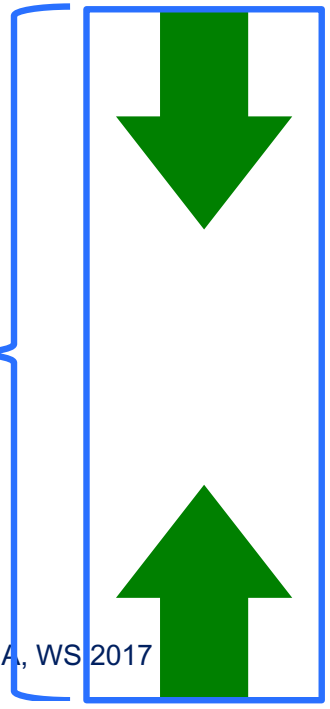
Speicherorganisation: der Stack

- Speicherorganisation: Stack und Heap
- Aktivierungssatz für eine Methode, Speicherbedarf
- Beispiele für das Wachsen und Schrumpfen des Stacks
  - Methode mit Rückgabewert
  - Methode ohne Rückgabewert
- Illustration in Eclipse
- (Keine) Namenskonflikte

Speicherverwaltung in Java 1/2

- Der Speicher in Java wird (wie in vielen Programmiersprachen) in zwei Bereiche aufgeteilt
  - Stack und
  - Heap
- Beim Programmstart ist nicht klar, wie viel Platz für diese Bereiche benötigt wird
- Diese Bereiche sollen sich nicht gegenseitig stören
- Lösung:
  Beide wachsen, je nach Bedarf, aufeinander zu

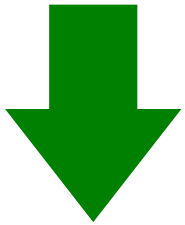


Verfügbarer Speicher für den Java-Prozess

Von "Oben" nach "Unten" im Speicher "wächst" der *Heap*

Von "Unten" nach "Oben" im Speicher "wächst" der *Stack*

Speicherverwaltung in Java 2/2

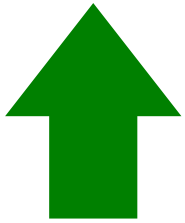


- Auf dem sog. *Heap* (eng. für Haufen) wird der Speicherplatz für *benutzerdefinierte* Datentypen und Arrays angelegt
- Benutzerdefinierte Datentypen (und String) beginnen alle mit einem Großbuchstaben

  Das ist nur eine Verabredung von Entwicklern; dem Compiler ist das egal!

- Auf dem sog. *Laufzeit-Stack* (engl. für Stapel) wird der Speicherplatz für Variable und formale Parameter von *elementaren* Datentypen angelegt
- Elementare (in die Sprache eingebaute) Datentypen sind
  - byte, short, int, long
  - char
  - float, double
  - boolean

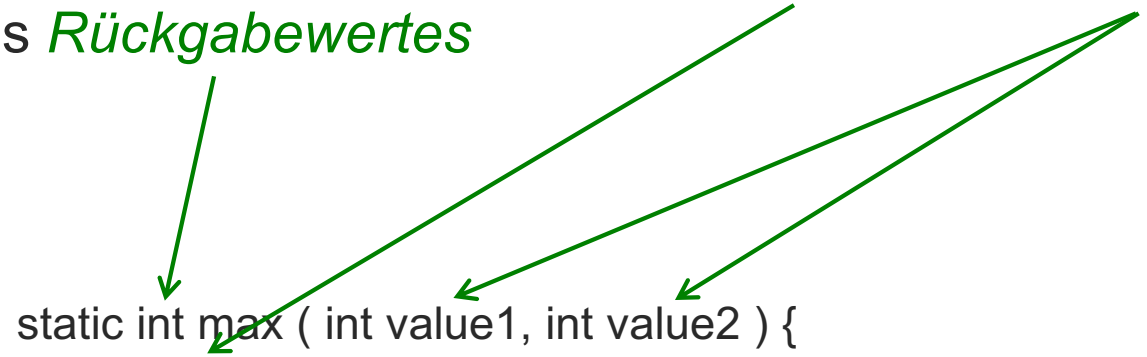
  -> Alle beginnen mit einem Kleinbuchstaben



- Für benutzerdefinierte Datentypen, Strings und Arrays wird dort ein Verweis (Deskriptor) auf den Heap angelegt

Aktivierungssatz von Methoden 1/2

- Je nach Bedarf wird für jede Methode im Speicher Platz reserviert
- Der Bedarf richtet sich nach dem Umfang (Speicherplatzbedarf) aller *lokalen Variablen*, *Parameter* und des *Rückgabewertes*



