Live Vote

PIN: M4I2

https://ilias.uni-marburg.de/vote/M4I2

2.2 Methoden in Java

- Algorithmen lassen sich in Java direkt als Methoden implementieren.
- Beispiel: Algorithmus zur Berechnung der Kreisfläche.

Dokumentation

Methodenkopf

Methodenrumpf

```
/** Die Methode kreisFlaeche liefert zum Radius r die Kreisfläche.
  * @param r ist der Radius
  * @return liefert die Fläche eines Kreises mit Radius r
  */
double kreisFlaeche (double r)
{
        double pi = 3.14;
        return pi*r*r;
}
```

- Am Ende des Kapitels sollten Sie dieses Beispiel verstehen!

2.2 Methoden in Java

- Algorithmen lassen sich in Java direkt als Methoden implementieren.
- Beispiel: Algorithmus zur Berechnung der Kreisfläche.

Dokumentation

Methodenkopf

Methodenrumpf

```
/** Die Methode kreisFlaeche liefert zum Radius r die Kreisfläche.
 * @param r ist der Ra
 * @return liefert die Fl                dius r
 */
double kreisFlaeche (d
{
        double pi = 3.14;
        return pi*r*r;
}
```

In einem Methodenrumpf muss immer ein Semikolon (;) am Ende jeder Anweisung stehen.

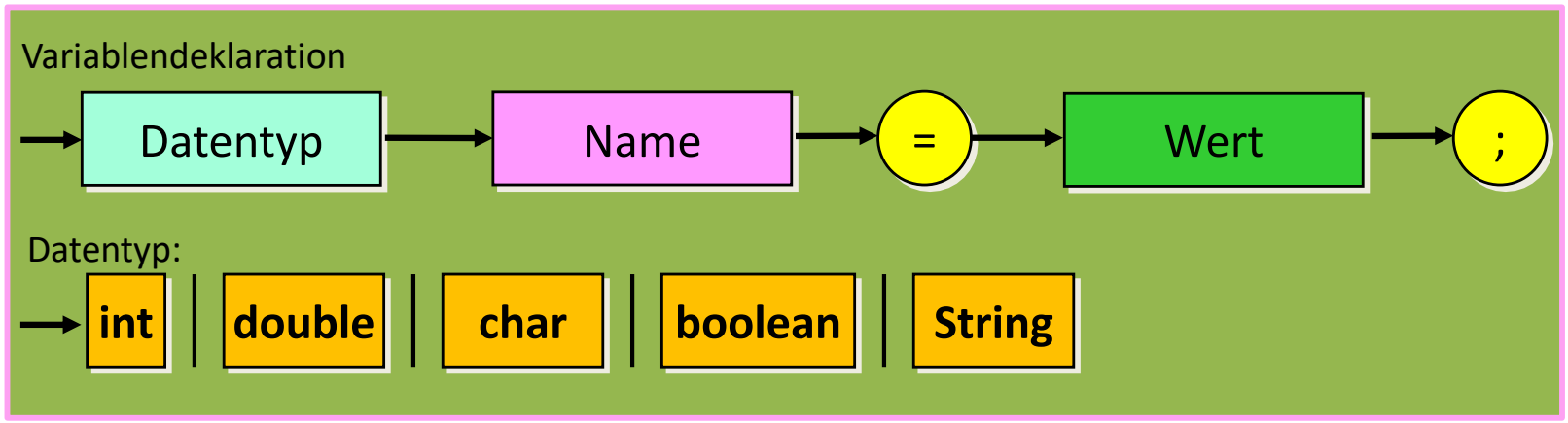
- Am Ende des Kapitels sollten sie dieses Beispiel verstehen!

Syntax einer Sprache

- Unter der Syntax einer formalen Sprache versteht man die Regeln, um aus elementaren Bausteinen korrekte Programme zu erstellen.
  - Elementare Bausteine einer Programmiersprache sind
    - Schlüsselwörter
    - Operatoren
    - Werte
    - Bezeichner eines Benutzers
  - Beispiel (Deklaration einer Variable)
    - int i = 42;
- Darstellung der Regeln durch Syntaxdiagramme

Syntaxdiagramme

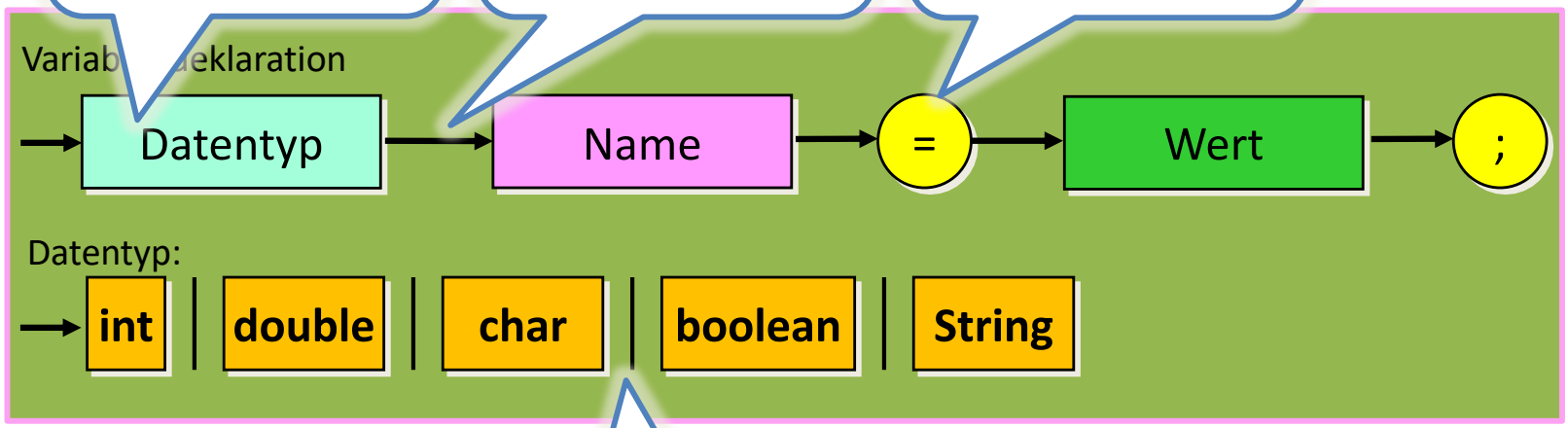
- Sehr vereinfachte Beispiele (für die Sprache Java)



  - Die erste Regel besagt, dass eine Variablendeklaration aus einem Datentyp, einen Namen, dem Zuweisungsoperator und einem Wert besteht.
  - Die zweite Regel besagt, dass ein Datentyp entweder int, double, char, boolean oder String ist.

