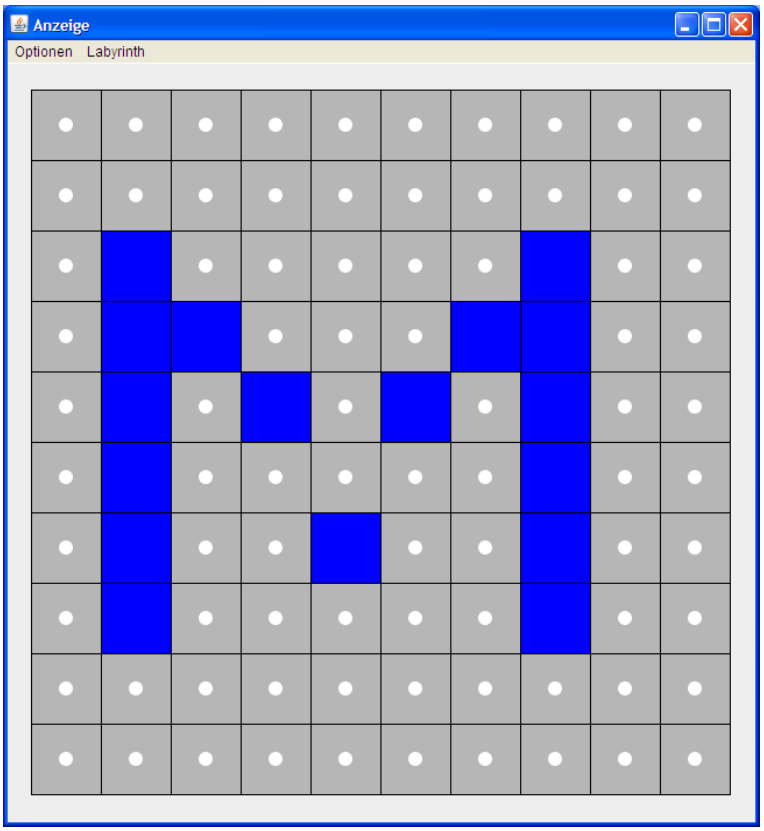
Kapitel 14 Das Labyrinth optimieren

Lernziele:

Erstellen mehrerer Methoden mit gleichem Namen in einer einzigen Klasse
Vertiefungen zur Objektkommunikation.

14.1 Labyrinth mit Powerkrümeln

Für das Spiel ist es wichtig, dass beim Erzeugen eines Labyrinths nicht nur normale Krümel wie in Abbildung 1 oben vorhanden sind, sondern auch einige Powerkrümel, die Mampfi die Chance geben, die Monster zu fressen (Abbildung 1 unten).



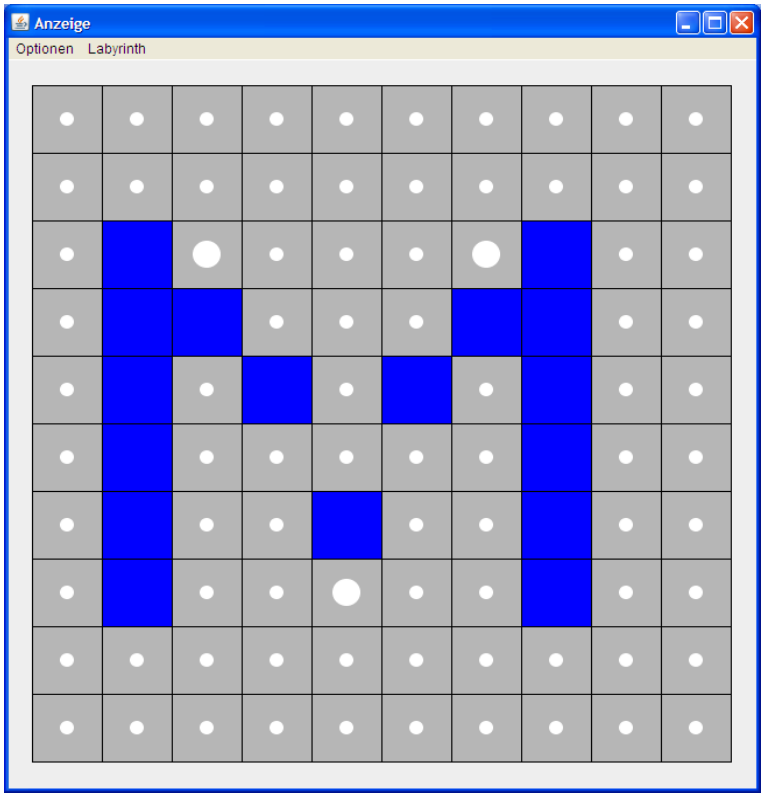


Abbildung 1: Labyrinth ohne Powerkrümel (oben) und mit Powerkrümel (unten)

