BoundedVersionedString

- A VersionedString with a bounded capacity
- Abstraction Function:

$$A: BVS \rightarrow \{\phi\} \cup S \cup S \times S \cup ... \cup S^{capacity}$$

where:

- BVS = {vs | vs is a BoundedVersionedString state}
- S = { s | s is a String}

$$vs \in BVS \xrightarrow{A} \begin{cases} \phi \\ s \in S \\ (s_1, s_2) \in S \times S \\ (s_1, s_2, s_3) \in S \times S \times S \end{cases}$$

Software 1

Recitation No. 5
(Class Verification)

The Contract
Pre/Post Conditions

- BoundedVersionedString(int capacity):
  - Requires:
    capacity > 0
  - Ensures:
    nothing

Method Specification

```
public class BoundedVersionedString {
    public BoundedVersionedString(int capacity) {…}
    public void add(String s) {…}
    public int length() {…}
    public String getLastVersion() {…}
    public String getVersion(int i) {…}
}
```

The Contract
Pre/Post Conditions (cont.)

- int length():
  - Requires:
    nothing
  - Ensures:

Since length() is a query

$$A(new) = A(old)$$

$$returned\ \ value = \begin{cases} 0 & if \quad A(old) = \phi \\ k & if \quad A(old) = (s_1, ..., s_{k \geq 1}) \end{cases}$$

The Contract
Pre/Post Conditions (cont.)

- void add(String s):
  - Requires:
    s != null
  - Ensures:

$$A(old) = \phi \Rightarrow A(new) = (s)$$

$$A(old) = (s_1, s_2, ..., s_{k \geq 1}) \Rightarrow A(new) = \begin{cases} (s_1, s_2, ..., s_k, s) & if\ k < capacity \\ (s_2, s_3, ..., s_k, s) & if\ k = capacity \end{cases}$$

The Contract
Pre/Post Conditions (cont.)

- String getLastVersion():
  - Requires:

$$A(this) \neq \phi$$

  - Ensures:

$$A(new) = A(old)$$
$$A(old) = (s_1,...,s_k) \Rightarrow returned \quad value = s_k$$

The Contract
Pre/Post Conditions (cont.)

- String getVersion(int i):
  - Requires:

$$A(this) = (s_1, s_2,..., s_{k \geq i \geq 1})$$

  - Ensures:

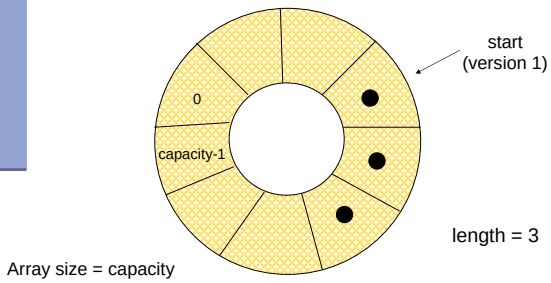
$$A(new) = A(old)$$
$$A(old) = (s_1,...,s_{k \geq i \geq 1}) \Rightarrow returned \quad value = s_i$$

Implementation (cont.)
Fields

```
// Stores the remembered versions
private String[] versions ;

// The maximum number of versions to remember
private int capacity;

// The actual number of remembered versions
private int length = 0;

// The position of the oldest remembered version
private int start = 0;
```

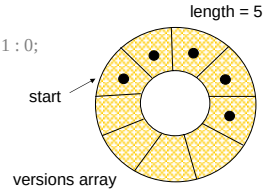
Implementation

- Based on a circular array



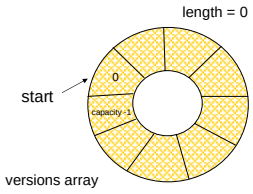
Implementation (cont.)
Command

```
public void add(String s) {
    int pos = (start + length) % capacity;
  versions[pos] = s;

  if ((length > 0) && (start == pos)) {
        start = (start+1) % capacity;
    }

    length += (length < capacity) ? 1 : 0;
}
```



Implementation (cont.)
Constructor

```
public BoundedVersionString(int capacity) {
    this.capacity = capacity;
    versions = new String[capacity];
}
```



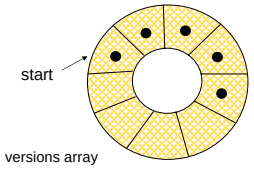
Abstraction Function Definition

$$A : BVS \rightarrow \{\phi\} \cup S \cup S \times S \cup ... \cup S^{capacity}$$

where:

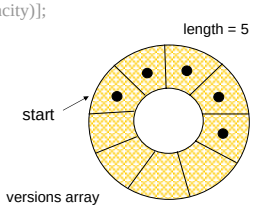
- BVS = {vs | vs is a BoundedVersionedString state}
- S = { s | s is a String}

$A(this) = (verions\ [start], ..., versions\ [(start + length - 1)\%\ capacity])$



Implementation (cont.)
Queries

```
public int length() {
    return length;
}

public String getVersion(int i) {
    return versions[(start+i-1)%capacity)];
}

public String getLastVersion() {
    return getVersion(length());
}
```



Class Verification

- Invariant Verification in initial state:
  - *capacity* > *0* (construction precondition) 1
  - *start* = *0* < *capacity* (initial state, 1) 2
  - *length* = *0* < *capacity*, *start* = *0* (initial state) 3
  - *A(this)* = *(versions[0],…,versions[-1])* = Ø 4
    length = *0*

Representation Invariant

1. $capacity > 0$
2. $0 \le start < capacity$
3. $length < capacity \Rightarrow start = 0$
4. $length = \begin{cases} 0 & if \quad A(this) = \phi \\ k & if \quad A(this) = (s_1, ..., s_{k \ge 1}) \end{cases}$

Class Verification (cont.)

- Method Verification:
  - Assume: The invariant and pre-condition are satisfied **before** the call
  - Prove: The invariant and post-condition are satisfied **after** the call
- Other Code Verification:
  - Prove: No other code in the program change the invariant.