# Konstruktoren

Wir nähern uns dem Thema von zwei Seiten:

1. Wenn wir ein Objekt erzeugen, z.B. in  
     
    Dreieck d1 = new Dreieck**()**;  
     
   fragen wir uns,  
   a) warum wir die abhängige Klasse,  
   von der ein Objekt erzeugt werden soll,  
   zweimal nennen müssen und

b) warum dahinter ein leeres rundes Klammerpaar steht,  
das wir sonst nur von Methoden her kennen.

1. Wir finden es umständlich,  
   bei Objekten mir vielen Attributen  
   diese erst in der abhängigen Klasse zu initialisieren  
   u. gleich danach in der Hauptklasse einzeln mit mit Werten belegn zu müssen.

class Dreieck {

2 int a = 0;

3 int b = 0;

4 int c = 0;

5 boolean ist\_schraffiert = false;

}

class Dreieck\_Haupt\_1{

… main …{

1 Dreieck d1 = new Dreieck();

6 d1.a = …;

7 d1.b = …;

8 d1.c = …;

9 d1.ist\_schraffiert = …;

}

}

Wir benötigen 9 (!) Anweisungen, bis das Dreieck "fertig vor uns liegt",

darunter 5 im Hauptprogramm.

Wir suchen nach einer Möglichkeit, dies zu reduzieren,

insbesondere aus dem Hauptprogramm auszulagern.

# Schritt 1:

Antworten zur Frage, warum

1. der Klassenname doppelt genannt wird und
2. warum ein leeres rundes Klammerpaar hinter der zweiten Nennung des Klassennamens steht

zu 1a): Die zweite Nennung des Klassennamens ist der Aufruf des sog. "Standardkonstruktors".

Ein Konstruktor ist eine spezielle Methode, die ein Objekt einer Klasse erzeugt.

Bei Definition einer abhängigen Klasse wird automatisch der sog. Standardkonstruktor ins Leben gerufen,

damit wir ein Objekt der Klasse erzeugen können.

zu 1b): Weil der Standardkonstruktor eine besondere Methode ist,  
stehen dort auch die von Methoden her bekannten runden Klammern.

Die Erkenntnis aus Schritt 1 geben die Antwort für die Fragen unter Punkt 1,

helfen uns aber bei dem Wunsch unter Punkt 2 noch nicht weiter,

mit weniger Aufwand eine Objekt erzeugen u. belegen zu können.

# Schritt 2:

Wir können selbst Konstruktoren definieren.

Dazu müssen wir uns die Besonderheiten von Konstruktoren gegenüber "normalen" Methoden,

wie wir sie bisher verwendet haben, klar machen:

1. Konstruktoren heißen immer genauso wie die Klasse.  
   Das ist ungewöhnlich, verschiedenen Dingen den gleichen Namen zu geben!
2. Entgegen der Konvention für Methoden werden Konstruktoren groß geschrieben.
3. Konstruktoren haben keinen Rückgabetyp.  
   Auch das Schlüssselwort "void" darf nicht aufgeführt werden!
4. Konstruktoren schreiben wir gleich nach der Deklaration der Attribute,  
   vor anderen Methoden, auch vor getter- u. setter-Methoden!

Als Inhalt des Konstruktors nehmen wir die Standardinitialisierung auf,

die bisher bei der Deklaration erfolgte.

Wenn wir einen eigenen Konstruktor definieren,

überschreiben wir damit den bisherigen Standardkonstuktor.

