Kapitel 9 Das Labyrinth

Lernziel:
Wiederholung: eindimensionales Feld, Wiederholung mit fester Anzahl
neu: Feld mit Objektreferenzen, zweidimensionales Feld

9.1 Felder

Zu Beginn des letzten Kapitels war es Aufgabe, die Klasse LABYRINTH zu modellieren und dabei auf die Beziehungen zu anderen Klasse zu achten. Eine mögliche Lösung zeigt Abbildung 1.

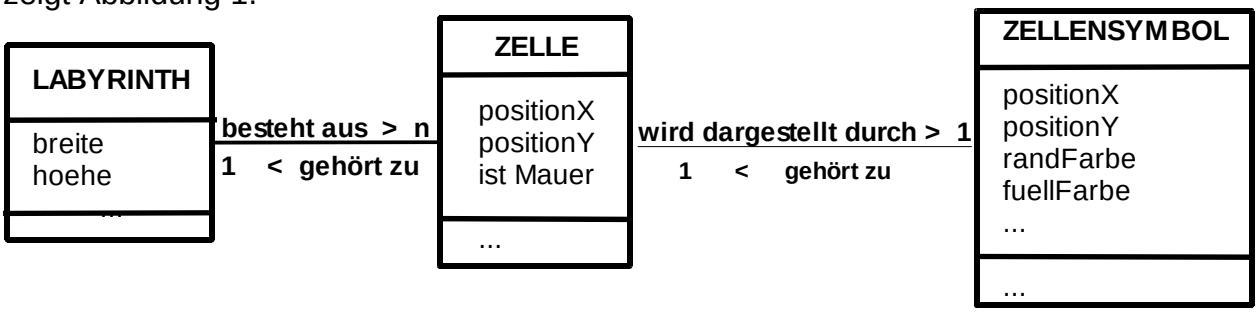


Abbildung 1:
Klassendiagramm mit Beziehungen zwischen den Klassen LABYRINTH, ZELLE und ZELLENSYMBOL

Die Beziehung zwischen Objekten der Klasse ZELLE und ZELLENSYMBOL wurde bereits in Kapitel 8 umgesetzt. Im Folgenden geht es nun um die Beziehung zwischen Objekten der Klasse LABYRINTH und ZELLE

Aufgabe 9.1
a) Wie werden Beziehungen in Java umgesetzt?
b) Wie sieht das erweiterte Klassendiagramm der Klasse ZELLE unter Berücksichtigung der Beziehung zur Klasse ZELLENSYMBOL aus?

c) Überlege dir, wie die Beziehung zwischen LABYRINTH und ZELLE in Abbildung 1 effizient[1] umgesetzt werden kann. Es ist durchaus sinnvoll, die Überlegungen zunächst auf eine Zeile des Labyrinths zu beschränken, und erst in einem zweiten Schritt zu erweitern. Denke insbesondere an das in Kapitel 6 vorgestellte Konzept zur Verwaltung von Informationen gleichen Typs.
Ein erweitertes Klassendiagramm wäre eine sinnvolle Zusammenfassung der Überlegungen dieser Teilaufgabe.

1 mit möglichst wenig Aufwand

Wenn das Labyrinth aus 100 Zellen besteht, bedeutet dies, dass 100 Referenzattribute in der Klasse LABYRINTH deklariert und initialisiert werden müssen. Eine sehr aufwändige Möglichkeit, dies zu tun, zeigt der Quelltextauszug in Abbildung 2.

```
class LABYRINTH
{
      ZELLE zelle1;
      ZELLE zelle2;
      ZELLE zelle3;
      ZELLE zelle4;
      ...
      ZELLE zelle100;

      LABYRINTH()
      {
            zelle1 = new ZELLE(0, 1);
            zelle2 = new ZELLE(0, 2);
            zelle3 = new ZELLE(0, 3);
            zelle4 = new ZELLE(0, 4);
            ...
            zelle100 = new ZELLE(9, 9);
      }
}
```

Deklaration

Initialisierung

Abbildung 2: Aufwändiger Quelltextauszug für eine Klasse LABYRINTH

Viel kürzer und damit auch übersichtlicher wird der Quelltext, wenn man das Konzept des Feldes (engl. **array**) aus Kapitel 6 nützt. In diesem Fall muss ein Feld deklariert werden, bei dem jedes Feldelement ein Zellobjekt referenziert. Als Namen für das Array wird hier im Skript spielFlaeche verwendet:

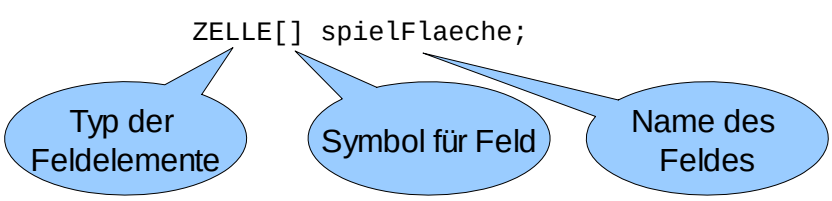


Abbildung 3: Deklaration eines Feldes am Beispiel der spielFlaeche

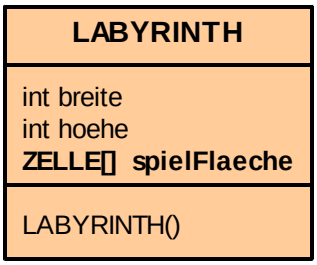


Abbildung 4: Erweitertes Klassendiagramm mit der Deklaration des Feldes spielFlaeche als Umsetzung der Beziehung aus Abbildung 1

Hinweis:
Das erweiterte Klassendiagramm enthält die Deklaration des Feldes spielFlaeche. (Abbildung 4).

Aufgabe 9.2
Welche Anweisungen bzw. Zuweisungen müssen im Konstruktor *LABYRINTH()* ausgeführt werden, um die Feldelemente zu initialisieren? Wie lassen sich diese Anweisungen sehr effizient formulieren? (Beschränke dich zunächst auf die Zellobjekte der ersten Zeile!)

Die eckigen Klammern geben an, dass das Feld spielFlaeche mehrere Objekte der Klasse ZELLE referenziert. Aber wie viele genau? Dies wird bei der Erzeugung des Feldes im Konstruktor festgelegt. Wenn man sich zunächst auf die erste Zeile des Labyrinths beschränkt, so wird ein Feld der Länge 10 benötigt. Die Anweisung dazu lautet

```
new ZELLE[10];
```

Das erzeugte Feld muss dem in Abbildung 3 deklarierten Feld zugewiesen werden. Die Erzeugung des Feldes und Zuweisung wird in Java in einer Zeile realisiert:

```
spielFlaeche = new ZELLE[10];
```

Aufgabe 9.3

a) Schreibe eine Klasse LABYRINTH entsprechend dem Klassendiagramm aus Abbildung 1. Die Attribute breite und hoehe sollen auf 10 initialisiert werden. Die Umsetzung der Beziehung soll durch ein Feld spielFlaeche mit 10 Feldelementen erfolgen.
(Die oben angegebenen Quelltextbestandteile kannst du als Hilfe verwenden.)

b) Erzeuge ein Objekt der Klasse LABYRINTH und betrachte das Feld spielFlaeche im Objektinspektor. Beantworte folgende beiden Fragen:
- Ab welcher Nummer wird gezählt, d.h. welchen Wert hat der kleinste Index?
- Enthält das Feld spielFlaeche Referenzen auf Objekte der Klasse ZELLE, d.h. kannst du dir im Objektinspektor Objekte der Klasse ZELLE anzeigen lassen?

Abbildung 5 zeigt den Objektinspektor des Feldes. Das Feld hat ein Attribut length, dessen Wert 10 die Anzahl der referenzierbaren Zellen angibt. Jedoch ist kein einziges Objekt der Klasse ZELLE referenziert, der Eintrag lautet jeweils „null“ für „kein Wert“. Im Bild des Aktenschranks wurde der Aktenschrank mit 10 Schubladen zwar gebaut, aber alle Schubladen sind noch leer.
Abbildung 6 zeigt die gleiche Situation im Objektdiagramm.

