Javakurs 12: Methoden

Johannes Heinemann Sadik Hasanovic

basierend auf der Vorlage von
Mario Bodemann und Sebastian Dyroff

Technische Universität Berlin

6. März 2012

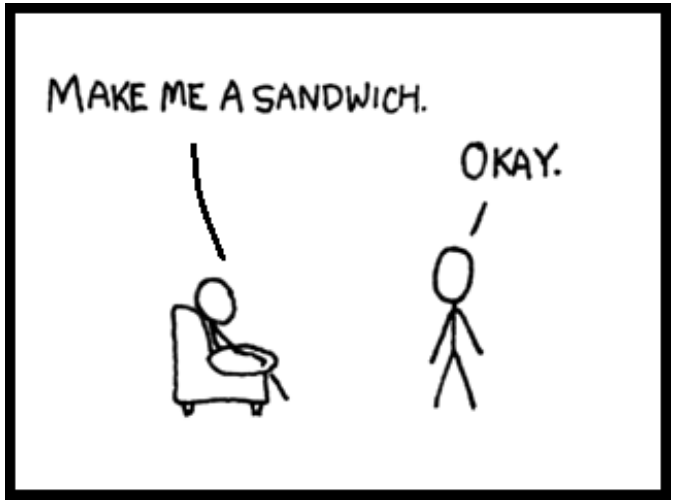
Inhaltsverzeichnis

Inhaltsverzeichnis

Was sind Methoden?

Was sind Methoden?

- Eine Methode ist eine Aufforderung etwas zu tun.

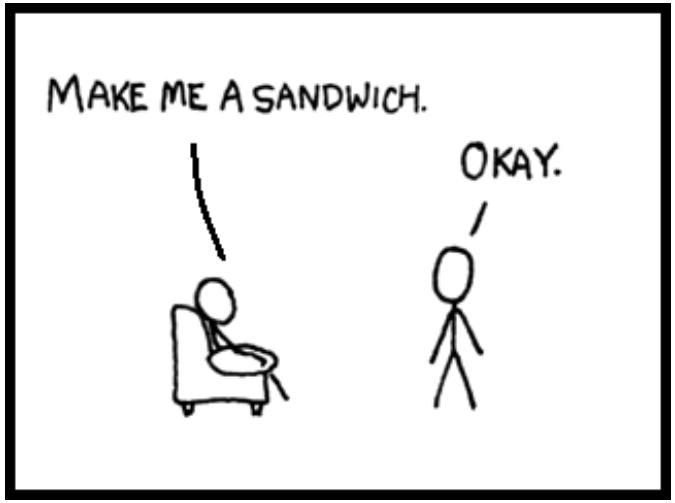


[1]

[1] xkcd. http://xkcd.com/149/

Was sind Methoden?

- Eine Methode ist eine Aufforderung etwas zu tun.
- Im Leben könnte das zum Beispiel so aussehen:

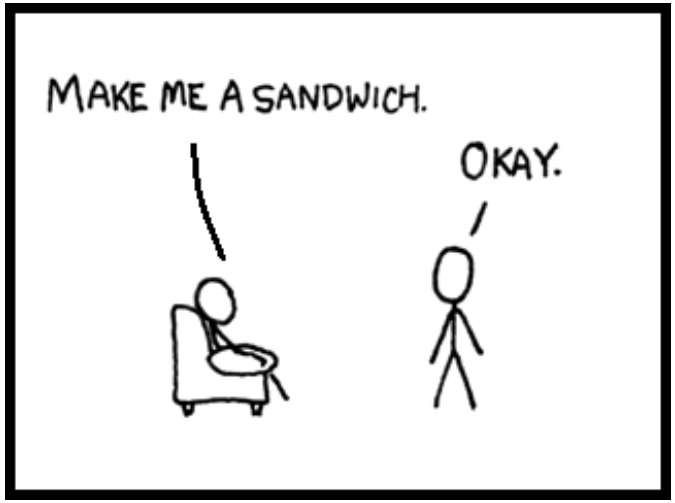


[1]

[1] xkcd. http://xkcd.com/149/

Was sind Methoden?

- Eine Methode ist eine Aufforderung etwas zu tun.
- Im Leben könnte das zum Beispiel so aussehen:



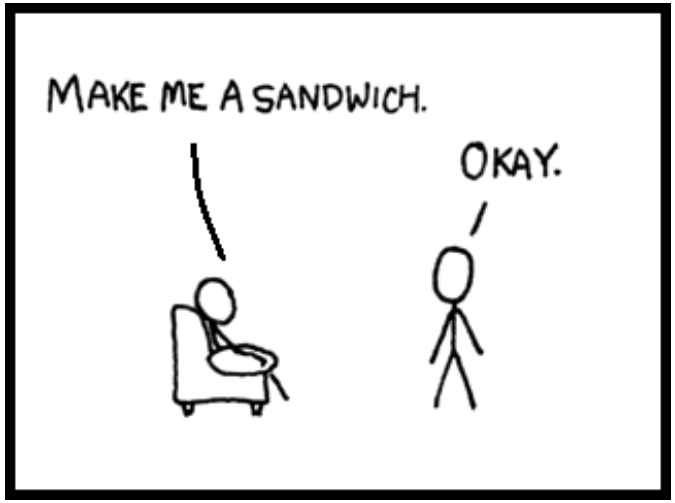
[1]

- Aufforderung:
  Make me a Sandwich!

[1] xkcd. http://xkcd.com/149/

Was sind Methoden?

- Eine Methode ist eine Aufforderung etwas zu tun.
- Im Leben könnte das zum Beispiel so aussehen:



[1]

- Aufforderung:
  Make me a Sandwich!
- Ergebnis: Sandwich

[1] xkcd. http://xkcd.com/149/

Methoden mit Rückgabe

Methoden mit Rückgabe - Bauanleitung

Methoden

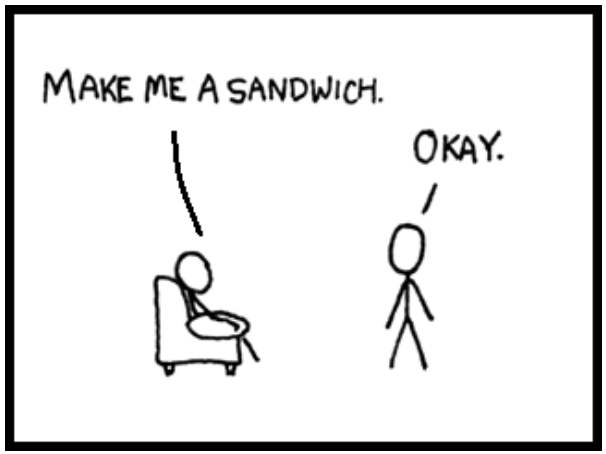
```
<Ergebnistyp> <Name> (<ParameterListe>){
    // do something
    return something;
}
```

Parameterliste

$<\text{Typ}_1> <\text{Name}_1>, \ldots, <\text{Typ}_n> <\text{Name}_n>$

Methoden mit Rückgabe - Bauanleitung

- Wie sieht unser voriges Beispiel in Java aus?



Listing 1:

```
1 public Sandwich makeSandwich(){
2 Salad salad = getSalad();
3 Mustard mustard = getMustard();
4 Bread bread = getBread();
5 Sandwich sandwich =
6 putTogether(bread, mustard,
        salad);
7 return sandwich;
8 }
```

Methoden mit Rückgabe - Bauanleitung

- Wie sehen Methoden in Java aus?