Er ist nicht mehr zugreifbar.

class Dreieck {

int a = 0;

int b = 0;

int c = 0;

boolean ist\_schraffiert = false;

public Dreieck (){

}

}

class Dreieck\_Haupt\_1{

… main …{

Dreieck d1 = new Dreieck();

d1.a = …;

d1.b = …;

d1.c = …;

d1.ist\_schraffiert = …;

}

}

class Dreieck {

int a;

int b;

int c;

boolean ist\_schraffiert;

public Dreieck (){

a=0;

b=0;

c=0;

ist\_schraffiert = false;

}

}

class Dreieck\_Haupt\_1{

… main …{

Dreieck d1 = new **Dreieck()**;

d1.a = …;

d1.b = …;

d1.c = …;

d1.ist\_schraffiert = …;

}

}

Schreibe ich im Hauptprogramm den gleichen Text wie bisher, nämlich

Dreieck d1 = new Dreieck();

geschieht im Hintergrung etwas anderes, obwohl sich der Text nicht geändert hat:

Es wird nicht mehr der Standardkonstruktor ausgeführt, sondern der selbst definierte.

class Dreieck {

int a;

int b;

int c;

boolean ist\_schraffiert;

public Dreieck (){

2 a=0;

3 b=0;

4 c=0;

5 ist\_schraffiert = false;

}

}

class Dreieck\_Haupt\_1{

… main …{

1 Dreieck d1 = new Dreieck();

6 d1.a = …;

7 d1.b = …;

8 d1.c = …;

9 d1.ist\_schraffiert = …;

}

}

Hat es uns bis hierher eine Ersparnis gebracht? Nein!

Wir haben die Standardinitialisierung nur an eine andere Stelle verschoben, aber immer noch genauso viele Anweisungen.

Ja noch schlimmer:

Durch die Deklaration des selbstdefinierten Konstruktors haben wir sogar mehr Schreibarbeit als zuvor!

# Schritt 3:

Wesentlicher Schritt:

Selbst definierten Konstruktoren kann ich Parameter spendieren nach Belieben!

Dabei ist es insbesondere interessant, genauso viele Parameter vorzusehen, wie die Klasse Attribute hat.

Dann könnten wir beim Aufruf dieses Konstruktors aus dem Hauptprogramm

mit nur diesem einen Aufruf das Objekt erzeugen u. Werte für sämtliche Attribute übergeben.

Den bisherigen selbst definierten Konstruktor ohne Parameter kommentieren wir (vorübergehend) aus,

weil noch unklar ist, ob mehrere Konstruktoren parallel existieren dürfen:

class Dreieck {

int a;

int b;

int c;

boolean ist\_schraffiert;

/\* public Dreieck (){

a=0;

b=0;

c=0;

ist\_schraffiert = false;

}

\*/

public Dreieck(**int a, int b, int c, boolean is**){

2 this.a = a;

3 this.b = b;

4 this.c = c;

5 ist\_schraffiert = is;

}

}

class Dreieck\_Haupt\_1{

… main …{

**1** Dreieck d1 = new Dreieck(**3,4,5,true**);

}

}

Nun haben wir nur noch 5 Anweisungen, davon nur noch eine im Hauptprogramm!

Das ist eine erhebliche Ersparnis, insbesondere im Hauptprogramm!

Vorher: 9 Anweisungen, davon 5 im Hauptprogramm

Nachher: 6 Anweisungen, davon nur noch eine im Hauptprogramm (!)

Auf den Umweg über die Standardinitialisierung können wir verzichten,

weil durch den Konstruktor sichergestellt ist,

dass jedem Attribut explizit ein Wert zugewiesen wird.

# Schritt 4:

Eben haben wir, als wir den Konsturktor mit mehreren Parametern implementiert haben,

den ohne Parameter deaktiviert durch Auskommentierung,

weil unklar ist, ob beide nebeneinander existieren dürfen.

An dieser Stelle benötigen wir die Aussage des "Polymorphismus":

*"Es darf mehrere Methoden gleichen Namens parallel geben,*

*sofern sie sich in der Parameterliste unterscheiden."*

Dies übertragen wir auf Konstruktoren, die ja spezielle Methoden sind:

Alle Konstruktoren heißen gleich, nämlich wie die Klasse.

