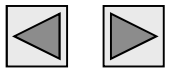
Softwareentwicklung

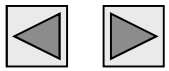
Programmierung mit Java

Teil 3



Objektorientierte Prinzipien

- Klassenbildung
  - Objekte mit gleichen Eigenschaften werden zu **Klassen** zusammengefasst
  - Objekte haben **Attribute** und **Methoden**
- Kapselung
  - Zugriff auf Objekte nur über wohldefinierte Schnittstellen (Methoden)
  - Verbergen von unwichtigen Details
- Vererbung (später ...)
- Polymorphismus (später ...)



Klassen

- zunächst nur 'einfache' Klassen
  - haben nur Attribute (Eigenschaften)
  - keine Methoden (erst später)
    ➔ zunächst keine Kapselung
- Klasse besteht aus
  - einem Klassennamen
  - beliebig vielen Attributen
  - Attribute haben einen (einfachen) Typ, z.B. int, double
  - Beispiel: Klasse „Punkt“ mit x- und y-Koordinate (Typ double) und Nummer (Typ int).

Definition einer Klasse

Klassenname

- **Syntax:**

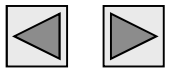
```
class Klassenname {
        Typ1  Variablenname1;
        Typ2  Variablenname2;
        . . .
}
```

Attribute werden deklariert wie Variablen, stehen jedoch innerhalb einer Klasse.

Deklaration der Attribute: jede Instanz (Objekt) der Klasse hat gleichnamige Variablen (Attribute); die Werte können aber für jedes Objekt individuell zugewiesen werden.

- **Beispiel:**

```
class Punkt {
        double xKoordinate;
        double yKoordinate;
        int punktNummer;
}
```



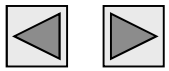
Erzeugen von Objekten einer Klasse

- **Syntax:**

```
Klassenname Variable;                 // Deklaration
Variable = new Klassenname ();        //Instantiierung
```

- **Beispiel:**

```
Punkt p1;                   //Deklaration der Variablen p1
p1 = new Punkt();           // Erzeugung einer Instanz der
                            //Klasse Punkt; p1 ist eine
                            //Referenz auf die Instanz

Punkt p2;
p2 = new Punkt();
Punkt p3 = new Punkt();
```



Zugriff auf Attribute eines Objekts

- **Syntax:**
  Variablennam
- **Beispiel:**

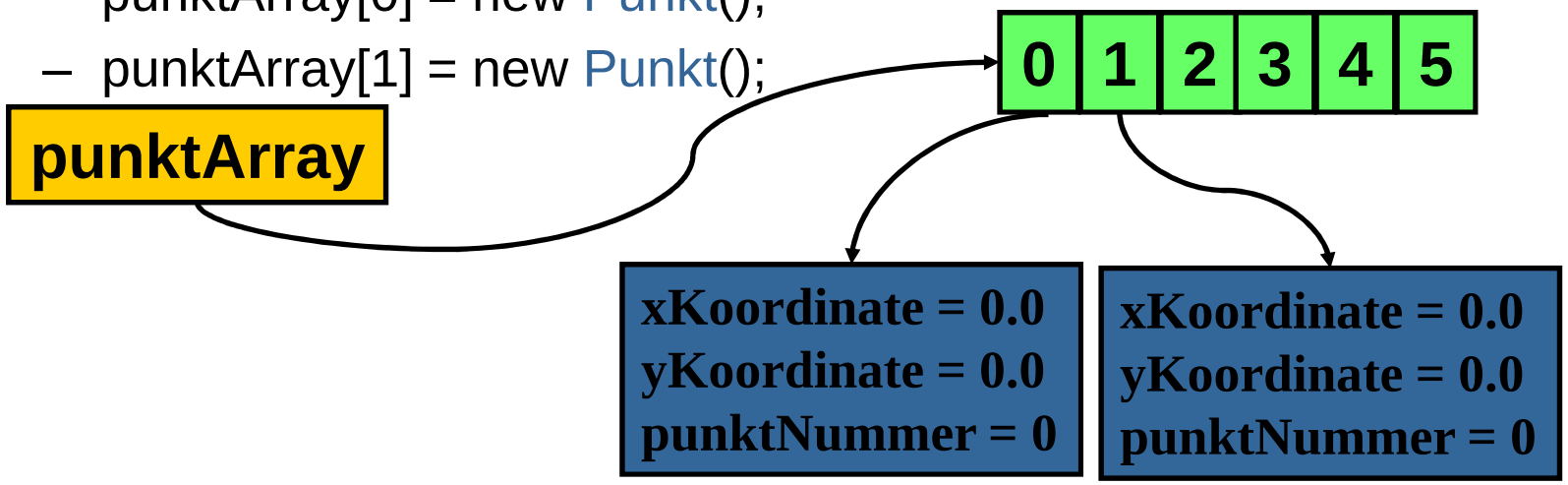
```
class Punkt {
        double xKoordinate;
        double yKoordinate;
        int punktNummer;
}
```

```
Punkt p1;
p1 = new Punkt();
p1.xKoordinate = 56987.43;
p1.yKoordinate = 4365.43;
p1.punktNummer = 6564;
double Abstand = ... + p1.xKoordinate + ...
```