Methoden mit Rückgabe - Bauanleitung

- Wie sehen Methoden in Java aus?
- Mit einem Parameter:

Listing 3:

```
1 public static float kreisFlaeche(
    float radius){
2     return radius * radius * 3,1416F;
3 }
```

Methoden mit Rückgabe - Bauanleitung

- Wie sehen Methoden in Java aus?

Methoden mit Rückgabe - Bauanleitung

- Wie sehen Methoden in Java aus?
- Mit mehreren Parametern und geschachtelt:

Listing 5:

```
1 public static float zylinderVolumen(
    float hoehe, float radius){
2     return hoehe * kreisFlaeche(radius);
3 }
```

Methoden mit Rückgabe - Beispiel

Listing 6: src/MyMath.java

```
1  public class MyMath {
2    public static void main (String[] args) {
3      int fakFour = factorial(4);
4      System.out.println("4! = " + fakFour);
5    }
6    public static int factorial (int zahl) {
7      int result = 1;
8      for(int i=1 ; i<=zahl ; i++) {
9        result = result * i;
10     }
11     return result;
12   }
13 }
```

Methoden mit Rückgabe - Beispiel

- Was sollt ihr aus dem Beispiel mitnehmen?

Methoden mit Rückgabe - Beispiel

- Was sollt ihr aus dem Beispiel mitnehmen?
- Gründe für die Deklaration von Methoden:
  1. Wiederkehrende Programmteile müssen nicht immer wieder programmiert werden!
  2. Bei Umfangreichen Programmen können einzelne Teile in Methoden zusammengefasst werden, um das Programm übersichtlicher zu gestalten!

Methoden mit Rückgabe - Beliebte Fehler

Listing 7: src/Foo.java

```
1 public static int foo( int zahl){
2   int i = 1;
3   return i;
4   i = i + zahl; // Unreachable code!
5 }
```

Methoden mit Rückgabe - Beliebte Fehler

- Richtig wäre:

Listing 8: src/Foo.java

```
1 public static int foo( int zahl){
2   int i = 1;
3   i = i + zahl; // Reachable code!
4   return i;
5 }
```

Methoden mit Rückgabe - Beliebte Fehler

- Wo ist der Fehler?

Listing 9: src/EvenOrOdd.java

```
1 public static boolean isEven( int d ){
2   // Compilerfehler
3   if ( d%2 == 0 ){
4     return "true";
5   }else{
6     return "false";
7   }
8 }
```

Methoden mit Rückgabe - Beliebte Fehler

Fehlerausgabe:

```
EvenOrOdd.java:4: incompatible types
found   : java.lang.String
required: boolean
                        return "true";
                               ^
EvenOrOdd.java:6: incompatible types
found   : java.lang.String
required: boolean
                        return "false";
                               ^
2 errors
```

Methoden mit Rückgabe - Beliebte Fehler

- Richtig wäre:

Listing 10: src/EvenOrOdd.java

```
1 public static boolean isEven( int d ){
2
3   if ( d%2 == 0 ){
4     return true;
5   }else{
6     return false;
7   }
8 }
```

Methoden mit Rückgabe - Beliebte Fehler

- Wo ist der Fehler?

Listing 11: src/PosOrNeg.java

```
1 public static String posOrNeg( int d ){
2   // Compilerfehler
3   if ( d >= 0 ){
4     return "pos";
5   }
6   if ( d < 0 ){
7     return "neg";
8   }
9 }
```

Methoden mit Rückgabe - Beliebte Fehler

Fehlerausgabe:

```
PosOrNeg.java:9: missing return statement
        }
        ^
1 error
```

Methoden mit Rückgabe - Beliebte Fehler

- Richtig wäre:

Listing 12: src/PosOrNeg.java

```
1 public static String posOrNeg( int d ){
2   if ( d >= 0 ){
3     return "pos";
4   }
5   if ( d < 0 ){
6     return "neg";
7   }
8   return "keines von beiden";
9 }
```

Methoden ohne Rückgabe

Methoden ohne Rückgabe - Bauanleitung

Methoden

```
void <Name> (<ParameterListe>){
    // do something
}
```

Parameterliste

$<\text{Typ}_1> <\text{Name}_1>, \ldots, <\text{Typ}_n> <\text{Name}_n>$

Methoden ohne Rückgabe - Beispiele

- Wie sieht das in Java aus?

Methoden ohne Rückgabe - Beispiele

- Wie sieht das in Java aus?
- Ein altbekanntes Beispiel ohne Parameter:

Listing 14: src/HelloWorld.java

```
1 public class HelloWorld {
2   public static void main(String[] args) {
3     helloWorld();
4   }
5   public static void helloWorld() {
6     System.out.println("Hello World!");
7   }
8 }
```

Methoden ohne Rückgabe - Beispiele

- Ein einfaches Beispiel mit Parametern:

Listing 15: src/PrintMax.java

```
1  public class Max{
2   public static void main (String args[]) {
3      printMax(1,2);
4    }
5    public static void printMax(int a,int b){
6      if ( a > b ){
7        System.out.println( a );
8      } else {
9        System.out.println( b );
10     }
11   }
12 }
```

Methoden ohne Rückgabe - Beispiele

Listing 16: src/Main.java

```
1 public static void main( String[] args ){
2   System.out.println( "Hello World!" );
3 }
```

- Die statische Methode liefert keine Rückgabe, daher ist der Rückgabetyp void.

Methoden ohne Rückgabe - Beispiele

Listing 17: src/Main.java

```
1 public static void main( String[] args ){
2   System.out.println( "Hello World!" );
3 }
```

- Die statische Methode liefert keine Rückgabe, daher ist der Rückgabetyp void.
- Der Methodenname ist main.

Methoden ohne Rückgabe - Beispiele

Listing 18: src/Main.java

```
1 public static void main( String[] args ){
2   System.out.println( "Hello World!" );
3 }
```

- Die statische Methode liefert keine Rückgabe, daher ist der Rückgabetyp void.
- Der Methodenname ist main.
- Die Parameterliste ist String[] args.

Methoden ohne Rückgabe - Beispiele

Listing 19: src/Main.java

```
1 public static void main( String[] args ){
2   System.out.println( "Hello World!" );
3 }
```

- Die statische Methode liefert keine Rückgabe, daher ist der Rückgabetyp void.
- Der Methodenname ist main.
- Die Parameterliste ist String[] args.
- Der Rumpf besteht nur aus der Bildschirmausgabe.

Zusammenfassung

Methoden

```
<Ergebnistyp> <Name> (<ParameterListe>){
    // do something
    return something;
}
```

Methoden	Ohne Rückgabe	Mit Rückgabe
ohne Parameter	helloWorld()	makeSandwich()
mit Parameter	printMax(a,b)	kreisFlaeche(f)

Noch Fragen?

... dann eine kurze Pause!

Inhaltsverzeichnis

Variablen in Methoden

Der Befehl

Listing 20:

```
1 int i = 4;
```

führt genau genommen 2 Befehle aus:

Variablen in Methoden

Der Befehl

Listing 22:

```
int i = 4;
```

führt genau genommen 2 Befehle aus:

Listing 23:

```
int i;
i = 4;
```

Zuerst wird die Variable deklariert, danach initialisiert.

