.
 - **MyTask** allows to create "requests". Its attributes are a **taskName** (name of the task to do which allows to be recognized by the Server when this is received), **inputData** (on which we apply our Task) and **outputData** (we store results into it, before returning the results to the client).
 - **DataExchange** allows to send or receive data. It uses the principle of Serialization and Deserialization of **MyTask** Objects throw a **NamedPipe Stream**.
- We created what we called **ClientProcess**. This allows to execute all Task on the server. It first sends Mapping Requests (in Threads), then Reduce Requests (in threads) and finally get the results by reading results on all active client of **NamedPipe Stream** (stored in a **ConcurrentDictionary**).

3. UML diagrams

a. Class diagram of the solution



b. Sequence diagrams



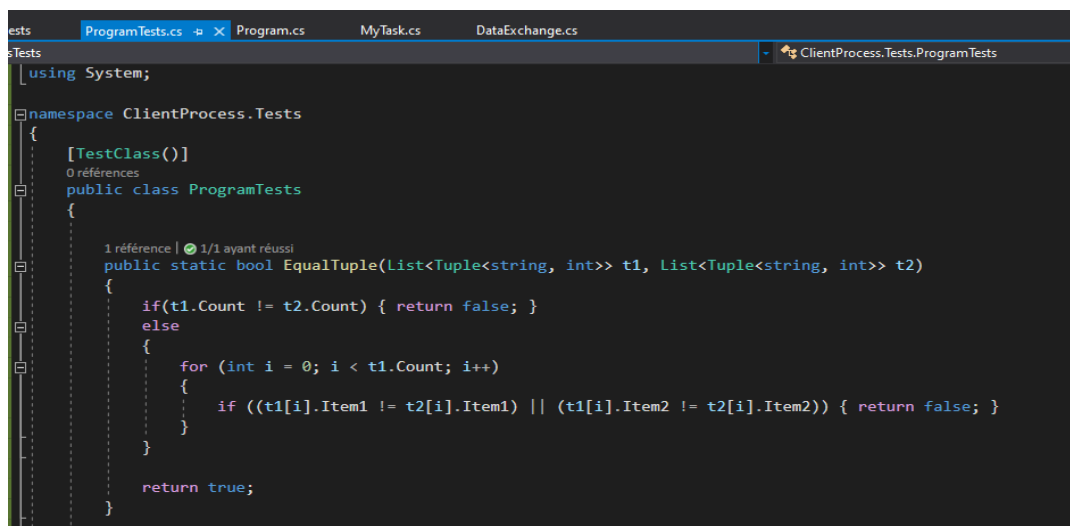
4. Test cases

We use **UnitTesting** package of Microsoft VS.

Test 1: Load the server & load the tests.


In this test we want to **compare reel** and **expected output** of the function **Map()**.

In order to do this, we wrote a util function that test if 2 Tuple are equals.



```
using System;

namespace ClientProcess.Tests
{
    [TestClass()]
    public class ProgramTests
    {
        public static bool EqualTuple(List<Tuple<string, int>> t1, List<Tuple<string, int>> t2)
        {
            if (t1.Count != t2.Count) { return false; }
            else
            {
                for (int i = 0; i < t1.Count; i++)
                {
                    if ((t1[i].Item1 != t2[i].Item1) || (t1[i].Item2 != t2[i].Item2)) { return false; }
                }
            }
            return true;
        }
    }
}
```



```
[TestMethod()]

// Load the Server for this Test case
public void MapTest()
{
    string line = "Bonjour je m'appelle";
    List<Tuple<string, int>> myTupleListExpected = new List<Tuple<string, int>>();

    var t1 = new Tuple<string, int>("Bonjour", 1);
    var t2 = new Tuple<string, int>("je", 1);
    var t3 = new Tuple<string, int>("m'appelle", 1);

    myTupleListExpected.Add(t1);
    myTupleListExpected.Add(t2);
    myTupleListExpected.Add(t3);

    List<Tuple<string, int>> myTupleListResult = Program.Map(line);

    Assert.IsTrue(EqualTuple(myTupleListExpected, myTupleListResult));
}
```

Test 2: Load the server & then load the test.