Syntaxdiagramme

- S… …t …ür … Java)

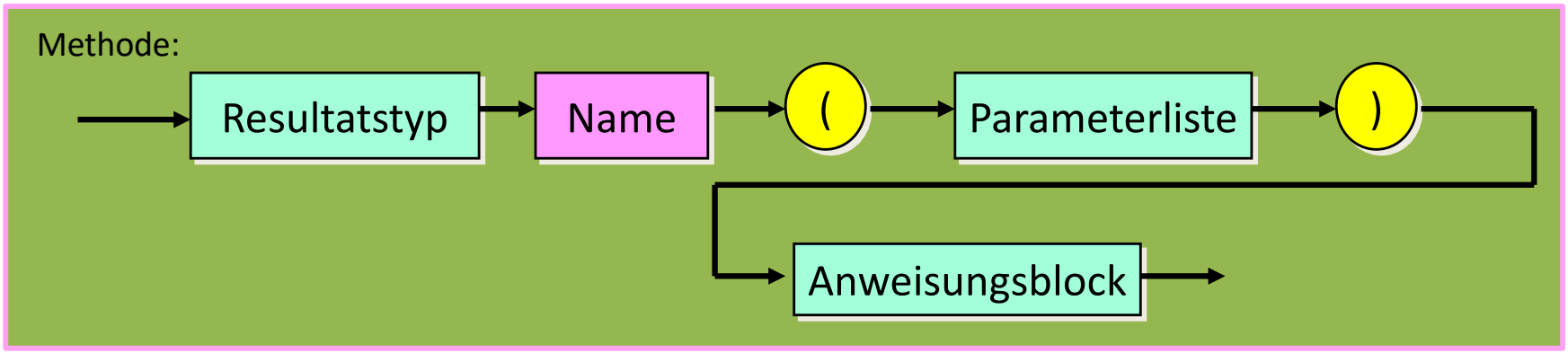


- Die erste Regel besa… …ass eine Variablendeklaration aus einem Datentyp, einen Na… …isungsoperator und einem Wert besteht.
- Die zweite Regel b… …Datentyp entweder int, double, char, boolean oder String ist.

Senkrechter Strich trennt Alternativen

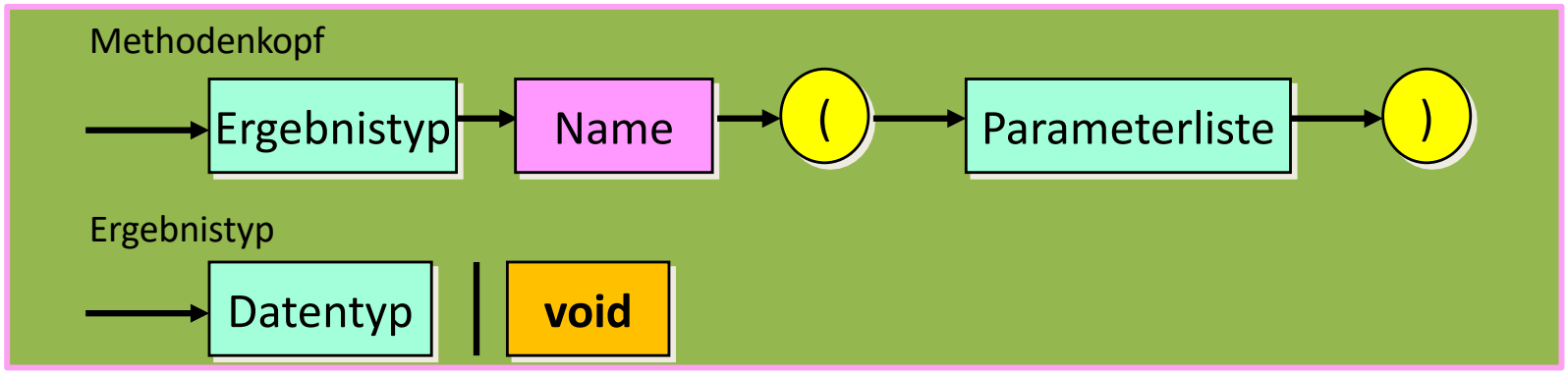
Syntax einer Methode (für die jshell)

- Eine Methode kann durch folgende Regel beschrieben werden.



- Der **Methodenrumpf** entspricht einem **Anweisungsblock** (auch **Sequenz** genannt).

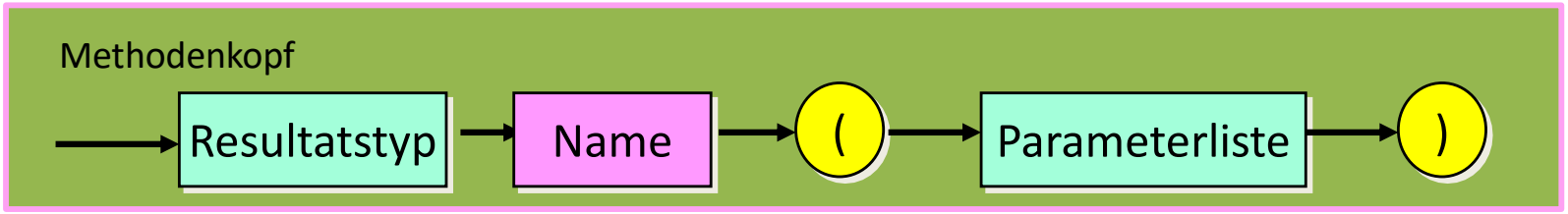
2.2.1 Methodenkopf: Ergebnistyp



- **Ergebnistyp**
  - *Datentyp der Ausgabe*
    - In unserem Beispiel kreisFlaeche ist es der Typ double.
      - Die Methode muss ein Ergebnis aus der Wertemenge der Gleitkommazahlen liefern.
  - *Schlüsselwort void* wird verwendet, wenn die Methode kein Ergebnis produziert.
    - Die Methode System.out.println liefert kein Ergebnis, sondern eine Ausgabe auf dem Bildschirm.

```
double kreisFlaeche (double r)
```

