Konzepte objektorientierter Programmierung

Prof. Dr. Peter Müller

Chair of Programming Methodology

Exercise 14: Specification

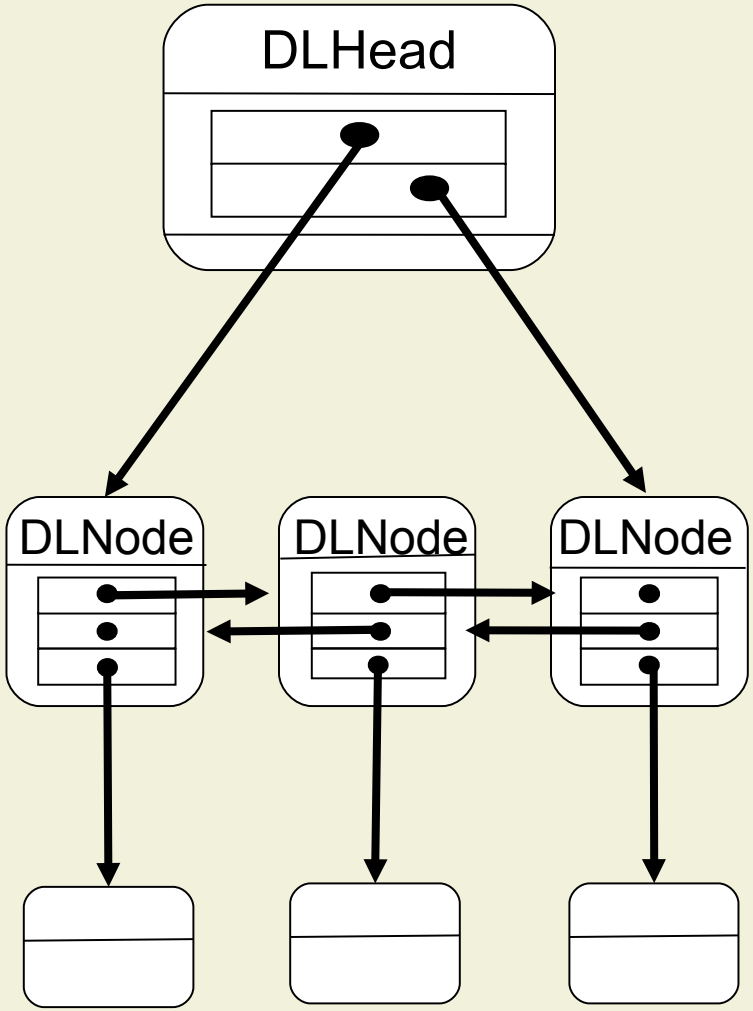
ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Exercise 1: Doubly-linked List

```
public class DLHead {
  private class DLNode {
    DLNode prev, next;
    Object elem;
  }
  private DLNode first, last;
  …

}
```

What we want:

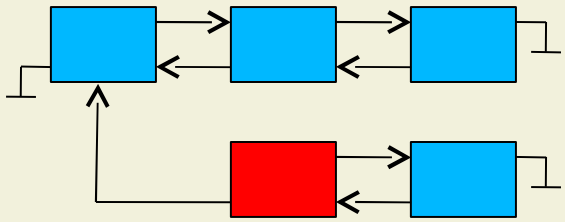


Invariant 1: I'm my successor's predecessor

```
private class DLNode {
   // invariant this.next != null =>
   //                this.next.prev == this
   …
```

- But is this enough?

No! `prev` could link into another List

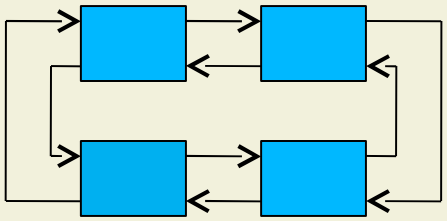


Invariant 2: I'm my predecessor's successor

```
private class DLNode {
   // invariant this.next != null =>
   //                this.next.prev == this
   // invariant this.prev != null =>
   //                this.prev.next == this
   ...
```

- Is this enough now?

No! We could have a circular list

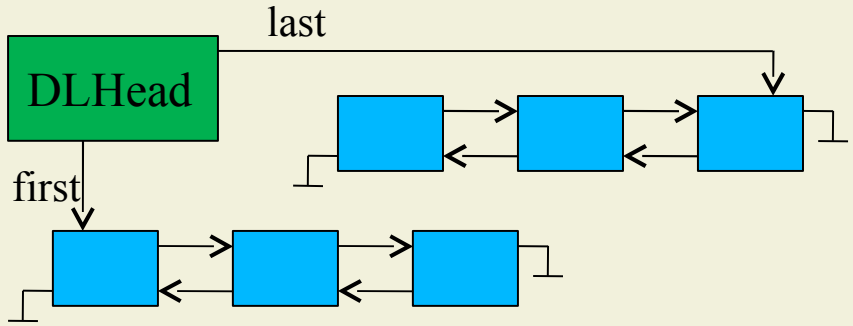


Invariant 3: first / last come / go nowhere

```
public class DLHead {
   private DLNode first, last;
   // invariant first != null =>
   //         first.prev == null
   // invariant last != null =>
   //         last.next == null
```

- Are we done?

No! we could have two independent lists



Invariant 4: first and last use same list

Approach 1:

```
public class DLHead {
  private DLNode first, last;

  // invariant first != null =>
  //   first.next* == last
  or
  // invariant last != null =>
  //   last.prev* == first
```

Invariant 4: first and last use same list 2

Approach 2:

```
public class DLHead {
  private DLNode first, last;

  // invariant reachable( first, last )

  pure boolean
    reachable(DLNode from, DLNode to)
  { … }
```

Using dummy-nodes

- Both the specification and implementation get simpler by always having a dummy first and last element
- These elements are always there and do not contain any data
- Example:

```
// invariant first.prev == null
// invariant last.next == null
```