Freie Universität Berlin

Programmieren

Barry Linnert

Sommersemester 2020

Gliederung der heutigen Vorlesung

- Kurze Wiederholung
- Arbeiten mit Objekten – Konstruktoren
- Modularisierung – Packages
- Sichtbarkeit von Variablen und Methoden in Java
- Referenzvariablen und Parameterübergabe
- Zusammenfassung

Einführung in die Objektorientierte Programmierung (Teil 3)

WIEDERHOLUNG

Freie Universität Berlin

Ebenen der Objektorientierten Denkweise

Höhere Ebene

“Arbeiten” mit Objekten

Niedrigere Ebene

Definition der benötigten Objekte (Klassen) mit imperativen Elementen

Imperative Programmierung in Methoden

Operatoren

- Arithmetische Operatoren
- Vergleichsoperatoren
- Zuweisungsoperatoren
- Logische Operatoren

Anweisungen

- Einfache Anweisung
- Alternativanweisung
- Switch-Anweisung
- Schleifen
- For-Anweisung

Einführung in die Objektorientierte Programmierung (Teil 3)

ARBEITEN MIT OBJEKTEN

Beispiel: Die Haus-Klasse

Eigenschaften:
Etagen
Wohnfläche
Nutzfläche
Adresse

Operation:
sanieren
renovieren
verkaufen

Klassendefinition

Zustand
Zwei Etagen
100 m^2 Wohnfläche
200 m^2 Nutzfläche
Takostraße 20

Operationen
kann saniert werden
kann renoviert werden
kann verkauft werden

Ein konkretes Haus Objekt

Objekterzeugung

- Objekte werden durch den Aufruf von Konstruktoren erzeugt. Ein Konstruktor wird mit Hilfe der **new**-Operatoren aufgerufen. Zum Beispiel:

```
Rectangle r1 = new Rectangle();
```

- Eine Klassendefinition kann mehrere Konstruktoren haben mit verschiedenen Initialisierungen der Objekteigenschaften.
- Wenn in einer Klasse keine Konstruktoren definiert worden sind, werden die Eigenschaften von Objekten mit Defaultwerten initialisiert.

Möglichkeiten der Objekterzeugung

```
...

Kreis first_circle = new Kreis ( 0.0, 0.0, 1.0 );

Kreis second_circle = new Kreis ( first_circle );

Kreis four_circle = new Kreis ( 5.0 );

Kreis third_circle = new Kreis ( );

...
```

Konstruktoren

- Ein guter OOP-Stil bedeutet, geeignete *Konstruktoren* zu definieren, die Objekte initialisieren und evtl. initiale Berechnungen durchführen.

- Syntaktisch sind Konstruktoren Methoden mit einigen speziellen Eigenschaften:
  - ihr Name muss mit dem Namen der Klasse übereinstimmen,
  - sie dürfen keinen Ergebnistyp, auch nicht void, haben,
  - sie werden mit dem new-Operator aufgerufen werden und erzeugen ein Objekt der Klasse.

Freie Universität Berlin

Implizite vs. explizite Kontruktoren

- Ist kein Konstruktor definiert, wird ein impliziter Konstruktor ohne Argumente angenommen.

```
public class Kreis {
     double x, y, radio;
     ...
}
```

=

```
public class Kreis {
     double x, y, radio;
     public Kreis() {
     }
     ...
}
```

- Sobald ein expliziter Konstruktor definiert ist, fällt der implizite Konstruktor weg!

Freie Universität Berlin

```
public class Rectangle {

// Attribute
    private int x, y;
    private int width, height;

// Konstruktur
    public Rectangle() {
        x = 0;
        y = 0;
        width = 10;
        height = 10;
    }
// Methoden
   ...
} // end of class Rectangle
```

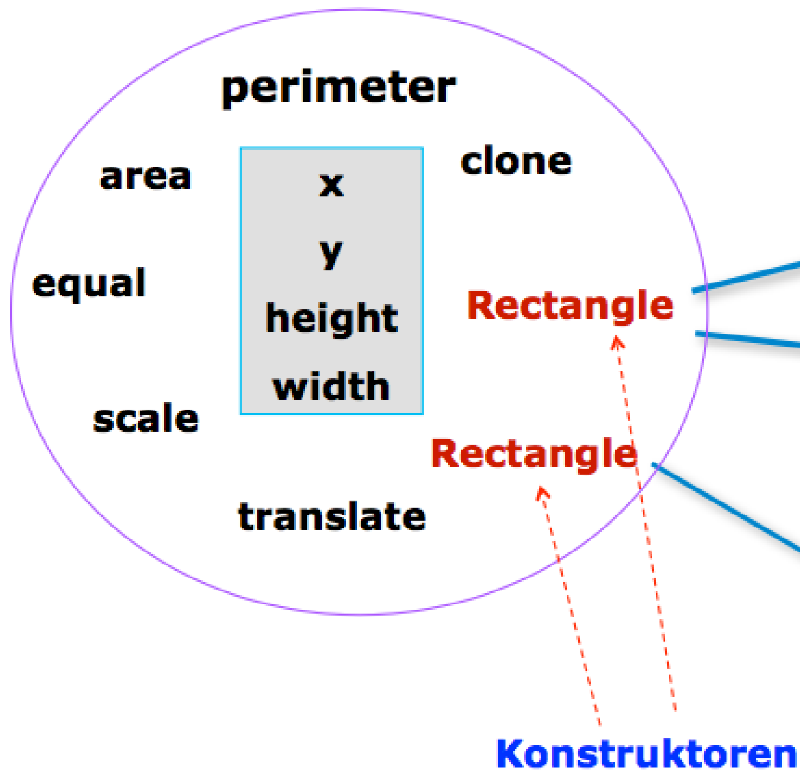
Definition eines parameterlosen Konstruktors