In this test we want to compare 2 text files, that's why we wrote a util function **FileCompare()**. If expected and reel text files are equals, test succeed.

```
[TestMethod()]
public void compareOutputFiles()
{
    Assert.IsTrue(FileCompare("output1Expected.txt", "output.txt"));
}

1 référence | 1/1 ayant réussi
private bool FileCompare(string file1, string file2)
{
    int file1byte;
    int file2byte;
    FileStream fs1;
    FileStream fs2;

    if (file1 == file2)
    {
        return true;
    }

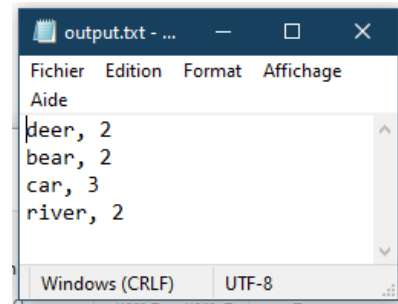
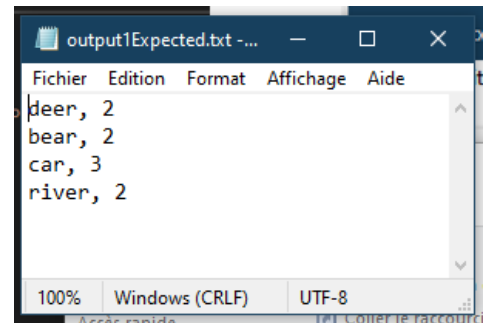
    fs1 = new FileStream(file1, FileMode.Open);
    fs2 = new FileStream(file2, FileMode.Open);

    if (fs1.Length != fs2.Length)
    {
        fs1.Close();
        fs2.Close();
        return false;
    }

    do
    {
        file1byte = fs1.ReadByte();
        file2byte = fs2.ReadByte();
    }
    while ((file1byte == file2byte) && (file1byte != -1));

    fs1.Close();
    fs2.Close();

    return ((file1byte - file2byte) == 0);
}
```



Test Results:

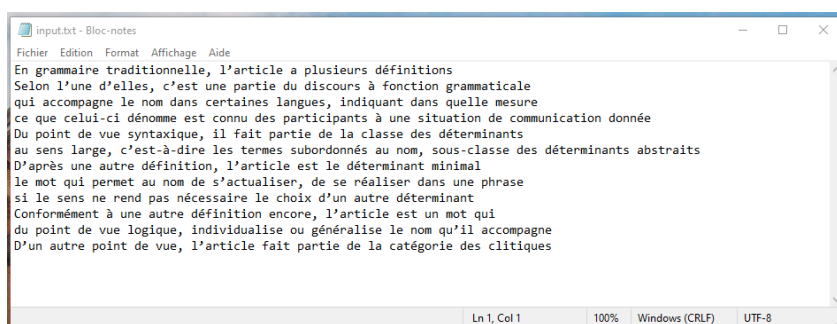
Explorateur de tests			
ProgramTests.cs			
Program.cs			
MyTask.cs			
DataExchange.cs			
2			
Test			
		Durée	Caractéris...
		Message d'erreur	
✓	ClientProcessTests (2)	36 ms	
✓	ClientProcess.Tests (2)	36 ms	
✓	ProgramTests (2)	36 ms	
✓	MapTest	26 ms	
✓	compareOutputFiles	10 ms	

All tests succeed !

