Informatik II Übung, Woche 14

Giuseppe Accaputo

7. April, 2016

Plan für heute

1. Java Klassen Beispiel:
   Implementation eines Vorlesungsverzeichnis (VVZ)
2. Vorbesprechung Übung 6

Was sollte unser VVZ anbieten?

Anforderungen:

1. Vorlesungen für das aktuelle Semester anzeigen
2. Informationen zu einzelnen Vorlesungen anzeigen
3. Eintragen von Vorlesungen ermöglichen
4. Studenten können sich für Vorlesungen anmelden

Welche Klassen benötigen wir?

- `Vorlesungsverzeichnis`
- `Student`
- `Dozent`
- `Vorlesung`

Definition der Klassen

Welche Variablen und Methoden benötigen wir pro Klasse?

Vorschläge!

Stack

- Stapelspeicher: Arbeitet nach dem Last-In-First-Out-Prinzip (LIFO)
- Bietet folgende Methoden an:
  1. `Push`: Legt das Objekt oben auf den Stapel
  2. `Pop`: Liefert das oberste Objekt und entfernt es vom Stapel

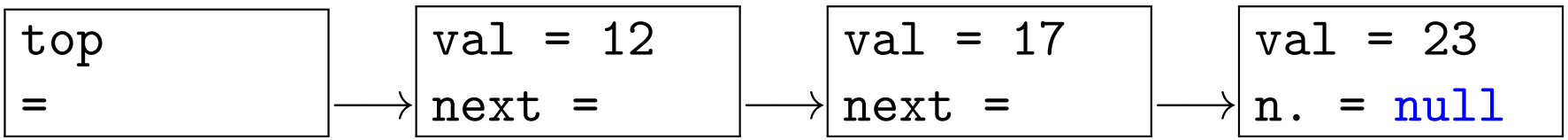
Stack Implementation: Node Klasse

```
package Stack;

class Node
{
    double value; // Data
    Node next;  // Reference to next Node

    Node (double v, Node nxt) // Constructor
    {
        value=v;
        next = nxt;
    }
}
```

Stack Implementation: Stack Klasse

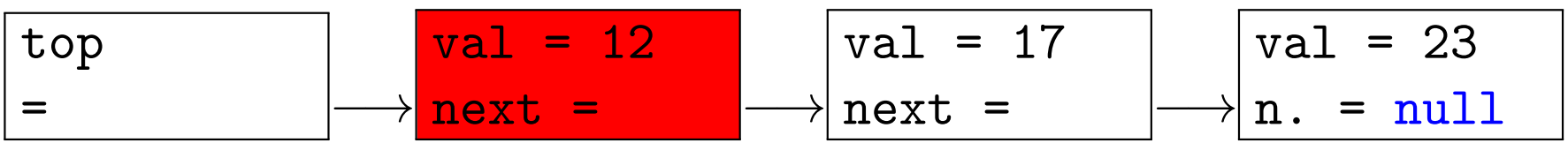
```
package Stack;

public class Stack {
    private Node top = null;

    public void Push(double value){ ... }

    public double Pop(){ ... }
}
```

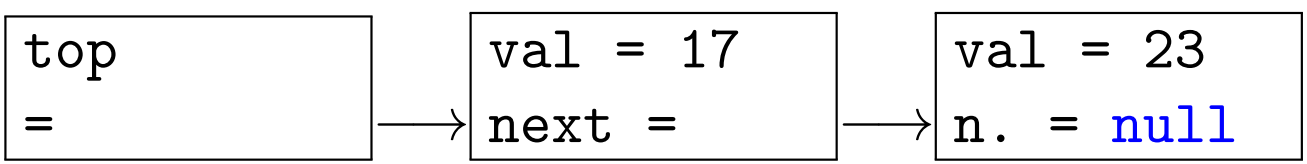
Beispiel Stack



Stack Implementation: Pop Methode

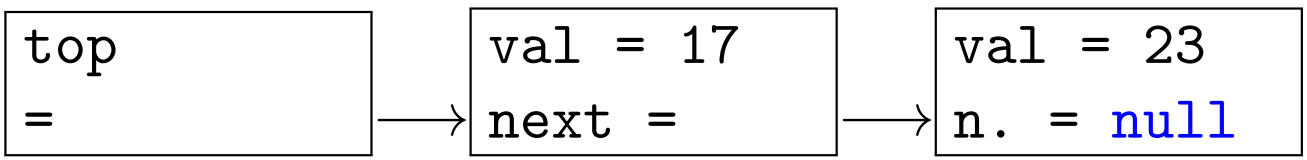
```
public double Pop() {
    double value = top.value;
    top = top.next;
    return value
}
```