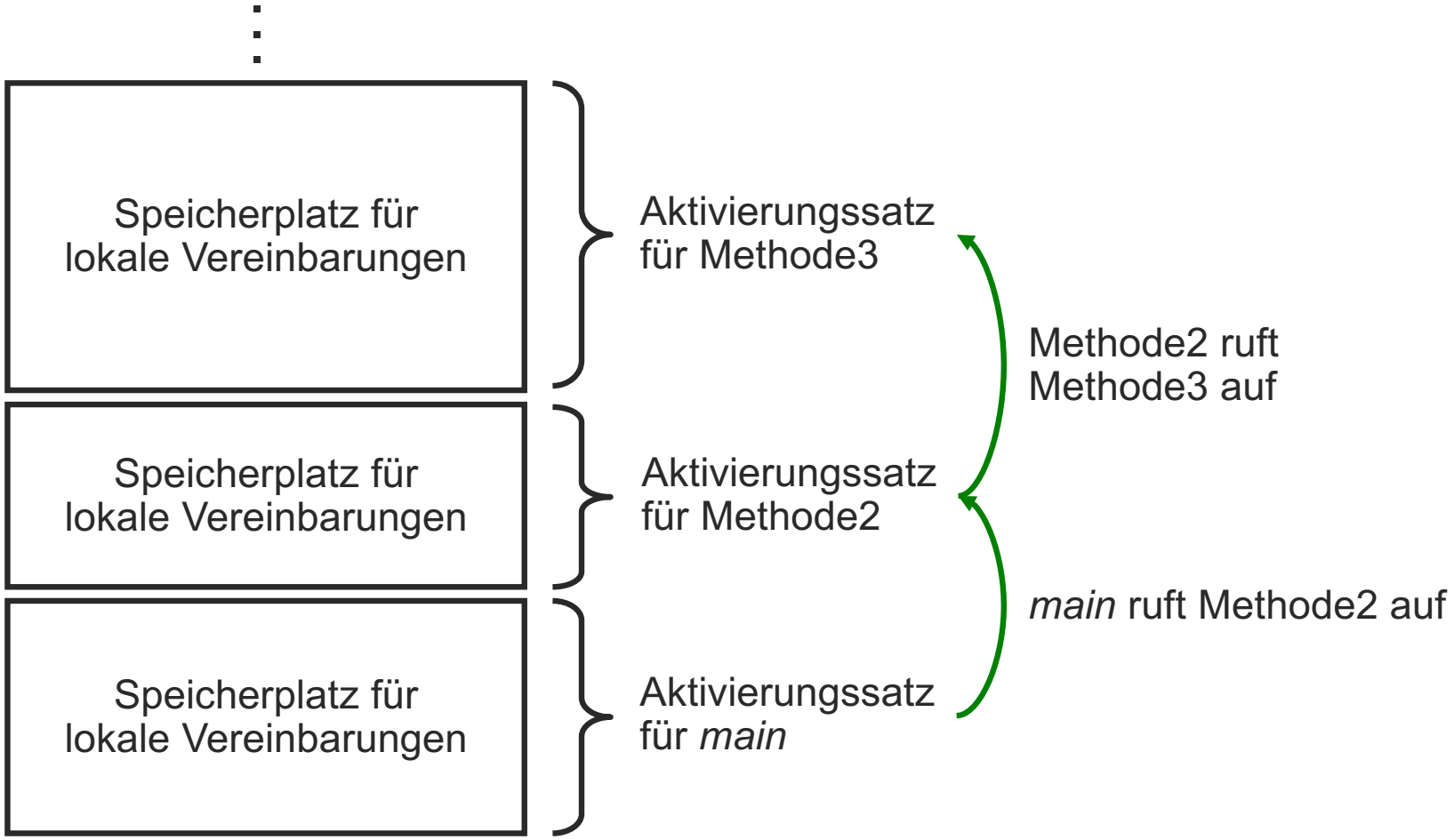
```
static int max ( int value1, int value2 ) {
    int result;
    if ( value1 >= value2 )
            result = value1;
    else
            result = value2;
    return result;
}
```

Aktivierungssatz von Methoden 2/2

- Je nach Bedarf wird für jede Methode im Speicher Platz reserviert
- Der Bedarf richtet sich nach dem Umfang (Speicherplatzbedarf) aller *lokalen Variablen*, *Parameter* und des *Rückgabewertes*
- Der reservierte Speicherbereich wird *Aktivierungssatz* (eigentlich: Aktivierungs*daten*satz) genannt, da er beim Aufruf, also der *Aktivierung* der Methode angelegt wird
- Der Aktivierungssatz wird *bei jeder* Aktivierung einer Methode (im Anschluss an den jeweils vorherigen auf dem *Stack*) angelegt

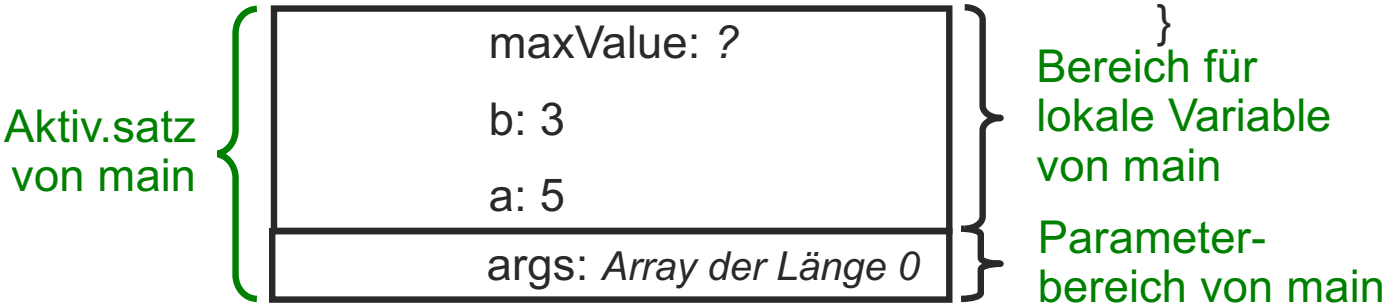
Aktivierungssätze auf dem Stack

- Die Aktivierungssätze verschiedener Methoden werden im Speicher in der Reihenfolge ihrer Aufrufe "aufeinander" gelegt



Beispiel für Wachsen und Schrumpfen des Stack

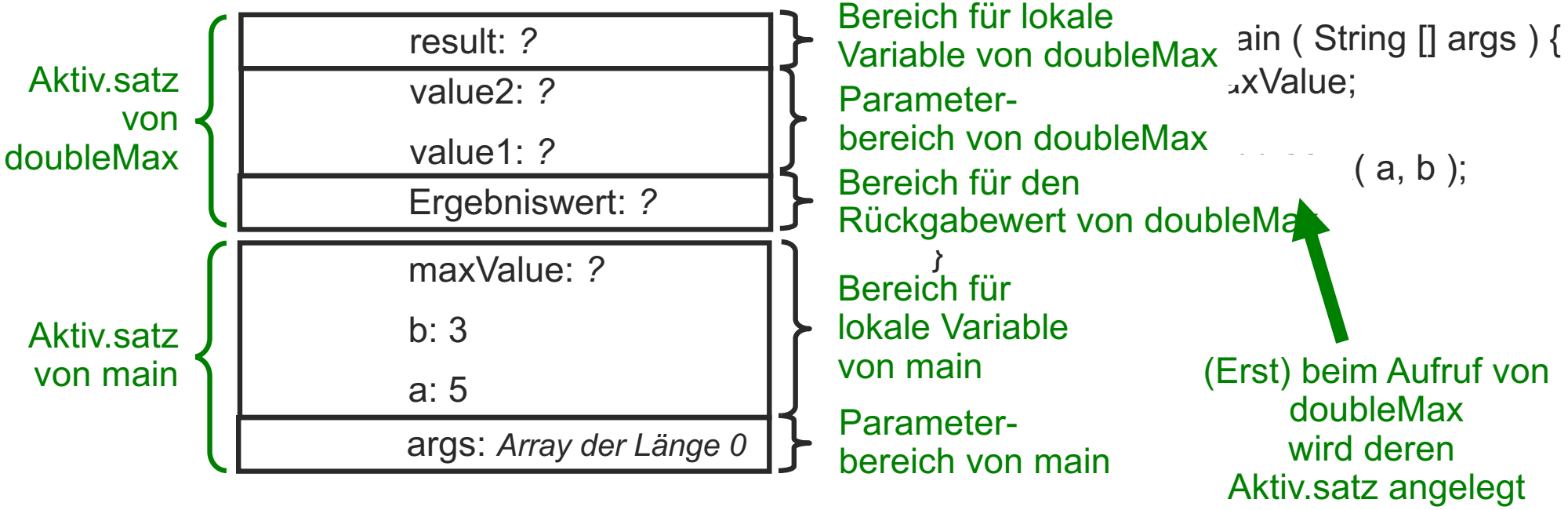
```
static int doubleMax(int value1, int value2)
{    int result;
     if ( value1 >= value2 )
          result = value1;
     else
          result = value2;
     return result * 2;
}


public static void main ( String [] args ) {
     int a=5, b=3, maxValue;
     ...
     maxValue = doubleMax( a, b );
     ...
}
```