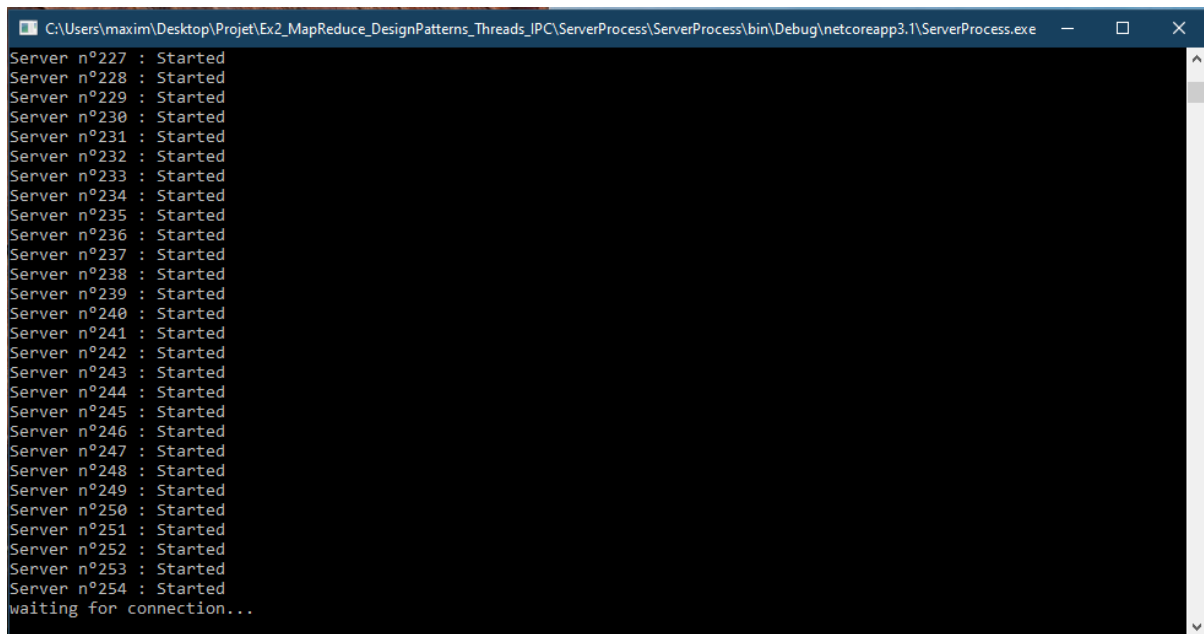
5. Additional / Final remarks

Capture of the execution :

Input.txt file :

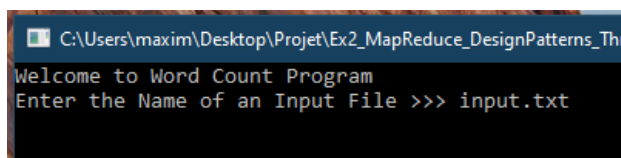


First, load the Server Process:



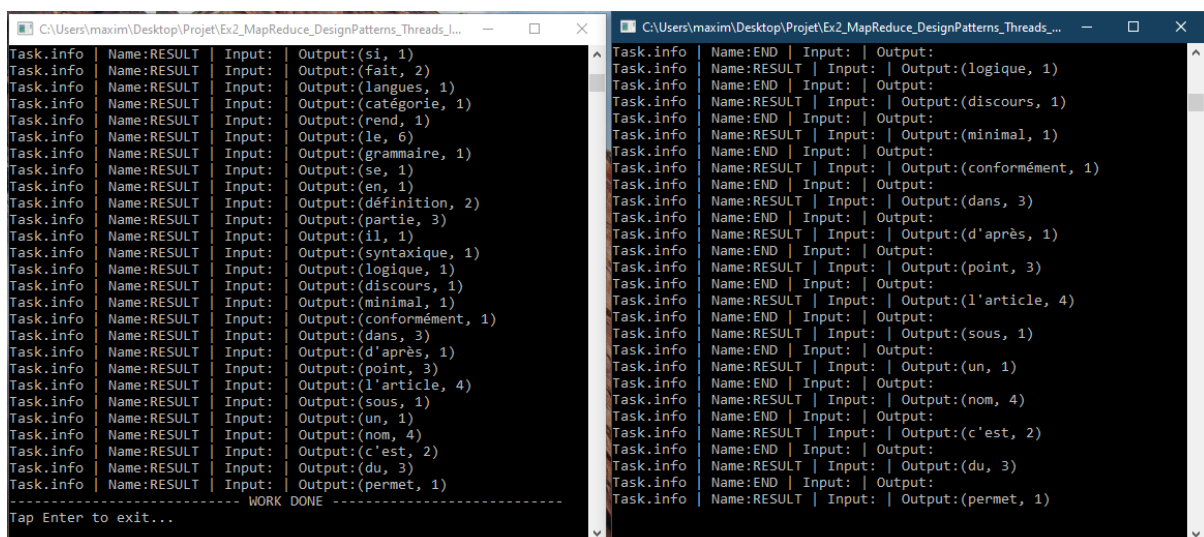
```
C:\Users\maxim\Desktop\Projet\Ex2_MapReduce_DesignPatterns_Threads_IPC\ServerProcess\ServerProcess.exe
Server n°227 : Started
Server n°228 : Started
Server n°229 : Started
Server n°230 : Started
Server n°231 : Started
Server n°232 : Started
Server n°233 : Started
Server n°234 : Started
Server n°235 : Started
Server n°236 : Started
Server n°237 : Started
Server n°238 : Started
Server n°239 : Started
Server n°240 : Started
Server n°241 : Started
Server n°242 : Started
Server n°243 : Started
Server n°244 : Started
Server n°245 : Started
Server n°246 : Started
Server n°247 : Started
Server n°248 : Started
Server n°249 : Started
Server n°250 : Started
Server n°251 : Started
Server n°252 : Started
Server n°253 : Started
Server n°254 : Started
waiting for connection...
```