Variablen in Methoden

Deklaration: Es wird Speicher angefordert.

Variablen in Methoden

Deklaration: Es wird Speicher angefordert.
Initialisierung: Wir weisen der Variable einen Wert zu.

Scope

Scope

Der Scope definiert den Sichtbarkeitsbereich einer Variablen. Eine Variable ist nur dort sichtbar, wo sie deklariert wurde.

Scope: Beispiel

Listing 24: src/Scope.java

```
1  public class Scope{
2
3    public static void main(String args[]) {
4      int zahl;
5      zahl = 5;
6      zeigeZahl();
7    }
8
9    public static void zeigeZahl() {
10     System.out.println(zahl);
11   }
12 }
```

Scope: Beispiel

Ausgabe

```
Scope.java:10: cannot find symbol
symbol  : variable zahl
location: class Scope
System.out.println(zahl);
                   ^
1 error
```

Die Variable `int zahl` wurde in der main-Methode deklariert und ist nur in dieser sichbar, nicht aber in der Methode `zeigeZahl()`.

Scope: Beispiel 2

Listing 25: src/Scope2.java

```
1   public static void main(String args[]) {
2     int ergebnis = addiere(8,subtrahiere(3,4));
3     System.out.println(ergebnis);
4   }
5   public static int addiere(int x, int y) {
6     int ergebnis = x+y;
7     return ergebnis;
8   }
9   public static int subtrahiere(int x, int y) {
10    int ergebnis = x-y;
11    return ergebnis;
12  }
```

Scope: Beispiel

Die Variable `int ergebnis` wurde insgesamt drei mal deklariert. Ist das nicht falsch?

Scope: Beispiel

Die Variable `int ergebnis` wurde insgesamt drei mal deklariert. Ist das nicht falsch?
Nein. Jede Deklaration erfolgt in einer anderen Methode, deshalb hat jede Variable ihren eigenen Scope.

lokale Variablen

Manchmal möchte man Variablen haben, auf die man in jeder Methode zugreifen kann.

lokale Variablen

Manchmal möchte man Variablen haben, auf die man in jeder Methode zugreifen kann.
Lösung:

Klassenvariablen

Eine Variable, die innerhalb der Klasse deklariert wird, ist in allen Methoden der Klasse sichtbar.

lokale Variablen

Listing 26: src/Locals.java

```
1 public class Locals{
2
3   int x = 8;
4
5   public static void main(String args[]) {
6     // ...
7   }
8 }
```

lokale und globale Variablen

Listing 27: src/Variables.java

```
1  public class Variables{
2
3    int zahl = 7;
4
5    public static void main(String args[]) {
6      int b = foo(3);
7      System.out.println(b);
8    }
9    public static int foo(int x) {
10     int zahl = x*x;
11     return zahl;
12   }
13 }
```

Verschattung

Verschattung

Wird eine Variable lokal deklariert, so wird die gleichnamige Klassenvariable verschattet, d.h. sie ist innerhalb der Methode temporär nicht sichtbar. Dies gilt auch für Variablen aus Parameterlisten.

lokale und globale Variablen

Listing 28: src/Variables2.java

```
1  public class Variables2{
2
3    int zahl = 7;
4
5    public static void main(String args[]) {
6      int b = foo(3);
7      System.out.println(b);
8    }
9    public static int foo(int zahl) {
10     return zahl*zahl;
11   }
12 }
```

Call by Value

Wie werden Parameterwerte übergeben?

Call by Value

Listing 29: src/DoubleNumber.java

```
1 public class DoubleNumber{
2   public static void main (String args[]) {
3     int i = 3;
4     redouble(i);
5     System.out.println(i);
6   }
7   public static  void redouble(int i) {
8     i = i * 2;
9   }
```

Call by Value

Ausgabe: 3
Der Wert wurde nicht verdoppelt.

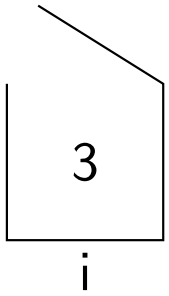
Primitive Datentypen

main()

Primitive Datentypen

main()

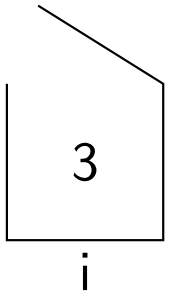
int i = 3;



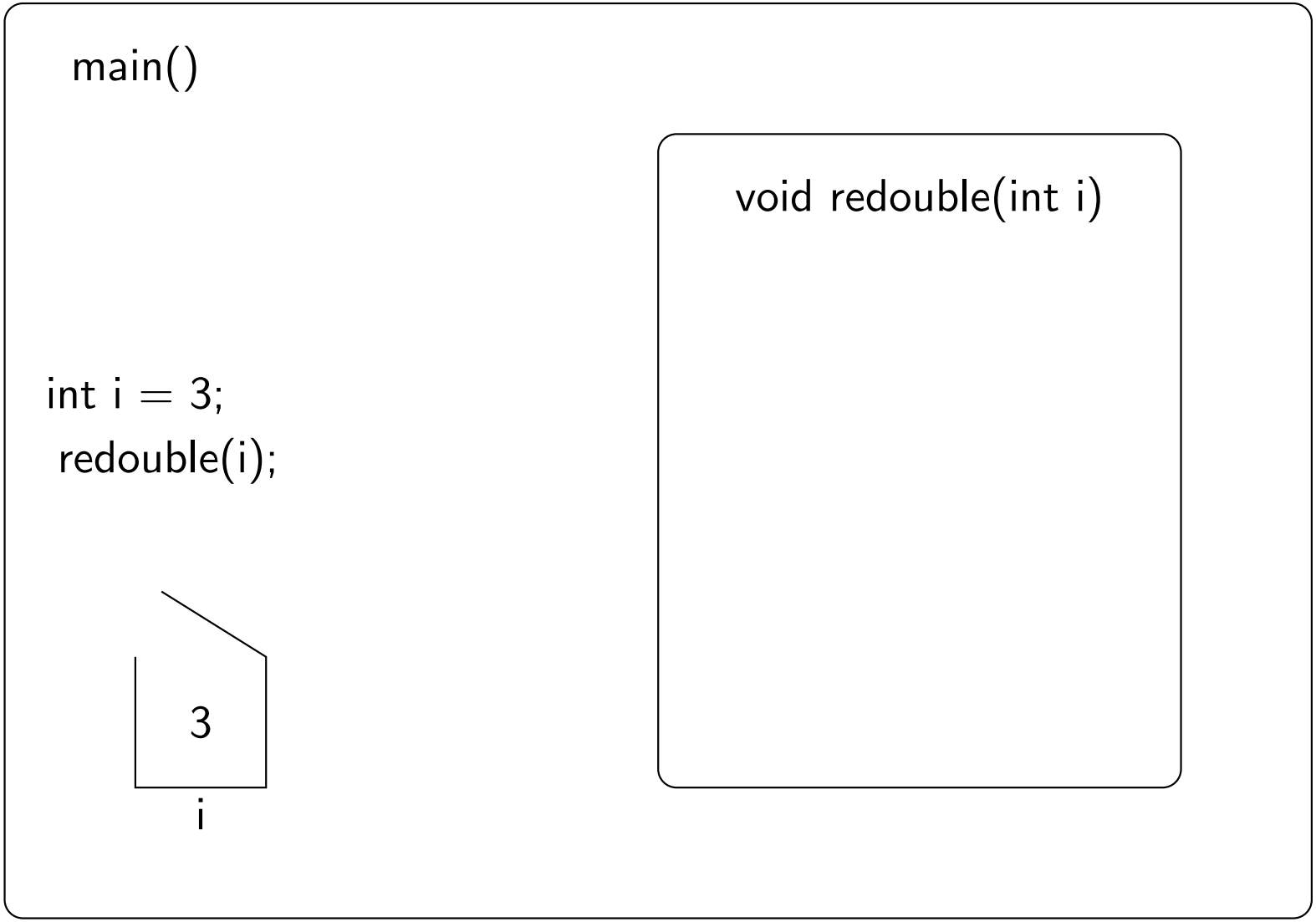
Primitive Datentypen

main()