Sie haben verschiedene Parmeterlisten,

in unserem Fall der erste keine Parameter, der zweite so viele wie die Klasse Attribute.

Konstruktoren sind spezielle Methoden.

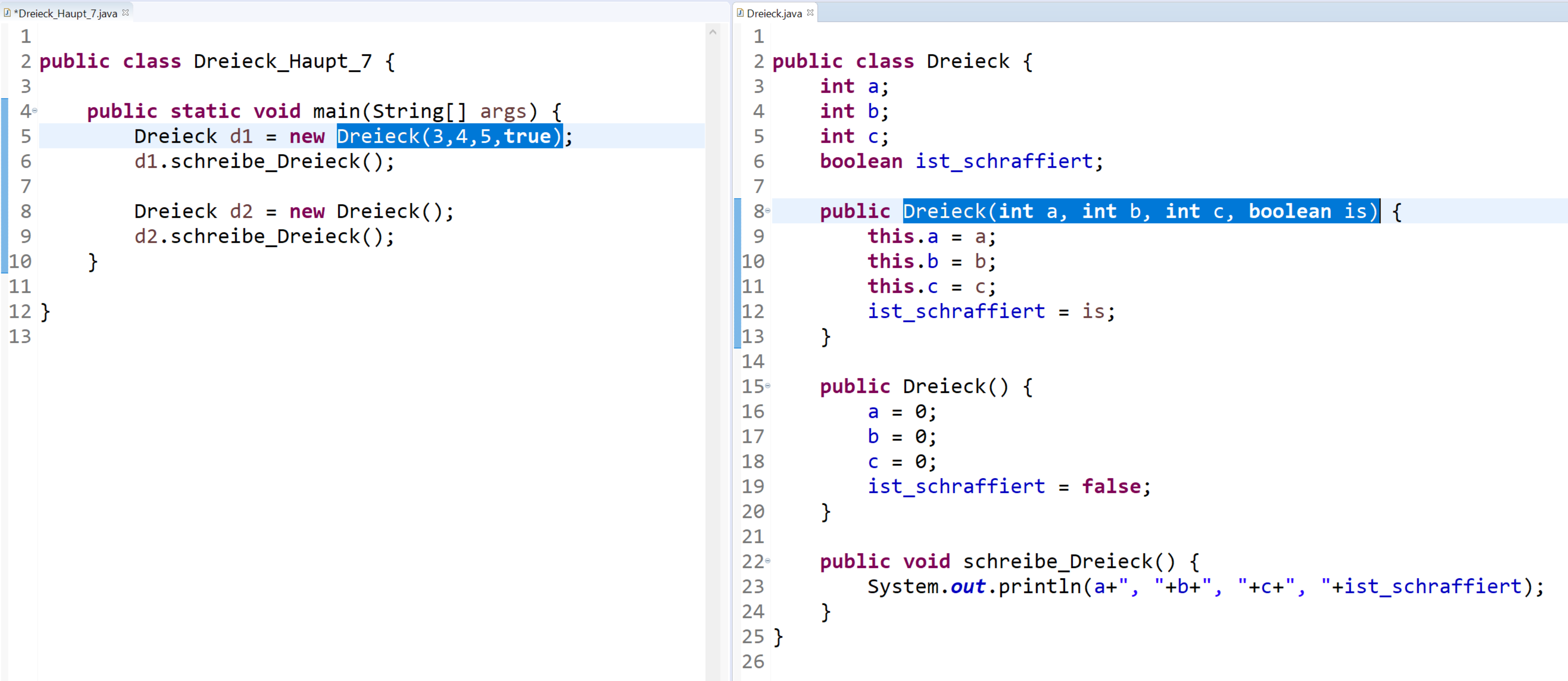
Deshalb können wir die Erkenntnis des Polymorphismus übertragen:

**Es darf mehrere Konstruktoren parallel geben,**

**sofern sie sich in der Parameterliste unterscheiden.**

Zur Laufzeit sucht der Compiler aufgrund der aktuellen Parameterliste die passende Implementierung heraus.