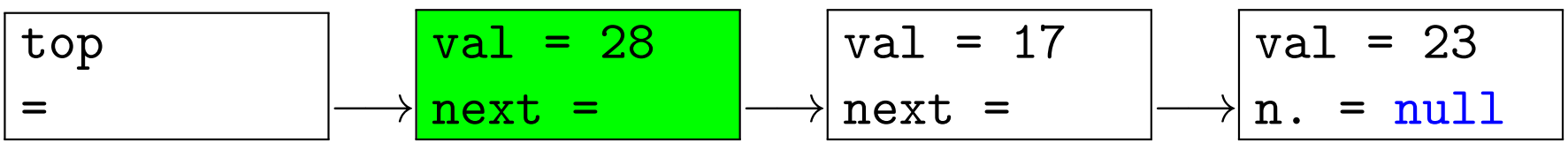




Stack Implementation: Push Methode

```
public void Push(double val) {
    Node next = new Node(val, top);
    top = next;
}
```

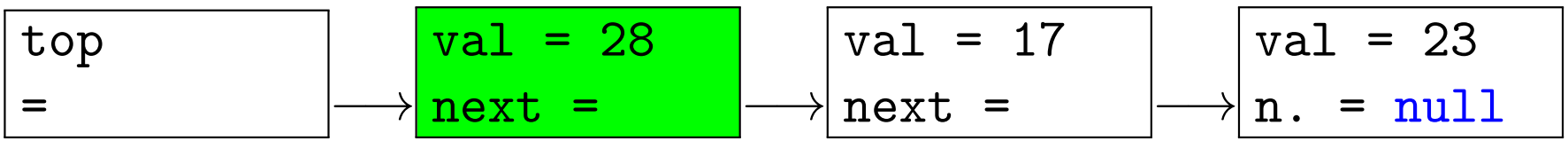




Stack Implementation: Stack-Inhalt ausgeben

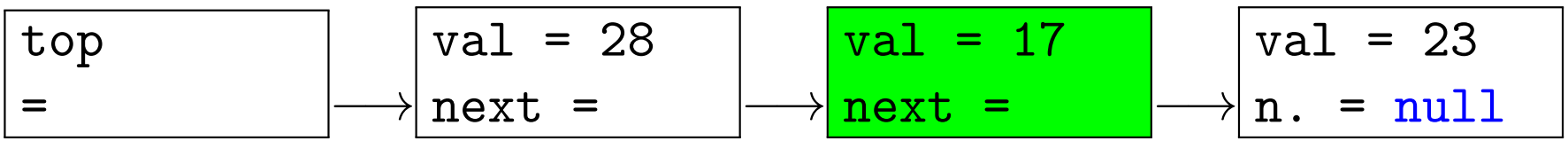
```
public void print()
{
    Node t = top;
    while(t != null)
    {
        t.print();
        t = t.next;
    }
}
```

Stack Implementation: Stack-Inhalt ausgeben



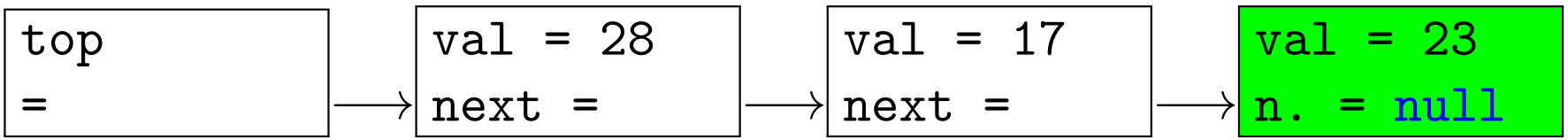
Konsole: 28

Stack Implementation: Stack-Inhalt ausgeben



Konsole: 28
17

Stack Implementation: Stack-Inhalt ausgeben

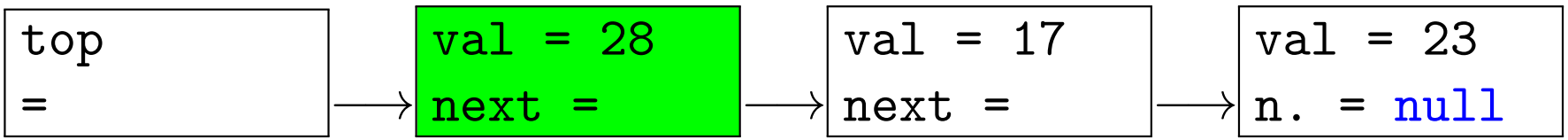


Konsole:
28
17
23

Stack Implementation: Reverse Printing

```
public void print_reverse()
{
    Node t = top;
    if (t != null)
        print_recurse(t);
}

private void print_recurse(Node nxt){
    if(nxt != null)
        print_recurse(nxt.next);
    nxt.print();
}
```