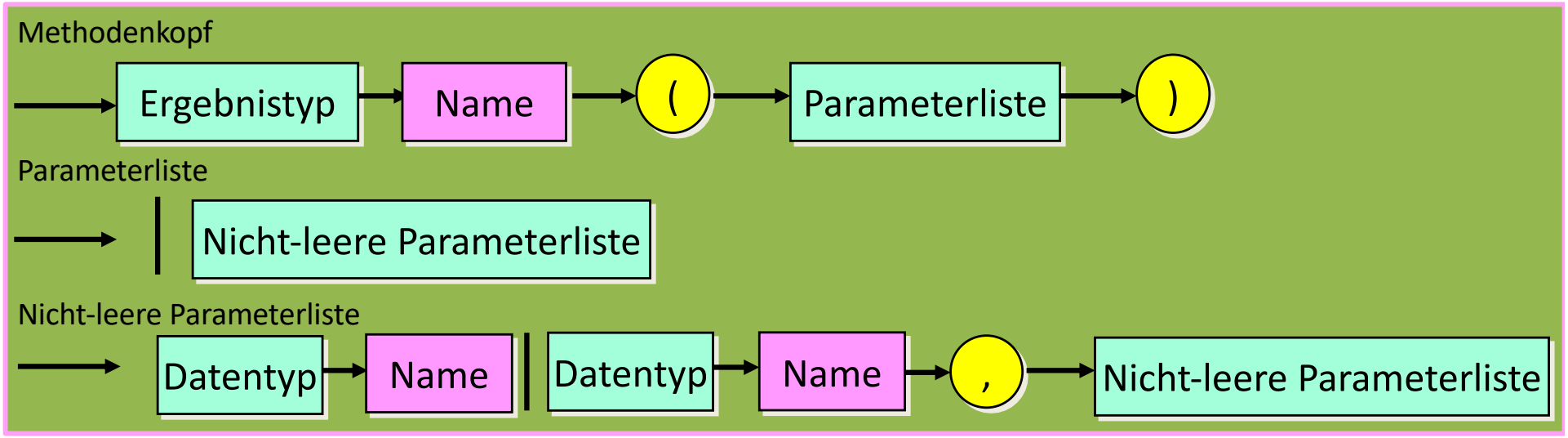
Methodenkopf: Name der Methode



- Name der Methode
  - Dieser wird vom Programmierer gewählt werden und muss eindeutig sein.
    - In unserem Beispiel kreisFlaeche.
  - Bei der Namenswahl müssen noch gewisse Regeln beachtet werden. (→ Details dazu später)

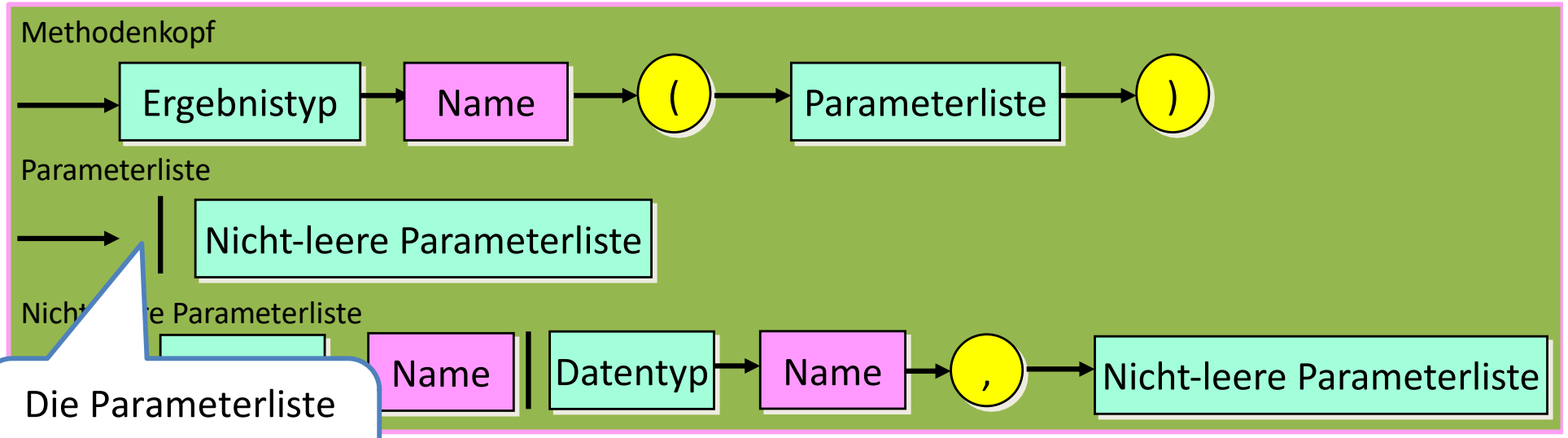
```
double kreisFlaeche (double r)
```

Methodenkopf: Parameterliste



- Parameterliste
  - Diese Liste kann leer sein. Wir sprechen dann von einer ***parameterlosen Methode***.
- Nicht-leere Parameterliste
  - Diese bestehen entweder aus einem Parameter, der aus einem Datentyp und einem Namen besteht.
  - Oder mehreren Parametern, die mit Komma getrennt sind.
    - Dies wird durch eine rekursive Regel definiert.(→ Details Rekursion später)

Methodenkopf: Parameterliste



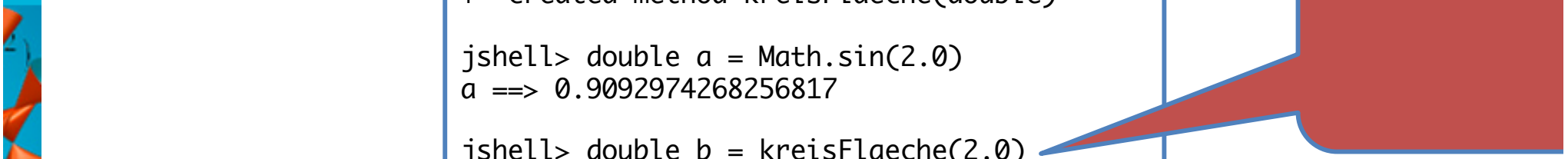
Die Parameterliste kann auch „nichts“ sein.

- ann leer sein. Wir sprechen dann von einer ***parameterlosen Methode***.
  - Die Liste kann nicht leer sein.
- Nicht-leere Parameterliste
  - Diese bestehen entweder aus einem Parameter, der aus einem Datentyp und einem Namen besteht.
  - Oder mehreren Parametern, die mit Komma getrennt sind.
    - Dies wird durch eine rekursive Regel definiert.(→ Details Rekursion später)