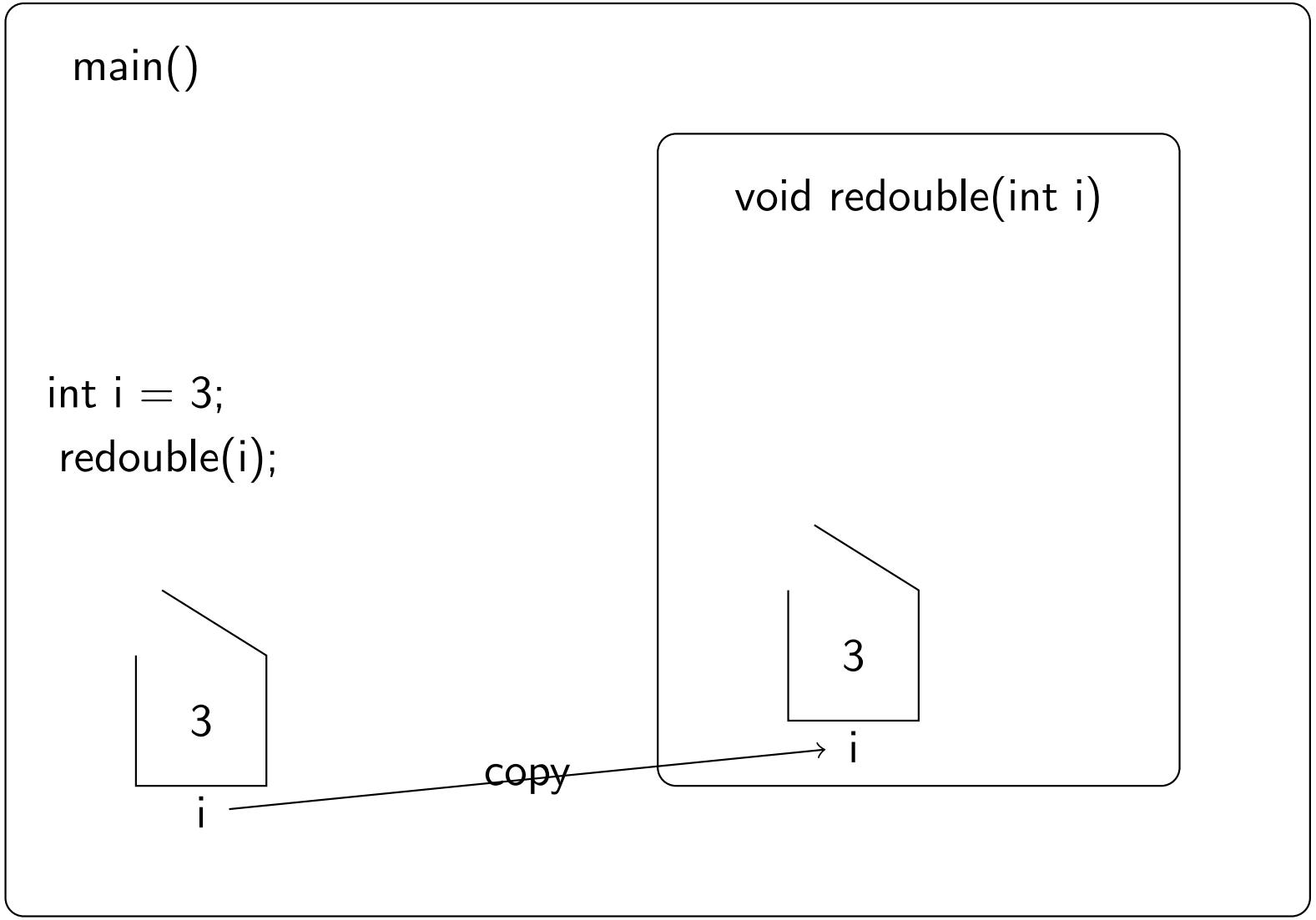
```
int i = 3;
redouble(i);
```



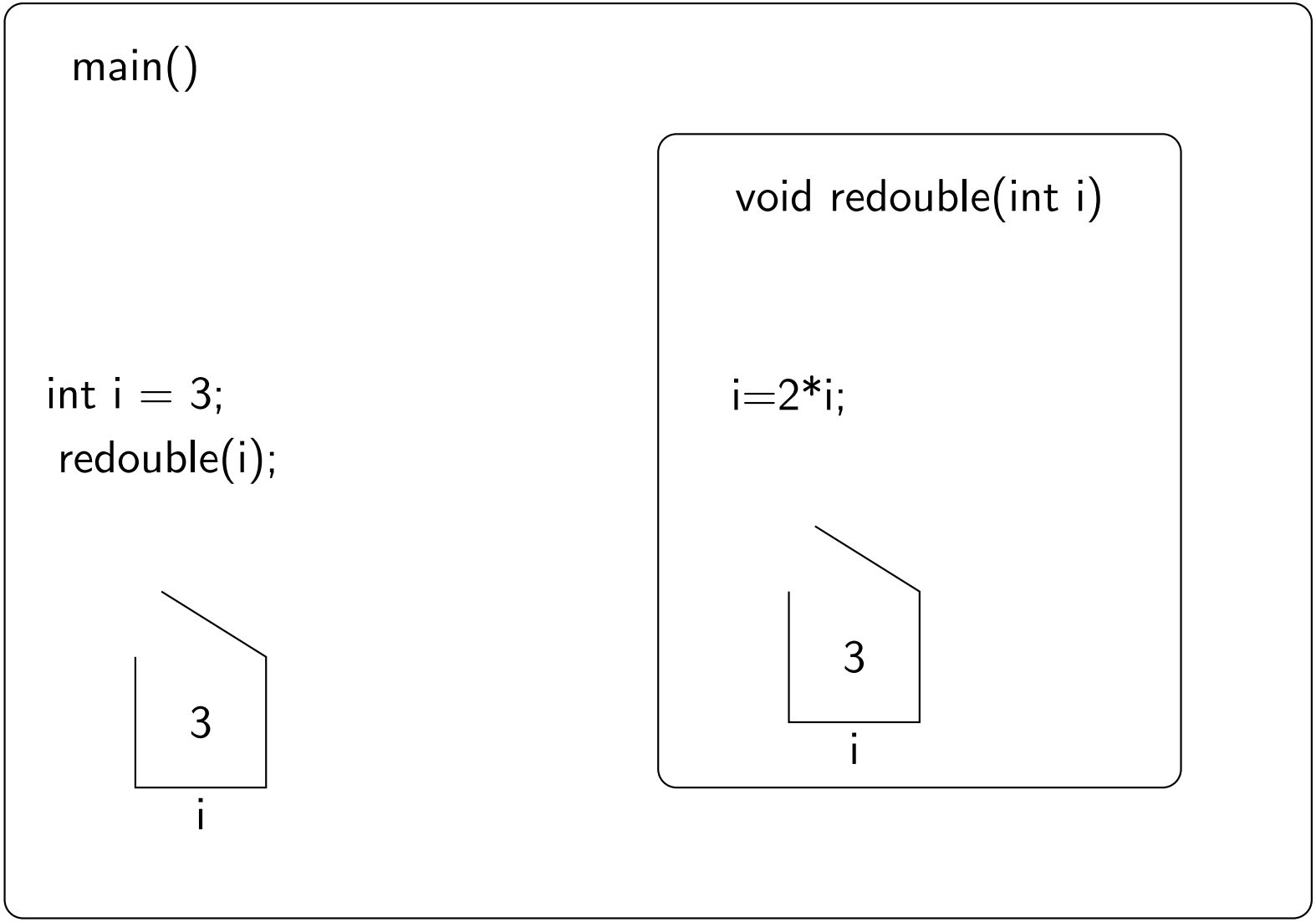
Primitive Datentypen



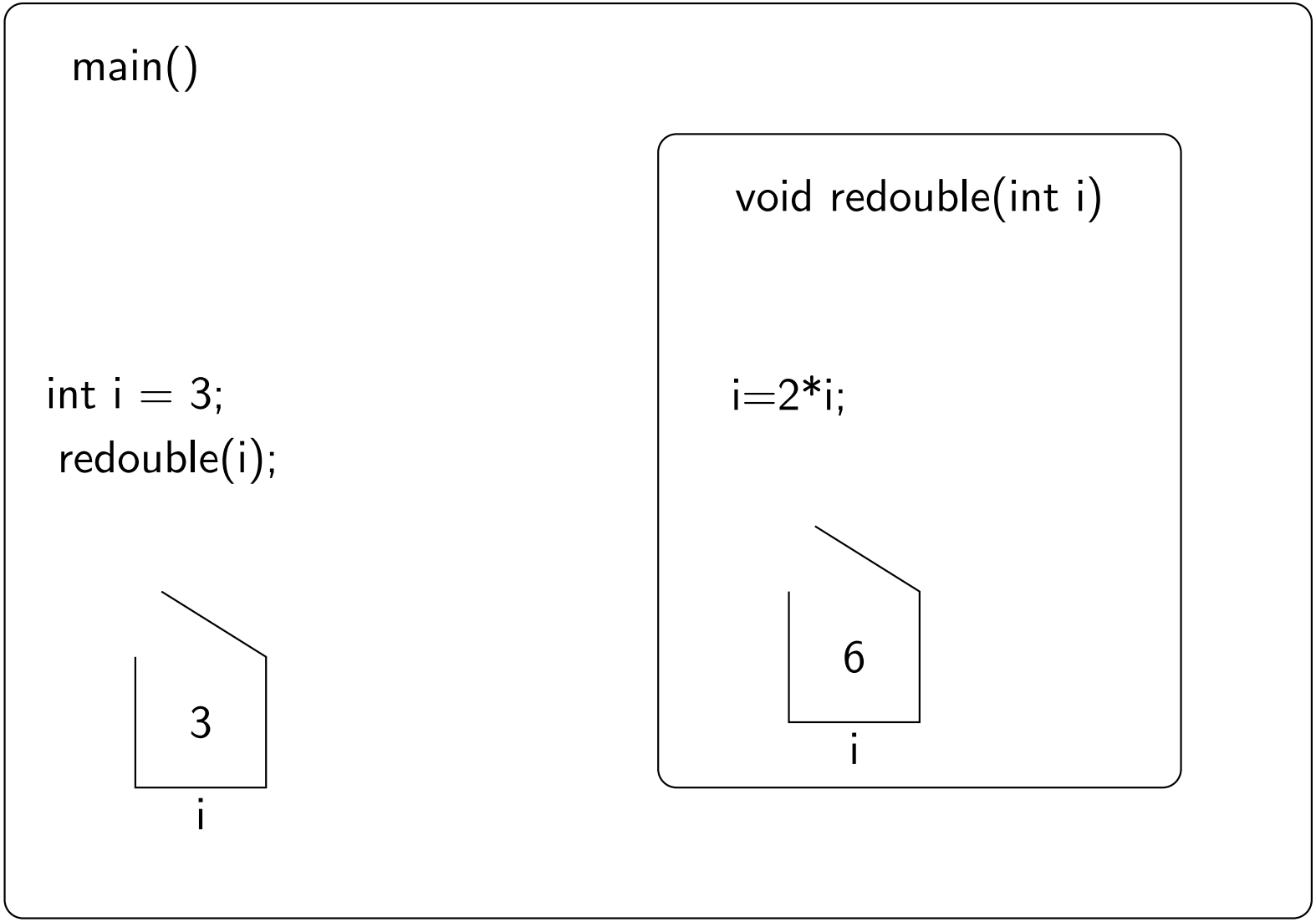
Primitive Datentypen



Primitive Datentypen



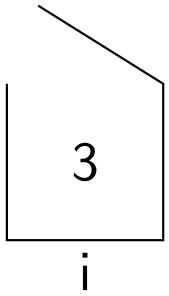
Primitive Datentypen



Primitive Datentypen