Freie Universität Berlin

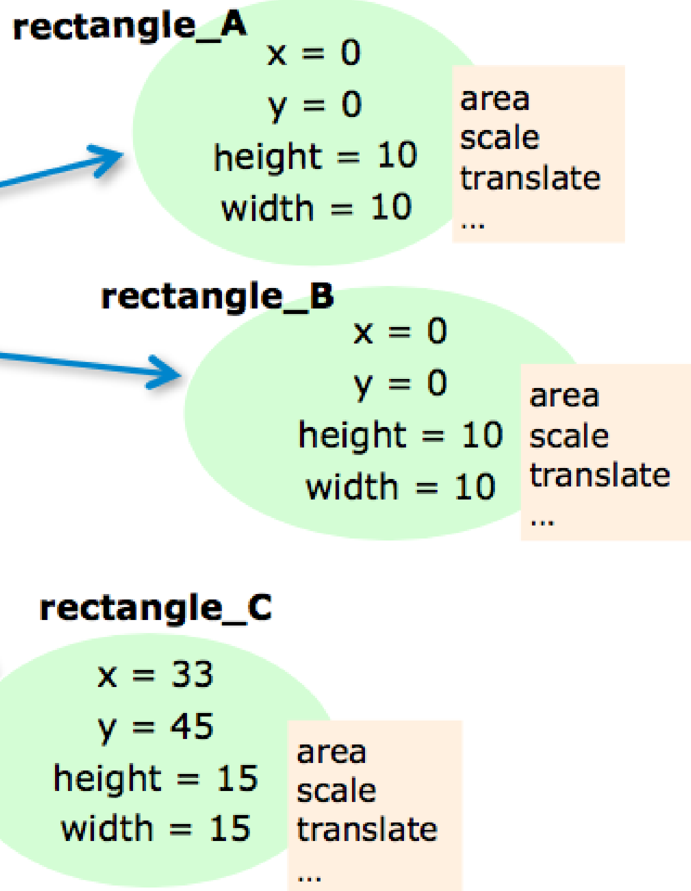
Erzeuge Rechteck-Objekte

```
public class TestRectangle{

// main-Methode

public static void main( String[] args ) {

  Rectangle r1 = new Rectangle();
  Rectangle r2 = new Rectangle();

  int u = r1.area();
  int f = r2.perimeter(); }

} // end of class TestRectangle
```

Rectangle-Klasse



Rectangle-Objekte



Freie Universität Berlin

```java
public class Rectangle {

// Attribute
    private int x, y;
    private int width, height;

// Konstruktur
    public Rectangle() {
        x = 0; y = 0; width = 10; height = 10;}
    public Rectangle(int a, int b) {
        x = a; y = b; width = 10; height = 10;}
    public Rectangle(int a, int b, int w, int h) {
        x = a; y = b; width = w; height = h;}

// Methoden
   ...
} // end of class Rectangle
```

Die Klasse verfügt über drei Konstruktoren.
Der erste ist parameterlos.
Der zweite erlaubt die Koordinate
zu ändern und
der dritte alle Eigenschaften
des Rechtecks.

Freie Universität Berlin

Erzeuge Rechteck-Objekte

```
public class TestRectangle{

// main-Methode

public static void main( String[] args ) {

Rectangle r1 = new Rectangle();
Rectangle r2 = new Rectangle(10,10);

int u = r1.area();
int f = r2.perimeter(); }

} // end of class TestRectangle
```

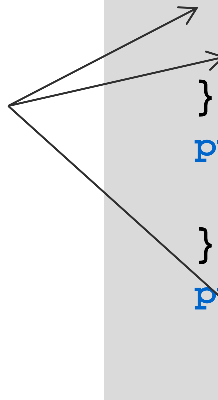
Freie Universität Berlin

"Goldene Regel" bei mehreren Konstruktoren

- Schreibe *genau einen* Konstruktor, der alle Initialisierungen vornimmt und rufe ihn aus den anderen mit geeigneten Parametern auf. Dies vermindert die Zahl potentieller Fehler.

```
public class Circle {
   double x, y, r;

   public Circle(double x, double y, double r){
      this.x = x; this.y = y; this.r = r; }

   public Circle(double r) {this(0.0, 0.0, r);}
   public Circle(Circle c) {this(c.x, c.y, c.r);}
   public Circle() {this(1.0);}}
```

Referenz auf das aktuelle Objekt selbst



```
public class Kreis {

    public final double PI = 3.141598;
    public double x, y;
    private double radio = 10;

    public Kreis( double x, double y ){
        this.x = x;
        this.y = y;
    }
    public double getRadio() {
        return this.radio;
    }
    public void setRadio( double r ) {
        if ( r>0 )
            this.radio = r;
        else
            System.err.println( "Fehler:...." );
   }
...
```

this

- Das Schlüsselwort **this** bezeichnet immer eine Referenz auf das aktuelle Objekt selbst.

- **this** kann in Methoden und in Konstruktoren verwendet werden, um durch Argumentnamen “verschattete” Variablennamen zu erreichen.

Weitere vordefinierte Operationen, die mit Referenz-Variablen erlaubt sind

Freie Universität Berlin

- (Typ)-Operator

```
Object objekt = null;
Button button = (Button) objekt;
```

- Der (.)-Operator

  Mit dem (.)-Operator hat man Zugriff auf die Eigenschaften und Methoden eines Objekts.

```
Punkt p1 = new Punkt(10,35);
int x_koord = p1.x ;
```

Freie Universität Berlin

null

- Das Schlüsselwort **null** bezeichnet immer ungültige, d.h. nicht initialisierte Referenzen.

- **null** kann überall da verwendet werden, wo eine Referenz erwartet wird. Zugriff auf eine Referenz, die gleich null ist, erzeugt einen Laufzeitfehler.

```
Rechteck r1;
Rechteck r2 = null;
...
r1.gleich( r2 );
```

(NullPointerException)

Verursacht einen Laufzeitfehler

Java Anwendung

Rechtecke.java

```
class Rechteck {
  // Attribute
    . . . .
  // Konstruktoren
    . . . .
  // Methoden
    . . . .
    . . . .
   . . . .
}
```

Kreis.java

```
class Kreis {
  // Attribute
    . . . .
  // Konstruktoren
    . . . .
  // Methoden
    . . . .
    . . . .
   . . . .
   . . . .
}
```

Blatt.java

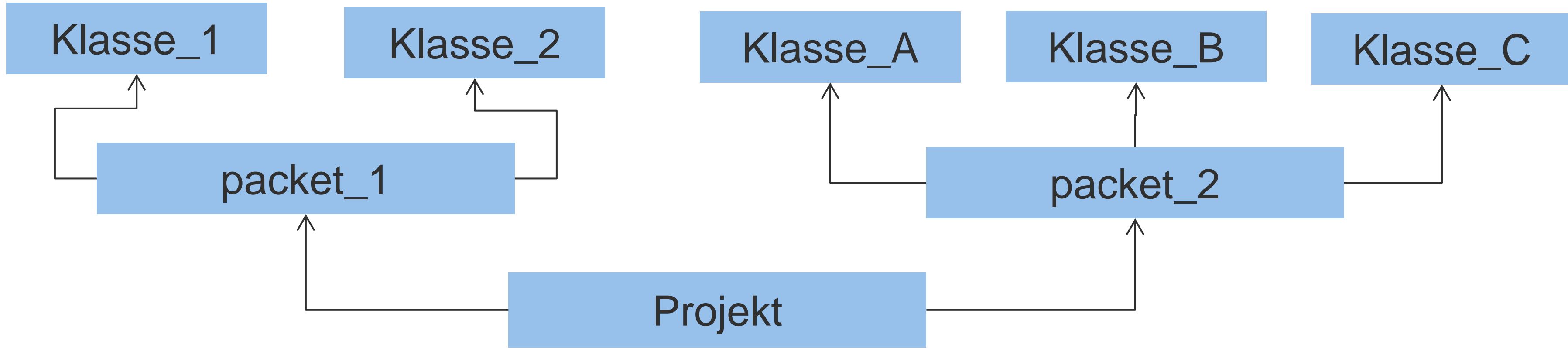
```
class Blatt {
  // Attribute
    . . . .
  // Konstruktoren
    . . . .
  // Methoden
    . . . .
    . . . .
 . . main ( . . . . ) {. .}   <- start
   . . . .

}
```