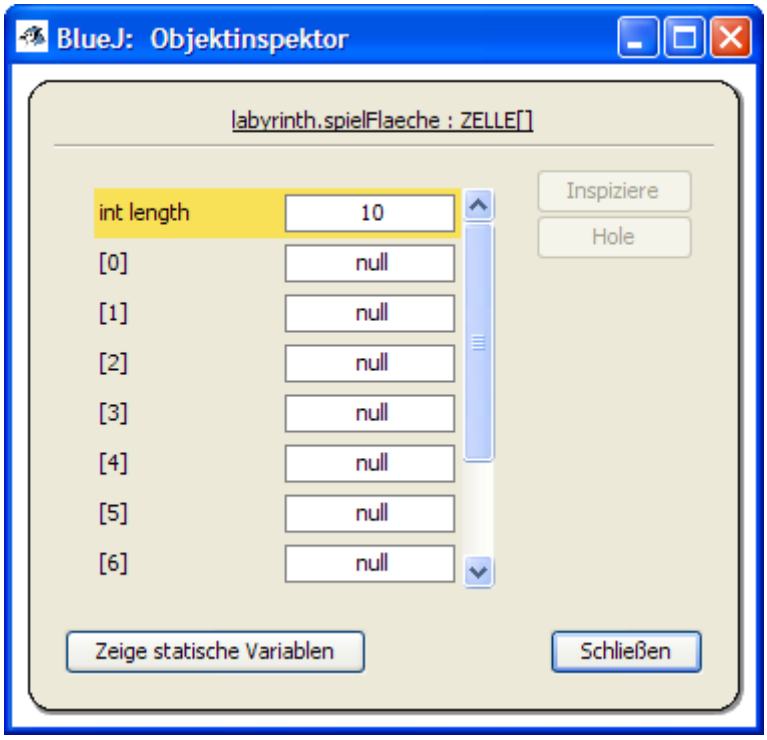


Abbildung 5: Objektinspektor des Felds spielFlaeche : Bisher wird kein Objekt der Klasse Zelle referenziert.

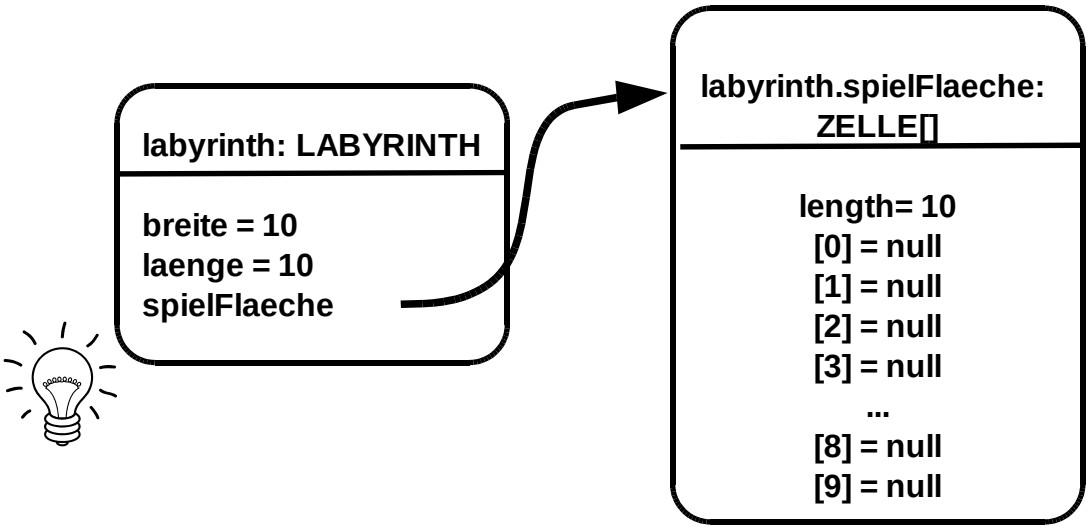


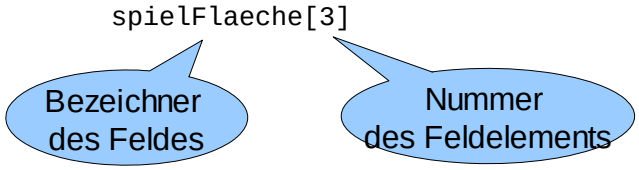
Abbildung 6: - Objektdiagramm passend zu der Situation im Objektinspektor: Die Feldelemente des Objekts spielFlaeche referenzieren keine Zell-Objekte.

Aufgabe 9.4
Warum existieren bisher keine Objekte der Klasse ZELLE?

Um den „leeren“ Feldelementen Objekte zuordnen zu können, müssen einerseits die Objekte der Klasse ZELLE erzeugt werden und dann passend den Feldelementen („Schubladen“) zugewiesen werden.
Objekte der Klassse ZELLE werden wie gewohnt mit dem new-Operator erzeugt, beispielsweise

```
new ZELLE(0,3)
```

Um die Objektreferenz einem konkreten Feldelement zuweisen zu können, wird der entsprechende Index in eckigen Klammern angegeben.