main()

```
int i = 3;
redouble(i);
```



Referenztypen (z.B. Arrays)

main()

Referenztypen (z.B. Arrays)

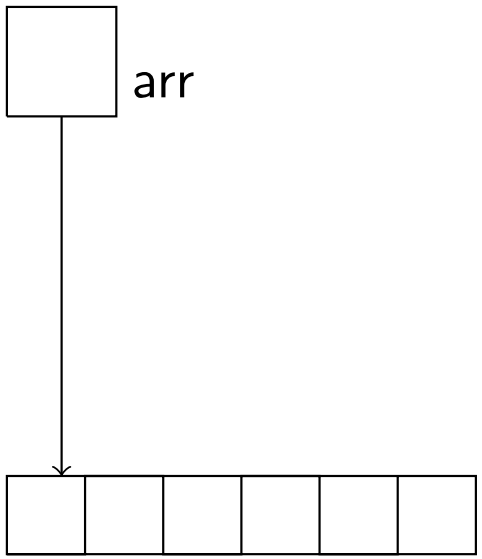
main()

```
int[] arr = new int[6];
```

Referenztypen (z.B. Arrays)

main()

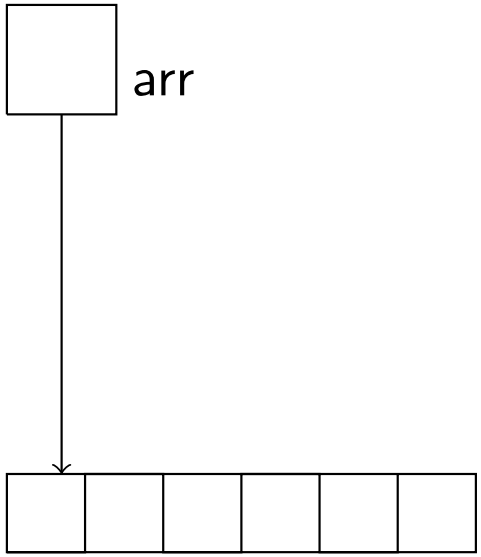
int[] arr = new int[6];



