|  |
| --- |
| ESILV |
| Report Final project |
| Design Pattern & Soft dev |

|  |
| --- |
| GOUBET Victor - Daems Chloé – DIA2  11/01/2021 |



**Exercice 1 – CustomQueue – Generics**

1. Introduction

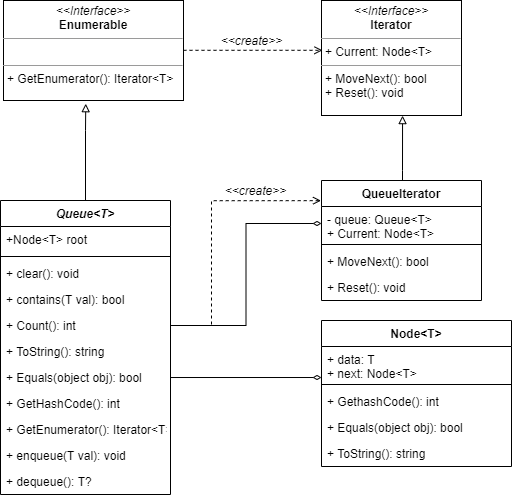
The goal here is to create from scratch an entire generic queue class. It must have all the major functions and characteristics of the provided queue as the capacity to be used in a foreach loops.

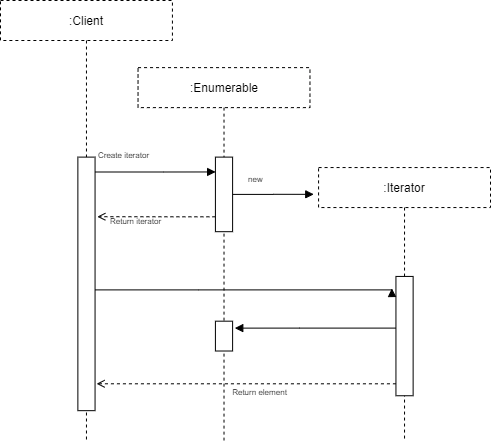
1. Design Hypotheses

For this problem, we use the model of a queue composed of nodes. Each Nodes have a generic value and a next node. A queue has a root which is the foot of the queue. In order to make the custom queue iterable, we have implemented the iterator model. Thus, we can iterate throw the nodes of a queue.

1. UML diagrams

**Class diagramm :**





**Sequence diagramm :**

1. Test cases

We made a total of 10 tests in order to assert that our Custom queue and its methods works properly. We checked that dequeue(), clear(), enqueue(), count(),contains() works well even with empty queues. We’re happy to say that all our tests appeared to be correct, which lets us think that our code works properly.

1. Additional / Final remarks

We add the constraint where T: struct to be able to return a nullable type when we dequeue an empty queue. Thus, the dequeue method return a T? type.

Moreover, an improvement would be to add the possibility to iterate throw the T data instead of Node<T>

**Exercice 2 – MapReduce**

1. Introduction

We want here to create a basic implementation of the MapReduce function. The goal is to give an input to the algorithm, it divides it in subset and apply a map function on each subset. Then it reduce / resume all results in one. It allows to process huge number of data quickly.

1. Design Hypotheses

For this exercise we will make some hypotheses. First, we simulate independent machine thanks to a class called. This class take an input and a really generic function as arguments. Thus, we have a first step where we apply all map function on each subset throw different workers (each worker executes a map function with a thread), then we shuffle the result and finally we apply all the reduce function in parallel ways.

Moreover, we must define a data exchange protocol so that the data and map/reduce function be understandable by our MapReduce class. To do that we use three types: the type of the key and value of the input, the type of the intermediate values after the map step and finally the type of the result. The input must be an iterable collection of Key/Value pair , for example it can be a . This choice seems to be relevant because all type of input can be formatted under this shape. In all our class we use a lot this type: because it is really generic and that why we want!

To force users to define well their map/reduce function we define delegates with these signatures:

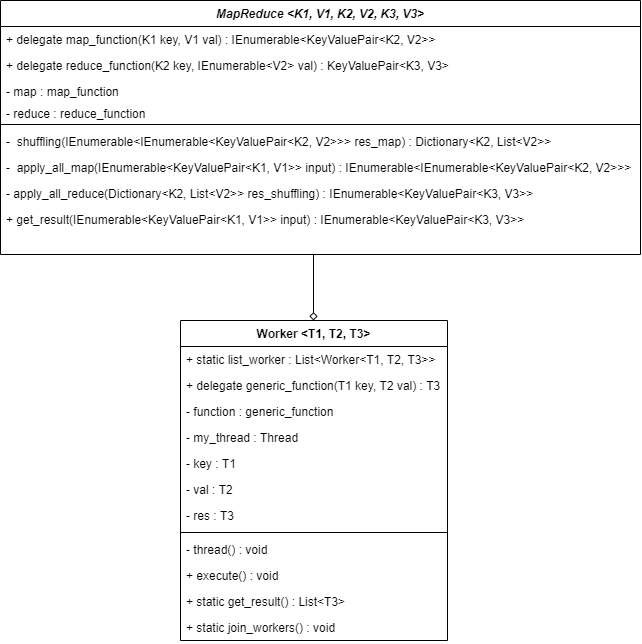


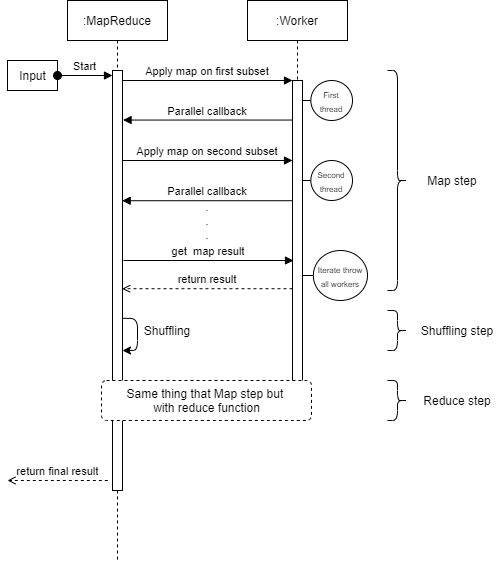
And the signature of the function of a worker have this signature (more generic):

Finally, to get the result of workers, each worker keeps his result in his property “res” and at the end of a step we iterate throw all workers and get all results.

1. UML diagrams

**Class diagramm :**





**Sequence diagramm :**

1. Test cases

For the tests, we have defined two-unit test:

* The first check if our MapReduce works on the word counting example
* The second check if our MapReduce works on the matrix columns sum. By the way, for the example we have define a dictionary of matrix. We can imagine that the original matrix, before being split, was all these matrixes concatenate by rows.

**Exercice 3 – CustomQueue – Generics**

1. Introduction
2. Design Hypotheses
3. UML diagrams
4. Test cases
5. Additional / Final remarks