Freie Universität Berlin

Einführung in die Objektorientierte Programmierung (Teil 3)

MODULARISIERUNG

Ebenen Modualisierung




```
package packet_1 ;
public class Klasse_1 {
.....
}
```

Freie Universität Berlin

Packages

- Java bietet die Möglichkeit, miteinander in Beziehung stehende Klassen zu Paketen (packages) zusammenzufassen. Die Zugehörigkeit zu einem Paket wird über die **package**-Direktive deklariert.

- Einzelne oder alle Klassen eines anderen Pakets werden mit der **import**-Direktive "sichtbar" gemacht.

Packages

```
package beispiele;

public class Person { ... }
```

```
import beispiele.*; ...

Person p1;

java.lang.String str = "";

String str2 = "";

...
```

import java.lang.*; **// Standard**

Packages und Klassennamen

- Wird kein Package bestimmt, so gehört die Klasse automatisch ins globale, unbenannte Package.
- Der vollständig qualifizierte Name einer Klasse wird aus dem Klassennamen und allen umschließenden Package-Namen gebildet, z.B.

```
java.awt.Button button;
```

wenn eine entsprechende **import**-Anweisung vorhanden ist.

schreibe ich nur →

```
import java.awt.*; ...
Button button;
```

Einführung in die Objektorientierte Programmierung (Teil 3)

SICHTBARKEIT VON VARIABLEN UND METHODEN IN JAVA

Freie Universität Berlin

Motivation anhand der Methodennutzung

- Jede Methode in Java ist typisiert. Der Typ einer Methode wird zum Zeitpunkt der Definition festgelegt und bestimmt den Typ des Rückgabewerts.

- Dieser kann von einem beliebigen primitiven Typ, einem Objekttyp oder vom Typ void sein. Die Methoden vom Typ void haben gar keinen Rückgabewert und dürfen nicht in Ausdrücken verwendet werden.

- Hat eine Methode einen Rückgabewert (ist also nicht vom Typ void), so muss sie mit Hilfe der return-Anweisung einen Wert an den Aufrufer zurückgeben.

Instanzmethoden

- Instanzmethoden definieren das Verhalten von Objekten. Sie werden innerhalb einer Klassendefinition angelegt und haben Zugriff auf alle Variablen des Objekts.
- Sie haben immer den impliziten Parameter this

```
class Fahrzeug {
   ...
   String getName() {
      return name;
   }
   void setName(String name) {
      this.name = name;
   }
}
```

Klassenmethoden

- Analog zu Klassenvariablen gibt es Klassenmethoden
- Klassenmethoden werden mit static markiert und werden über den Klassennamen aufgerufen:

```
class Fahrzeug {
   static void setzeAnzahl(int n){
       Fahrzeug.anzahl = n;
}}
Fahrzeug.setzeAnzahl(29);
```

- Klassenmethoden haben weder Zugriff auf Instanzvariablen, noch können direkt Methoden der eigenen Klasse aufgerufen werden, die nicht selbst Klassen-methoden sind.
- Innerhalb von Klassenmethoden gibt es kein this.

Wiederholung: Zugriffskontrolle auf Methoden

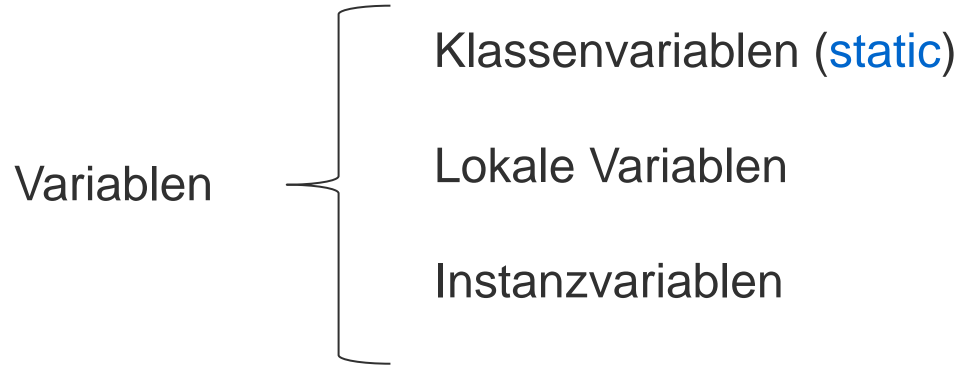
- Der Zugriff auf Methoden kann genauso wie bei Variablen durch ***Modifizierer*** gesteuert werden:

  - **public**: überall zugänglich.
  - **private**: nur innerhalb der eigenen Klasse zugänglich.
  - **protected**: in anderen Klassen desselben Packages und in Unterklassen zugänglich.

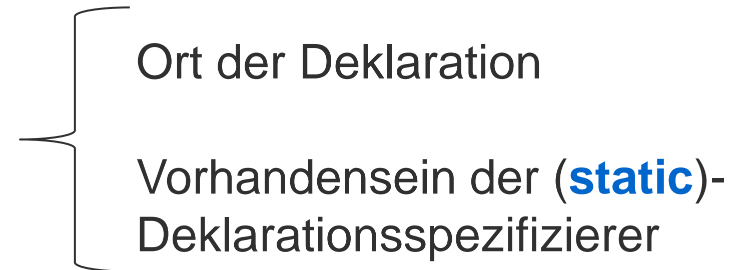
- Methoden ohne **Modifizierer** sind nur im selben ***Paket*** (package) zugänglich.