Attribute eines Objekts können genauso verwendet werden wie andere Variablen desselben Typs

Einschub: Arrays vom Typ einer Klasse

- Wie erhält man Arrays mit Instanzen einer Klasse, z.B. Punkt?
- int [ ] integerArray = new int[6];
- Punkt [ ] punktArray = new Punkt[6];
- Typ der Arrayelemente ist Punkt (statt int)
- erforderlich: Erzeugen von Instanzen der Punkte:
  - punktArray[0] = new Punkt();
  - punktArray[1] = new Punkt();



Zugriff auf Attribute von Instanzen in Array

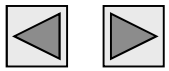
- Zugriff auf normales Objekt:

  Referenzvariable.Attributname

  - Beispiel: p1.xKoordinate
- Zugriff auf Objekt in einem Array:

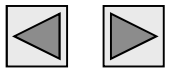
  Arrayname[index].Attributname

  - Beispiel: punktArray[0].xKoordinate = 565.99;



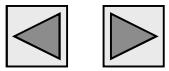
Übungsaufgabe

- Erzeugen Sie ein Array von 5 Instanzen der Klasse Punkt und belegen Sie die Koordinaten mit Werten Ihrer Wahl.



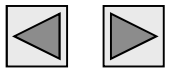
Übungsaufgabe: Lösung

```
//Datei Punkt.java:
class Punkt {
    double xKoordinate, yKoordinate;
}


…
//innerhalb von main einer anderen Datei:
Punkt []punktArray = new Punkt[5];      //Deklaration des Arrays

punktArray[0]=new Punkt();              //Instanzen anlegen
punktArray[1]=new Punkt();
punktArray[2]=new Punkt();
punktArray[3]=new Punkt();
punktArray[4]=new Punkt();

punktArray[0].xKoordinate=12.3;         //Werte festlegen für die
punktArray[0].yKoordinate=1.5;          //angelegten Instanzen
punktArray[1].xKoordinate=13.4;
punktArray[1].yKoordinate=1.6;
…
```



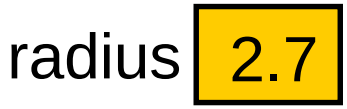
Klassen und Objekte

- **Klassen** definieren allgemeine Eigenschaften (z.B. Definition geometrischer Figuren oder den Bauplan für ein Haus)
- Von einer Klasse können **Objekte** (**Instanzen**) erzeugt werden (konkrete Ausprägung einer Klasse, z.B.: Kreis mit Radius 2.7)