Aufgabe 14.1
In der Klasse LABYRINTH soll die Methode *PowerKruemelVerteilen* hinzugefügt werden, die Powerkrümel im Labyrinth ergänzt. Überlege welche Objektkommunikationen dazu ablaufen müssen. Fasse das Ergebnis deiner Überlegungen in einem Sequenzdiagramm zusammen.
Schreibe auf, in welchen Klassen Methodenrümpfe ergänzt bzw. neue Methoden hinzugefügt werden müssen.

(Wenn du dich sicher fühlst, kannst du deine Planungen aus dieser Aufgabe in deinem BlueJ-Projekt umsetzen und dann mit Kapitel 14.2 fortfahren)

Das Labyrinth soll die Methode *PowerKruemelVerteilen* erhalten. Beim Aufruf dieser Methode müssen nromale Krümel in Powerkrümel umgewandelt werden, d.h. u.a. muss die Methode *MachtUnverwundbarSetzen* der Krümelobjekte aufgreufen werden. Jedoch hat das Labyrinth keine einzige Referenz auf einen Krümel. Die Kommunikation muss ähnlich wie bei der Methode *GaengeErstelllen* indirekt über die "Zwischenstation" von Zellen erfolgen:

a) Das Labyrinth gibt der ersten betroffenen Zelle durch einen Methodenaufruf den Auftrag dort einen Powerkrümel hinzulegen.
b) Die Zelle gibt den Auftrag an den Krümel weiter, indem sie die Methode *MachtUnverwundbarSetzen* mit dem Eingabewert `true` aufruft.
c) Der Krümel sendet seinem Symbol die Nachricht sein Aussehen zu verändern, damit man erkennt, dass es ein Powerkrümel ist, z.B. durch das Vergrößern des Radius.
d) Das Vorgehen a) bis c) wird an weitere Zellen wiederholt. Im Fall von Abbildung 9 muss insgesamt 3 Zellen die Nachricht *PowerKruemelHinlegen* gesendet werden.

Das Sequenzdiagramm in Abbildung 2 veranschaulicht die Schritte a) bis c)

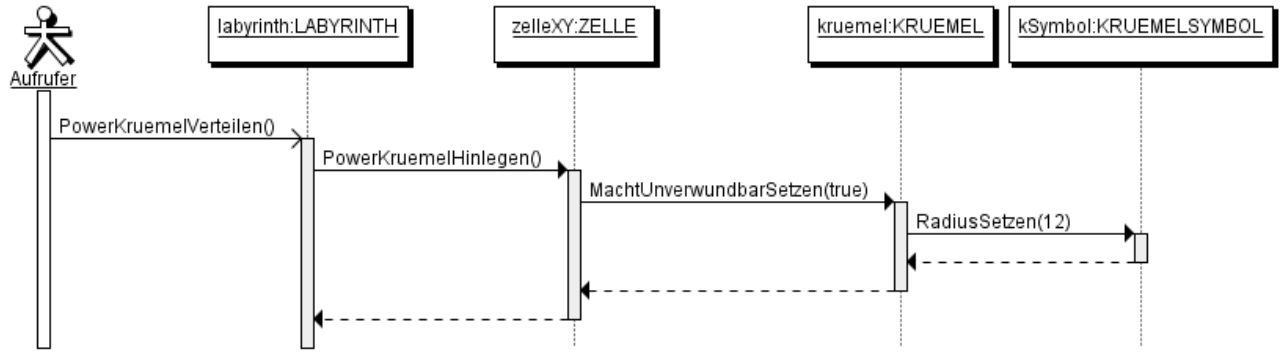


Abbildung 2: Sequenzdiagramm zur Veranschaulichung der Objektlkommunikation beim Aufruf der Methode *PowerKruemelVerteilen*

Die Methode *MachtUnverwundbarSetzen* gibt es bereits in der Klasse KRUEMEL. Schritt c) ist schon implementiert.

