

# **OOP mit Java**

## **Daten schützen mit Zugriffsmodifikatoren**

# Beispiel: Konto / Bank

```
class Bank
{
    Konto k1, k2;
    ...

    void main()
    {
        k1.einzahlen(100);
        ...
    }
}
```

# Beispiel: Konto / Bank

```
class Konto
{
    double stand;

    void einzahlen(double pb)
    {
        stand = stand + pb;
    }
    ...
}
```

# Neu: Zugriffsmodifikatoren

```
class Konto
{
    private double stand;

    public void einzahlen(double pb)
    {
        stand = stand + pb;
    }
    ...
}
```

# Neu: Zugriffsmodifikatoren

```
class Konto
{
    private double stand;

    public void einzahlen(double pb)
    {
        stand = stand + pb;
    }
}
```

Private: kein Zugriff für andere Klassen

Public: Zugriff für andere Klasse möglich

# Methoden: Zugriff **erlaubt**

```
class Bank
{
    Konto k1, k2;
    ...

    void main()
    {
        k1.einzahlen(100);
        ...
    }
}
```

# Attribute: Zugriff **nicht erlaubt**

```
class Bank
{
    Konto k1, k2;
    ...

void main()
{
    k1.stand = k1.stand + 100;
    ...
}
}
```

# Warum Zugriff verbieten?

Direkter Zugriff Attribute fremder Klassen:  
**Fehlerquelle!**



# Warum Zugriff verbieten?

Direkter Zugriff Attribute fremder Klassen:  
**Fehlerquelle!**

```
class Bank
{
    Konto k1, k2;
    ...
    void main()
    {
        k1.stand = k1.stand - 100;
        ...
    }
}
```

# Warum Zugriff verbieten?

Negativer Kontostand nicht erlaubt!  
→ Programm führt zu Problemen

```
class Bank
{
    Konto k1, k2;
    ...
    void main()
    {
        k1.stand = k1.stand - 100;
        ...
    }
}
```

# Vermeiden von Fehlern

```
class Konto
{
    private double stand;

    public void einzahlen(double pb)
    {
        if (pb >= 0)
        {
            stand = stand + pb;
        }
    }
}
```

# Vermeiden von Fehlern

```
public void abheben(double pb)
{
    if (      ???      )
    {
        stand = stand - pb;
    }
}
```

# Vermeiden von Fehlern

```
public void abheben(double pb)
{
    if (stand - pb >= 0)
    {
        stand = stand - pb;
    }
}
```

Vermeidet negativen Kontostand

# Vermeiden von Fehlern

```
public void abheben(double pb)
{
    if (stand - pb >= 0 && pb > 0)
    {
        stand = stand - pb;
    }
}
```

Vermeidet negativen Kontostand **und**  
Abheben eines negativen Betrags

# Vermeiden von Fehlern

```
public void abheben(double pb)
{
    if (stand - pb >= 0 && pb > 0)
    {
        stand = stand - pb;
    }
}
```

Attribute sollen private sein

- **nur** durch Methoden veränderbar
- Methoden können prüfen, ob Änderung (durch einen Parameter) sinnvoll ist