Referenztypen (z.B. Arrays)

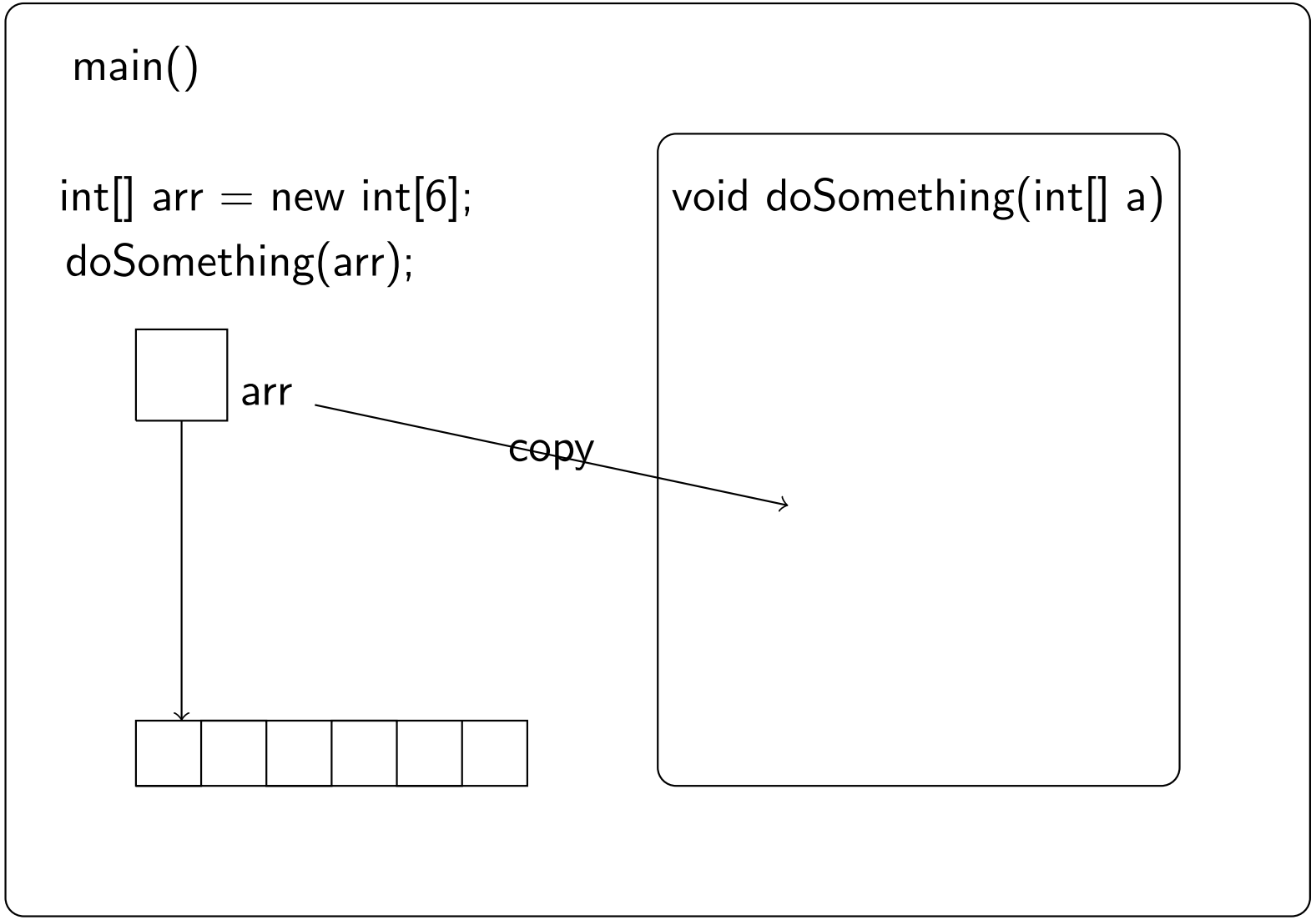
main()

```
int[] arr = new int[6];
doSomething(arr);
```