Für Schritt a) und b) muss in der Klasse LABYRINTH die Methode *PowerKruemelVerteilen* und in der Klasse ZELLE eine Methode entsrpechend b) neu hinzugefügt werden. Ein aussagekräftiger Name für die neue Methode der Klasse ZELLE ist *PowerKruemelHinlegen* (siehe Fettdruck in den Klassendiagrammen).

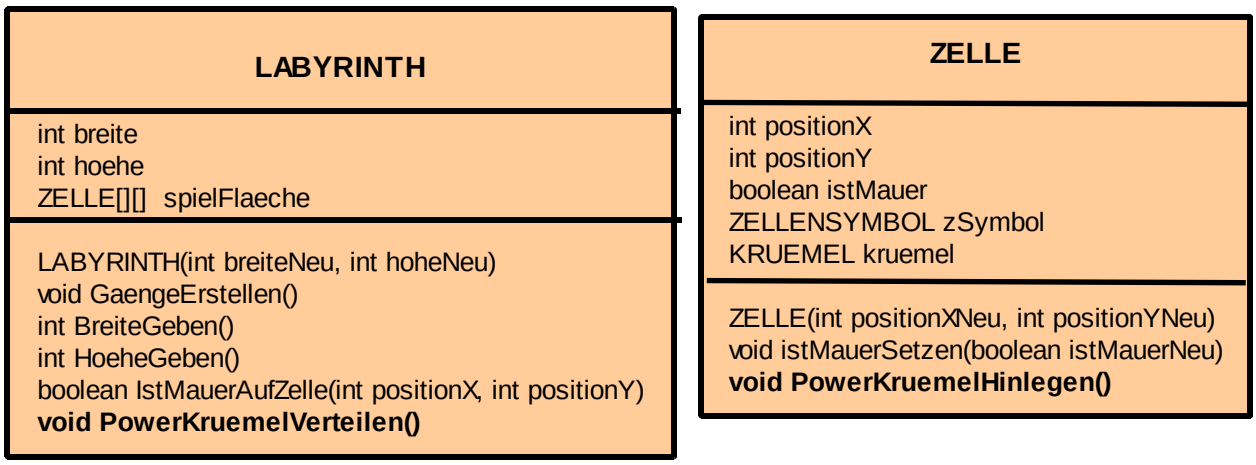


Abbildung 3: erweiterte Klassendiagramme von LABYRINTH und ZELLE mit den Änderungen fett markiert

Aufgabe 14.2

Setze die besprochenen Änderungen in den Klassen ZELLE und LABYRINTH um. Teste danach, indem du ein Objekt der Klasse Labyrinth erzeugst und dann hintereinander die Methode *GaengeErstellen* und *PowerKruemelVerteilen* aufrufst.
Dokumentiere die Veränderungen in den Klassen.
(Solltest du dich unsicher fühlen, spicke bei der Methode GaengeErstellen und versuche das Vorgehen dort zu übertragen. Solltest du immer noch Probleme haben, gibt es einen Tipp im Anhang.)

14.2 Optimierung der Klasse LABYRINTH

Ziel dieses kurzen Teilkapitels ist es, die Klasse LABYRINTH so zu optimieren, dass in den folgenden Kapiteln effektiver und schneller gearbeitet werden kann. Als Einstieg dazu, wird der Konstruktor der Klasse LABYRINTH betrachtet (Abbildung 4):

```
/**
* Konstruktor der Klasse LAYBRINTH mit Eingangsparamtern
* @param breiteNeu Breite des Labyrinths
* @param hoheNeu H&ouml;he des Labyrinths
*/
public LABYRINTH(int breiteNeu, int hoeheNeu)
{
    breite = breiteNeu;
    hoehe = hoeheNeu;

    spielFlaeche = new ZELLE[breite][hoehe];
    for(int zaehlerX = 0; zaehlerX <= breite-1; zaehlerX = zaehlerX+1)
    {
        for(int zaehlerY = 0; zaehlerY <= hoehe-1; zaehlerY = zaehlerY+1)
        {
            spielFlaeche[zaehlerX][zaehlerY] = new ZELLE(zaehlerX,zaehlerY);
        }
    }

}
```

Abbildung 4: Konstruktor der Klasse LABYRINTH

Aufgabe 14.3
a) Beschreibe knapp und prägnant die Bestandteile des Konstruktors.
b) Warum könnte man auf die Eingabeparameter beim Konstruktor in Kapitel 13 und auch hier in Kapitel 14 verzichten?
c) Welche Methoden müssen nach dem Erzeugen des Labyrinths aufgerufen werden, um das Labyrinth in Abbildung 1 unten zu erhalten?

