5. Übungsblatt, Thema: Sortierung und Queue

Einführung:

In der Welt von Computern ist es oft erforderlich Elemente zu sortieren. Deswegen sollte man auch in der Lage sein so was selbst zu implementieren. Verkette Datenstrukturen können sehr komplexe Operationen beinhalten. Manchmal reicht auch eine simplere Variante aus. Solche Varianten lassen sich dann auch leichter verwalten. Als Beispiel wird hier eine Queue implementiert.

1. GetStringArray Method

Beschreibung:

Um später leichter den Code zu Debuggen, soll erst mal eine Funktion implementiert werden, welche eine textuelle Presentation eines Arrays ausgibt.

Beispiel einer textuellen Ausgabe zum einem Array

> *mit array = new int[] {2, 4, 5 } => [2, 4, 5]*

Falls ein leeres array kommt soll folgende Ausgabe kommen
[]

Deklaration:

```
static string GetStringArray(int[] array)
```

Test Program:

```
int[] array = new int[] { 2, 4, 5};
Console.WriteLine(GetStringArray(array));
array = new int[0];
Console.WriteLine(GetStringArray(array));
```

Ausgabe sollte sein:

[2 4 5]
[]

---

2. Sortierung eines Arrays

Beschreibung:

Nun soll die Sortierung eines Arrays vorgenommen werden. Diese Function soll eine gegebenes Array direkt sortieren.

Diese Function soll der übersichtshalber statisch in einer Klasse Namens Utility geschrieben werden.

Tipp: Es gibt viele Wege eine Array zu sortieren. Eine einfache Variante ein Array zu sortieren ist der BubbleSort. Ein Suchmaschine kann dazu mehr Informationen liefern.

Deklaration:

```
public static class Utility
{
    public static void Sort(int[] numbers)
    {

    }
}
```

Test Program:

```
var array = new int[] { 0, -2, 7 };
Utility.Sort(array);
Console.WriteLine(GetStringArray(array));
array = new int[] { -2, -90, -5, -89 };
Utility.Sort(array);
Console.WriteLine(GetStringArray(array));
array = new int[] { -8, 5, -89, 0, 2, 4 };
Utility.Sort(array);
Console.WriteLine(GetStringArray(array));
array = new int[] { 0 };
Utility.Sort(array);
Console.WriteLine(GetStringArray(array));
array = new int[0];
Utility.Sort(array);
Console.WriteLine(GetStringArray(array));
array = null;
Utility.Sort(array);
Console.WriteLine(GetStringArray(array));
array = new int[] { -8, 5, 5, 0, 2, 4 };
Utility.Sort(array);
Console.WriteLine(GetStringArray(array));
```

Ausgabe sollte sein:

[-2 0 7]
[-90 -89 -5 -2]
[-89 -8 0 2 4 5]
[0]
[]
[]
[-8 0 2 4 5 5]

---

3. NumberQueue | Enqueue und ToNumberArray

Beschreibung:

Es soll eine Klasse namens NumberQueue erstellt werden. Diese Klasse soll eine Queue darstellen.

Elemente befinden sich in der Queue und jedes Element hat einen Wert. Innerhalb der Queue wird mit Elementen gearbeitet. Außerhalb der Klasse werden Werte übergeben und erhalten.

Es soll als erstes die 2 Methoden Enqueue und ToNumberArray implementiert werden.

Bei Enqueue soll ein Integrer Wert angenommen werden. Nach der Ausführung soll sich ein Element mit diesen Integrer Wert am Ende der Queue befinden.

Bei ToNumberArray soll ein Array zurück gegeben werden. Diese Array soll alle Werte der Queue enthalten. Wichtig ist, dass die Queue nach der Ausführung unverändert bleibt. Array soll nur so groß sein wie die Anzahl der Werte in der Queue.

Tipp: Queues sollten die Properties Head und Tail haben. Head ist das 1. Element in der Queue und Tail ist das letzte hinzugefügte Element in der Queue. In einer Queue muss jedes Element nur seinen Nachfolge kennen.

Deklaration:

```
public void Enqueue(int number)
public int[] ToNumberArray()
```

Test Program:

```
var queue = new NumberQueue();
Console.WriteLine(GetStringArray(queue.ToNumberArray()));
queue.Enqueue(5);
queue.Enqueue(10);
Console.WriteLine(GetStringArray(queue.ToNumberArray()));
queue.Enqueue(-5);
queue.Enqueue(-10);
queue.Enqueue(20);
Console.WriteLine(GetStringArray(queue.ToNumberArray()));
```

Ausgabe sollte sein:

[]
[5 10]
[5 10 -5 -10 20]

---

4. NumberQueue | Dequeue

Beschreibung:

Nun soll eine Method Dequeue implementiert werden. Nach der Ausführung soll der Wert von dem letztem Element zurück gegeben werden. Nach der Ausführung soll das letzte Element aus der Queue entfernt sein.

Deklaration:

```
public int Dequeue()
```

Test Program:

```
var queue = new NumberQueue();
queue.Enqueue(2);
queue.Dequeue();
Console.WriteLine(GetStringArray(queue.ToNumberArray()));
queue.Enqueue(2);
queue.Enqueue(8);
queue.Dequeue();
Console.WriteLine(GetStringArray(queue.ToNumberArray()));
```

Ausgabe sollte sein:

[]
[8]

---

5. NumberQueue | Count und Empty

Beschreibung:

Es sollen nun die Properties Count and Empty implementiert werden.

Das Property Count soll die Anzahl aller Elemente in der Queue zurück gegeben. Das Property Empty soll true ausgeben falls die Queue keine Elemente hat.

Deklaration:

```
public bool Empty
public int Count
```

Test Program:

```
var queue = new NumberQueue();
queue.Enqueue(2);
queue.Dequeue();
Console.WriteLine($"queue.Count = {queue.Count}");
queue.Enqueue(2);
queue.Enqueue(8);
queue.Dequeue();
Console.WriteLine($"queue.Count = {queue.Count}");
queue.Enqueue(8);
queue.Enqueue(8);
queue.Enqueue(8);
Console.WriteLine($"queue.Count = {queue.Count}");
queue = new NumberQueue();
Console.WriteLine($"queue.Empty = {queue.Empty}");
queue.Enqueue(2);
Console.WriteLine($"queue.Empty = {queue.Empty}");
queue.Dequeue();
Console.WriteLine($"queue.Empty = {queue.Empty}");
```

Ausgabe sollte sein:

queue.Count = 0
queue.Count = 1
queue.Count = 4
queue.Empty = True
queue.Empty = False
queue.Empty = True