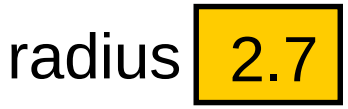
Referenzvariablen

- Variablen für einfache Datentypen (Z.B. int, double)
  - Stehen als Platzhalter für Werte
  - Variablen **beinhaltet** ihren Wert

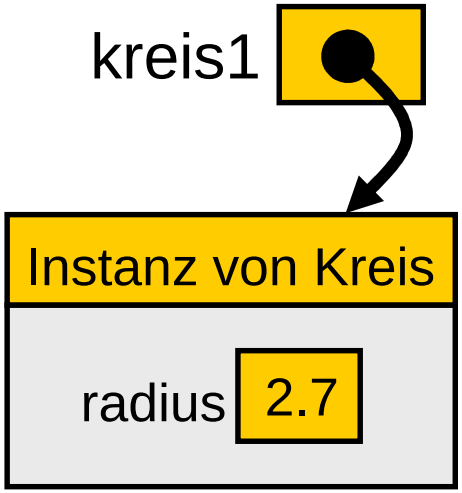


Referenzvariablen

- Variablen für einfache Datentypen (Z.B. int, double)
  - Stehen als Platzhalter für Werte
  - Variablen **beinhaltet** ihren Wert



- Referenzvariablen **zeigen** auf Objekte; sie **enthalten nicht** selber **das Objekt (wie Array-Variablen)**
  - **Kreis kreis1 = new Kreis();**
    kreis1.radius = 2.7;

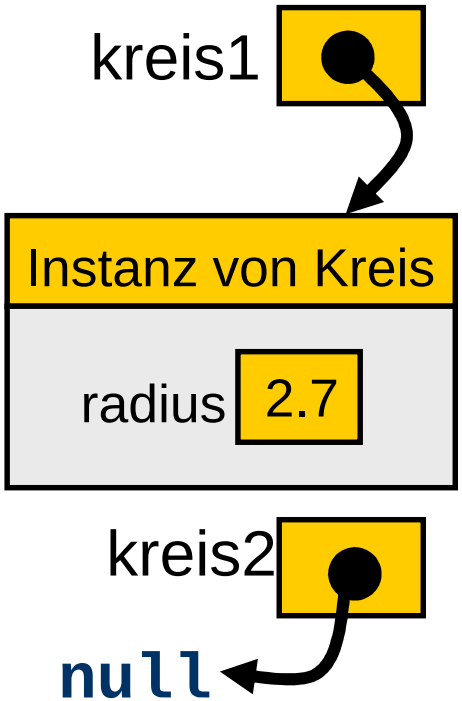


Referenzvariablen

- Variablen für einfache Datentypen (Z.B. int, double)
  - Stehen als Platzhalter für Werte
  - Variablen **beinhaltet** ihren Wert

- Referenzvariablen **zeigen** auf Objekte; sie **enthalten nicht** selber **das Objekt**
  - Referenzvariablen, die auf kein konkretes Objekt verweisen, sollte der Wert ***null*** zugewiesen werden (z.B. direkt bei der Deklaration)

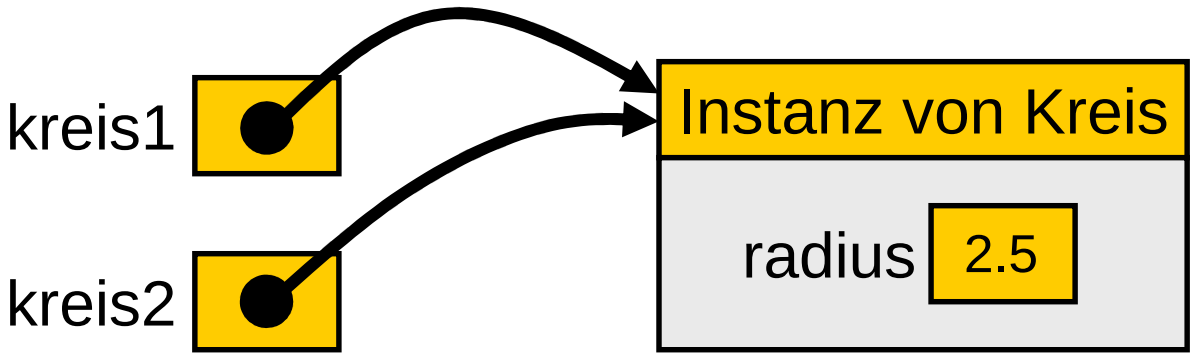
  Beispiel:
  ```
  Kreis kreis2 = null;
  ```