Im Konstruktor in Abbildung 4 werden zunächst die Attribute breite und hohe initialisiert, dann das Referenzattribut spielflaeche, indem zuvor zweidimensionales Feld (im Schachtelmodell ein Aktenschrank) erzeugt und dann die Referenz darauf dem Attribut spielflaeche zugewiesen wird.
Durch die geschachtelten Wiederholungen mit fester Anzahl werden alle Feldelemente erzeugt und entsprechend zugewiesen (Im Schachtelmodell alle Schubladen des Aktenschranks gefüllt)
Um ein Labyrinth wie in Abbildung 1 unten zu erhalten müssen folgende drei Anweisungen ausgeführt werden:
- Erzeugen eines Labyrinths mit den Eingabewerten 10 und 10.
- Aufruf der Methode *GaengeErstellen*
- Aufruf der Methode *PowerKruemelVerteilen*

Da in den folgenden Kapiteln immer wieder genau diese drei Anweisungen durchgeführt werden müssen, ist es sinnvoll dies zu automatisieren.

Aufgabe 14.4
Wie wäre so eine Automatisierung möglich?

Im Konstruktor könnten die Aufrufe der Methoden *GaengeErstellen* und *PowerKruemelVerteilen* ergänzt werden. Weiterhin wäre es möglich, die Eingangsparameter zu löschen und die breite und hoehe ohne Wahlmöglichkeit auf 10 zu initialisieren.
Auch wenn bis auf weiteres die Spielfeldgröße bei 10 x 10 bleiben wird, sollte die Option auf Eingabewerte beim Konstruktor nicht gelöscht werden. Es ist möglich, in einer Klasse mehrere Konstruktoren zu schreiben. Dies ist jedoch nur gestattet, wenn sich die Konstruktoren in den Datentypen der Eingangsparameter unterscheiden. Das ist für unser Ziel keine Einschränkung:
bisher: LABYRINTH(int breiteNeu, int hoeheNeu) –
zwei Ganzzahlen (int) als Datentypen der Eingangsparameter
zusätzlich LABYRINTH() –
kein Eingangparameter

Aufgabe 14.5
Ergänze in der Klasse LABYRINTH einen zweiten Konstruktor *LABYRINTH()*.
Teste und Dokumentiere.
(Bei Bedarf findest du Tipps im Anhang.)

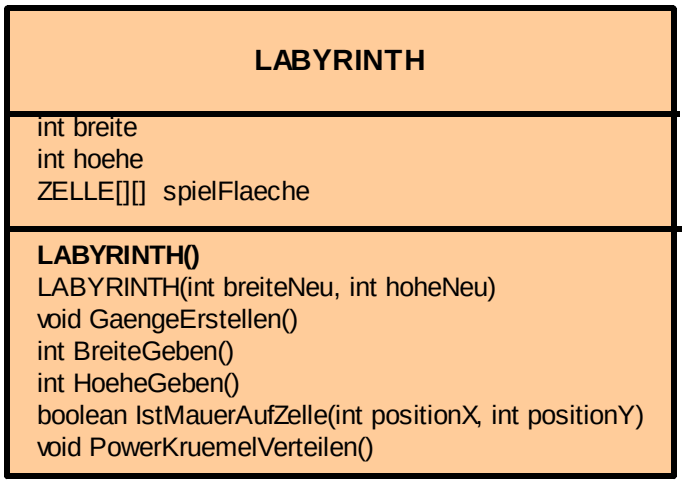


Abbildung 5: Ergänzung eines zweiten Konstruktors in der Klasse LABYRINTH

Hinweise:
- Die Parameterliste muss sich in den Datentypen der Eingangsparameter unterscheiden, die Namen und Bedeutungen der Eingangsparameter sind egal. So wäre folgender Konstruktor als Ergänzung zur Klasse LABYRINTH nicht möglich:
  LABYRINTH(int hoheNeu, int breiteNeu)
  Dieser Konstruktur unterscheidet sich von *LABYRINTH(int breiteNeu, int hoheNeu)* dadurch, dass die Breite und Höhe vertauscht sind, aber bei beiden Konstruktoren sind die Datentypen der Eingangsparameter int, int
- Es ist in einer Klasse möglich, dass auch andere Methoden mehrfach implementiert werden. Wie beim Konstruktor gilt, dass sie sich in den Datentypen der Eingangsparameter unterschieden müssen. Man nennt dies überladen einer Methode.