Definition und Aufruf einer Methode in jshell

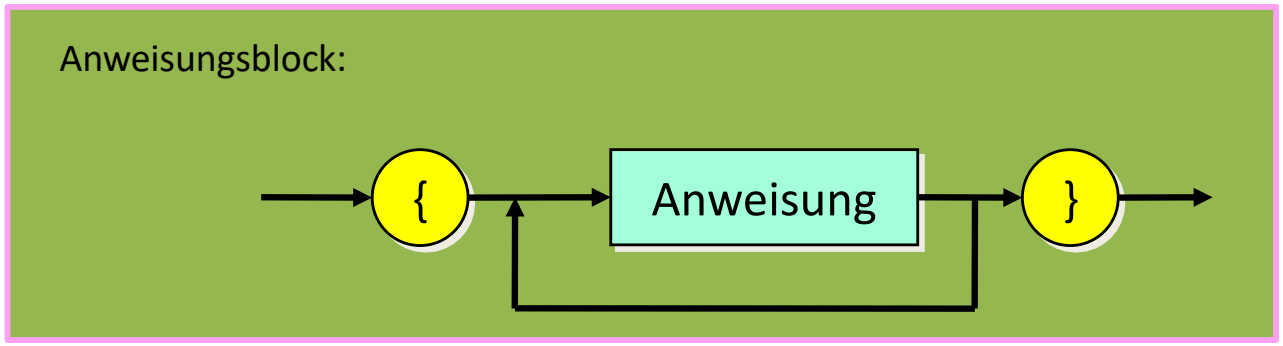
- Eine Methode wird durch Angabe des Methodennamens und eines Werts oder eines Ausdrucks für jeden Parameter aufgerufen.
  - Der Methodenrumpf hat keinen Einfluss auf die Syntax des Aufrufs.

```
jshell> double kreisFlaeche(double r) {
   ...> double pi = 3.14;
   ...> return r*r*pi;
   ...> }
|  created method kreisFlaeche(double)

jshell> double a = Math.sin(2.0)
a ==> 0.9092974268256817

jshell> double b = kreisFlaeche(2.0)
b ==> 8.56

jshell> double c = kreisFlaeche(b)
c ==> 156.805504

jshell> double c = kreisFlaeche(a * b)
c ==> 129.65021070295182
```



2.2.2 Anweisungsblock in Java-Methoden

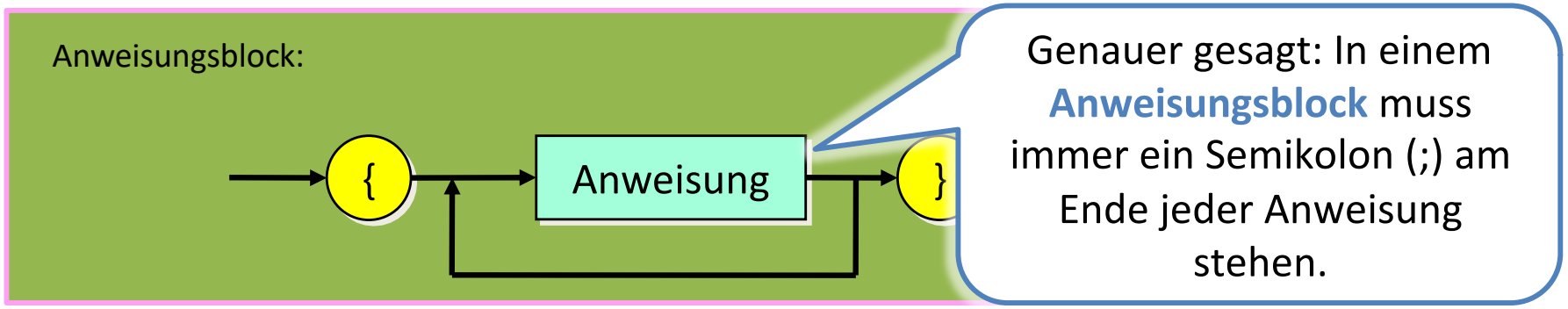
- In Java wird ein Verarbeitungsschritt als **Anweisung** bezeichnet.
- Eine Sequenz von Anweisungen, die durch ein Paar von geschweiften Klammern umgeben ist, wird als **Anweisungsblock** bezeichnet.



- Ein Anweisungsblock kann logisch wieder als eine einzige Anweisung aufgefasst werden!
  - Wir benutzen deshalb auch dann den Begriff Anweisung.
- Zur Unterscheidung werden Anweisungen, die nicht weiter strukturiert sind, auch als **atomar** bezeichnet.

2.2.2 Anweisungsblock in Java-Methoden

- In Java wird ein Verarbeitungsschritt als **Anweisung** bezeichnet.
- Eine Sequenz von Anweisungen, die durch ein Paar von geschweiften Klammern umgeben ist, wird als **Anweisungsblock** bezeichnet.



- Ein Anweisungsblock kann logisch wieder als eine einzige Anweisung aufgefasst werden!
  - Wir benutzen deshalb auch dann den Begriff Anweisung.
- Zur Unterscheidung werden Anweisungen, die nicht weiter strukturiert sind, auch als **atomar** bezeichnet.

Beispiel: Methoden kreisFlaeche

```
double kreisFlaeche (double r)
{
        double pi = 3.14;
        return pi*r*r;
}
```

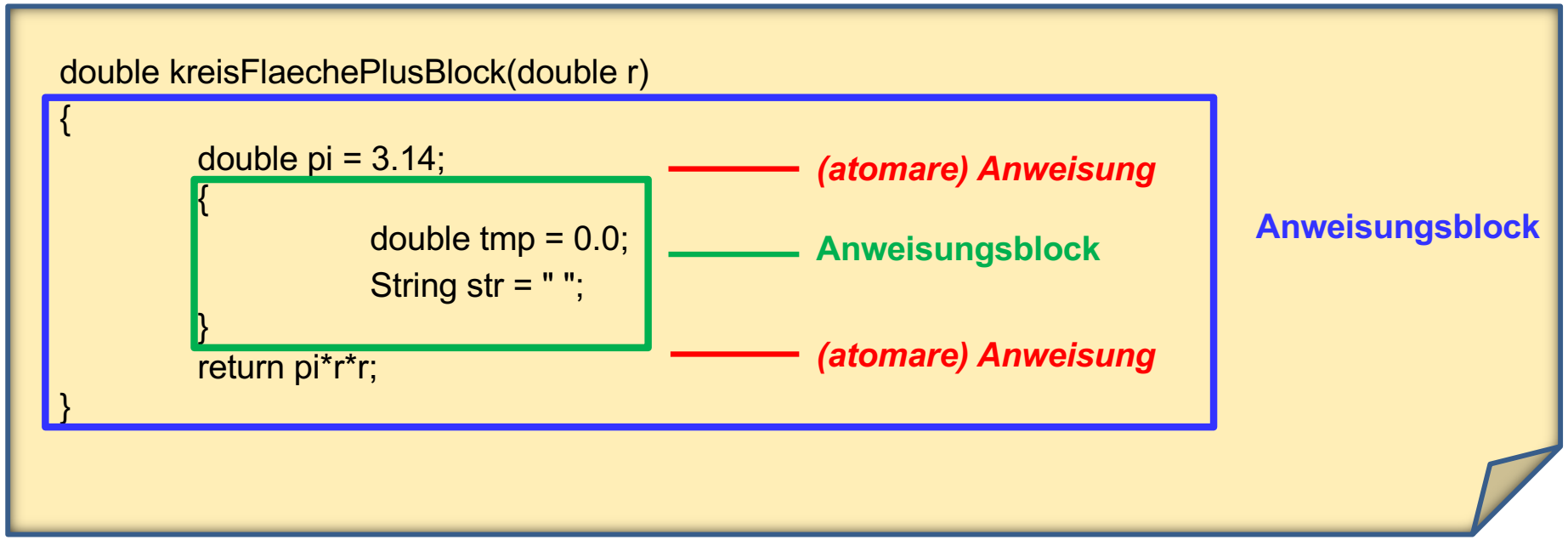
(atomare) Anweisung

(atomare) Anweisung

Anweisungsblock

- *Der Methodenrumpf besteht aus einem Anweisungsblock, der aus zwei atomaren Anweisungen besteht.*