Referenzvariablen - Bei

Beispiel:

```
Kreis kreis1, kreis2;
kreis1 = new Kreis();
kreis1.radius = 1.0;
kreis2 = kreis1;
kreis2.radius = 2.5;
```

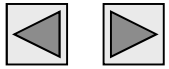


Fazit:

- Zum Kopieren von Objekten oder Erzeugen neuer Objekte reicht es nicht, die Referenzen zu kopieren
- Zum Vergleich zweier unterschiedlicher Objekte reicht es nicht auf Gleichheit der Referenzen zu prüfen

Methoden

- Klassen bestehen aus Attributen und Methoden
- Attribute speichern Werte einer Instanz
  - z.B. Radius = 3.5 bei einem Kreis
- Methoden werden für eine Instanz aufgerufen
- Methoden führen Berechnungen auf der Instanz aus
  - greifen auf Attributwerte der Instanz zu
  - geben diese z.B. aus
  - berechnen z.B. den Flächeninhalt eines Rechtecks
  - ändern z.B. Attributwerte
  - .....
- Methoden realisieren das Prinzip der Kapselung



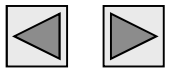
Methodendefinition

- **Syntax:**

```
class Klassenname {
        Typ1  Variablenname1;
        . . .
        Rückgabetyp  Methodenname (Parameterliste) {
                //Programmcode der Methode
        }
}
```

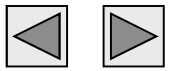
- **Beispiel:**

```
class Rechteck {
        double breite, hoehe;
        double flaeche () {
                double flaecheninhalt = breite * hoehe;
                return flaecheninhalt;
        }
}
```



Beispiele für Methodendefinitionen

```
class Rechteck {
    double breite, hoehe;             // Attribute

    // Methode zur Berechnung des Umfangs
    double umfang () {                    // keine Parameter; Rückgabetyp double
        double umfang;                    // lokale Variable (ex. nur in der Methode)
        umfang = 2*breite + 2*hoehe;  // Berechnung anhand der Attributwerte
        return umfang;                    // Rückgabe des berechneten Wertes
    }

    // Methode zur Berechnung des Flächeninhalts
    double flaeche () {
            double flaecheninhalt = breite * hoehe;
            return flaecheninhalt;
    }

    //Methode zum Ändern der Breite
    void aendereBreite(double neueBreite){
        breite = neueBreite;
    }

    // Methode zur Ausgabe des Objekts (als Text)
    void ausgeben() {             // keine Parameter; kein Rückgabewert
        System.out.println("Dieses Rechteck hat folgende Eigenschaften:");
        System.out.print("Breite: " + breite + "Höhe: " + hoehe);
        System.out.print("Umfang: " + umfang());      //Methodenaufruf!
    }
}
```



Aufruf von Methoden

Syntax:

// Methode ohne Rückgabewert (*void*)

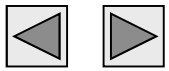
objektvariable*.*methode1(Parameterliste);

// Methode mit Rückgabewert

***variable* = *objektvariable*.*methode2(Parameterliste)*;**

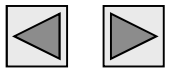
Beispiel:

```
myRechteck.ausgeben();

double flaecheninhalt;
flaecheninhalt = myRechteck.flaeche();
```



