Exercises: Unit Testing

This document defines the **in-class exercises** assignments for the "High-Quality Code" course @ Software University.

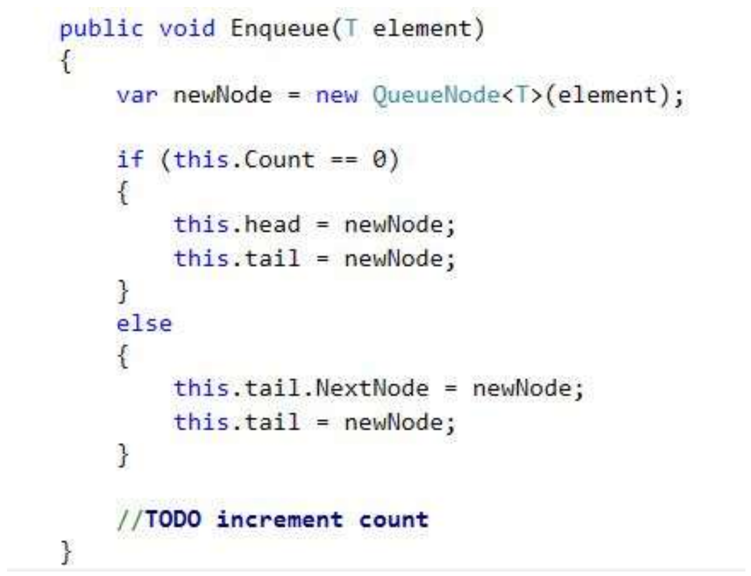
Problem 1. Get Implement Linked List Based Queue

A queue is a basic linear data structure which allows for pushing elements and its end, and retrieving elements from its front. It's really simple to implement, maintain and test.

You are given a skeleton which contains the basic methods a queue should support, but the methods are not implemented yet. It's your task to write them and then test them.

Implement Enqueue Method

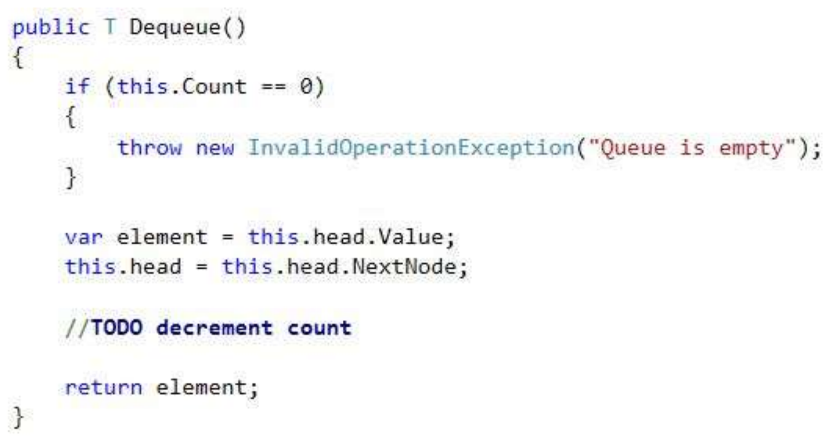
The Enqueue() method should add a new element to the end of the queue. You've already noticed that we keep two pointers – **head** and **tail.** If the queue has no elements in it, both the head and the tail should point to the same **queue node.** Otherwise, the new element is set as the **next** element pointed at by the queue **tail**, after which we change the tail. In both scenarios, we should increment the count of elements;



```
public void Enqueue(T element)
{
    var newNode = new QueueNode<T>(element);

    if (this.Count == 0)
    {
        this.head = newNode;
        this.tail = newNode;
    }
    else
    {
        this.tail.NextNode = newNode;
        this.tail = newNode;
    }

    //TODO increment count
}
```