(Erst) beim Aufruf von doubleMax wird deren Aktiv.satz angelegt

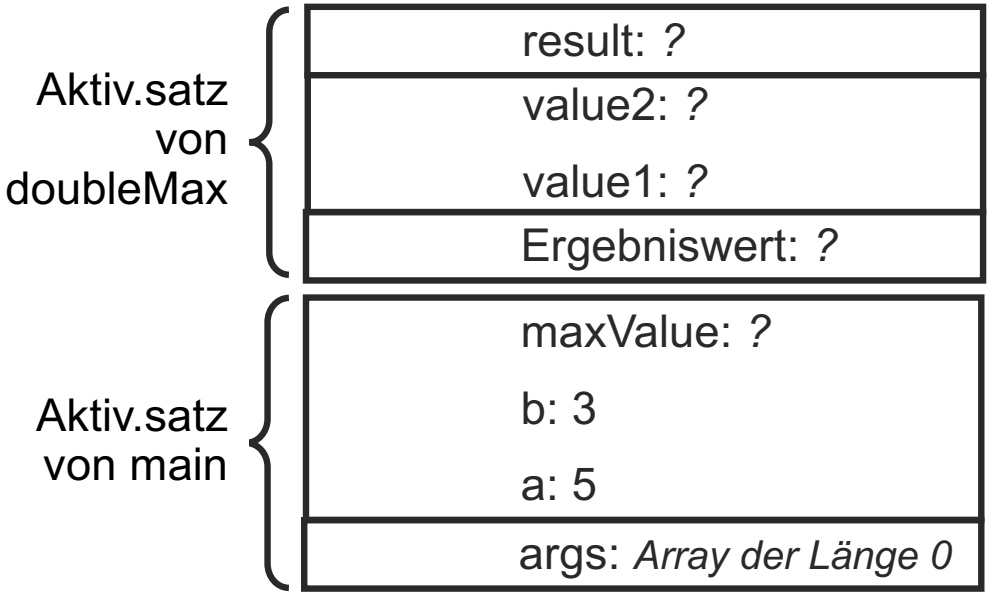
Beispiel für Wachsen und Schrumpfen des Stack

```
static int doubleMax(int value1, int value2)
{   int result;
    if ( value1 >= value2 )
        result = value1;
    else
        result = value2;
    return result * 2;
}
```



Beispiel für Wachsen und Schrumpfen des Stack

```
static int doubleMax(int value1, int value2)
{   int result;
    if ( value1 >= value2 )
        result = value1;
    else
        result = value2;
    return result * 2;
}
```



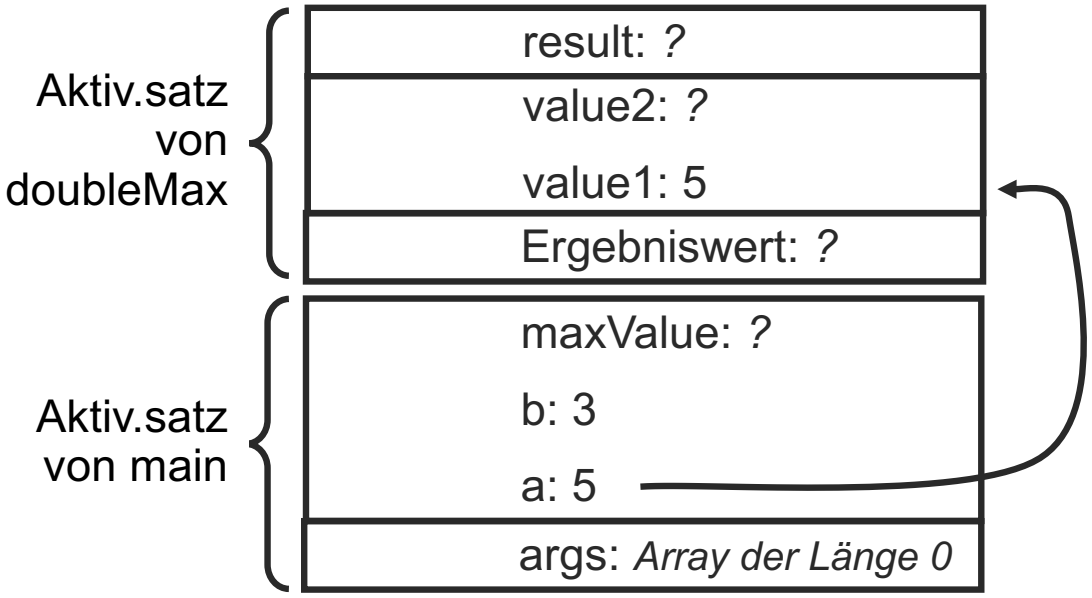

```
public static void main ( String [] args ) {
    int a=5, b=3, maxValue;
    ...
    maxValue = doubleMax( a, b );
    ...
}
```

(Erst) beim Aufruf von doubleMax wird deren Aktiv.satz angelegt

... und die aktuellen Parameter werden den formalen zugewiesen

Beispiel für Wachsen und Schrumpfen des Stack

```
static int doubleMax(int value1, int value2)
{   int result;
    if ( value1 >= value2 )
        result = value1;
    else
        result = value2;
    return result * 2;
}
```