Methoden mit verschachtelten Anweisungsblöcken

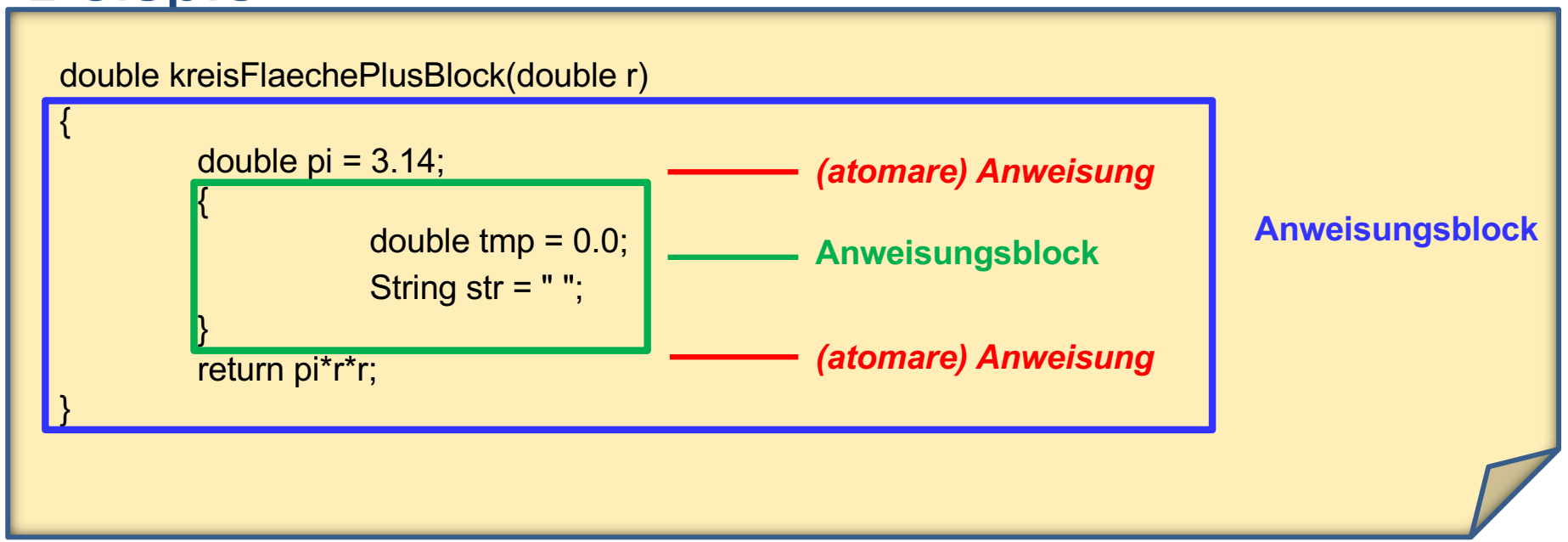


- *Der Methodenrumpf besteht aus einem Anweisungsblock, der aus zwei atomaren Anweisungen und einem Anweisungsblock besteht.*
  - Es ist eine beliebige Verschachtelung von Anweisungsblöcken möglich.

Abarbeitung eines Anweisungsblocks

- **Ein Prozessor** führt einen Anweisungsblock nach folgenden Regeln aus:
  - Die Bearbeitung des Anweisungsblocks beginnt mit der ersten Anweisung.
  - Zu einem Zeitpunkt wird genau eine Anweisung ausgeführt.
  - Jede Anweisung wird genau einmal ausgeführt.
    - Keine Wiederholung, kein Auslassen einer Anweisung. (Mehr dazu später)
  - Die Reihenfolge der Bearbeitung der Anweisungen ist identisch mit der im Algorithmus.
  - Die Bearbeitung des Anweisungsblocks endet mit der letzten Anweisung.

Beispiel



- Der blaue Anweisungsblock besteht aus drei Anweisungen
  - Die zweite Anweisung ist ein Anweisungsblock.
- Die Abarbeitung verläuft sequentiell.
  1. final double pi = 3.14;
  2. Komplette Abarbeitung des grünen Anweisungsblock
  3. return pi*r*r;

Anweisungen

- In unserem Beispiel gibt es zwei Arten von Anweisungen.
  - Eine Variablendeklaration mit einer Initialisierung.

```
double pi = 3.14;
```

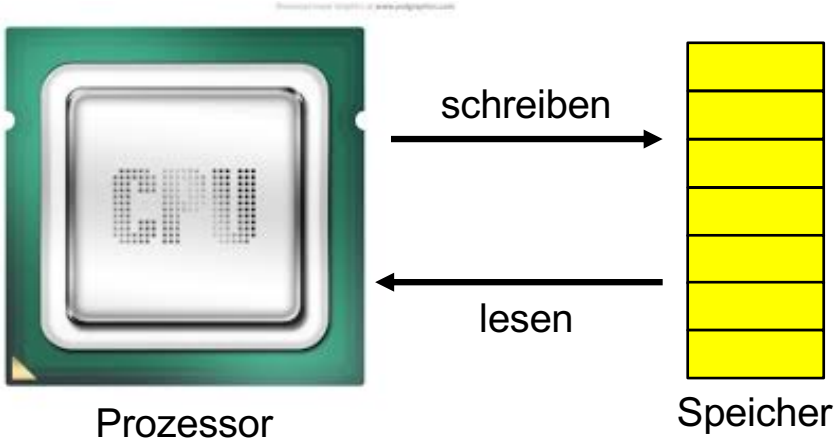
  - Eine return-Anweisung.

```
return pi*r*r;
```

- Im Folgenden werden wir nochmals genauer das Thema Variablen behandeln.

2.2.3 Variablen

- Beim Ausführen einer Methode durch den Prozessor müssen häufig Werte aus Berechnungen in einer Speicherzelle des Computers zwischengespeichert werden.



- **Variablen** bieten in höheren Programmiersprachen, wie z. B. Java, die einzige Möglichkeit, um auf den Speicher des Computers zuzugreifen.
  - Man kann mit einer Variablen den Inhalt einer Speicherzelle **lesen**.
  - Man kann mit einer Variablen den Inhalt einer Speicherzelle **schreiben**.