Implement Dequeue Method

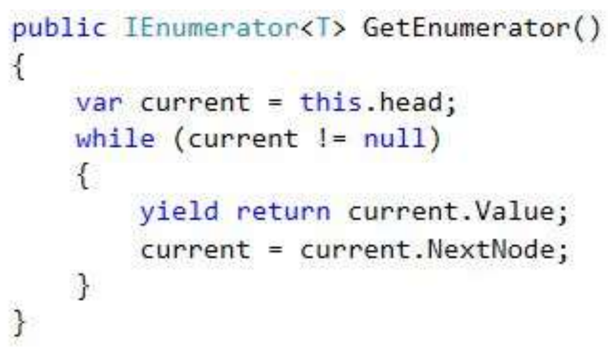
The Dequeue() should return and remove the first element of our queue. The only specific thing is that we should throw and exception of the queue is empty, to notify the developer that something is wrong.



```
public T Dequeue()
{
    if (this.Count == 0)
    {
        throw new InvalidOperationException("Queue is empty");
    }

    var element = this.head.Value;
    this.head = this.head.NextNode;

    //TODO decrement count

    return element;
}
```

Implement IEnumerable<T>

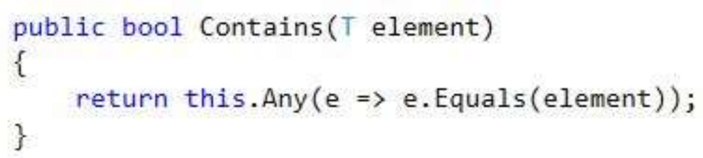
The IEnumerable<T> allows us to use our data structure in a **foreach** loop. We should implement the GetEnumerator() method. We will go through our elements in our collection until the NextNode points at null.



```
public IEnumerator<T> GetEnumerator()
{
    var current = this.head;
    while (current != null)
    {
        yield return current.Value;
        current = current.NextNode;
    }
}
```

Implement Contains

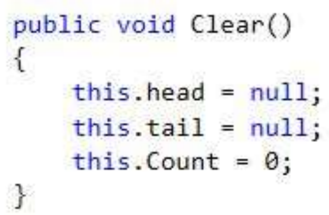
The contains method will return a boolean which tells us whether we have a specific element in our collection or not. Since we are lazy programmers and have already implemented IEnumerable<T>, we can use the always handy LINQ extensions



```
public bool Contains(T element)
{
    return this.Any(e => e.Equals(element));
}
```

Implement Clear

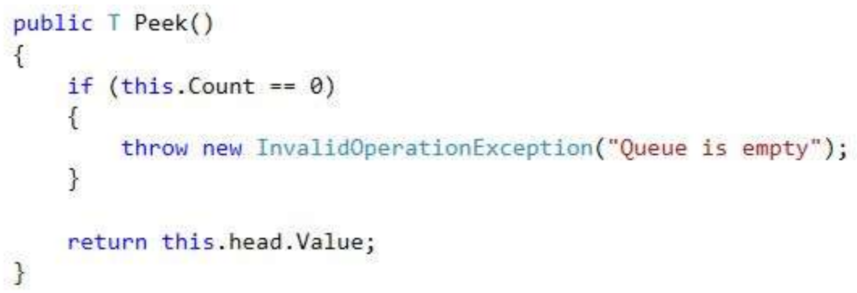
The clear should remove all the contents of our queue and set its count to zero. We will make the head and tail point to null and let the garbage collector take care of any remaining queue nodes.



```
public void Clear()
{
    this.head = null;
    this.tail = null;
    this.Count = 0;
}
```

Implement Peek

The Peek() method is usually implemented in stacks and queues. It should return the value of the front element without removing it from the data structure. It's really trivial to create. We should only check if our collection is not empty.



```
public T Peek()
{
    if (this.Count == 0)
    {
        throw new InvalidOperationException("Queue is empty");
    }

    return this.head.Value;
}
```

Problem 2. Create a New Test Project

In Visual Studio, select **File > New > Project…** In the left part of the dialog box, select **Visual C# > Test** and then select **Unit Test Project**. Name your new project and solution accordingly.

Write tests for the Dequeue method.

- Enqueue an element and then Dequeue it. Test if the queue returns the expected element.
- Enqueue several elements and then Dequeue them. Test if each of the elements is the expected one.
- Enqueue several elements and test the Count of the queue.
- Enqueue several elements, Dequeue several and the test the Count again
- Try to use Dequeue with and empty queue. Make sure that the collection throws the correct exception.