Then the Client Process, write the input file name, tap ENTER:



```
C:\Users\maxim\Desktop\Projet\Ex2_MapReduce_DesignPatterns_Threads_IPC\ClientProcess\ClientProcess.exe
Welcome to Word Count Program
Enter the Name of an Input File >>> input.txt
```

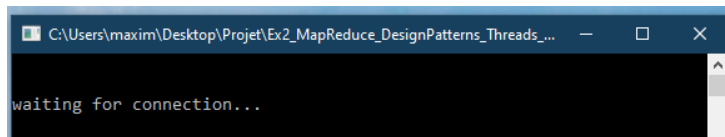
Client (at Left) and Server (at Right) are communicating together to the end of the MapReduce Request.



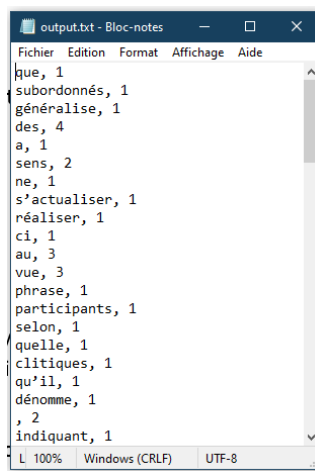
```
C:\Users\maxim\Desktop\Projet\Ex2_MapReduce_DesignPatterns_Threads_IPC\ClientProcess\ClientProcess.exe
Task.info | Name:RESULT | Input: | Output:(si, 1)
Task.info | Name:RESULT | Input: | Output:(fait, 2)
Task.info | Name:RESULT | Input: | Output:(langues, 1)
Task.info | Name:RESULT | Input: | Output:(catégorie, 1)
Task.info | Name:RESULT | Input: | Output:(rend, 1)
Task.info | Name:RESULT | Input: | Output:(le, 6)
Task.info | Name:RESULT | Input: | Output:(grammaire, 1)
Task.info | Name:RESULT | Input: | Output:(se, 1)
Task.info | Name:RESULT | Input: | Output:(en, 1)
Task.info | Name:RESULT | Input: | Output:(définition, 2)
Task.info | Name:RESULT | Input: | Output:(partie, 3)
Task.info | Name:RESULT | Input: | Output:(il, 1)
Task.info | Name:RESULT | Input: | Output:(syntaxique, 1)
Task.info | Name:RESULT | Input: | Output:(logique, 1)
Task.info | Name:RESULT | Input: | Output:(discours, 1)
Task.info | Name:RESULT | Input: | Output:(minimal, 1)
Task.info | Name:RESULT | Input: | Output:(conformément, 1)
Task.info | Name:RESULT | Input: | Output:(dans, 3)
Task.info | Name:RESULT | Input: | Output:(d'après, 1)
Task.info | Name:RESULT | Input: | Output:(point, 3)
Task.info | Name:RESULT | Input: | Output:(l'article, 4)
Task.info | Name:RESULT | Input: | Output:(sous, 1)
Task.info | Name:RESULT | Input: | Output:(un, 1)
Task.info | Name:RESULT | Input: | Output:(nom, 4)
Task.info | Name:RESULT | Input: | Output:(c'est, 2)
Task.info | Name:RESULT | Input: | Output:(du, 3)
Task.info | Name:RESULT | Input: | Output:(permet, 1)
----- WORK DONE -----
Tap Enter to exit...

C:\Users\maxim\Desktop\Projet\Ex2_MapReduce_DesignPatterns_Threads_IPC\ServerProcess\ServerProcess.exe
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(logique, 1)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(discours, 1)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(minimal, 1)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(conformément, 1)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(dans, 3)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(d'après, 1)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(point, 3)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(l'article, 4)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(sous, 1)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(un, 1)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(nom, 4)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(c'est, 2)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(du, 3)
Task.info | Name:END | Input: | Output:
Task.info | Name:RESULT | Input: | Output:(permet, 1)
```