Beispiele für Methodenaufrufe

```
class Rechteck {
    double breite, hoehe;            // Attribute
    // Methode zur Berechnung des Umfangs
    double umfang () {               // keine Parameter; Rückgabetyp double
        double umfang;               // lokale Variable (ex. nur in der Methode)
        umfang = 2*breite + 2*hoehe; // Berechnung anhand der Attributwerte
        return umfang;               // Rückgabe des berechneten Wertes
    }
    //Methode zum Ändern der Breite
    void aendereBreite(double neueBreite){
        breite = neueBreite;
    }
    // Methode zur Ausgabe des Objekts (als Text)
    void ausgeben() {         // keine Parameter; kein Rückgabewert
        System.out.println("Dieses Rechteck hat folgende Eigenschaften:");
        System.out.print("Breite: " + breite + "Höhe: " + hoehe);
        System.out.print("Umfang: " + umfang());    //Methodenaufruf!
    }
}

//Aufruf (dies steht in einer anderen Datei...)
    Rechteck r = new Rechteck();
    r.aendereBreite(5.2);
    r.ausgeben();
```



Methoden mit und ohne Rückgabewert

Ohne Rückgabewert

- Deklaration:
  void methode(){
  .....

  }

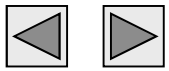
- Aufruf
  instanz.methode();
- Methode muss kein „return“ haben

Mit Rückgabewert

- Deklaration:
  double methode(){

  ...
  return double_Wert;
  }

- Aufruf:
  double d = instanz.methode()
- Methode muss „return Wert“ haben;
- Typ des Wertes = Rückgabetyp
- Zweck: Liefert Ergebnis nach außen



Methoden mit Rückgabewert

- „return“ beendet Ausführung der Methode
- Methode mit Rückgabewert darf nie ohne return beendet werden
- Gegenbeispiel:

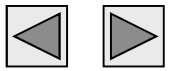
```
boolean methode1(){
        if(Bedingung1)   return true;
        if(Bedingung2)   return false;
}
```

Ende ohne „return“ wenn Bedingung1 falsch und Bedingung2 falsch

- Richtig:

```
boolean methode2(){
        if(Bedingung3) return true;
                else return false;
}
```