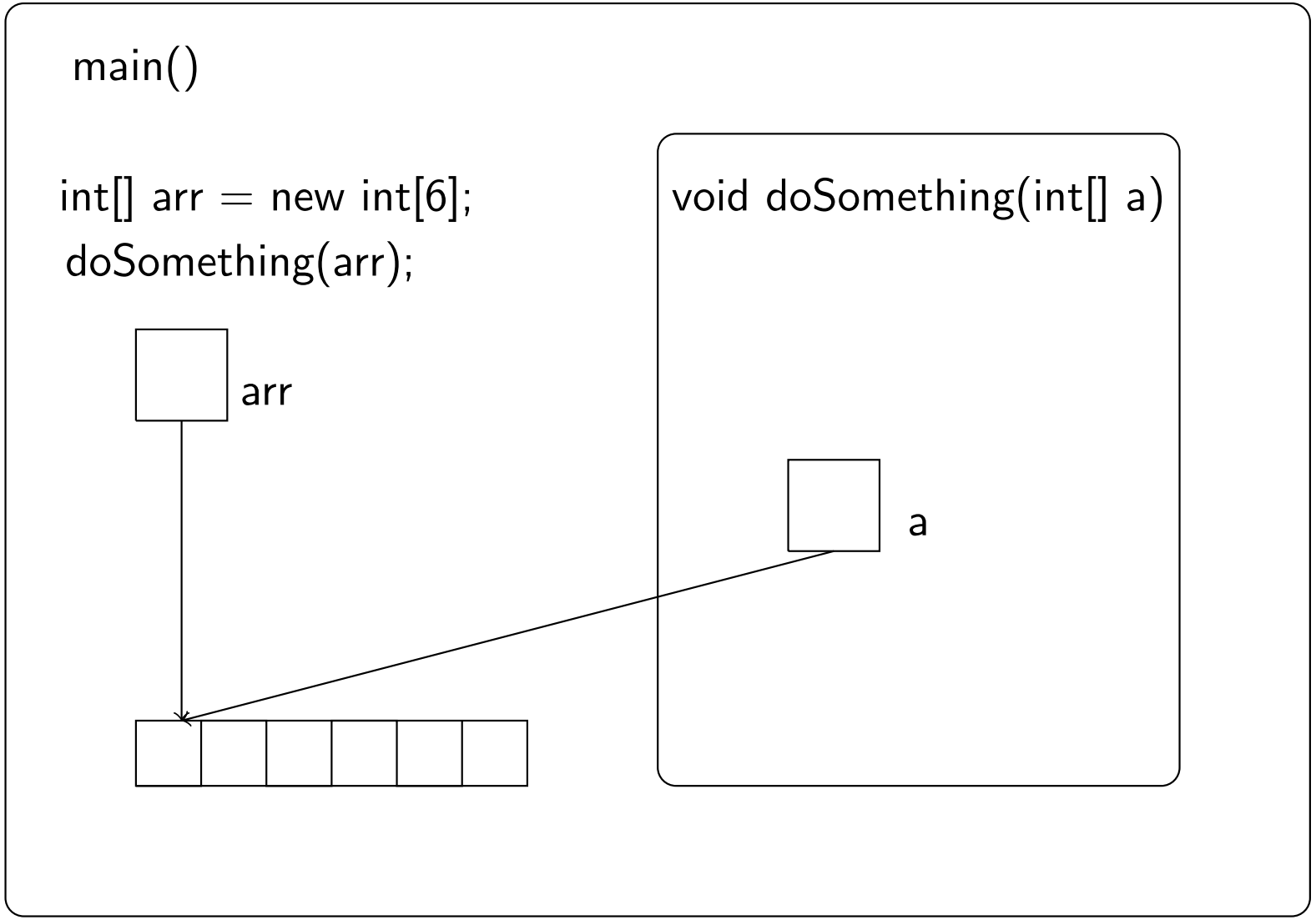



```
void doSomething(int[] a)
```

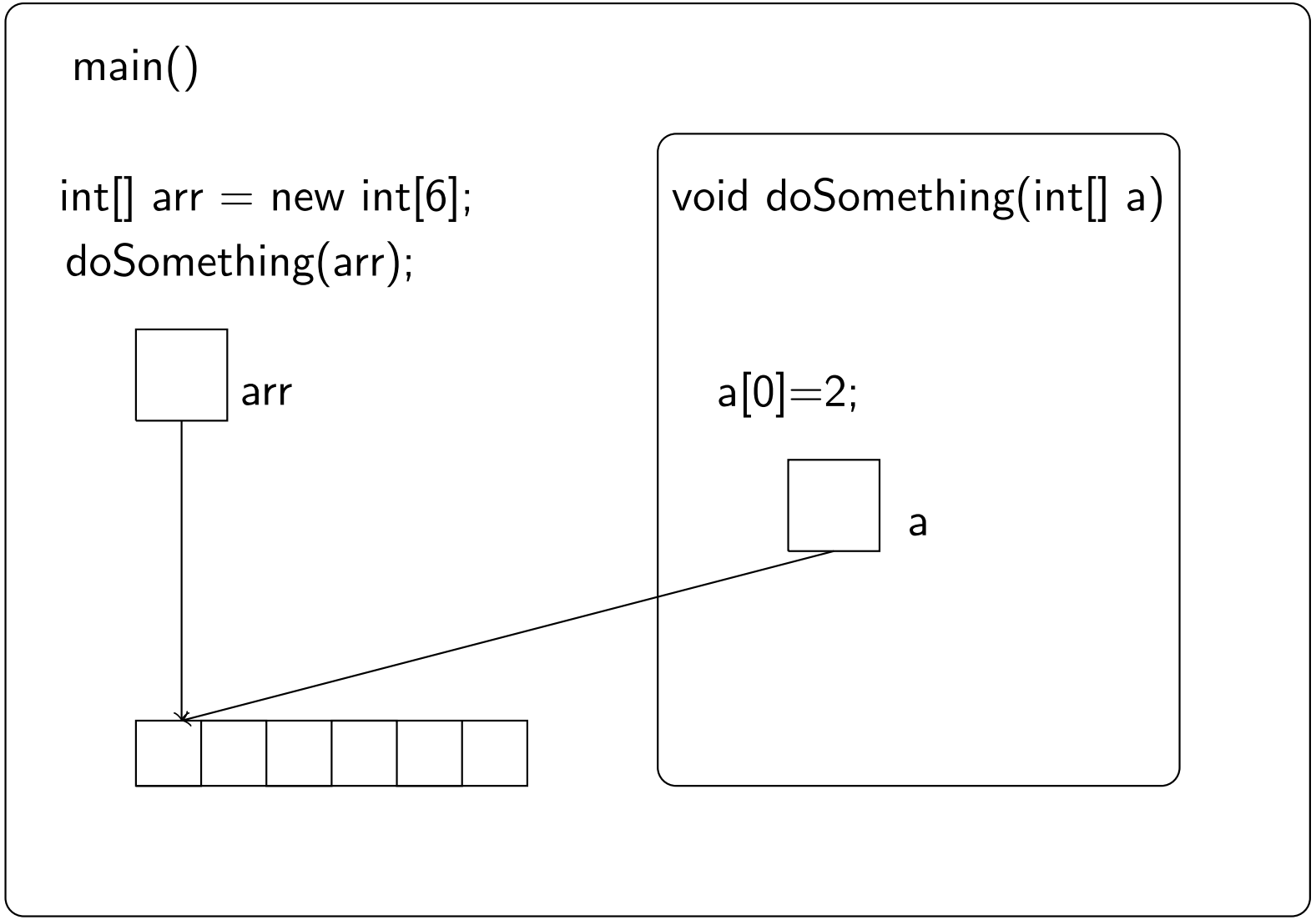
Referenztypen (z.B. Arrays)



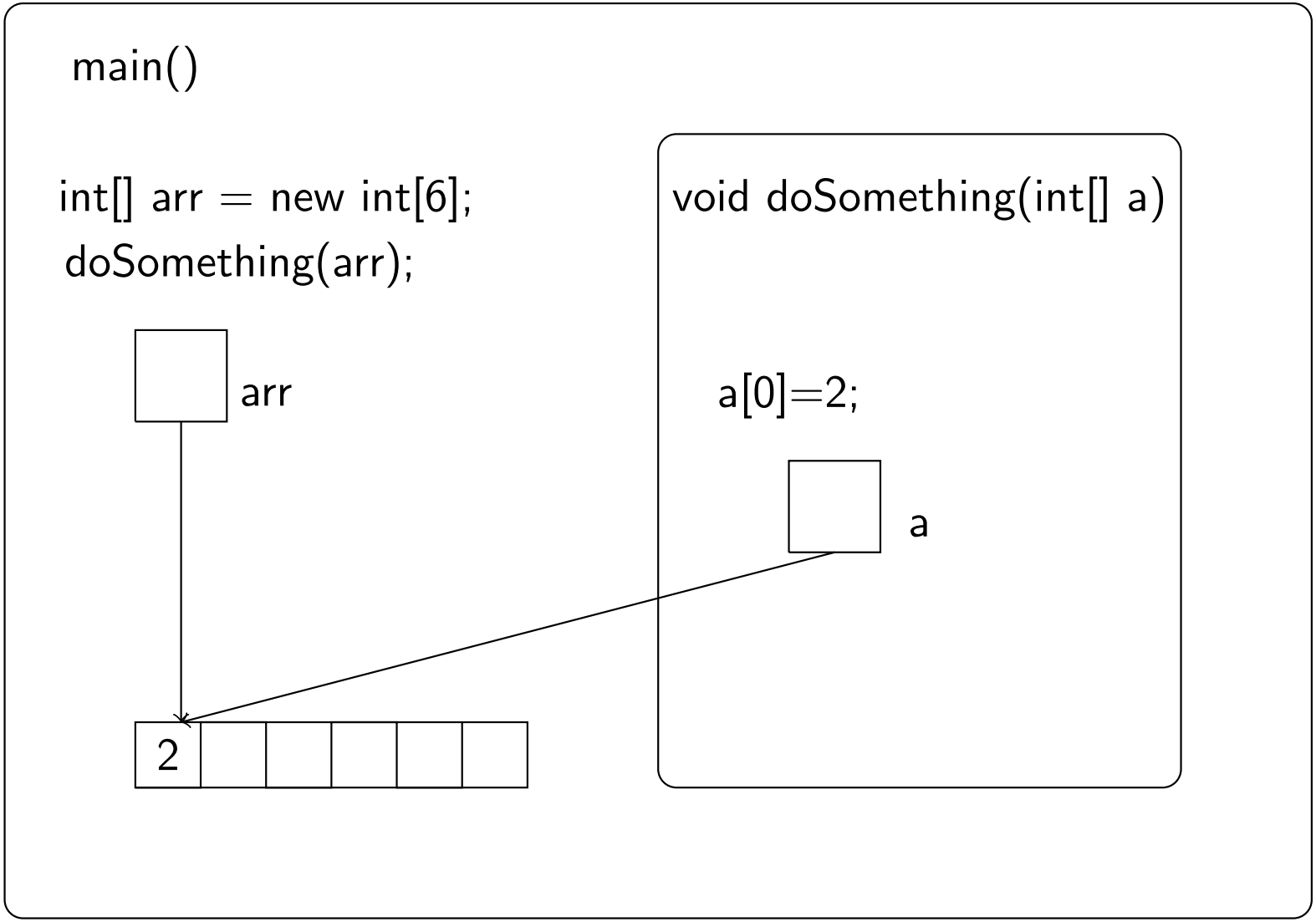
Referenztypen (z.B. Arrays)