Aufgabe 14.6
Begründe jeweils stichhaltig, ob folgende zusätzliche Methoden in der Klasse LABYRINTH aus Abbildung 5 ergänzt werden könnten.

a) *LABYRINTH(int seitenlaenge) – erzeugt ein quadratisches Labyrinth mit der als Eingabewert eingegebenen Seitenlänge*
b) *LABYRINTH(int breiteNew, int hoeheNew)* – erzeugt ein Labyrinth mit der als Eingabewerte eingegebenen Breite und Höhe.
c) *LABYRINTH() – erzeugt ein rechteckiges Labyrinth mit der festen Breite 8 und festen Länge 4.*
d) *GaengeErstellen() – erzeugt andere Gänge als die Methode, die in Abbildung 5 aufgelistet ist.*
e) *GaengeErstellen3() – erzeugt andere Gänge als die Methode, die in Abbildung 5 aufgelistet ist.*
f) *GaengeErstellen(int auswahlnummer) – abhängig von der eingegebenen Nummer werden unterschiedliche Gänge erstellt:*

Eingabewert	Tätigkeit der Methode beim Aufruf
1	*Gänge wie bei der Methode GaengeErstellen aus Abbildung 5 erstellt*
2	Mauern werden nur in der obersten und untersten Zeile und in den Spalten ganz rechts und links erzeugt. (Es wird sozusagen ein Rahmen gebildet)
3	*Gänge wie bei der Methode GaengeErstellen3 aus Teilaufgabe e werden erstellt*
sonst	keine Gänge werden erstellt

14.3 Zusammenfassung

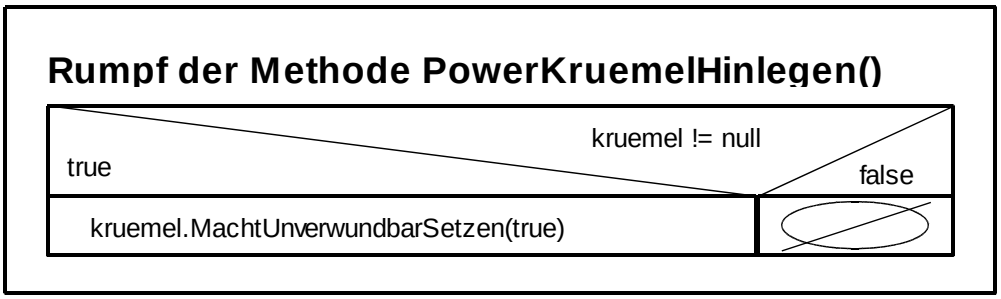
Aufgabe 14.7
In diesem Kapitel ist neu das Überladen von Methoden. Ergänze in deiner bisherigen Zusammenfassung eine Erklärung und Beispiele dazu.

siehe Anhang B

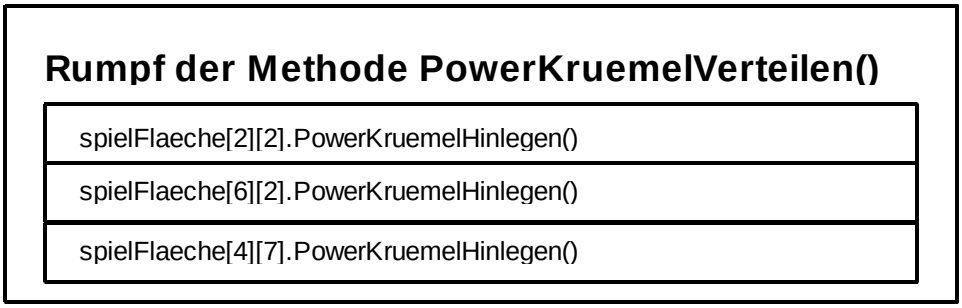
Anhang A: Tipps zu den Aufgaben

Tipps zu Aufgabe 14.2:

Die Methode *PowerKruemelHinlegen* in der Klasse ZELLE hat nur die Aufgabe die Methode MachtUnverwundbarSetzen ihres Krümels aufzurufen. Wichtig ist eine Überprüfung mit Hilfe einer bedingten Anweisung, ob ein Krümel existiert, um Laufzeitfehler zu vermeiden. Folgende Abbildung zeigt das Struktogramm:



Die Methode *PowerKruemelVerteilen* in der Klasse LABYRINTH hat nur die Aufgabe die Methode *PowerKruemelHinlegen* der betroffenen Zellen aufzurufen. Für das Beispiel aus Abbildung 1 unten sieht das Struktorgramm wie folgt aus:



Tipps zu Aufgabe 14.5:

Der neue Konstruktor hat folgende Form. Veränderungen gegenüber dem alten Konstruktor sind in blauer Farbe.

```
/**
 * Konstruktor der klasse LABYRINTH ohne Eingangsparameter:
 * Erzeugung eines Labyrinths mit den Standardwerten 10 f&uuml;r
 * die Breite und die H&ouml;he. Automatisch werden Mauern und
 * Powerkr&uuml;mel passen zum 10x10 Labyrinth erzeugt
 * (&uuml;ber den Aufruf der Methoden GaengeErstellen und PowerKruemelVerteilen)
 */
public LABYRINTH()
{
    breite = 10;
    hoehe = 10;

    // Spielflaeche erstellen
    spielFlaeche = new ZELLE[breite][hoehe];
    for(int zaehlerX = 0; zaehlerX <= breite-1; zaehlerX = zaehlerX+1)
    {
        for(int zaehlerY = 0; zaehlerY <= hoehe-1; zaehlerY = zaehlerY+1)
        {
            spielFlaeche[zaehlerX][zaehlerY] = new ZELLE(zaehlerX,zaehlerY);
        }
    }

     // Gänge durch Mauern erstellen und Powerkrümel verteilen
     GaengeErstellen();
      PowerKruemelVerteilen();
}
```


Anhang B: Schlüsselwort this und dessen Einsatzmöglichkeiten beim Konstruktor der Klasse LABYRINTH

Viele Programmierer verwenden bei den Setzen-Methoden bzw. beim Initialisieren von den Attributen im Konstruktor als Bezeichner für Eingangsparameter die Attributnamen selbst und nicht andere Namen. Beispielsweise lautet bei Ihnen der Methodenkopf des Konstruktors der Klasse LABYRINTH nicht

LABYRINTH(int breiteNeu, int hoeheNeu)

sondern

LABYRINTH(int breite, int hoehe)

Damit ergibt sich folgende Umformulierung des Konstruktors: (Die Stellen mit Veränderungen sind im Quelltext blau und fett ausgezeichnet.)

```
class LABYRINTH
{
    // Attribute
    int breite;
    int hoehe;

    // ....

    /**
    * Konstruktor der Klasse LAYBRINTH mit Eingangsparamtern
    * @param breite Breite des Labyrinths
    * @param hoehe H&ouml;he des Labyrinths
    */
    public LABYRINTH(int breite, int hoehe)
    {
        breite = breite;
        hoehe = hoehe;

        spielFlaeche = new ZELLE[breite][hoehe];
        for(int zaehlerX = 0; zaehlerX <= breite-1; zaehlerX = zaehlerX+1)
        {
            for(int zaehlerY = 0; zaehlerY <= hoehe-1; zaehlerY = zaehlerY+1)
            {
                spielFlaeche[zaehlerX][zaehlerY] = new ZELLE(zaehlerX,zaehlerY);
            }
        }
    }

    / ...
}
```