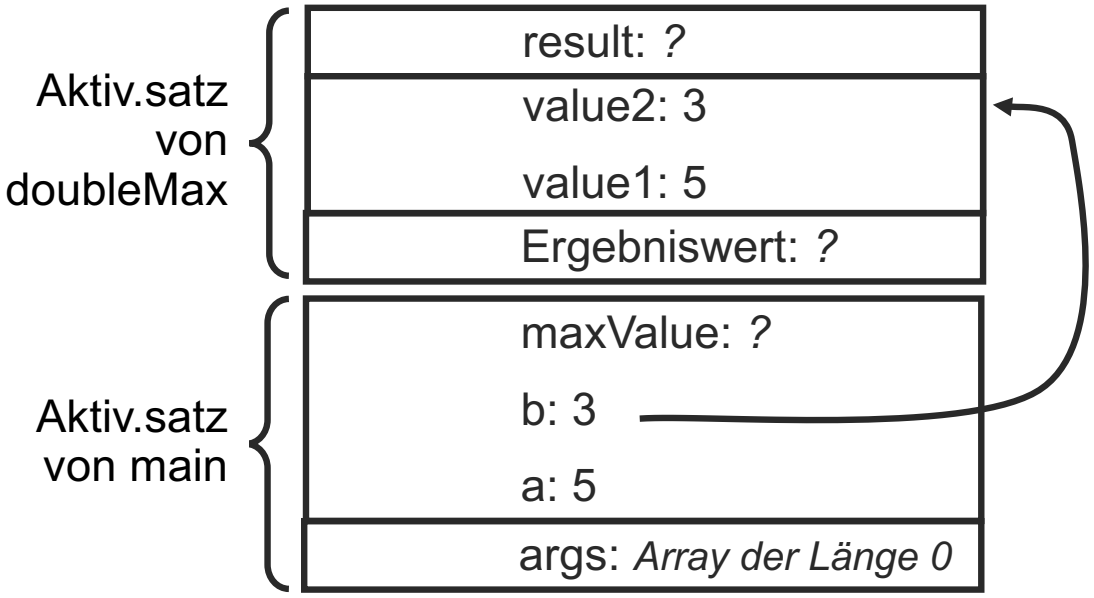

```
public static void main ( String [] args ) {
    int a=5, b=3, maxValue;
    ...
    maxValue = doubleMax( a, b );
    ...
}
```

(Erst) beim Aufruf von doubleMax wird deren Aktiv.satz angelegt

... und die aktuellen Parameter werden den formalen zugewiesen

Beispiel für Wachsen und Schrumpfen des Stack

```
static int doubleMax(int value1, int value2)
{   int result;
    if ( value1 >= value2 )
        result = value1;
    else
        result = value2;
    return result * 2;
}
```



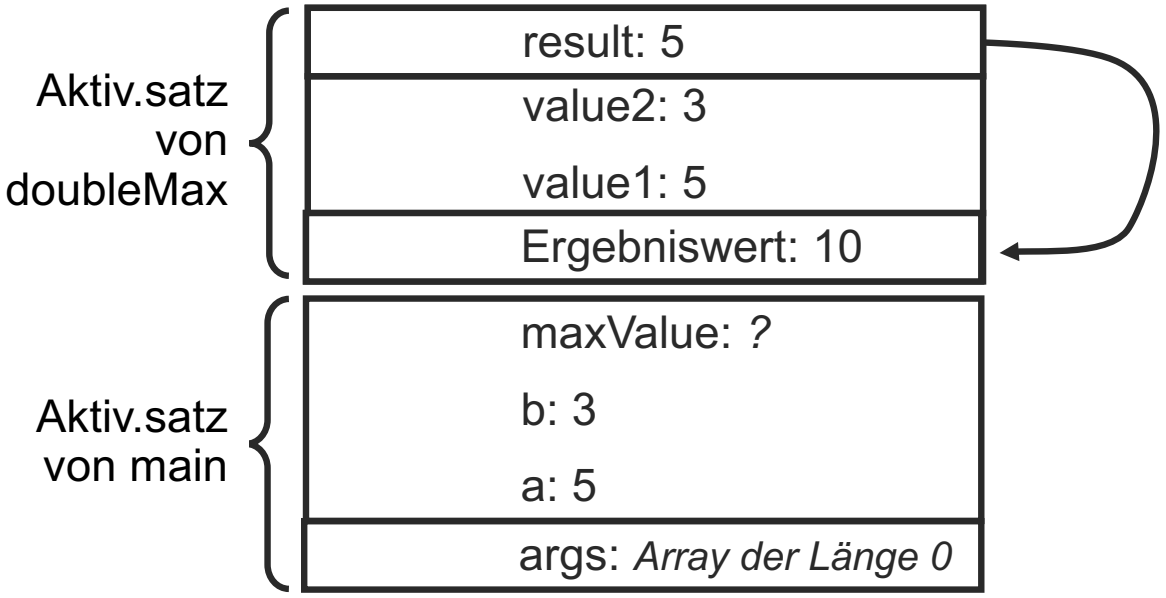

```
public static void main ( String [] args ) {
    int a=5, b=3, maxValue;
    ...
    maxValue = doubleMax( a, b );
    ...
}
```

(Erst) beim Aufruf von doubleMax wird deren Aktiv.satz angelegt

... und die aktuellen Parameter werden den formalen zugewiesen

Beispiel für Wachsen und Schrumpfen des Stack

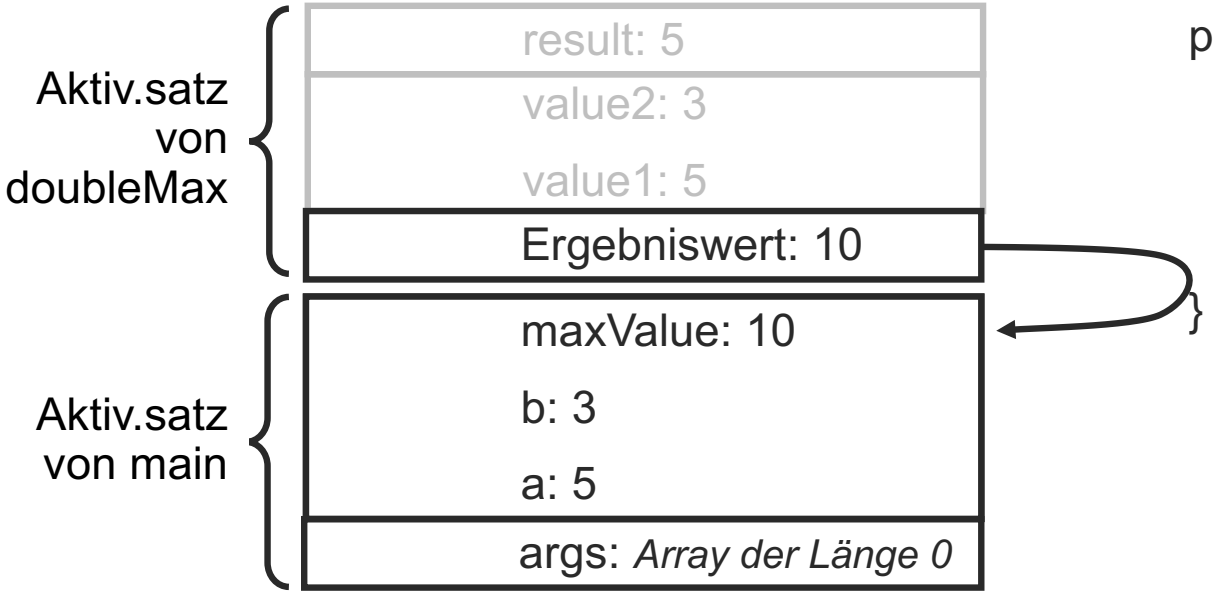
```
static int doubleMax(int value1, int value2)
{   int result;
    if ( value1 >= value2 )
        result = value1;
    else
        result = value2;
    return result * 2;
}
```




```
public static void main ( String [] args ) {
    int a=5, b=3, maxValue;
    ...
    maxValue = doubleMax( a, b );
    ...
}
```