# Zusammenfassung

Zugriffsmodifikatoren: public und private

**public:**

Zugriff durch andere Klassen erlaubt

**private:**

Zugriff durch andere Klassen nicht möglich

**Attribute** werden private deklariert

**Methoden** (meist) public

Grund: Attribute vor „unsachgemäßen Änderungen“ schützen



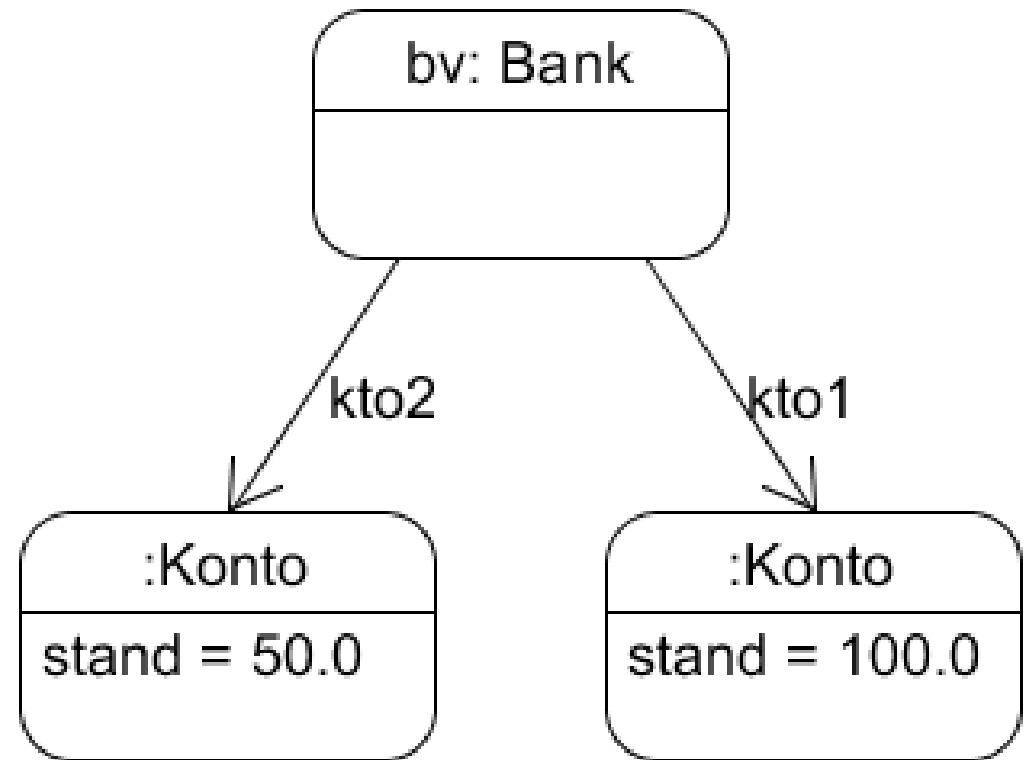
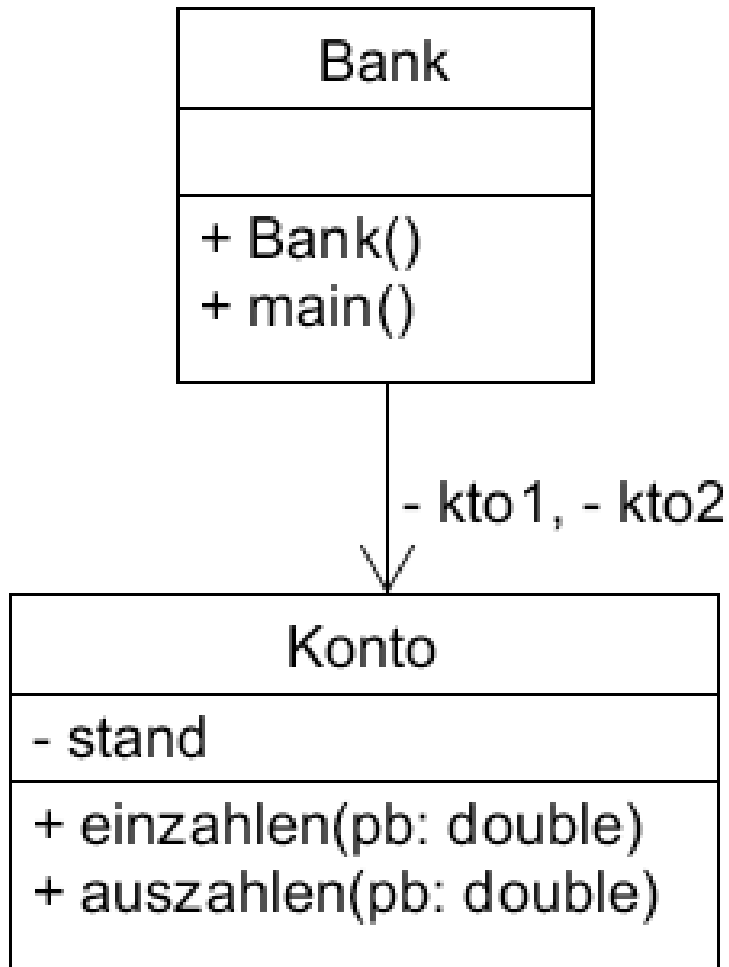
# Zusammenfassung

**Klassen:** in der Regel auch public

```
public class Konto  
{  
    private double stand;  
    ...  
}
```

Private Klassen sind zwar möglich, werden aber im Rahmen des Unterrichts nicht implementiert.

# Zugriffsmodifikatoren in UML



Klassendiagramm:  
public **+**, private **-**

Objektdiagramm:  
nicht dargestellt

# Bedingungen verknüpfen

Oft werden mehrere Bedingungen geprüft:  
Beispiel:

Ist Parameter größer 0 **und** kleiner 100 ?

Logisches UND: **&&**

Logisches ODER: **||**

Logisches NICHT: **!**

```
if ( pb > 0 && stand - pb > 0 ) ...
```

```
if ( px < 0 || px > window.getWidth() ) ...
```

```
while ( ! window.mouseButton1() ) ...
```

# Autor / Quellen

Autor:

- Christian Pothmann (cpothmann.de)  
Freigegeben unter CC BY-NC-SA 4.0, März 2021