Ende immer mit „return“

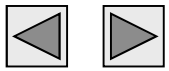


Übungsaufgabe

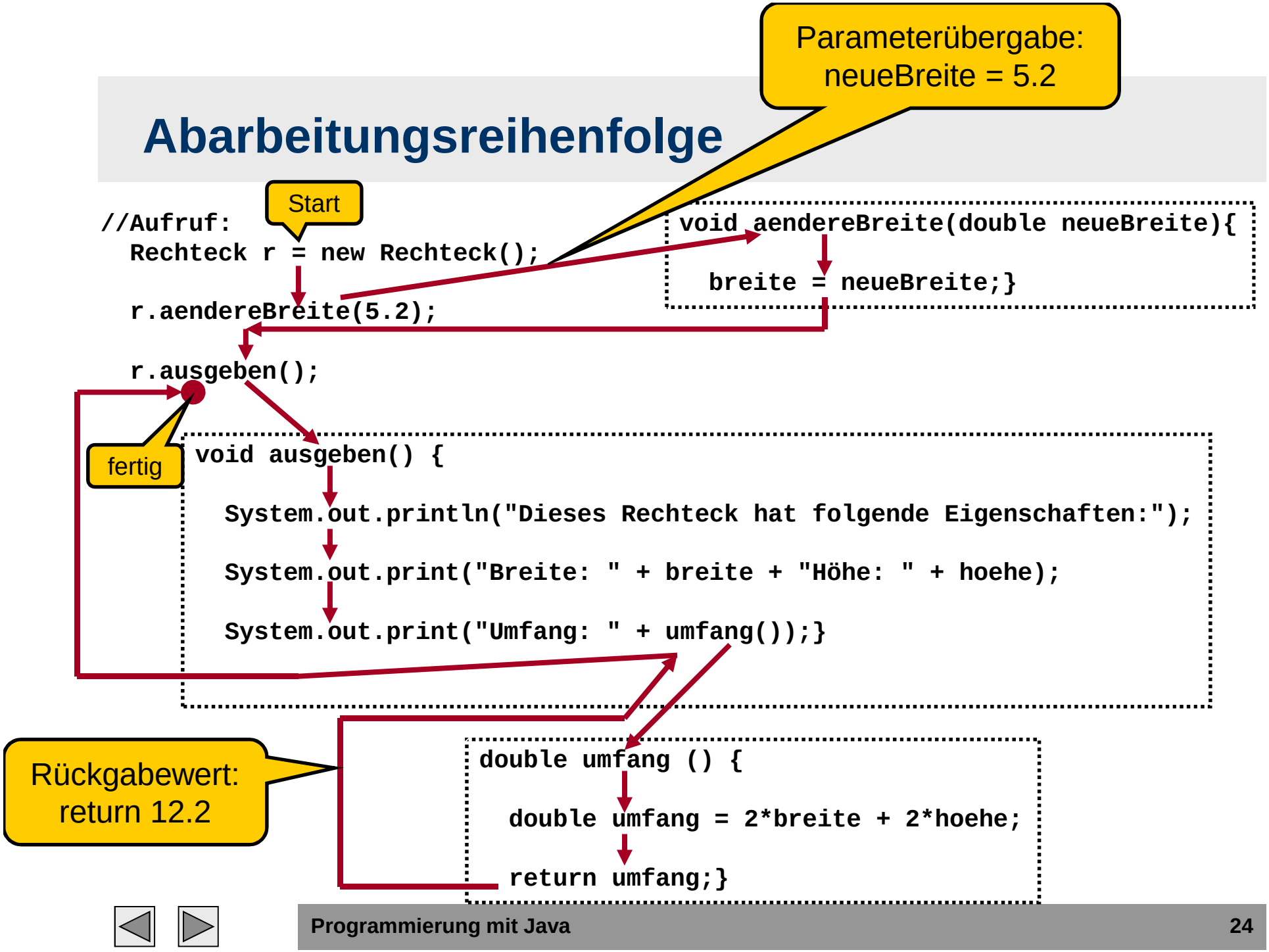
- Programmieren Sie zwei Methoden der Klasse „Kreis“, die den Flächeninhalt und den Umfang eines Kreises berechnen und zurückgeben. Berechnen Sie den Flächeninhalt und Umfang für eine Instanz und geben Sie sie aus.
- Hinweis: Die Zahl pi erhält man durch Math.PI
- Syntax:

```
class Kreis{

    double methode(){
            .............
            return double_Wert;
    }
}
```

Aufruf: Kreis k = new Kreis();
double d = k.methode();



Abarbeitungsreihenfolge



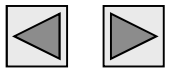
Konstruktoren (1): Motivation

- Beispiel: Erzeugen eines Dreiecks
  mit drei Attributen seite1, seite2, seite3

```
Dreieck d;
d = new Dreieck();
d.seite1 = 6.4;
d.seite2 = 3.1;
d.seite3 = 5.2;
```

- Mit Konstruktor:

```
Dreieck d;
d = new Dreieck(6.4 , 3.1, 5.2);
```



Konstruktoren (2)

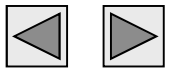
- Konstruktoren sind spezielle **Methoden**
- Name des Konstruktors = Klassenname
- Beispiel: Konstruktor für die Klasse *Punkt*
  (mit zwei Attributen xKoordinate und yKoordinate)

```
Punkt(double x, double y) {
        xKoordinate = x;
        yKoordinate = y;
}
```

- werden **implizit bei der Erzeugung** mit new aufgerufen:

```
Punkt p = new Punkt(4.9,8.5);
```

