# Class Kata „Ringbuffer“

Entwickle eine Klasse, die einen Ringbuffer [**1**, **2**] implementiert.

An einen Ringbuffer können Werte „hinten“ angehängt werden wie an eine Queue (*Add()*). Und sie können „vorne“ entnommen werden wie aus einer Queue (*Take()*). Allerdings hat der Ringbuffer eine begrenzte Kapazität (*Size()*). Wenn die ausgeschöpft ist, werden Werte „am Anfang“ „überschrieben“, d.h. dann haben neue Werte Vorrang vor alten.

Das Interface der Klasse soll so aussehen:

```
class Ringbuffer<T> {
        Ringbuffer(int size) {...}

        void Add(T value) {...}
        T Take() {...}
        int Count() {...} // Anzahl ungelesene Elemente (<= Size())
        int Size() {...} // Größe des Ringbuffers
}
```

Beispiel:

| **Aktion** | **Ringbufferinhalt** |
|---|---|
| `new Ringbuffer<int>(3)` | |
| `Add(1)` | 1 |
| `Add(2)` | 1,2 |
| `Size() -> 3` | |
| `Count() -> 2` | |
| `Take() -> 1` | 2 |
| `Add(3)` | 2,3 |
| `Add(4)` | 2,3,4 |
| `Add(5)` | 3,4,5 |
| `Take() -> 3` | 4,5 |
| `Add(6)` | 4,5,6 |
| `Add(7)` | 5,6,7 |

## Variationen

Zunächst kann der Fall, dass alte Werte zugunsten von neuen überschrieben werden, ohne spezielle Behandlung bleiben. Es kann still überschrieben werden.

Als Weiterentwicklung kann der Ringbuffer jedoch so ausgelegt sein, dass sein Nutzer bestimmen kann, ob still überschrieben oder eine Exception geworfen werden soll. Überlege, wie die Wahl einer Option am besten im Interface der Klasse abgebildet werden kann.

## Ressourcen

[1] Ringpuffer, **http://de.wikipedia.org/wiki/Warteschlange_(Datenstruktur)#Ringpuffer**

[2] Circular buffer, **http://en.wikipedia.org/wiki/Circular_buffer**