Beispiel für Wachsen und Schrumpfen des Stack

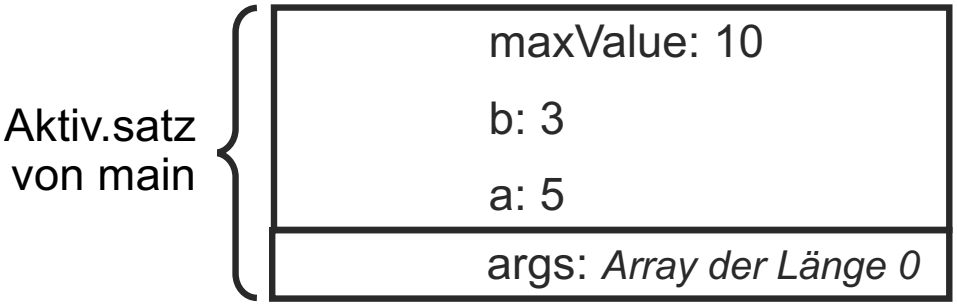
```
static int doubleMax(int value1, int value2)
{   int result;
    if ( value1 >= value2 )
        result = value1;
    else
        result = value2;
    return result * 2;
}
```




```
public static void main ( String [] args ) {
    int a=5, b=3, maxValue;
    ...
    maxValue = doubleMax( a, b );
    ...
}
```

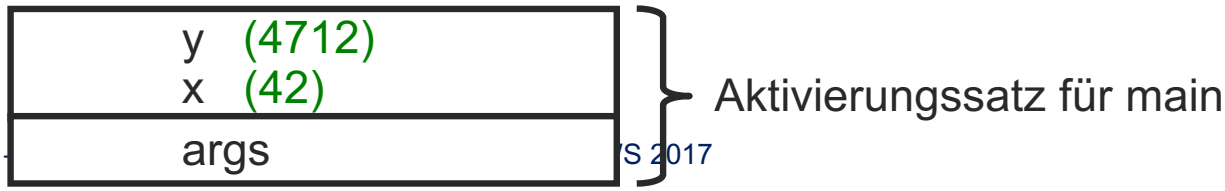
Beispiel für Wachsen und Schrumpfen des Stack

```
static int doubleMax(int value1, int value2)
{   int result;
    if ( value1 >= value2 )
        result = value1;
    else
        result = value2;
    return result * 2;
}


public static void main ( String [] args ) {
    int a=5, b=3, maxValue;
    ...
    maxValue = doubleMax( a, b );
    ...
}
```



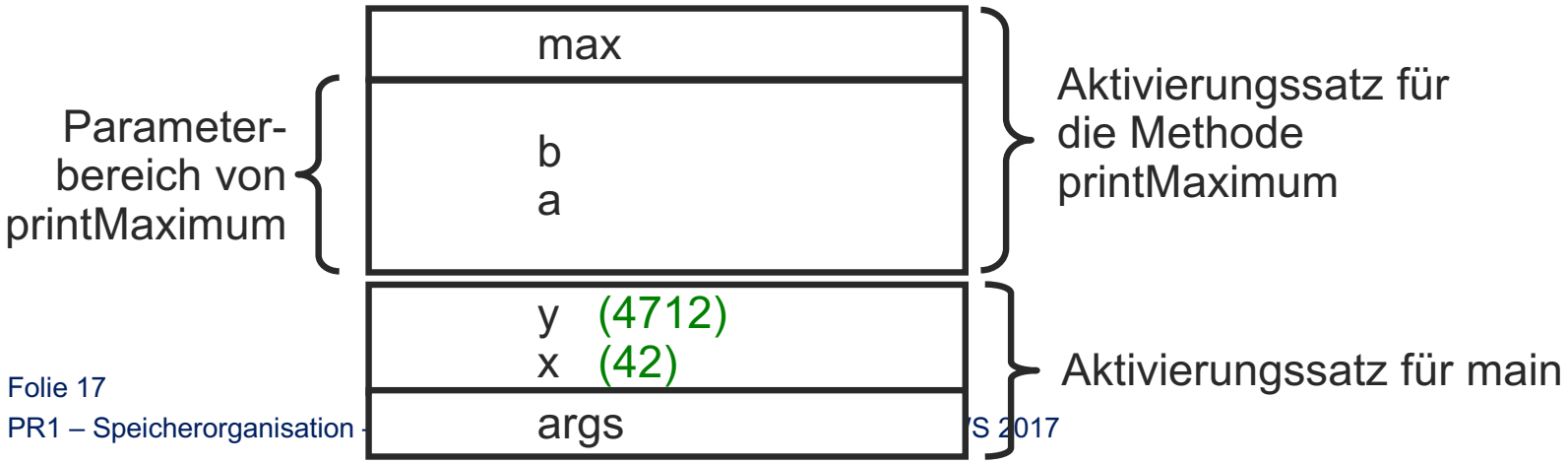
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        printMaximum( x, y );
}
```



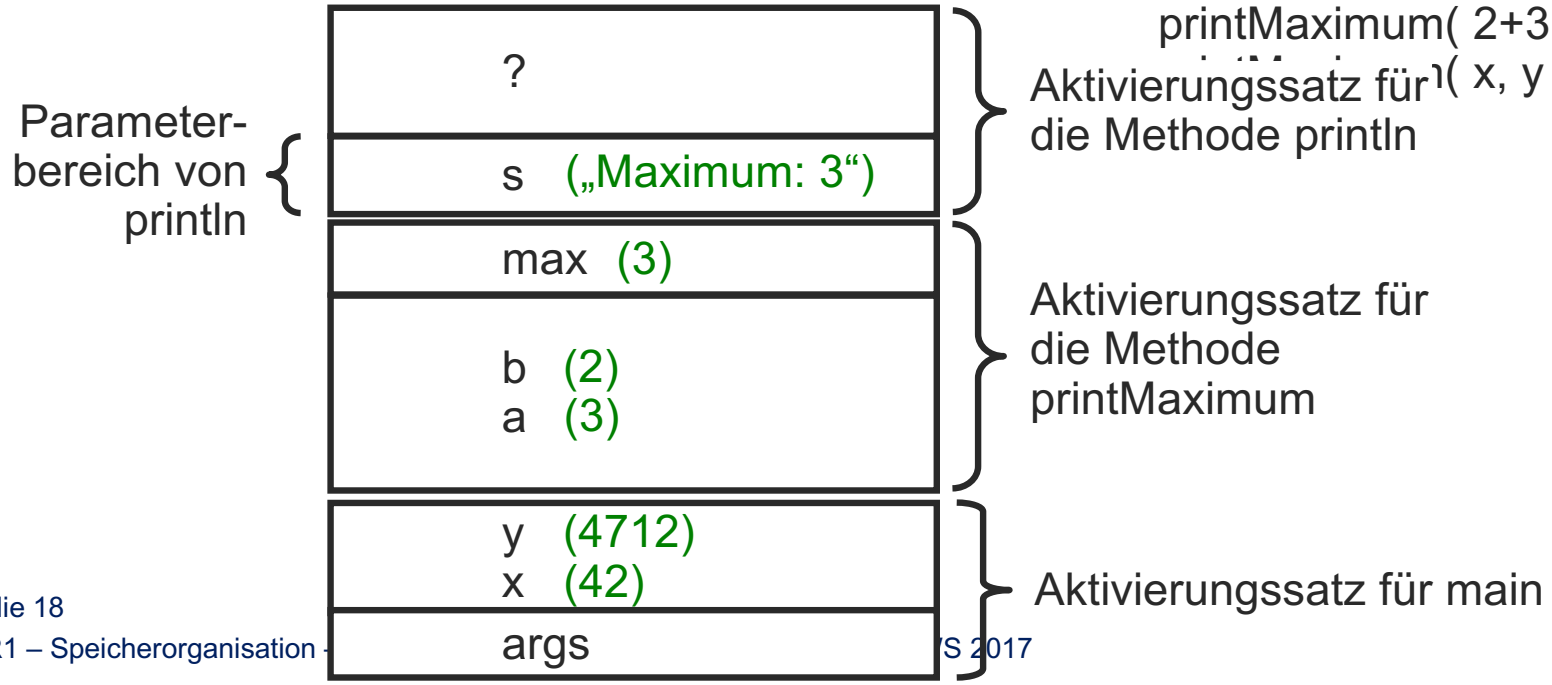
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        printMaximum( x, y );
}
```