When you Tap Enter at the Client side, Server side clear its Console and is ready for new connections.



Finally we can get the output.txt file in the bin repository (Client side):



Exercise 3 – A Monopoly™ game – Design patterns

1. Introduction

For the third part, we have to build a simple Monopoly Game implementing a game board with his players and a jail system. A player goes to jail if he lands on position 30 or has 3 times same dices' values.

2. Design Hypotheses

To implement this game, we created 5 classes: Player, Observer, PairOfDice, GameBoard and Game.

The class Player is used to instantiate new players having a name, a position, a jail Boolean and a jail counter.

To create a new game, we have to call the function Game, which also instantiates a new GameBoard for the game and a new PairOfDice.

Included in the Observer class:

- Observer Pattern:

Allows us to be notified when the status of a player changes (i.e. when he goes to jail).

- Factory Pattern:

Allows us to instantiate a new Player without having to call new Player().

+ Checks that 2 players don't have the same name when created.

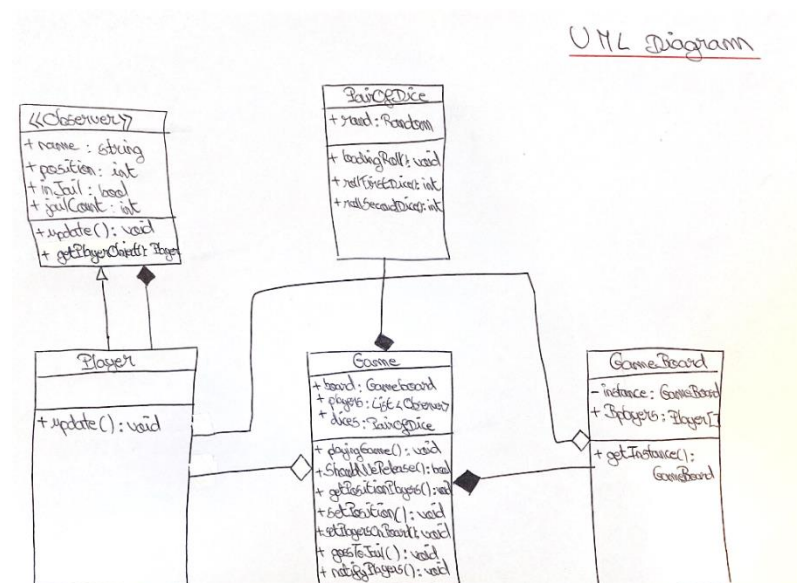
Included in the GameBoard class:

- Singleton Pattern:

Ensures that one and only one instance of GameBoard is created for the Game.

3. UML diagrams

a. Class diagram of the solution



4. Test cases

The function `shouldWeReleasePlayer` determines if a player should be released depending on his `jailCount` value. If this value reaches 3, then he is released. Otherwise, he stays in jail and the counter is incremented by one.