Bei der Erzeugung eines Objektes mit einem selbst definierten Konstruktor können wir dann wahlweise den einen oder anderen benutzen.



Wir haben gesehen, dass es geht, mehrere Konstruktoren parallel zu haben.

Macht es auch Sinn? Ja!

Sagen wir, wir wollen häufig Objekte mit gleichen Werten erzeugen,

bei Dreiecken z.B. "Standarddreiecke", angelehnt an einen "Standardkreis" mit Radius 1.

Ein Standarddreieck soll gleichseitig sein mit der Seitenlänge 1 u. nicht schraffiert.

Wenn wir diese "Default-Werte" bei dem Konstruktor ohne Parameter hinterlegen,

können wir mit minimalem Aufwand diese häufig benutzten Werte anwenden,

ohne sie neu eingeben zu müssen.

Wir brauchen nicht zu schreiben:

… = new Dreieck (1, 1, 1, false);

Sondern es reicht:

… = new Dreieck ();

Haben wir jedoch "einen Sonderwunsch", hier: ein Dreieck zu erzeugen mit speziellen Seiten,

nehmen wir den allgemeinen Konstruktor u. können genaue Angaben machen.

Vergleich mit einem Dialogfeld, z.B. bei der Installation eines Programmes:

Sie werden gefragt, in welchem Pfad das PRG installiert werden soll.

Im Feld für den Pfad haben Sie eine Vorgabe, den Standardpfad.

Sie haben nun die Wahl:

Bestätigen Sie bei minimalem Aufwand mit der Enter-Taste,

wird dieser am meisten verwendete Pfad übernommen.

Sie haben jedoch auch die Möglichkeit, einen anderen Pfad anzugeben.

Dieser überschreibt dann den vorgeschlagenen Standardpfad u. wird übernommen.

# Schritt 5:

Wir haben nur zwei Konstruktoren:

einen, bei dem wir nichts dazu sagen, einen anderen, bei dem wir alles angeben müssen.

Frage: Würde es Sinn machen, auch "Zwischengrößen" anzubieten?

Hier im Beispiel der Dreiecke:

Am häufigsten brauche ich Einheitsdreiecke: Gleichseitig u. mit Seitenlänge 1, unschraffiert.

Daneben brauche ich aber häufig unschraffierte gleichseitige Dreiecke mit einer von 1 differenten Seitenlänge.

Wenn ich ein solches mit bisherigen Mitteln erzeugen wollte, müsste ich schreiben:

Dreieck (3,3,3,false)

Alternativ könnte ich mir einen dritten Konstruktor defininieren,

der nur genau eine Seitenlänge übernimmt u. diese auf alle drei Seiten schreibt.

Dann könnte ich die dreifache Übergabe der gleichen Zahl sowie das "false" vermeiden:

Dreieck (3)

Dies wäre eine Vereinfachung.

Deshalb implementieren wir Konstruktoren mit beliebigen Anzahlen u. Kombinationen von Parmetern

u. definieren unseren Bedürfnissen entsprechend deren Bedeutung wie hier beim gleichseitigen Dreieck.

Man könnte auch hier den Vergleich mit dem Dialogfeld erweitern:

Wir haben nicht nur das Feld für den Installationspfad, sondern auch Buttons um z.B. festzulegen,

ob für das zu installierende Programm eine Desktopverknüpfung oder einen Eintrag ins Starfeld oder die Taskleiste erstellt werden soll.

Für alle gibt es Vorgaben, die beliebig definiert u. durch Drücken der Enter-Taste alle übernommen werden können.

Auch können alle Vorgaben geändert werden.

Nun sind aber auch Mischungen möglich:

Das eine ändere ich, von andern lasse ich die Standardvorgaben bestehen.

So kann ich mit minimalstem Aufwand das gewünschte Ergebnis erzielen.