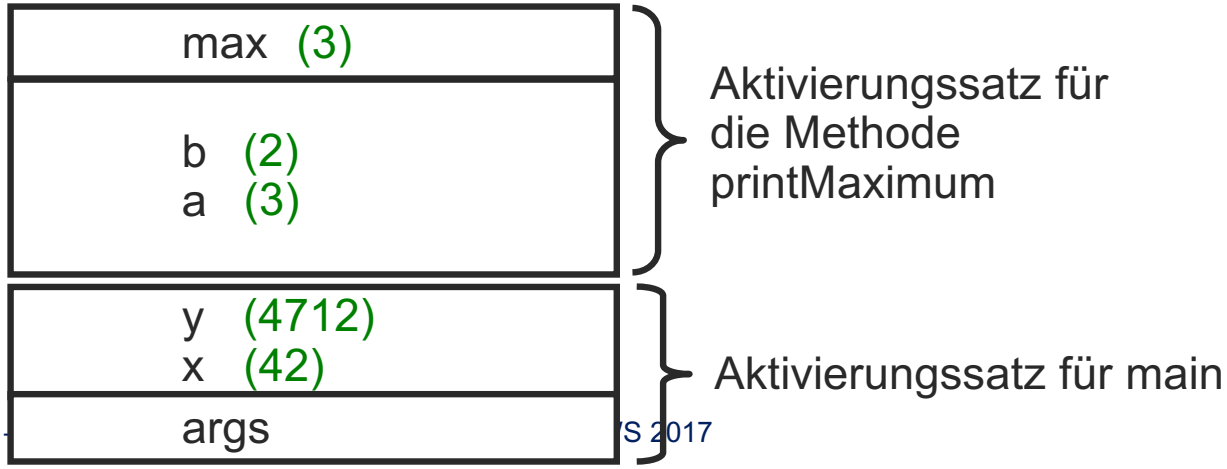
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        ...( x, y );
```



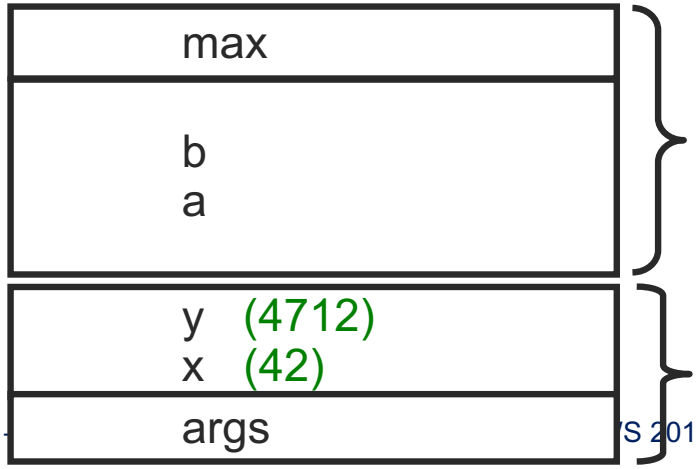
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}


public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        printMaximum( x, y );
}
```



Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        printMaximum( x, y );
}
```

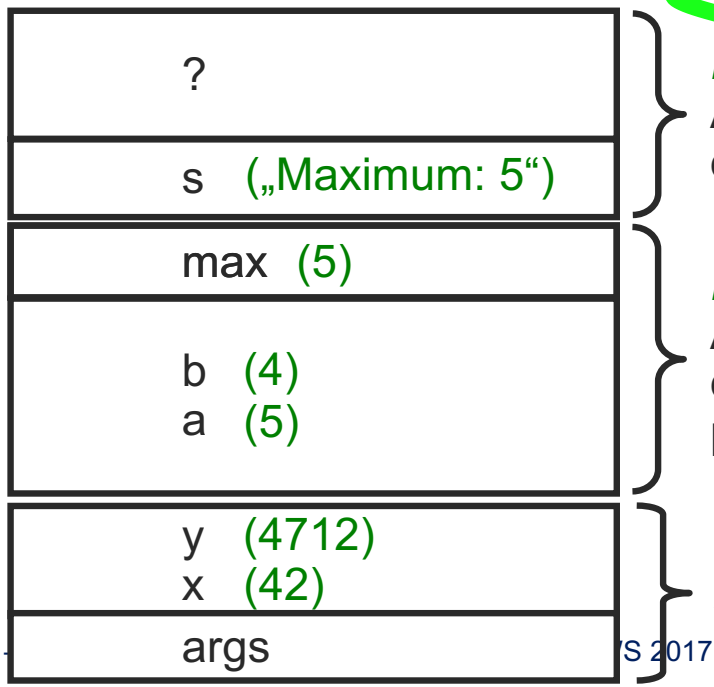


Neu angelegter Aktivierungssatz für die Methode printMaximum

Aktivierungssatz für main

Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 2+3, 4 );
```



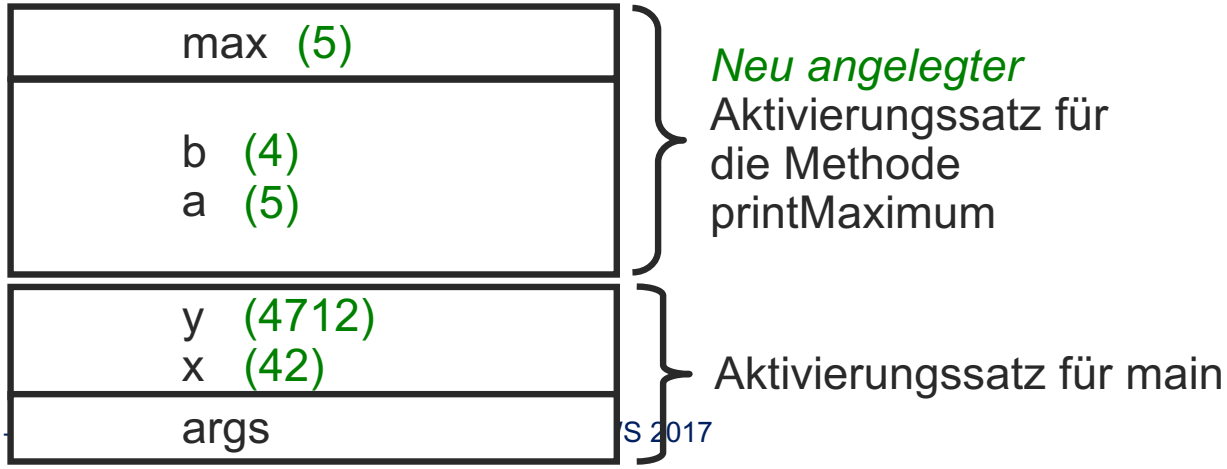
Neu angelegter Aktivierungssatz für die Methode println

Neu angelegter Aktivierungssatz für die Methode printMaximum

Aktivierungssatz für main

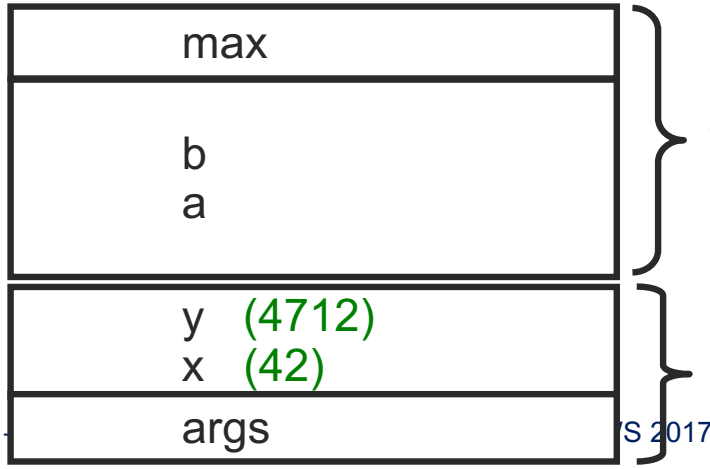
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}


public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        printMaximum( x, y );
}
```



Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 1 );
        printMaximum( x, y );
}
```