Freie Universität Berlin

Variablen in Java



Diese Klassifikation richtet sich nach folgenden zwei Aspekten:



Variablen in Java (2)

Lokale Variablen

- Hilfsvariable für Berechnungen
  Sie werden innerhalb von Methoden deklariert
- **Lebenszeit**: nur solange die Methode ausgeführt wird

Instanzvariablen

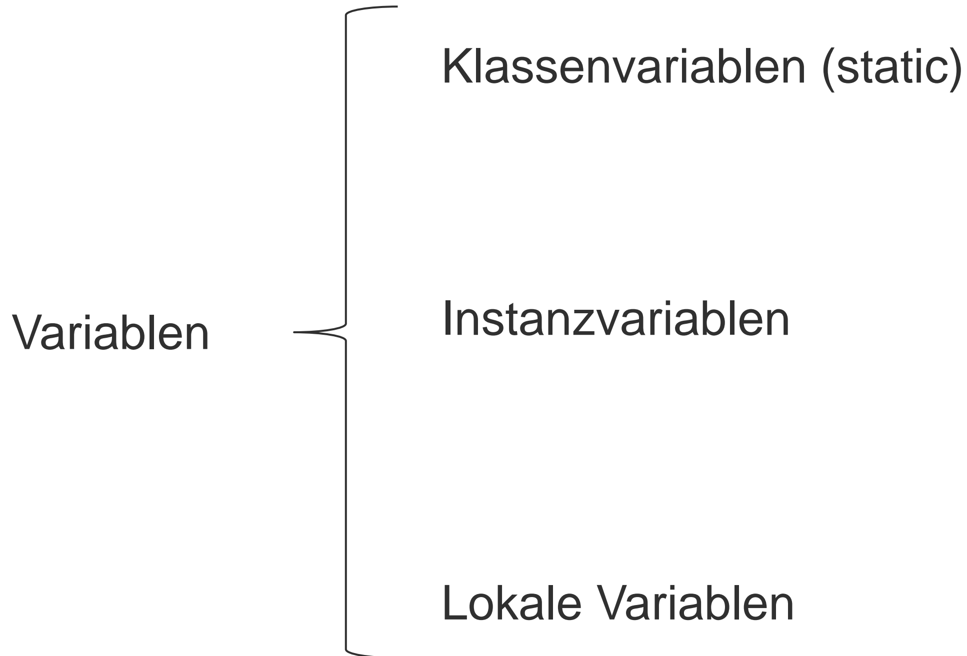
(Feldvariablen, Attribute)

- Variablen, in denen die Eigenschaften von Objekten gespeichert werden
- **Lebenszeit**: nur solange das Objekt existiert.

Klassenvariablen

- Variablen, die zu einer Klasse gehören
- **Lebenszeit**: solange das Programm ausgeführt wird.

Zugriff auf Variablen




```
float delta = Kreis.PI / 10;
```

Bezüglich des Klassennamens

```
Kreis k1 = new Kreis();
k1.radio = 5.0;
```

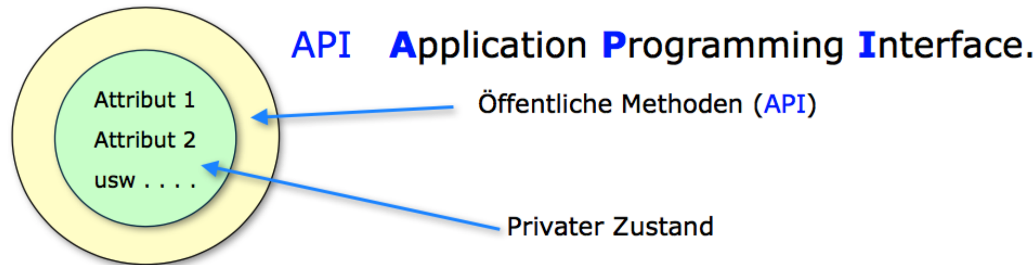
Nur in Bezug zu einem Objekt

```
int num;
num = 5;
```

Freie Universität Berlin

Was ist Kapselung?

- Kapselung ist die Einschränkung des Zugriffs auf die Instanzvariablen eines Objektes durch Objekte anderer Klassen. Man spricht von einer **Kapselung** des Objektzustands.



- **Kapselung** schützt so den Objektzustand vor “unsachgemäßer” Änderung und unterstützt **Datenabstraktion**.
- Kapselung erfolgt durch die Zugriffsmodifizierer.

Freie Universität Berlin

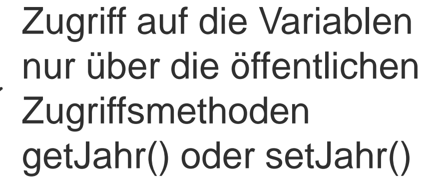
Vorteile der Kaselung

- Modularität:
  - Der Quellcode einer Klasse kann unabhängig vom Quellcode anderer Klassen gepflegt werden
  - Eine Klasse läßt sich leichter wiederverwenden

- „Information hiding“:
  - Die Implementierung kann sich ändern, ohne dass der Benutzer es merkt
  - Garantie von Datenkonsistenz
  - „Schlankere“ Klassen

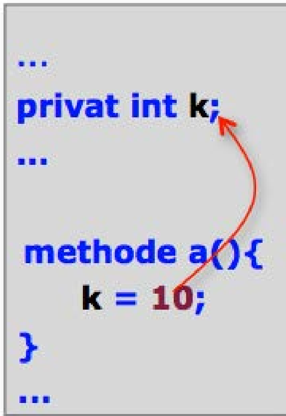
Beispiel für Datenkapslung

```java
class Datum{

    private int jahr = 2012, monat = 7, tag = 15;

    public int getJahr () { return jahr ;}
    public int getMonat(){ return monat;}
    public int getTag () { return tag ;}

    public void setJahr(int j){ jahr = j;}
    public void setMonat(int m){ monat = m;}
    public void setTag(int t){ tag = t;}
}
```