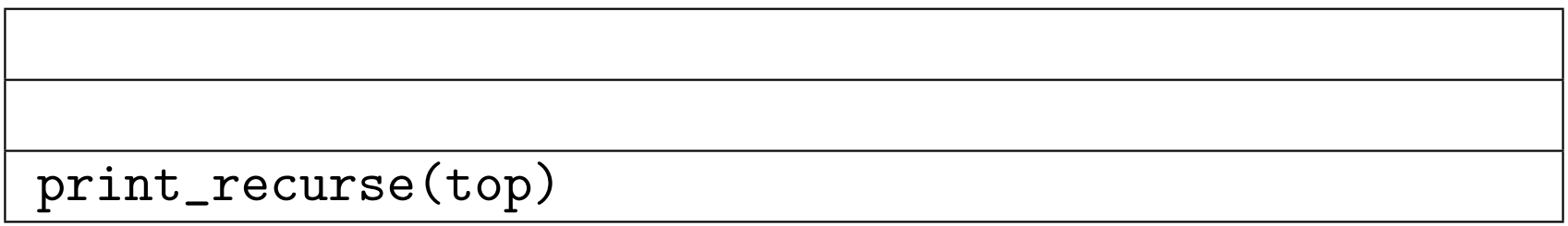
Stack Implementation: Reverse Printing



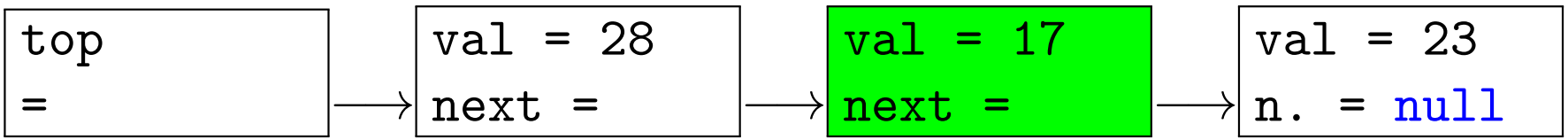
Funktionsaufruf:

```
print_recurse(top);
```

Rekursions-Stapel



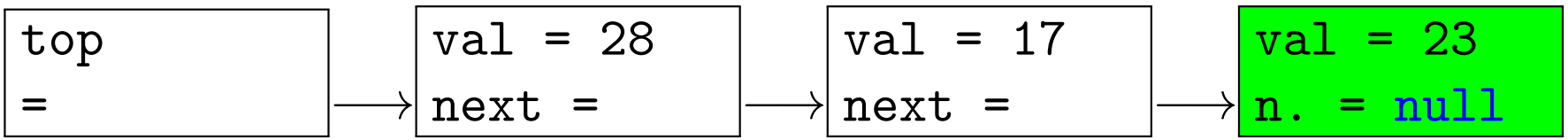
Stack Implementation: Reverse Printing



Rekursions-Stapel

print_recurse(Node17)
print_recurse(top)

Stack Implementation: Reverse Printing



Rekursions-Stapel

print_recurse(Node23)
print_recurse(Node17)
print_recurse(top)

Stack Implementation: Reverse Printing

Rekursions-Stapel

`print_recurse(Node23)`
`print_recurse(Node17)`
`print_recurse(top)`

Die Abbruchbedingung wurde nun nach dem Aufruf von `print_recurse(Node23)` erreicht, da `Node23.next == null`. In einem nächsten Schritt beginnt nun die Abarbeitung des Stapels von oben nach unten, begonnen bei `print_recurse(Node23)`.

Stack Implementation: Reverse Printing

Rekursions-Stapel

`print_recurse(Node23)`
`print_recurse(Node17)`
`print_recurse(top)`

Abarbeitung:

Konsole:

Stack Implementation: Reverse Printing

Rekursions-Stapel

`print_recurse(Node17)`
`print_recurse(top)`

Abarbeitung:

`print_recurse(Node23)`	`Node23.print()`

Konsole: 23

Stack Implementation: Reverse Printing

Rekursions-Stapel

`print_recurse(top)`

Abarbeitung:

`print_recurse(Node17)`	`Node17.print()`
`print_recurse(Node23)`	`Node23.print()`

Konsole: `23, 17`

Stack Implementation: Reverse Printing

Rekursions-Stapel

Abarbeitung:

`print_recurse(top)`	`top.print()`
`print_recurse(Node17)`	`Node17.print()`
`print_recurse(Node23)`	`Node23.print()`

Konsole: `23, 17, 28`