We have 2 Test : one for testing the case where he can leave the jail, and the other one where he can't.

```
[TestMethod]
✓ | 0 références
public void shouldWeReleasePlayer_PlayerIsInJail_ReturnsFalse()
{
    //Arrange
    var game = new Game();

    Player a = Observer.getPlayerObject("a");
    Player b = Observer.getPlayerObject("b");
    Player c = Observer.getPlayerObject("c");
    Player d = Observer.getPlayerObject("d");
    Player e = Observer.getPlayerObject("e");

    Player[] azertyuiop = new Player[] { a, b, c, d, e };

    game.setPlayersOnBoard(azertyuiop);

    //Act
    var result = game.shouldWeReleasePlayer(0);

    //Assert
    Assert.IsFalse(result);
}
```

```
[TestMethod]
✓ | 0 références
public void shouldWeReleasePlayer_PlayerIsInJail_ReturnsTrue()
{
    //Arrange
    var game = new Game();

    Player a = Observer.getPlayerObject("a");
    Player b = Observer.getPlayerObject("b");
    Player c = Observer.getPlayerObject("c");
    Player d = Observer.getPlayerObject("d");
    Player e = Observer.getPlayerObject("e");

    a.jailCount = 2;

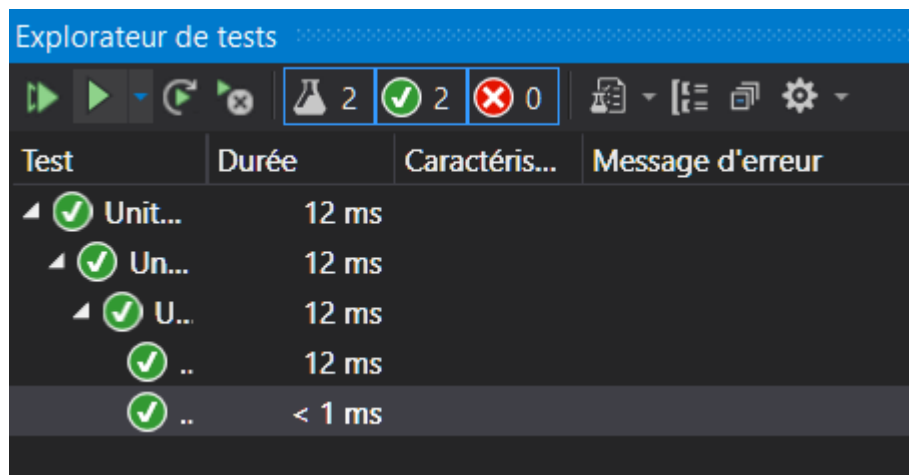
    Player[] azertyuiop = new Player[] { a, b, c, d, e };

    game.setPlayersOnBoard(azertyuiop);

    //Act
    var result = game.shouldWeReleasePlayer(0);

    //Assert
    Assert.IsTrue(result);
}
```

For both test, we get the expected results :



Test	Durée	Caractéris...	Message d'erreur
Unit...	12 ms		
Un...	12 ms		
U...	12 ms		
..	12 ms		
..	< 1 ms		

5. Additional / Final remarks

Screenshots of the beginning of the game and the first few players moving:

```
| Five players will confront each other in this Monopoly Game |
There will be 5 players in this game : Horse, Dog, Shoe, Iron, Hat.
How many rounds do you want to play ? Enter your choice :
5
Press Enter to continue...

Rolling Dices...
Player Horse moves to position 8 !

Press Enter to continue...

Rolling Dices...
Player Dog moves to position 8 !

Press Enter to continue...
```

Playing 2 times in a row and going to jail because the player moved to position 30:

```
Rolling Dices...
Player Hat moves to position 21 !
-- The two dices have the same value, Hat plays again !
Rolling Dices...
Player Hat moves to position 30 !
-----
- My buddy Hat is in jail ! -
-----
```

Screenshot of the program displaying when a player goes to jail when he had the same dices values 3 times in a row:

```
Rolling Dices...
Player Hat moves to position 28 !
-- The two dices have the same value, Hat plays again !
Rolling Dices...
Player Hat moves to position 0 !
-- The two dices have the same value, Hat plays again !
Rolling Dices...
Player Hat moves to position 6 !
-----
- My buddy Hat is in jail ! -
-----
```