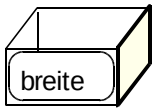
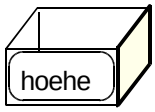
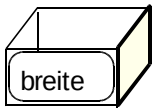
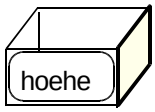
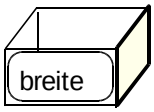
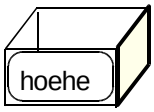
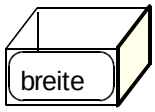
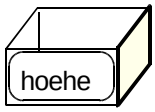
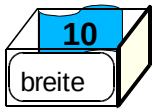
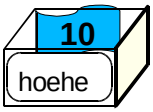
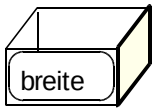
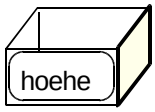
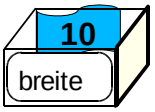
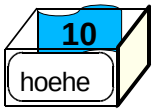
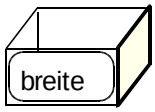
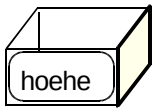
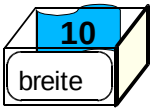
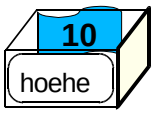
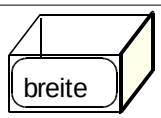
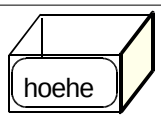
Deklaration der Attribute dieser Klasse

Deklaration von lokalen Attributen, die nur im Konstruktor gültig sind

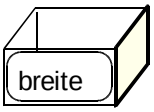
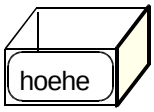
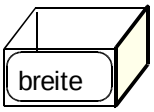
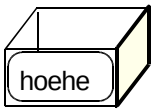
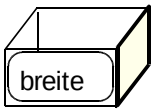
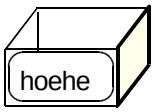
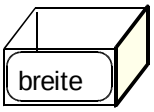
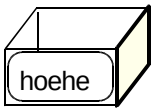
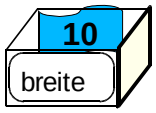
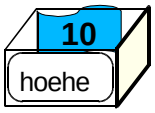
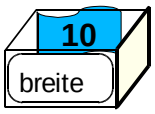
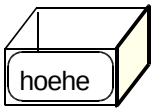
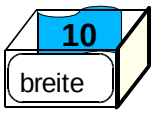
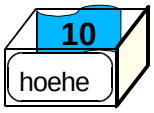
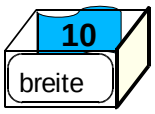
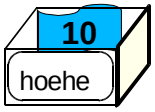
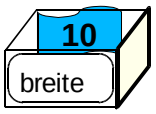
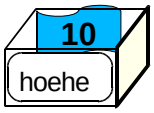
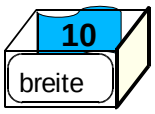
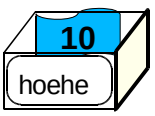
Fehler auf Grund von Namensgleichheit

Abbildung 4: Umformulierung des Konstruktors der Klasse LABYRINTH (fehlerhafte Ausführung!!)

Mit dieser Formulierung arbeitet das Programm jedoch nicht richtig.
Insgesamt werden in dem Quelltextauszug in Abbildung B.1 vier Attribute deklariert. Zwei davon sind (globale) Attribute, die so lange existieren, solange das Objekt existiert. Zwei davon sind lokale Attribute, die nur existieren, solange der Konstruktor ausgeführt wird und danach wieder gelöscht werden. Da jeweils zwei Attribute den gleichen Namen haben kommt, weiß das Programm nicht, welche Attribute nach Wunsch des Programmierers innerhalb des Konstruktors verwenden sollen. Die Abarbeitung erfolgt jedoch nach klaren Regeln: Innerhalb des Konstruktors werden die dort deklarierten Variablen verwendet. Somit ergibt sich folgender Ablauf Am Beispiel des Methodenaufrufs new LABYRINTH(10,10)

	(globales) Attribut breite	(globales) Attribut hoehe	lokales Attribut breite	lokales Attribut hoehe
Deklaration der (globalen) Attribute				
Deklaration der lokalen Attribute im Methdenkopf des Konstruktors				
Initialisierung der lokalen Attribute durch die Eingabewerte				
Zuweisung `breite = breite;`			 ausschließlicher Zugriff auf das lokale Attribut; keine Änderung: der Zettel in der Schachtel vorher und nachher haben den gleichen Wert	
Zuweisung `hoehe = hoehe;`				 ausschließlicher Zugriff auf das lokale Attribut; keine Änderung: der Zettel in der Schachtel vorher und nachher haben den gleichen Wert
Ende des Konstruktors			Gelöscht, da Gültigkeits-bereich zu Ende	Gelöscht, da Gültigkeits-bereich zu Ende
Ergebnis: Die globalen Attribute sind nicht initialisiert (bzw. erhalten den für int üblichen Standardinitialisieurngswert 0)				

Um nun deutlich zu machen, dass es sich in der Zuweisung `breite = breite` links um das globale Attribut handelt, muss das Schlüsselwort `this` ergänzt werden. This ist eine in Java zur Verfügung gestellte Referenzvariable, die auf das Objekt zeigt das gerade aktiv ist.