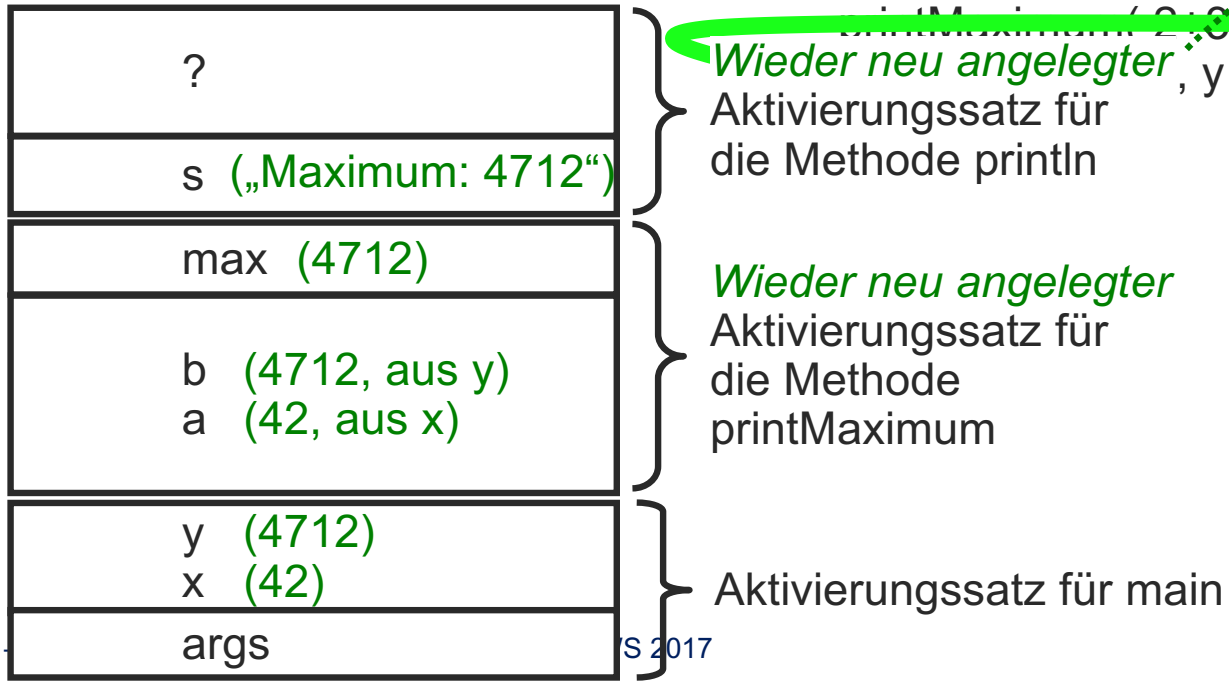


Wieder neu angelegter Aktivierungssatz für die Methode printMaximum

Aktivierungssatz für main

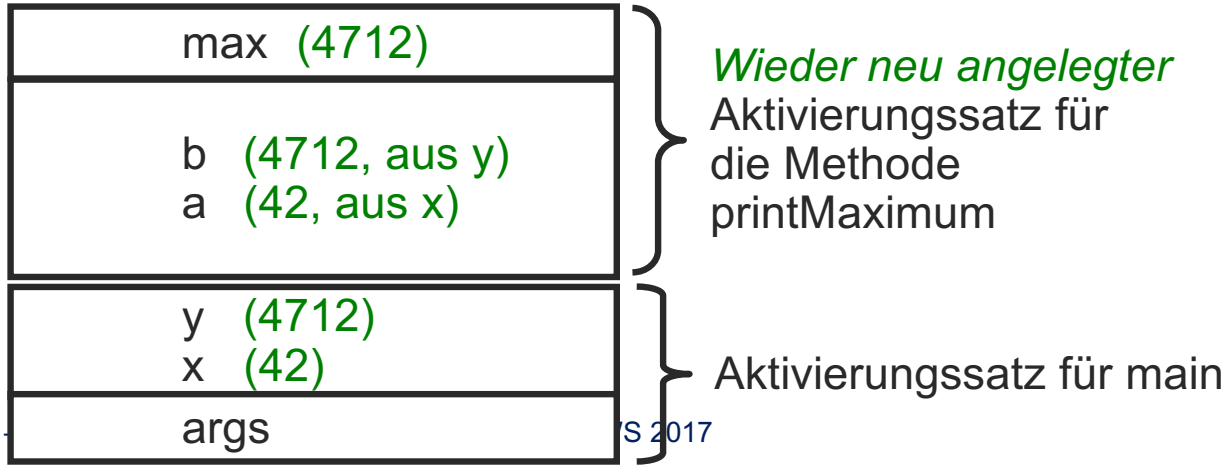
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        ...
        ...( ..., y );
```



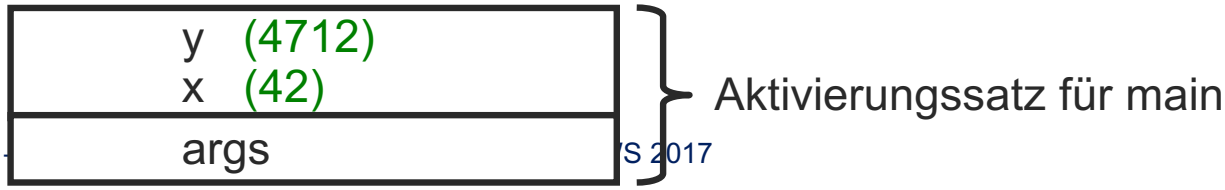
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2*3, 1 );
        printMaximum( x, y );
}
```



Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                max = a;
        else
                max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        printMaximum( x, y );
}
```

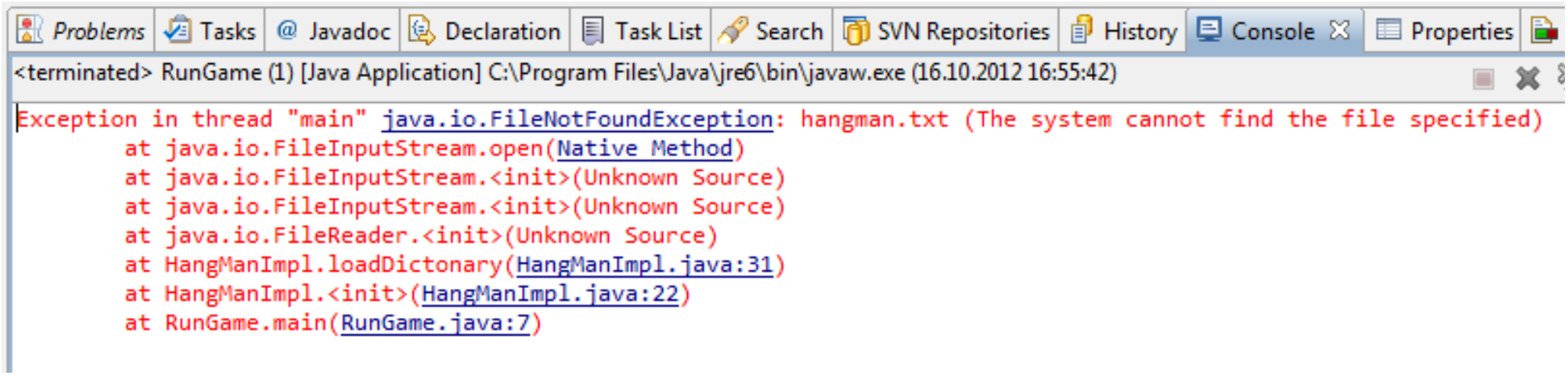


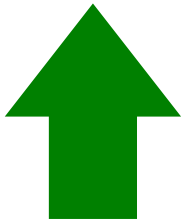
Speicherorganisation bei Prozeduren (Methoden ohne Rückgabewert)

```
static void printMaximum ( int a, int b )
{       int max;
        if ( a > b )
                        max = a;
        else
                        max = b
        println( “Maximum: “ + max );
}

public static void main ( String [ ] args )
{       int x = 42, y = 4712;
        printMaximum( 3, 5/2 );
        printMaximum( 2+3, 4 );
        printMaximum( x, y );
}
```

Haben Sie Fragen?

Bekannte Situation, in der man den Laufzeit-Stack sieht: bei Laufzeitfehlern





Von "Unten" nach "Oben" im Speicher "wächst" der *Stack*

Erklärung: keine Namenskonflikte

- Es gibt *keine Konflikte* bei gleichnamigen Variablen in *verschiedenen* Methoden
- Gegeben sei das folgende Programmfragment

```
public static void main ( String [ ] args ) {
    int zahl;
    ...
}
static void druckeAdresse ()
{
    int i, zahl; ........
```

- Daraus ergibt sich die folgende Struktur im Speicher