Variablen

- Unter einer Variablen verstehen wir einen Speicherbereich und einen Datentyp (wie z. B. int, double, String), auf den über einen **Variablennamen** (**Bezeichner**) zugegriffen werden kann.
  - Variablen müssen vor der ersten Benutzung in Java **deklariert** werden.

```
double y;                    // Variable y vom Typ double wird deklariert.
int x  = 23;                 // Variable x vom Typ int wird deklariert und initialisiert.
```

Bezeichner	Wert	Typ
x	23	int
y	uninitialisiert	double

Bemerkungen

- Der Bezeichner und der Typ einer Variablen ändern sich während der Lebenszeit nicht!
- Der Wert einer Variablen kann sich ändern
  - Schreibe den Wert 42 in die Variable x ergibt dann

Bezeichner	Wert	Typ
x	**42**	int
y	uninitialisiert	double

Variablen

- Unter einer Variablen verstehen wir einen Speicherbereich und einen Datentyp (wie z. B. int, double, String), auf den über einen **Variablennamen** (**Bezeichner**) zugegriffen werden kann.
  - Variablen müssen vor der ersten Benutzung in Java **deklariert** werden.

```
double y          // Variable y vom Typ double wird deklariert.
int x  = 23       // Variable x vom Typ wird deklariert und initialisiert.
```

Bezeichner	
x	23
y	uninitialisier

Variable darf erst nach Initialisierung gelesen werden! (Ausnahme: in JShell außerhalb von Methoden werden deklarierte Variablen automatisch mit einem Standardwert initialisiert)

Bemerkungen

- Der Bezeichner und der Typ ei benszeit nicht!
- Der Wert einer Variablen kann s
  - Schreibe den Wert 42 in die Variable x ergibt dann

Bezeichner	Wert	Typ
x	**42**	int
y	uninitialisiert	double

Variablen in Java-Methoden

- *Zunächst unterscheiden wir zwei Arten von Variablen*
  - Parametervariablen
    - In der Literatur wird auch der Begriff *formale Parameter* benutzt.
    - Diese Variablen werden im Methodenkopf deklariert.
  - Lokale Variablen
    - Deklaration erfolgt in einem Anweisungsblock einer Methode.

```
double kreisFlaeche (double r)  {

        double pi = 3.14;
        return pi*r*r;
}
```

Parametervariable

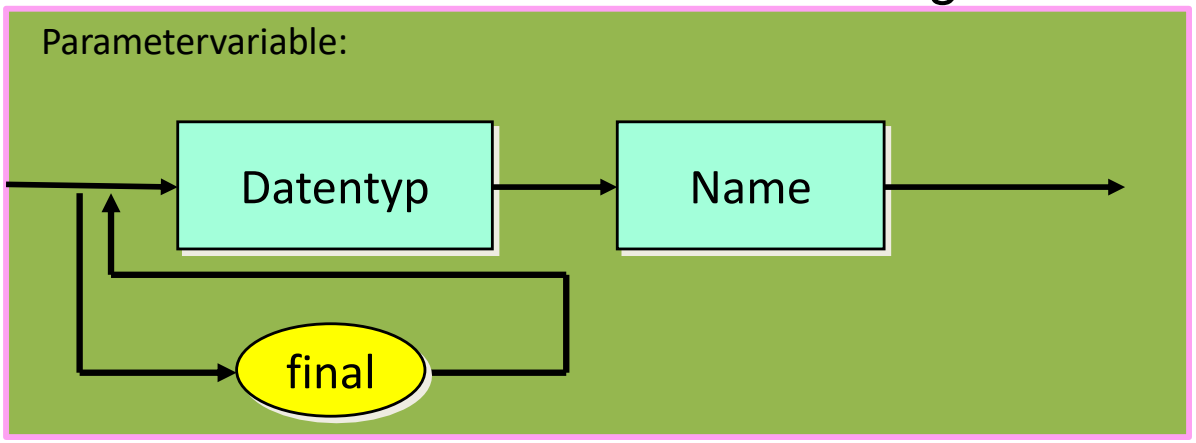
Lokale Variable

Wichtige Eigenschaften von Variablen

- Gültigkeit von Variablen
  - Teil des Programmtexts, in dem man die Variable verwenden darf.
- Lebensdauer von Variablen
  - Zeitspanne von der Erzeugung der Variablen im Speicher bis zum Löschen aus dem Speicher

Parametervariablen (mit final)

- Die Deklaration dieser Variablen erfolgt im Methodenkopf.

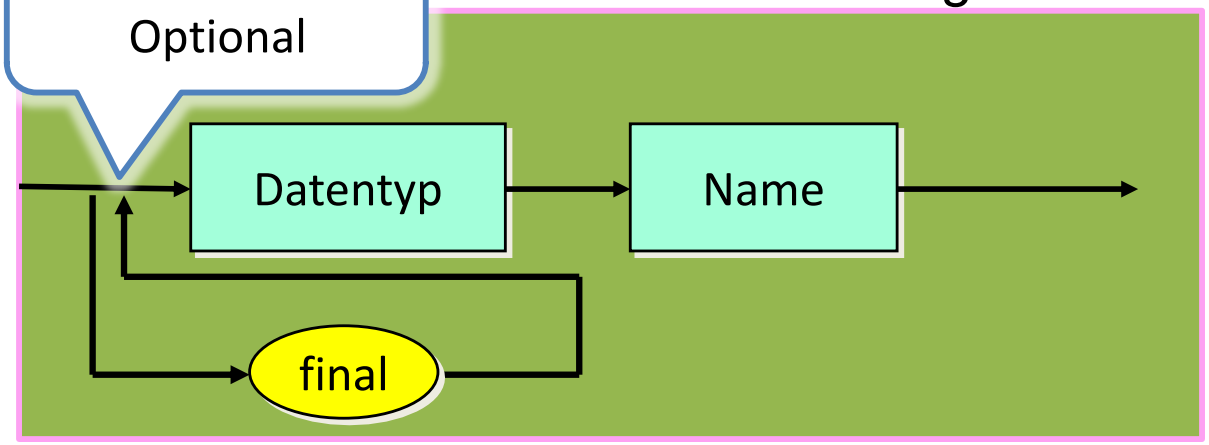


- Zusätzlich können wir das Schlüsselwort **final** voranstellen.
  - Alle Variablen, die als final deklariert werden, erhalten genau einmal einen Wert.
- Gültigkeit
  - Parametervariablen sind nur im Methodenrumpf verwendbar.
- Lebensdauer
  - Die Variable wird beim Methodenaufruf erzeugt und bekommt beim Aufruf einen Wert.
  - Nach dem Ende der Methode wird die Variable vom Speicher entfernt.

Parametervariablen (mit final)

- Verzweigung: Optional

ieser Variablen erfolgt im Methodenkopf.