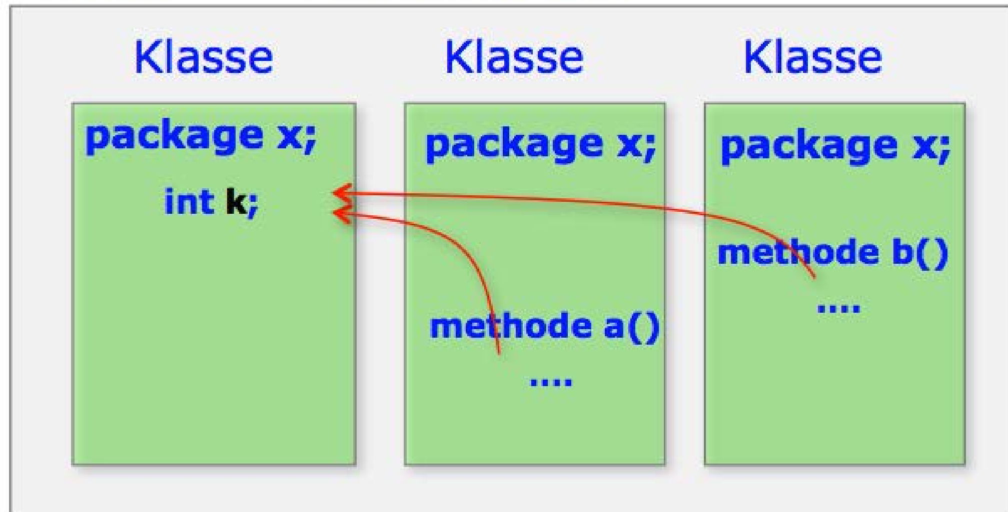


Freie Universität Berlin

Zugriffsangabe oder Sichtbarkeit von Variablen

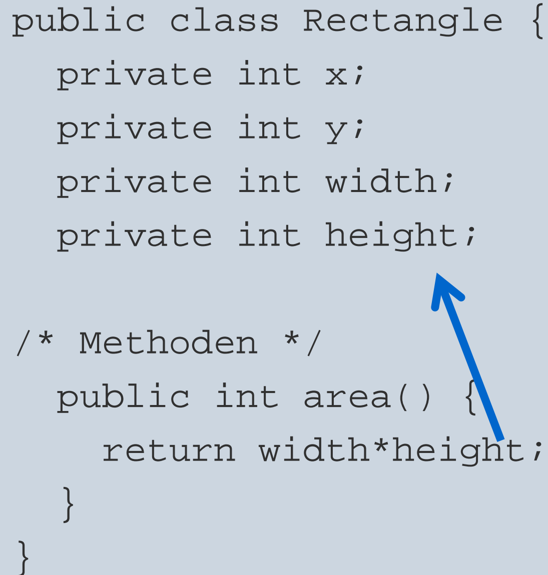


Nur das Objekt selbst kann den Inhalt einer privaten Variablen mittels seiner Methoden modifizieren.

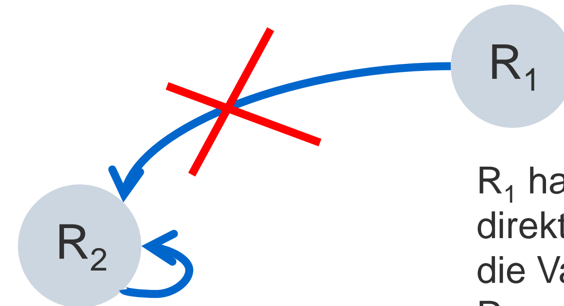


Alle Objekte innerhalb eines Verzeichnisses haben direkten Zugriff auf eine package-Variable.

Sichtbarkeit von private Variablen



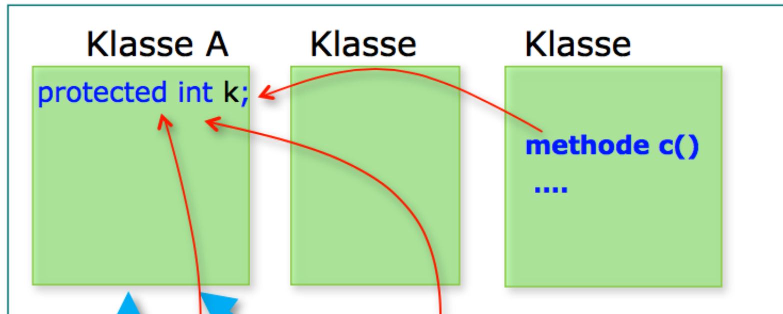
Zugriff nur innerhalb des jeweiligen Rechteck-Objektes

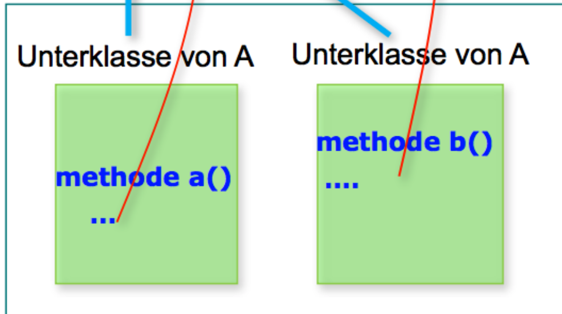


R_1 hat keinen direkten Zugriff auf die Variablen von R_2

R_2 hat Zugriff auf seine eigenen Variablen.

Zugriffsangabe oder Sichtbarkeit von Variablen





protected-Variablen

Zugriff innerhalb der Klassen-Hierarchie

Freie Universität Berlin

Sichtbarkeit von Java Variablen

Zugriffsangabe oder Sichtbarkeit	Klasse	Unterklasse	Paket	Welt
private	X			
kein Modifizierer (package)	X		X	
protected	X	X	X	
public	X	X	X	X

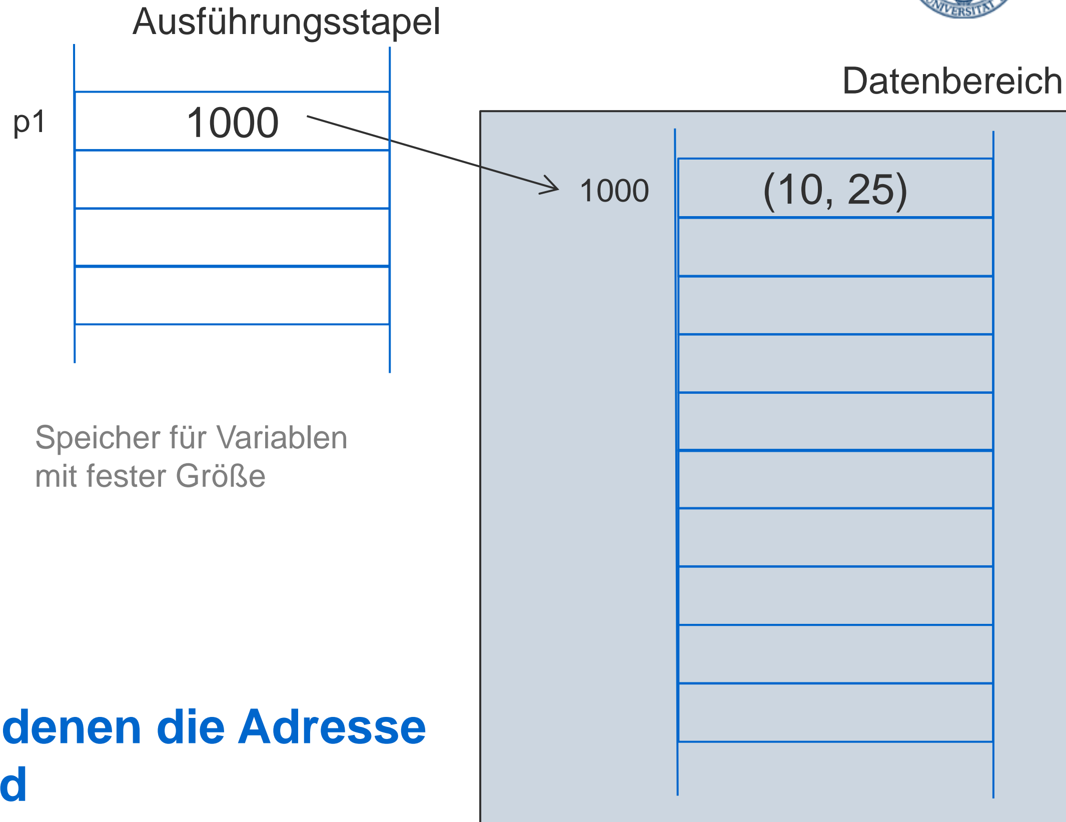
Freie Universität Berlin

Einführung in die Objektorientierte Programmierung (Teil 3)

REFERENZVARIABLEN UND PARAMETERÜBERGABE

Was ist eine Referenz?