Somit müssen die Zuweisungen im Quelltext wie folgt geschrieben werden:

```
spielFlaeche[0] = new ZELLE(0,0);
spielFlaeche[1] = new ZELLE(1,0);
spielFlaeche[2] = new ZELLE(2,0);
spielFlaeche[3] = new ZELLE(3,0);
...
```

Abbildung 7: Quelltextauszug der Initialisierung der Feldelemente des Feldes spielFlaeche

Mit Hilfe der Wiederholung mit fester Anzahl lässt sich der Quelltext der Initialisierung aus Abbildung 7 deutlich verkürzen:

```
for(int zaehlerX = 0; zaehlerX < 10 ; zaehlerX = zaehlerX+1)
{
   spielFlaeche[zaehlerX] = new ZELLE(zaehlerX,0);
}
```

Abbildung 8: Initialisierung der Feldelemente mit Hilfe der Wiederholung mit fester Anzahl

Um an späterer Stelle auch ein Labyrinth mit anderen Maßen als 10 x 10 zu erstellen, gibt es die Attribute breite und hoehe. Da sich alle Abbildungen in den Kapiteln 9.1 und 9.2 auf diese Maße beziehen, wird in einer ersten Version des Konstruktors der Klasse MAMPFI die Breite und Höhe auf 10 festgelegt (Abbildung 9). Eine Verallgemeinerung wird in Kapitel 9.3 erklärt.

```
LABYRINTH()
{
     breite = 10;
     hoehe = 10;

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

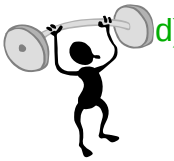
Abbildung 9: Erste Version des Konstruktors *LABYRINTH*

Aufgabe 9.5

a) Ergänze die Klasse LABYRINTH entsprechend den Erklärungen so, dass alle Zellen der ersten Zeile automatisch erzeugt und dem spielFlaeche zugeordnet werden.

b) Erzeuge ein Objekt der Klasse LABYRINTH und untersuche im Objektinspektor, ob es die Anforderungen erfüllt.

c) Gehe gedanklich nochmal den Konstruktor aus Abbildung 8 durch und erkläre jede Zeile des Quelltexts.



d) Falls du dich sicher fühlst kannst du gerne schon an dieser Stelle, den Konstruktor so allgemein formulieren, dass verschiedene Breiten des Labyrinths und damit auch der ersten Zeile möglich sind. (Darstellbar sind aktuell maximal 10 Zellen pro Zeile.)