- kein Rückgabewert (auch kein void)
- darf fehlen (dann wird Standardkonstruktor verwendet)
- Zweck: Initialisierung der Instanz, Zuweisung von Anfangswerten



Konstruktoren (3)

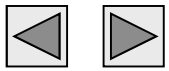
- Es kann mehrere Konstruk[illegible] geben
- diese müssen sich hinsi[illegible]ich Anzahl und Typ der Parameter unterscheiden
- Beispiel:

```
class Kreis{
        double mittelpunktX, mitt[illegible]iktY, radius;
        Kreis() {
                mittelpunktX = 0; mittelpunktY = 0; radius = 1}
        Kreis(double r){
                mittelpunktX = 0; mittelpunktY = 0; radius = r}
        Kreis(double r, double x, double y){
                mittelpunktX = x; mittelpunktY = y; radius = r}
        }
}
```

Aufruf:
Kreis k = new Kreis();

Aufruf:
Kreis k = new Kreis(56);

Aufruf:
kreis k = new Kreis(56,8,7);



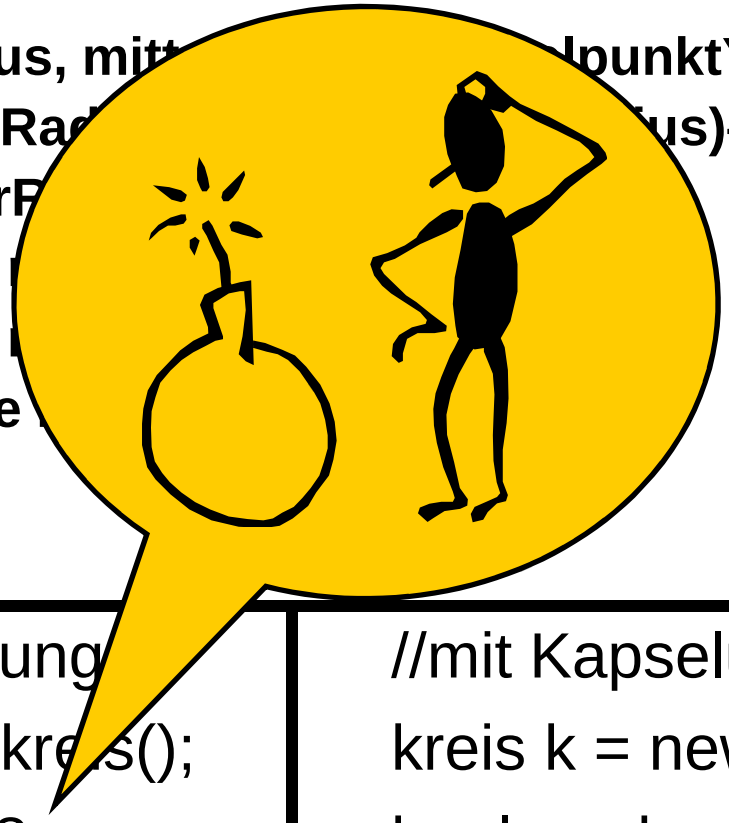
Kapselung

- auf Attribute sollte von außen nicht direkt zugegriffen werden
- Zugriff von außen nur über Methoden
- Mechanismen, um dies sicherzustellen:
  - private und protected statt public für Attribute

Modifikator	Eigene Klasse	Unterklassen	Klassen im gleichen Paket	Alle Klassen
`private`	X			
`protected`	X	X	X	
`public`	X	X	X	X
`(leer)`	X		X	

Wozu Kapselung?

```
class Kreis{
    double radius, mitt                    lpunktY;
    boolean setRa                          us){
        if(neuerR
            r
            r
        else r
    }
}
```



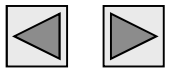
```
//ohne Kapselung
kreis k = new kreis();
k.radius = – 88;
// radius negativ!
```

```
//mit Kapselung
kreis k = new kreis();
boolean b = k.setRadius(– 88);
//radius nie negativ!
```

Auf welche Variablen darf in einer Methode zugegriffen werden?