- Zusätzlich können wir das Schlüsselwort **final** voranstellen.
  - Alle Variablen, die als final deklariert werden, erhalten genau einmal einen Wert.
- Gültigkeit
  - Parametervariablen sind nur im Methodenrumpf verwendbar.
- Lebensdauer
  - Die Variable wird beim Methodenaufruf erzeugt und bekommt beim Aufruf einen Wert.
  - Nach dem Ende der Methode wird die Variable vom Speicher entfernt.

Parametervariablen beim Methodenaufruf

- Eine Methode wird durch Angabe des Methodennamens und eines Werts oder eines Ausdrucks für jede Parametervariable aufgerufen.
  - kreisFlaeche(2.0)
  - kreisFlaeche(input)

- Beim Aufruf der Methode werden folgende Schritte ausgeführt.
  - Erzeugung der Parametervariablen im Speicher.
  - Initialisierung der Parametervariablen durch die Werte, die beim Aufruf übergeben werden.
  - Schrittweise Ausführung der Methode
  - Am Ende der Methode
    - Löschen aller erzeugter Variablen
    - Rücksprung an die Stelle im Programm, von der die Methode aufgerufen wurde.

Beispiel

```
double kreisFlaeche (double r)  {
        final double pi = 3.14;
        return pi*r*r;
}
double input = 2.5;
double result = kreisFlaeche(input);
System.out.println("Flaecheninhalt von Kreis mit Radius "  + result);
```

Aufruf der Methode kreisFlaeche

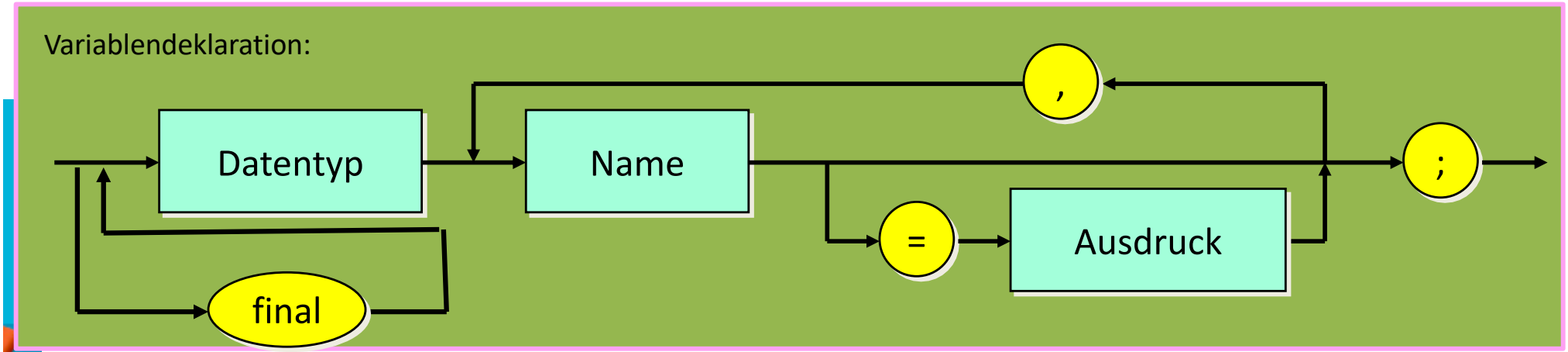
Ausführen der Methode kreisFlaeche

- Wert der Variable input wird gelesen und an Variable r übergeben.
- Am Ende wird das Ergebnis an den Aufrufenden geliefert.
- Parametervariablen und lokale Variablen werden gelöscht.

Es erfolgt ein Rücksprung an die Stelle des Aufrufs.

Lokale Variablen

- Deklaration erfolgt in einem der Anweisungsblöcke der Methode.

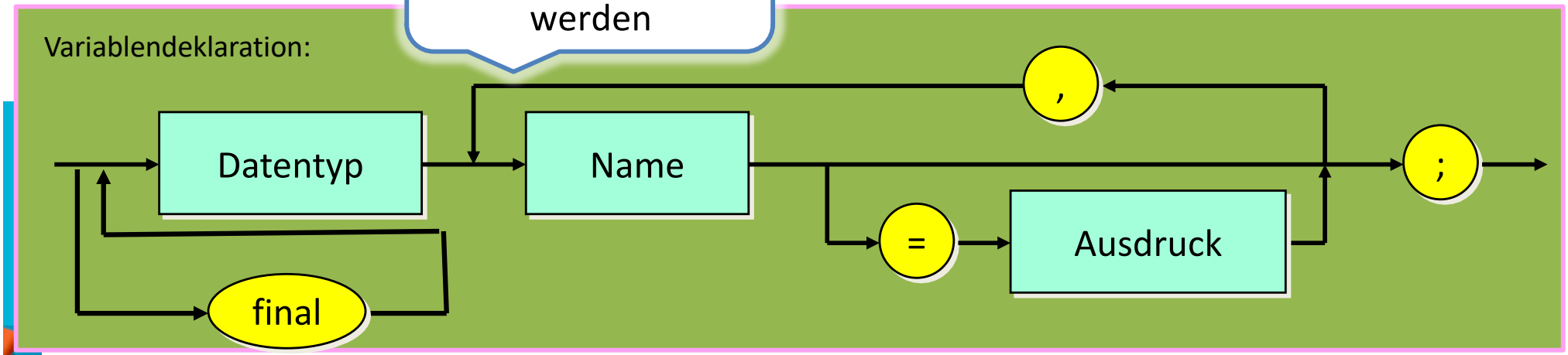


- Unterschiede zu Parametervariablen
  - Es können durch Komma getrennt mehrere Variablen in einer einzigen Deklaration definiert werden.
    - Alle Variablen haben den gleichen Datentyp.
  - Optional kann jede Variable einen initialen Wert bei der Deklaration erhalten.

Lokale Variablen

- Deklaration er nweisungsblöcke der Methode.

Rückverweis: Pfad kann wiederholt werden



- Unterschiede zu Parametervariablen
  - Es können durch Komma getrennt mehrere Variablen in einer einzigen Deklaration definiert werden.
    - Alle Variablen haben den gleichen Datentyp.
  - Optional kann jede Variable einen initialen Wert bei der Deklaraion erhalten.