Referenztypen (z.B. Arrays)



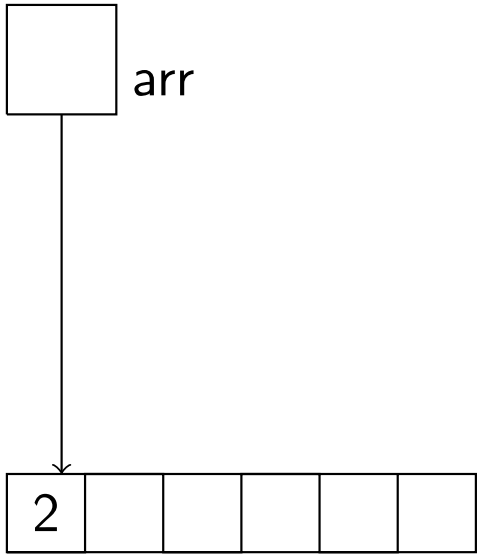
Referenztypen (z.B. Arrays)



Referenztypen (z.B. Arrays)

main()

```
int[] arr = new int[6];
doSomething(arr);
```



Referenztypen

Listing 30: src/EditArray.java

```
1  public class EditArray{
2    public static void main (String args[]) {
3      int[] arr = {2,3,4};
4      System.out.println("Vorher: " + arr[0]);
5      redouble(arr);
6      System.out.println("Nachher: " + arr[0]);
7    }
8    public static void redouble(int[] arr) {
9      arr[0] = 2 * arr[0];
10   }
```

Referenztypen

Ausgabe:
Vorher: 2
Nachher: 4

Inhaltsverzeichnis

Rekursion

Rekursion

Eine Methode kann sich selbst aufrufen. Das nennt man Rekursion. Dies ist besonders dann nützlich, wenn ein bestimmter Schritt immer wieder ausgeführt wird.

Rekursion

Beispiel: Fakultätsfunktion $n!$:

- $0! = 1$
- $n! = (n-1)! \cdot n$

Rekursion

Beispiel: Fakultätsfunktion $n!$:

- $0! = 1$
- $n! = (n-1)! \cdot n$

$$\begin{aligned} 3! &= 2! \cdot 3 = 1! \cdot 2 \cdot 3 \\ &= 0! \cdot 1 \cdot 2 \cdot 3 = 1 \cdot 1 \cdot 2 \cdot 3 = 6 \end{aligned}$$

Fakultät

Listing 31: src/Factorial.java

```
1  public class Factorial{
2    public static void main (String args[]) {
3      int f = factorial(3);
4    }
5    public static int factorial(int i) {
6      if(i <= 0) {
7        return 1;
8      }
9      else {
10       return i*factorial(i-1);
11     }
12   }
13 }
```

Fakultät

- $factorial(3)$

Fakultät

- $factorial(3)$
- $\rightsquigarrow 3 * factorial(2)$

Fakultät

- $factorial(3)$
- $\rightsquigarrow 3 * factorial(2)$
- $\rightsquigarrow 3 * (2 * factorial(1))$

Fakultät

- $factorial(3)$
- $\rightsquigarrow 3 * factorial(2)$
- $\rightsquigarrow 3 * (2 * factorial(1))$
- $\rightsquigarrow 3 * (2 * (1 * factorial(0)))$

Fakultät

- $factorial(3)$
- $\rightsquigarrow 3 * factorial(2)$
- $\rightsquigarrow 3 * (2 * factorial(1))$
- $\rightsquigarrow 3 * (2 * (1 * factorial(0)))$
- $\rightsquigarrow 3 * (2 * (1 * 1))$

Fakultät

- $factorial(3)$
- $\rightsquigarrow 3 * factorial(2)$
- $\rightsquigarrow 3 * (2 * factorial(1))$
- $\rightsquigarrow 3 * (2 * (1 * factorial(0)))$
- $\rightsquigarrow 3 * (2 * (1 * 1))$
- $\rightsquigarrow 3 * (2 * 1)$

Fakultät

- $factorial(3)$
- $\rightsquigarrow 3 * factorial(2)$
- $\rightsquigarrow 3 * (2 * factorial(1))$
- $\rightsquigarrow 3 * (2 * (1 * factorial(0)))$
- $\rightsquigarrow 3 * (2 * (1 * 1))$
- $\rightsquigarrow 3 * (2 * 1)$
- $\rightsquigarrow 3 * 2$

Fakultät

- $factorial(3)$
- $\rightsquigarrow 3 * factorial(2)$
- $\rightsquigarrow 3 * (2 * factorial(1))$
- $\rightsquigarrow 3 * (2 * (1 * factorial(0)))$
- $\rightsquigarrow 3 * (2 * (1 * 1))$
- $\rightsquigarrow 3 * (2 * 1)$
- $\rightsquigarrow 3 * 2$
- $\rightsquigarrow 6$

Rekursion

Vorteil der Rekursion:

- elegante, übersichtliche Programmierung bei kleinen Funktionen

Rekursion

Vorteil der Rekursion:

- elegante, übersichtliche Programmierung bei kleinen Funktionen

Nachteil der Rekursion:

- Quelltext chaotisch bei vielen Rekursionsfälle (siehe Übungsaufgabe "Rekursion vs Iteration")

Rekursion

Vorteil der Rekursion:

- elegante, übersichtliche Programmierung bei kleinen Funktionen

Nachteil der Rekursion:

- Quelltext chaotisch bei vielen Rekursionsfälle (siehe Übungsaufgabe "Rekursion vs Iteration")
- Gefahr der Endlosschleife

Rekursion

Vorteil der Rekursion:

- elegante, übersichtliche Programmierung bei kleinen Funktionen

Nachteil der Rekursion:

- Quelltext chaotisch bei vielen Rekursionsfälle (siehe Übungsaufgabe "Rekursion vs Iteration")
- Gefahr der Endlosschleife
- unter Umständen sehr ineffizient(siehe Übungsaufgabe "Fibonacci")

Zusammenfassung

Ihr solltet jetzt wissen, ...

- was ein Scope ist und was er bewirkt
- worin sich eine lokale Variable von einer globalen unterscheidet
- wie Parameter bei Methoden übergeben werden
- was Rekursion ist

Noch Fragen?