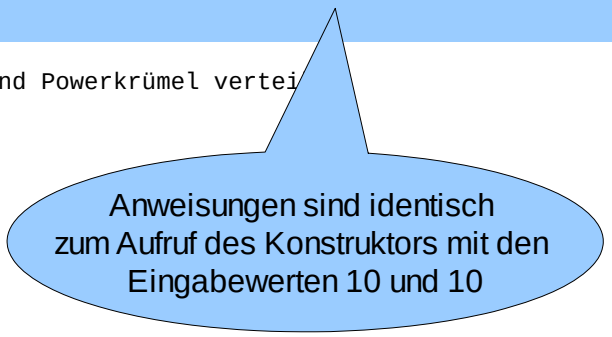
	(globales) Attribut breite auch erreichbar über this.breite	(globales) Attribut hoehe auch erreichbar über this.hoehe	lokales Attribut breite	lokales Attribut hoehe
Deklaration der (globalen) Attribute				
Deklaration der lokalen Attribute im Methdenkopf des Konstruktors				
Initialisierung der lokalen Attribute durch die Eingabewerte				
Zuweisung `this.breite = breite;`				
Zuweisung `this.hoehe = hoehe;`				
Ende des Konstruktors			Gelöscht, da Gültigkeits-bereich zu Ende	Gelöscht, da Gültigkeits-bereich zu Ende
Ergebnis: Die globalen Attribute sind korrekt initialisiert				

Das Schlüsselwort this lässt sich auch verwenden, um im neuen Konstruktor aus Aufgabe 14.5 redundanten Quelltext zu sparen. Die farbig hinterlegten Zeilen entsprechen dem Aufruf des anderen Konstruktors mit den Eingabewerten 10 und 10.

```
    * Konstruktor der Klasse LABYRINTH ohne Eingangsparameter:
    * Erzeugung eines Labyrinths mit den Standardwerten 10 f&uuml;r
    * die Breite und die H&ouml;he. Automatisch werden Mauern und
    * Powerkr&uuml;mel passen zum 10x10 Labyrinth erzeugt
    * (&uuml;ber den Aufruf der Methoden GaengeErstellen und PowerKruemelVerteilen)
    */
    public LABYRINTH()
    {
        breite = 10;
        hoehe = 10;

        // Spielflaeche erstellen
        spielFlaeche = new ZELLE[breite][hoehe];
        for(int zaehlerX = 0; zaehlerX <= breite-1; zaehlerX = zaehlerX+1)
        {
            for(int zaehlerY = 0; zaehlerY <= hoehe-1; zaehlerY = zaehlerY+1)
            {
                spielFlaeche[zaehlerX][zaehlerY] = new ZELLE(zaehlerX,zaehlerY);
            }
        }

        // Gänge durch Mauern erstellen und Powerkrümel verteilen
        GaengeErstellen();
        PowerKruemelVerteilen();
    }
```



Ein anderer Konstruktro der gleichen Klasse lässt sich mit dem Schlüsselwort this aufrufen. Damit kann der neue Konstruktor aus Aufgabe 14.5 wie folgt verkürzt werden.

```
    * Konstruktor der Klasse LABYRINTH ohne Eingangsparameter:
    * Erzeugung eines Labyrinths mit den Standardwerten 10 f&uuml;r
    * die Breite und die H&ouml;he. Automatisch werden Mauern und
    * Powerkr&uuml;mel passen zum 10x10 Labyrinth erzeugt
    * (&uuml;ber den Aufruf der Methoden GaengeErstellen und PowerKruemelVerteilen)
    */
    public LABYRINTH()
    {
        this(10,10);

        // Gänge durch Mauern erstellen und Powerkrümel verteilen
        GaengeErstellen();
        PowerKruemelVerteilen();
    }
```