Gültigkeit und Lebensdauer

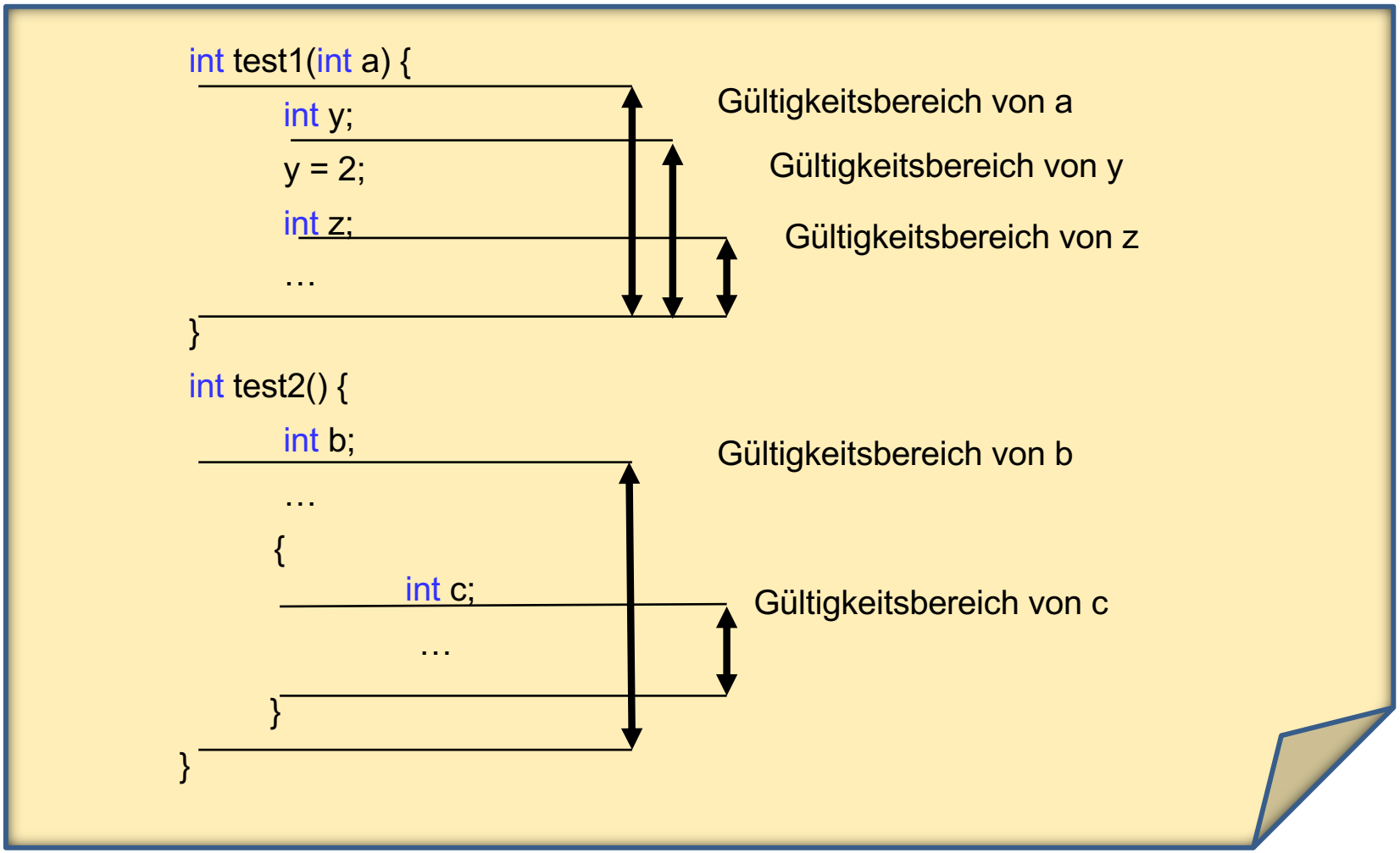
- Gültigkeitsbereich der lokalen Variable
  - erstreckt sich von der Deklaration bis zum Ende des umschließenden Blocks.
  - Innerhalb des Gültigkeitsbereich einer Variablen ist es nicht möglich eine Variable mit gleichem Namen zu deklarieren.
- Lebensdauer der lokalen Variable
  - Die Variable wird angelegt, wenn die Deklaration ausgeführt wird.
  - Die Variable wird am Ende der Ausführung des umschließenden Blocks gelöscht.
- Beispiel

```
double kreisFlaeche (double r)  {
        final double pi = 3.14;
        return pi*r*r;
}
```

Gültigkeitsbereich der Parametervariable r

Gültigkeitsbereich der lokalen Variable pi

Beispiel (Gültigkeitsbereich)



Live Vote

```
double flaecheZylinder(double h, double r) {
    double kreisFlaechen;
    {
        double eineKreisflaeche = 3.14 * r * r;
        kreisFlaechen = 2 * eineKreisflaeche;
    }
    double mantelFlaeche = 2 * 3.14 * h;
    return kreisFlaechen + mantelFlaeche;
}
```

- Wie viele Variablen sind an der markierten Stelle gültig?

PIN: EQYH

https://ilias.uni-marburg.de/vote/EQYH

Initialisierung lokaler Variablen

Zur Erinnerung

- Bei der Deklaration einer Variable wird ein Bezeichner eingeführt.
  - Vor dem Bezeichner steht der Datentyp.

```
int x = 42;
double y;
```

Programmschnipsel

Bezeichner	Wert	Typ
x	42	int
y	uninitialisiert	double

Speicher

- Im Unterschied zu Parametervariablen müssen diese explizit initialisiert werden.
  - Dies erfolgt unter Verwendung des Zuweisungsoperators
    - Entweder bereits bei der Deklaration (siehe Variable x)
      oder
    - zu einem späteren Zeitpunkt.
  - Das Lesen der Variablen vor ihrer Initialisierung ist nicht erlaubt.

Schreibender Zugriff auf Variablen

- Schreibend
  - Ein Schreiben der Variable erfolgt, wenn sie links von dem Zuweisungsoperator "=" steht.

```
x = 99;
```

  - Wiederholung
    - Das erste Schreiben einer Variable ist die Initialisierung.
    - Eine final-Variable kann nach der Initialisierung nicht wieder geschrieben werden.

Lesender Zugriff auf Variablen

- Lesend
  - Ein Lesen der Variablen erfolgt, wenn
    - sie auf der rechten Seite einer Zuweisung auftaucht.

```
z = 2*x;
```

    - sie als aktueller Parameter beim Aufruf einer Methode verwendet wird.

```
kreisFlaeche(x);
```

      Dabei bekommt die Parametervariable r den Wert von x zugewiesen, was der Zuweisung r = x; entspricht.
  - Lesen erfordert, dass zuvor die Variable initialisiert wurde.

Lesend und schreibender Zugriff auf Variablen

- Lesend und schreibender Zugriff in einer Anweisung

```
x = 2*x;
```

Dabei wird zunächst der Wert von x gelesen, mit zwei multipliziert und dann das Ergebnis wieder in x geschrieben.

- Diese Anweisungen haben mir schlaflose Nächte bereitet, bis ich verinnerlicht hatte, dass

'=' der Zuweisungsoperator ist (und nichts mit der mathematischen Gleichheit zu tun hat).