Information Hiding

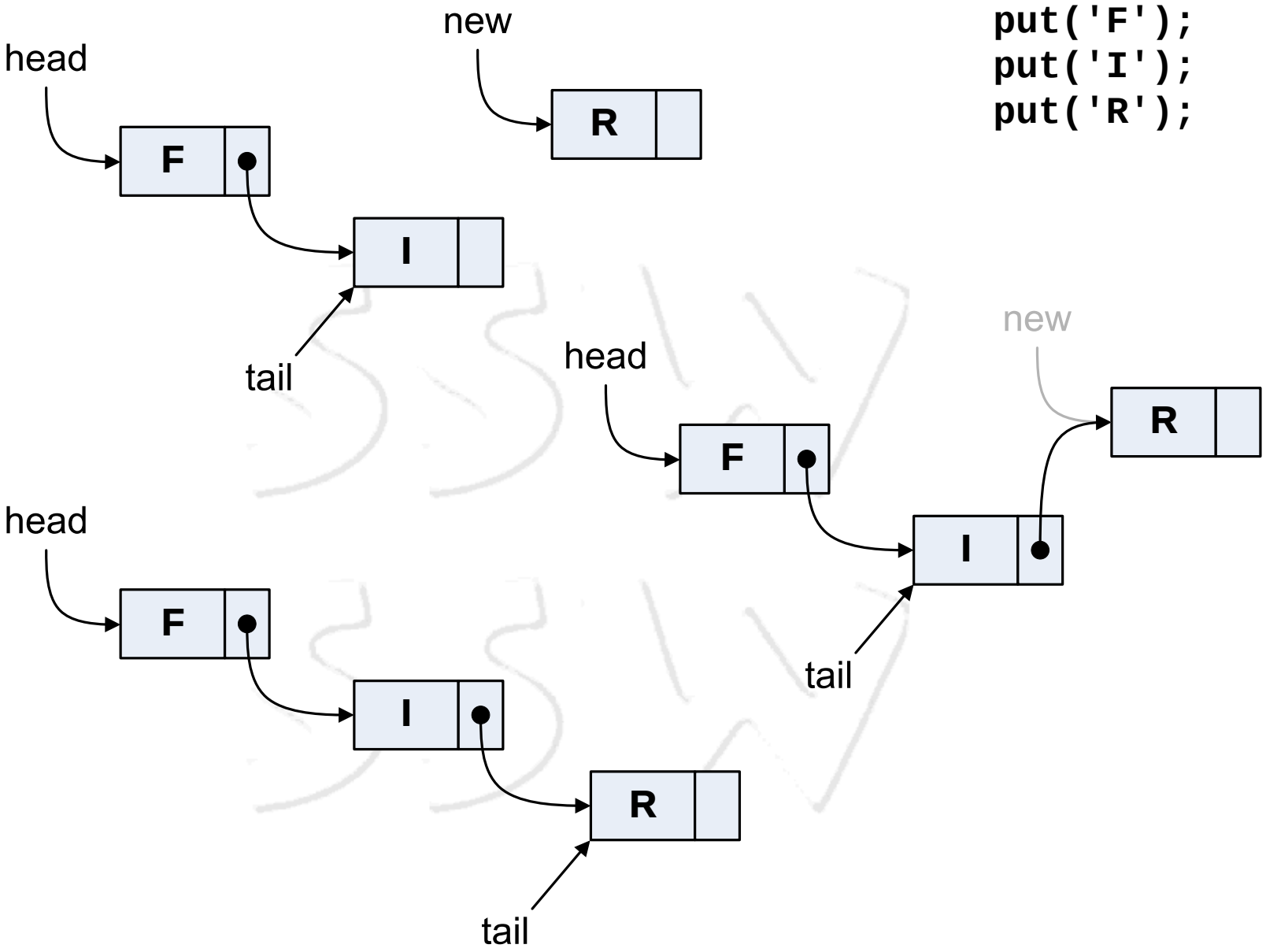
- Erklären Sie "Information hiding" (d.h. das Verstecken von Daten) und wozu ist das gut?

