Testaufgabe C# SS 2019

Entwicklung eines sicheren Zwischenspeichers für Integer Werte

Zu entwickeln ist in Einzelschritten die Erweiterung eines vorgegebenen Datenspeichers für Integer Werte.

Projekterstellung

In einem ersten Schritt ist ein C# Konsolenprojekt *BufferTest* zu erstellen.

Einbringen der Klasse *Buffer*

Vorgegeben ist eine Klasse *Buffer*:

```
class Buffer
{
    protected int[] data;
    protected int head, tail;
    protected int size;

    public Buffer(int size)
    {
        data = new int[size];
        this.size = size;
        head = 0;
        tail = 0;
    }

    public virtual void Put(int val)
    {
        data[tail] = val;
        tail = (tail + 1) % size;
    }

    public virtual int Get()
    {
        int val = data[head];
        head = (head + 1) % size;
        return val;
    }

    public int Length
    {
        get { return (tail + size - head) % size; }
    }
}
```

Mit Hilfe des Klassenassistenten ist eine Klasse *Buffer* dem Projekt hinzuzufügen und der entstehenden leeren Klasse der oben angegebene Code hinzuzufügen. Versuchen Sie aus dem Code der Klasse *Buffer* das Verhalten des Datenspeichers, insbesondere an seinen Grenzen, zu erschließen.

Erstellen eines Indexers für die Klasse *Buffer*

Die Klasse *Buffer* ist um einen Indexer zu erweitern, der es ermöglicht, ähnlich einem Array Zugriff, über einen Index auf die Daten des Zwischenspeichers zuzugreifen. Der Indexer soll einen lesenden und schreibenden Zugriff auf den Datenspeicher der Klasse erlauben.

Main Methode zum Test des Buffers

Zu einem ersten Test des um den Indexer erweiterten Datenspeichers kann die folgende Main Methode verwendet werden:

```
namespace BufferTest
{
    class Program
    {
        static void Main(string[] args)
        {
            Buffer buf = new Buffer(8);
            buf.Put(3);
            buf.Put(4);
            buf.Put(5);

            Console.WriteLine("First value = " + buf.Get());
            for (int i = 0; i < buf.Length; i++)
            {
                Console.WriteLine("buf[{0}] = {1}", i, buf[i]);
            }

            Console.ReadLine();
        }
    }
}
```

Erstellen der Klasse *SafeBuffer*

Mit Hilfe des Klassenassistenten ist nun das Projekt um eine Klasse *SafeBuffer* zu erweitern.

Im Modul der Klasse *SafeBuffer* definieren wir ein Delegat und ein Interface:

```
delegate void Processor(int val);

interface IProcessable
{
       void ApplyToEach(Processor proc);
}
```

Die Klasse *SafeBuffer* ist von der Klasse *Buffer* abgeleitet und implementiert das Interface *IProcessable.* Zudem erhält die Klasse das Attribut *int elements*, mit dem die Anzahl der im Puffer gehaltenen Elemente gezählt wird.

Für die Klasse *SafeBuffer* ist ein Konstruktor zu definieren, dem als Argument ebenso wie der Konstruktor der Klasse *Buffer* die Größe des Datenspeichers (*size*) übergeben wird. Dieser Konstruktor ruft den Konstruktor der Basisklasse auf und initialisiert das Attribut *elements* mit *0*.

Die Methode *ApplyToEach* des Interfaces *IProcessable* ist so zu implementieren, dass in einer Schleife über alle Elemente des Datenspeichers die an der Schnittstelle übergebene Methode auf das Element angewendet wird.

Im gleichen Modul wie die Klasse *SafeBuffer* sind zwei Ausnahmeklassen hinzuzufügen:

```
class BufferOverflowException : Exception
{
    public int val;
    public BufferOverflowException(int x)
    {
        val = x;
    }
}

class BufferUnderflowException : Exception { }
```

Die beiden Methoden *Put* und *Get* der Basisklasse *Buffer* sind nun zu überschreiben:

Für die Methode *Put* ist zu prüfen, ob noch Platz im Puffer ist. Ist dies der Fall, wird mit Hilfe der Methode *Put* der Basisklasse das als Argument übergebene Element dem Puffer zugefügt. Andernfalls wird die Exception *BufferOverflowException* geworfen.

Für die Methode *Get* ist zu prüfen, ob sich Elemente im Puffer befinden. Ist dies der Fall, so wird mit Hilfe der Methode *Get* der Basisklasse das oberste Element zurückgegeben. Im anderen Fall wird die Ausnahme *BufferUnderFlowException* geworfen.

Achten Sie auch darauf, dass die beiden überschriebenen Methoden *Put* und *Get* das Attribut *elements* ggfs. entsprechend anpassen müssen.

Main Methode zum Test der neuen Klasse *SafeBuffer*

Um die Klasse *SafeBuffer* testen zu können, kann unsere Main Methode durch das folgende Listing ersetzt werden. Dabei definieren wir eine statische Methode *Print* zur Ausgabe auf der Konsole, um ein Delegatobjekt zu erzeugen.

Die Elemente, die in den Buffer einzubringen sind, werden dem Hauptprogramm als Parameter übergeben. Diese Argumente können im Eigenschaftendialog des Projekts unter dem Punkt ‚Debuggen' eingetragen werden.

```
namespace BufferTest
{
    class Program
    {
        private static void Print(int val)
        {
            Console.WriteLine(val);
        }

        static void Main(string[] args)
        {
            SafeBuffer buf = new SafeBuffer(4);

            try
            {
                foreach (string s in args)
                {
                    buf.Put(Convert.ToInt32(s));
                }
            }
            catch (BufferOverflowException e)
            {
                Console.WriteLine("--{0} caused overflow", e.val);
            }
            catch (BufferUnderflowException)
            {
                Console.WriteLine("--buffer underflow");
            }
            catch
            {
                Console.WriteLine("--conversion error");
            }

            Console.ReadLine();
        }
    }
}
```