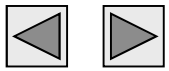
- auf die Attribute der Instanz, für die Methode aufgerufen wurde
- auf die formalen Parameter (bei Aufruf durch aktuelle ersetzt)
- auf lokale Variablen (in Methode deklariert und nur dort existent)
- Beispiel: (für Klasse Rechteck mit laenge und breite)

```
double volumen(double hoehe){
        double v;
        v = laenge * breite * hoehe;
        return v;
}
```



Methoden - Lokale Variablen

- Variablen, die innerhalb von Methoden deklariert werden, heißen lokale Variablen.
- Lokale Variablen werden bei jedem Aufruf einer Methode neu angelegt und existieren nur für die Zeitdauer der Abarbeitung der Methode.
- **Im Gegensatz:** Die evtl. innerhalb einer Methode erzeugten Objekte bestehen fort über die Methodenabarbeitung hinaus.
  - **Aber:** sollten alle Referenzen auf neue Objekte entfallen, weil am Ende einer Methode alle lokalen Referenzvariablen Aufhören zu existieren, so werden auch diese Objekte gelöscht (automatische Garbage Collection)

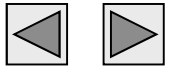


Methoden - Lokale Variablen - Beispiel

- Beispiel:

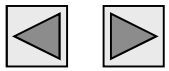
```
void methode1(){
    Punkt p = new Punkt(1,5, 6.7);
    ......
    nichtLokal = p;
}
```

- **p** ist eine **lokale Variable**
- nach Beendigung von methode1
  - existiert **p** nicht mehr
  - existiert die Instanz `Punkt(1,5, 6.7)` zunächst noch
  - wird die Instanz `Punkt(1,5, 6.7)` gelöscht, wenn keine weitere Referenz darauf existiert
  - wird die Instanz `Punkt(1,5, 6.7)` nicht gelöscht, wenn eine noch existierende Referenz darauf existiert



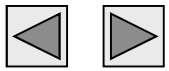
Beispiel: Klasse Punkt

```
public class Punkt{
    private double  xKoordinate;  //Attribut
    private double  yKoordinate;  //Attribut
    public Punkt(){ xKoordinate = 0.0; yKoordinate = 0.0;}  //Konstruktor
    public Punkt( double x, double y ){ xKoordinate = x; yKoordinate = y;}  //Konstruktor
    public double getxKoordinate() {return xKoordinate;}
    public double getyKoordinate() {return yKoordinate;}
    public void verschieben(double deltaX, double deltaY){
            xKoordinate = xKoordinate + deltaX;
            yKoordinate = yKoordinate + deltaY; }
    public double abstand(Punkt p){
            double a = Math.sqrt(Math.pow(xKoordinate - p.xKoordinate,2.0) +
                                 Math.pow(yKoordinate - p.yKoordinate,2.0));
            return a;  //a ist lokale Variable, p ein Parameter
    }
}
```



Beispiel: Benutzung der Klasse Punkt

```
Punkt p1 = new Punkt(6.0,8.0);
Punkt p2 = new Punkt(8.0,23.0);
p1.verschieben(2,12);
System.out.println("Der Abstand zwischen beiden Punkten beträgt " + p1.abstand(p2));
```



Übungsaufgabe

- Definieren Sie eine Klasse für Dreiecke, die in der Ebene eingebettet sind (d.h. alle drei Punkte haben x- und y-Koordinaten). Definieren Sie
  - einen komfortablen Konstruktor (der ein Dreieck in einem Schritt erzeugt)
  - eine Methode zur Berechnung des Umfangs,
  - eine Methode zum Verschieben des Dreiecks und
  - eine Methode zur textuellen Ausgabe.
- Hinweis: Es bietet sich an, die Klasse *Punkt* zu verwenden.