Übung FIFO-Warteschlange (Queue)

Implementieren Sie den ADT Queue mit Hilfe einer verketteten Liste ohne einen Dummy-Knoten zu verwenden.

```
public abstract class Queue
{
  public abstract void clear();
  public abstract char get();
  public abstract int getSize();
  public abstract boolean isEmpty();
  public abstract void put(char val);
}
```

Queue mit verketteter Liste



Lösung mit verketteter Liste (1)

```
public class Node {
  Node next;
  char val;
  Node(char val) {
    this.val = val;
    this.next = null; }
}
public class Queue0 extends Queue {
  private Node head, tail;
  private int size;

  public Queue0() {
    clear(); }

  public void clear() {
    head = tail = null;
    size = 0; }

  public int getSize() {
    return size; }
  ...
```

Lösung mit verketteter Liste (2)

```
  ...

  public char get() {
    char val = head.val;
    head = head.next;
    size--;
    return val; }

  public boolean isEmpty() {
    return (head == null); }

  public void put(char val)       {
    Node n = new Node(val);
    if(isEmpty())
      head = n;
    else
      tail.next = n;
    tail = n;
    size++; }
}
```

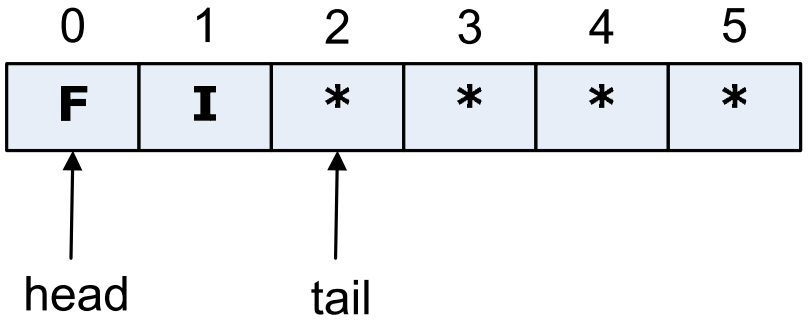
Übung FIFO-Warteschlange (Array)

Implementieren Sie den gleichen ADT Queue mit Hilfe eines Arrays.

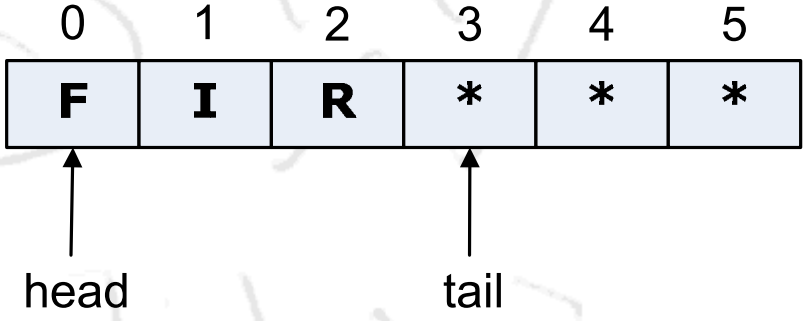
```
public abstract class Queue
{
  public abstract void clear();
  public abstract char get();
  public abstract int getSize();
  public abstract boolean isEmpty();
  public abstract void put(char val);
}
```

Queue mit Array

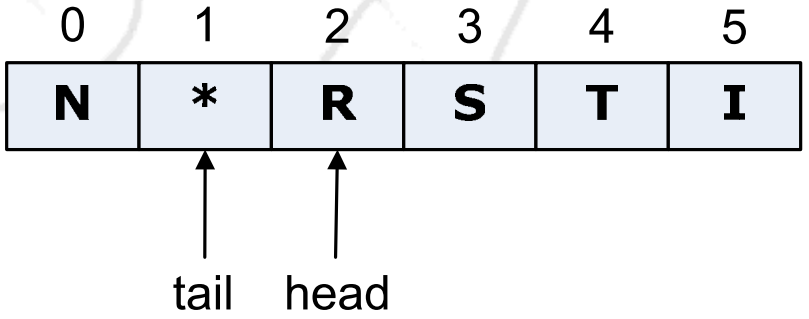
```
put('F');
put('I');
```




```
put('R');
```




```
put('S');
get();
put('T');
get();
put('I');
put('N');
```



Lösung mit Array (1)

```
public class Queue1 extends Queue {
  private int n, head, tail;
  private char[] q;

  public Queue1(int max) {
    q = new char[max+1];
    n = max + 1;
    clear();  }

  public void clear() {
    head = tail = 0; }

  public char get() {
    char val = q[head++];
    head = head % n;
    return val; }

  public int getSize() {
    return (tail >= head)
      ? tail - head : n - Math.abs(tail - head);
  }
  ...
```

Lösung mit Array (2)

```
  ...

  public boolean isEmpty()
  {
    return (head == tail);
  }

  public void put(char val)
  {
    q[tail++] = val;
    tail = tail % n;
  }
}
```