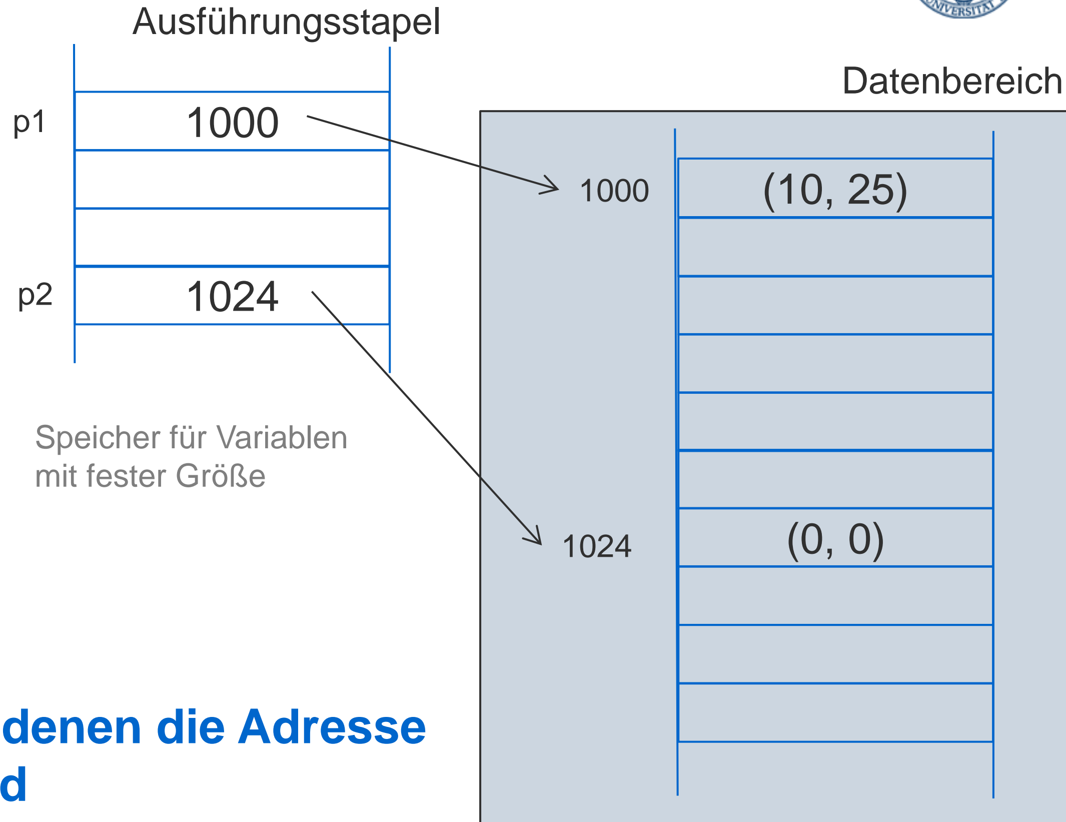
```
Point p1;
Point p2;
p1 = new Point (10, 25);
p2 = new Point (0, 0);
...
p1 = p2;
```



Referenzen sind Variablen, in denen die Adresse eines Objekts gespeichert wird

Was ist eine Referenz?

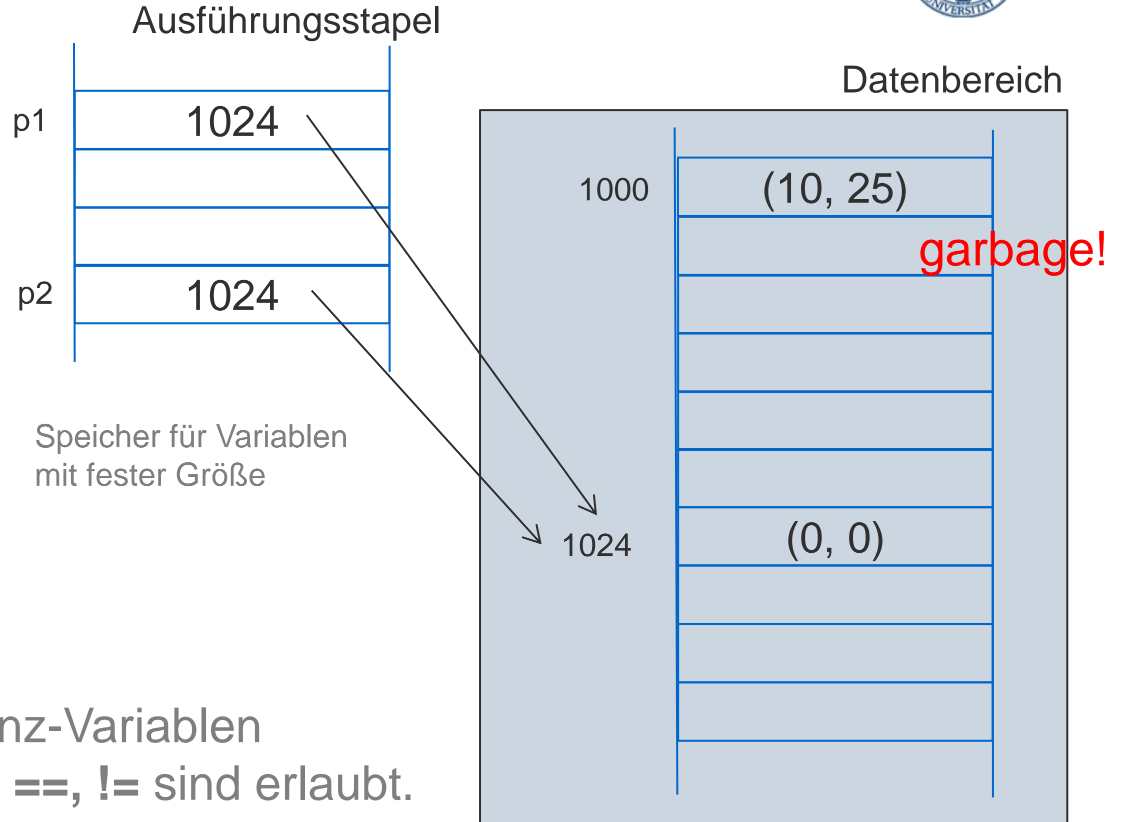
```
Point p1;
Point p2;
p1 = new Point (10, 25);
p2 = new Point (0, 0);
...
p1 = p2;
```



Referenzen sind Variablen, in denen die Adresse eines Objekts gespeichert wird

Was ist eine Referenz?

```
Point p1;

Point p2;

p1 = new Point (10, 25);

p2 = new Point (0, 0);

...

p1 = p2;
```



In Java ist Arithmetik mit Referenz-Variablen verboten. Nur die Operatoren =, ==, != sind erlaubt.

Parameterübergabe in Java

- Alle Parameter werden in Java per Wert (***by-value***) übergeben. Veränderungen der Parameter innerhalb der Methode bleiben **lokal**.
  - **Aber:** Bei Objekten sind die Referenzen die Werte!
- Die formalen Parameter einer Methode-Definition sind **Platzhalter**.
- Beim Aufruf der Methode werden die formalen Parameter durch reale Variablen ersetzt, die den gleichen Typ der formalen Parameter haben müssen.

```
...
int a = 5 ;
Point p1 = new Point ( 1, 2 );
g.methodenAufruf(a, p1);
...
```

Nach Beendigung des Methoden-Aufrufs haben sich der Wert von **a** sowie der Referenz-Wert **p1** nicht geändert.

Parameterübergabe mit primitiven Datentypen

Freie Universität Berlin

```
class Parameteruebergabe {

  public static int fakultaet ( int n ) {
    int fac = 1;
    if (n>1)
       while (n>1) {
           fac = fac*n;
           n--;
       }
    return fac;
  }

  public static void main ( String[] args ) {
    int n = 20;
    System.out.println ( "n=" + n );
    fakultaet ( n );
    System.out.println( "n=" + n);
  }
}
```

n=20

n=20

Die Variable n wird solange verändert, bis sie gleich 1 wird.

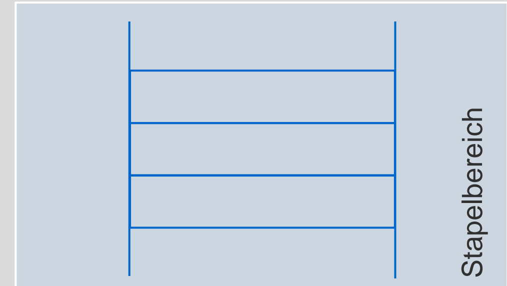
Nur eine Kopie des Parameterwertes wird übergeben.

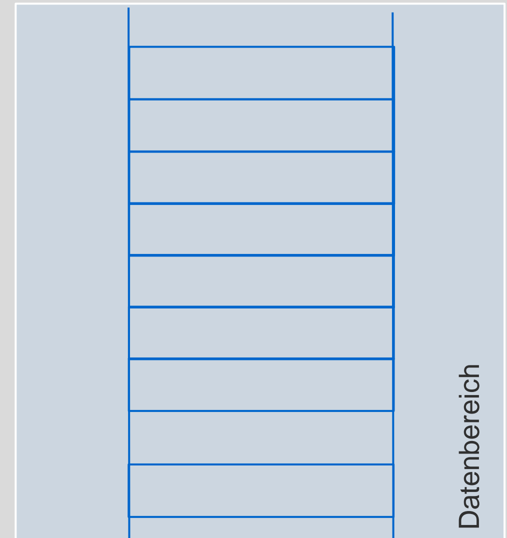
Nach Beendigung des Methoden-Aufrufs hat sich der Wert von n nicht geändert.

Parameterübergabe mit Referenz-Variablen (Objekte)

Freie Universität Berlin

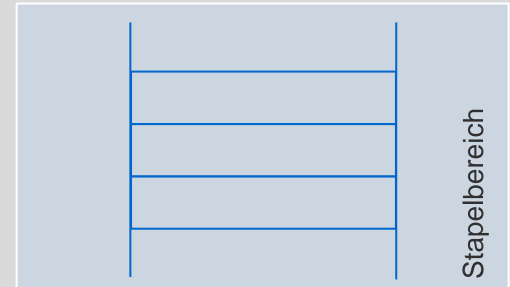
```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height ); }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

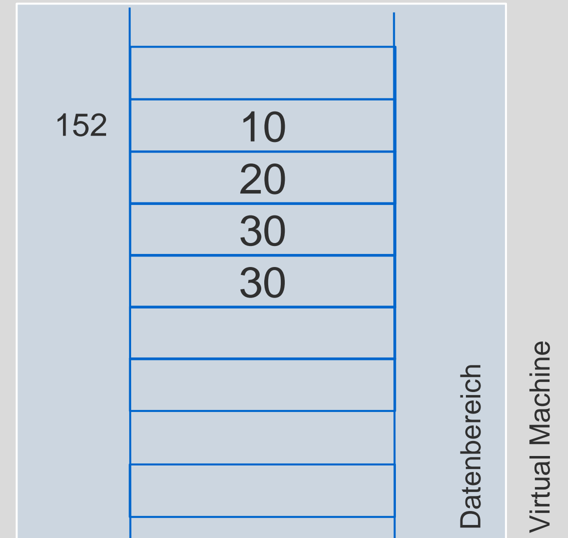




Parameterübergabe mit Referenz-Variablen (Objekte)

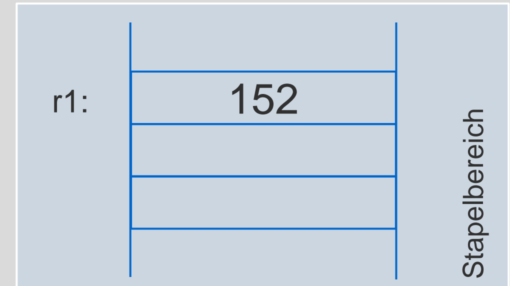
```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height ); }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

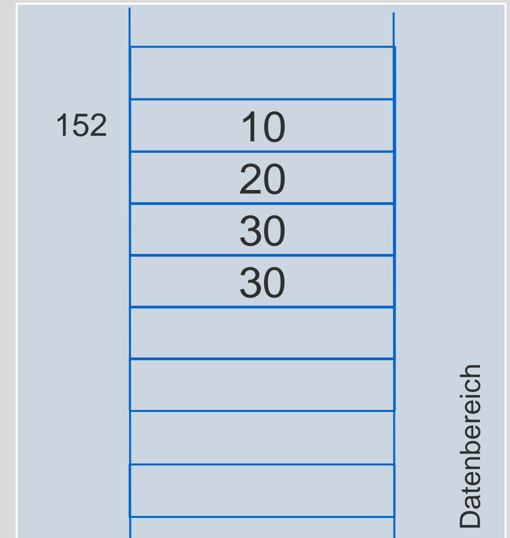




Parameterübergabe mit Referenz-Variablen (Objekte)

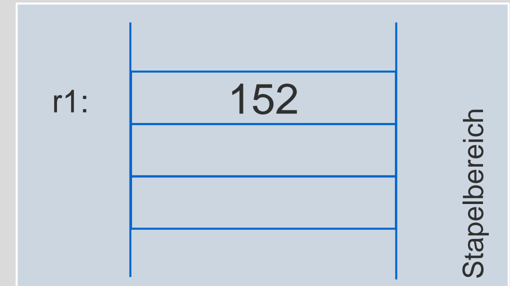
```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height ); }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

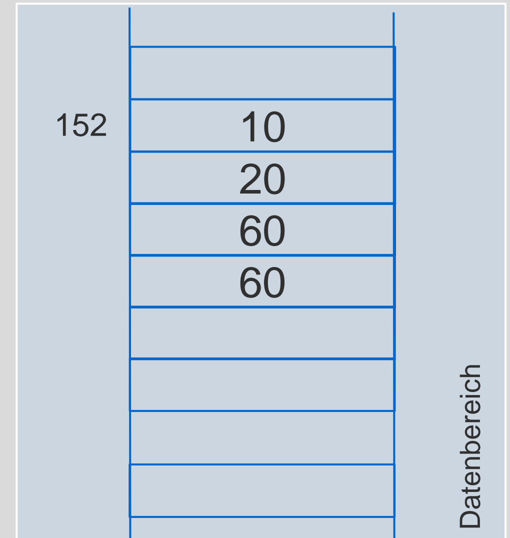




Parameterübergabe mit Referenz-Variablen (Objekte)

```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height ); }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

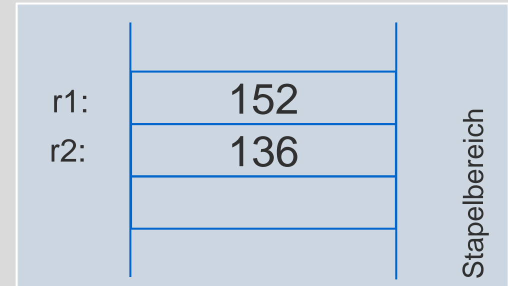


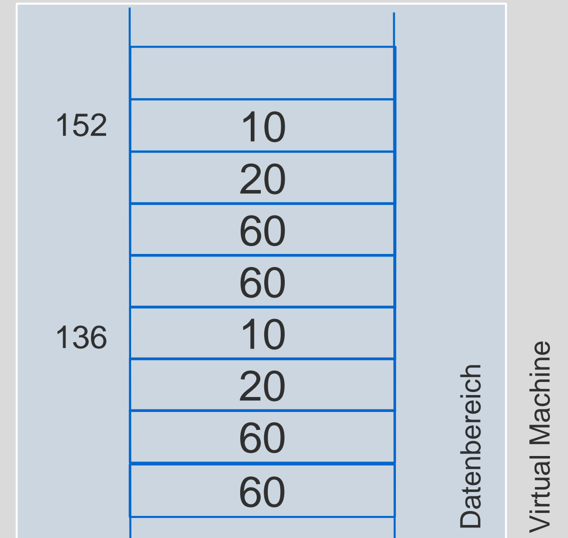


Parameterübergabe mit Referenz-Variablen (Objekte)

Freie Universität Berlin

```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height ); }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```



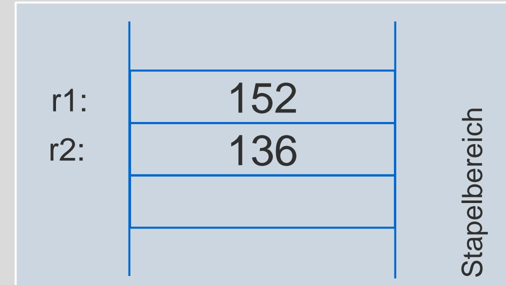


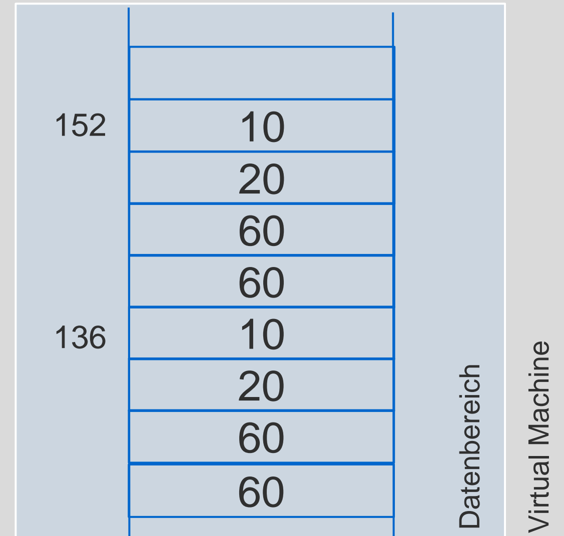
Parameterübergabe mit Referenz-Variablen (Objekte)

Freie Universität Berlin

```
public class Parameteruebergabe {

    public static void scale( Rechteck r, int s ) {
        r.height = r.height*s;
        r.width = r.width*s;
    }

    public static Rechteck clone( Rechteck r ){
        return new Rechteck(r.x, r.y, r.width, r.height ); }

    public static void main(String[] args) {
        Rechteck r1 = new Rechteck(10,20,30,30);
        scale( r1 , 2 );
        Rechteck r2 = clone( r1 );
        System.out.println( r2.flaeche() );
    }
}
```

Nach Beendigung des Methoden-Aufrufs hat sich der Referenz-Wert **r1** nicht verändert.





Einführung in die Objektorientierte Programmierung (Teil 3)

ZUSAMMENFASSUNG

Sie sollten wissen...

- wie Objekte erzeugt werden.
- welche syntaktischen Eigenschaften Kontruktoren haben und wie Sie diese verwenden.
- was ein impliziter bzw. Expliziter Konstruktor ist.
- was Modularisierung im Kontext von Java bedeutet.
- erklären können, welche Arten von Methoden existieren.
- wie Sie den Zugriff auf Variablen kontrollieren können.
- was eine Referenz ist.
- wie Parameterübergabe in Java erfolgt.