Durch mehrfaches Öffnen verschiedener Objektinspektoren ergibt sich eine Beziehung zwischen den in Abbildung 10 dargestellten Objekten.

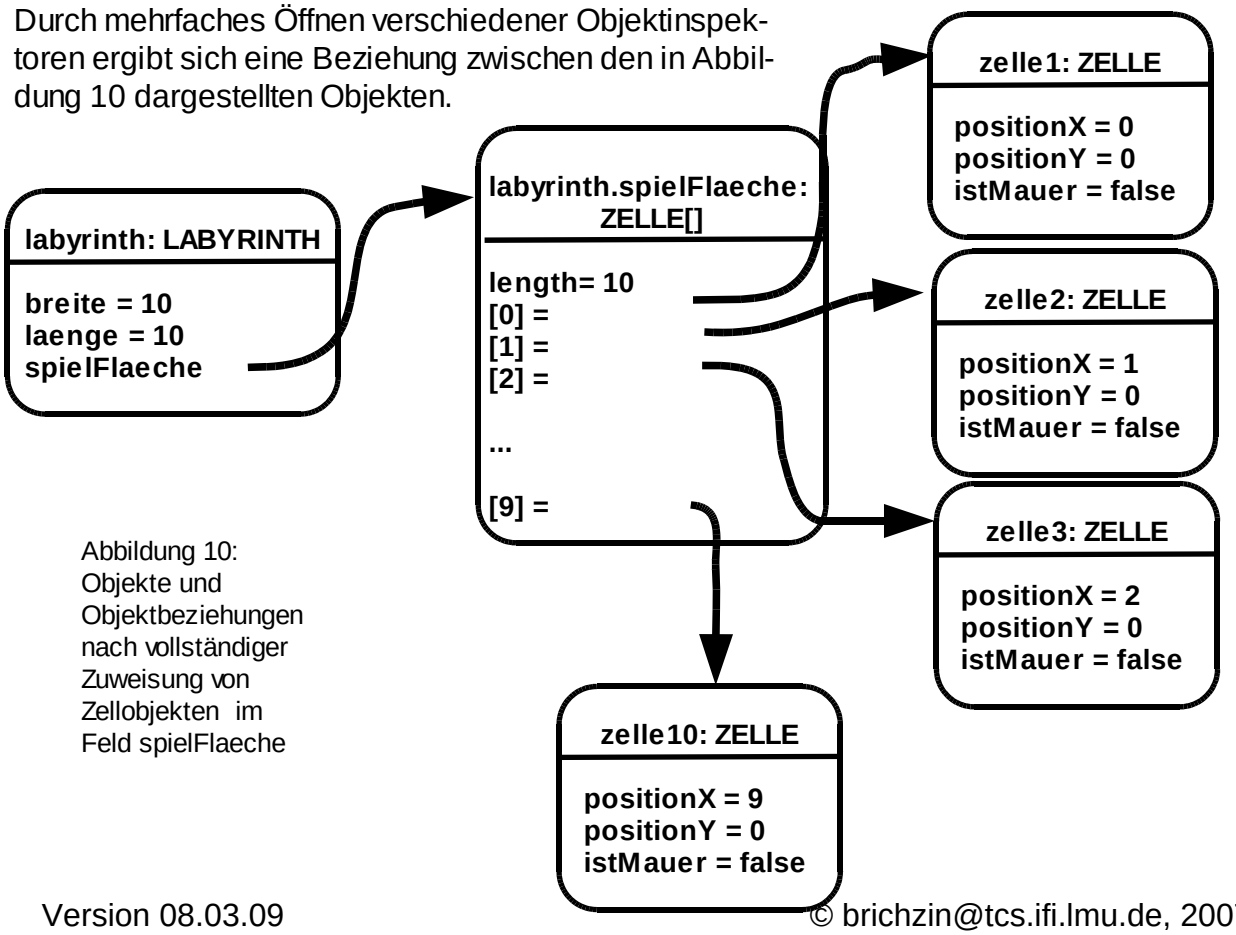


Abbildung 10: Objekte und Objektbeziehungen nach vollständiger Zuweisung von Zellobjekten im Feld spielFlaeche

Hinweis:
Ein referenziertes Objekt hat nicht den gleichen Namen wie das Referenzattribut. Der Objektname wird intern durch das System (Java Virtual Maschine) festgelegt und ist für den Nutzer unbekannt. Möchte man Objektreferenzen wie in Abbildung 10 veranschaulichen, muss man allen Objekten einen Namen geben. Da die echten Bezeichner wie gesagt unbekannt sind, hat der Autor die Freiheit selbst einen Namen zu vergeben.

9.2 Zweidimensionale Felder

Bisher wurde über die Klasse LABYRINTH nur die erste Zeile des Spielfelds eingebunden. Insgesamt hat das Spielfeld in der Abbildung rechts 10 mal 10 = 100 Felder. So könnte man die Lösung aus Kapitel 9.1 erweitern, indem man die Länge des Feldes auf 100 erhöht. Man müsste sich eine geschickte Nummerierung überlegen, so dass es nicht zu schwierig ist herauszufinden, welche x- und y-Position beispielsweise das Feldelement mit dem Index 43 hat.

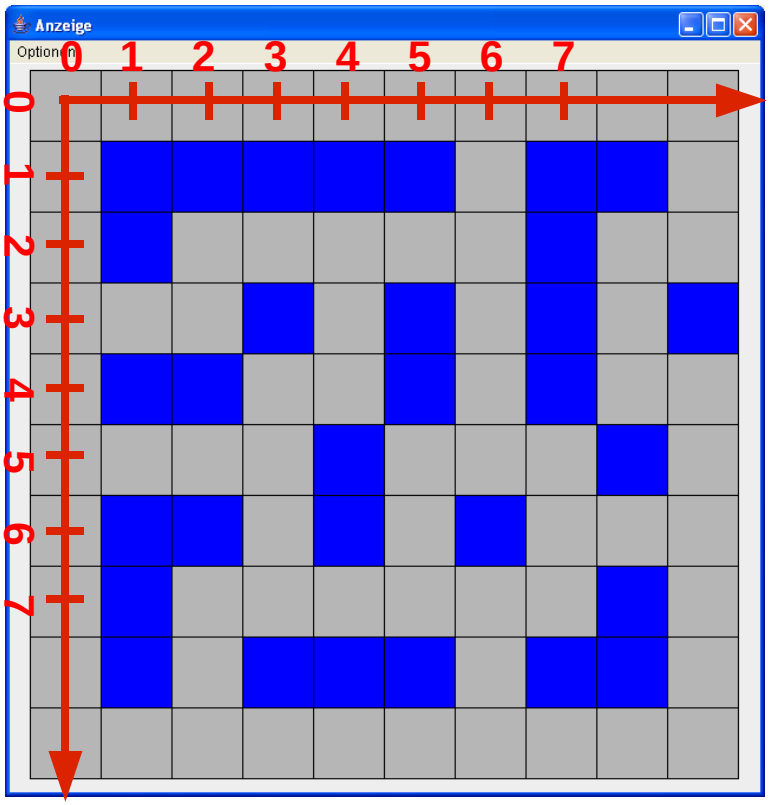


Abbildung 11:
Koordinatensystem für das Spielfeld von Krümel & Monster

Als alternative Lösung bietet sich in diesem Fall jedoch an, dass man ein zweidimensionales Feld verwendet, weil dies genau zum Spielfeld passt. Auch das Spielfeld hat zwei Dimensionen, eine in x-Richtung und eine in y Richtung (Abbildung 10).
Die Deklaration eines zweidimensionalen Feldes sieht in Java folgendermaßen aus:

```
ZELLE[][] spielFlaeche;
```

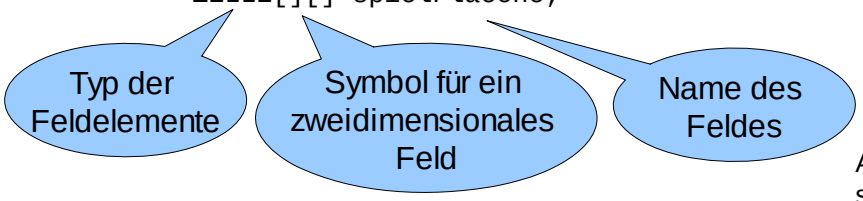


Abbildung 12: Deklaration eines Feldes am Beispiel der spielFlaeche

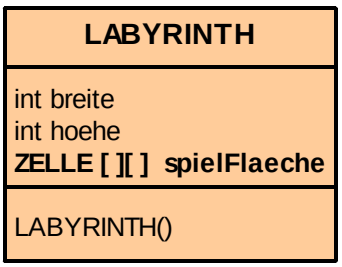


Abbildung 13: erweitertes Klassendiagramm mit dem zweidimensionalen Feld spielFlaeche

Ein Vergleich mit Abbildung 3 zeigt, dass der einzige Unterschied ein zweites Paar von eckigen Klammern ist. Dies legt fest, dass das Feld nicht ein- sondern zweidimensional ist.

Aufgabe 9.6
Überlege dir, wie ein zweidimensionales Feld im im Modell des Aktenschranks aussehen könnte.

Um nun ein zweidimensionales Feld zu erzeugen, benötigt man wieder den new-Operator. Einzige Änderung ist, dass eine Länge für beide Dimensionen angegeben werden muss. Für das Spielfeld von Mampfi, soll die Länge in beiden Dimensionen 10 sein. Der folgende Quelltext zeigt die Initialisierung des Feldes spielFlaeche.

```
spielFlaeche = new ZELLE[10][10];
```

Abbildung 14:
Erzeugung eines zweidimensionales Feldes und Zuweisung zum Referenzattribut spielFlaeche

Wie schon in Kapitel 9.1 wurde mit dieser Zeile nur der Aktenschrank erzeugt, aber noch keine einzige Schublade mit einem Inhalt gefüllt. Es müssen nun verschiedene Objekte der Klasse ZELLE erzeugt und den Feldelementen (Schubladen) zugewiesen werden.
Dabei hat im Schrankmodell der Aktenschrank nicht nur eine Reihe von Schubladen hat, sondern mehrere (Abbildung 15). Und jede Schublade ist eindeutig durch zwei Nummern identifizierbar.

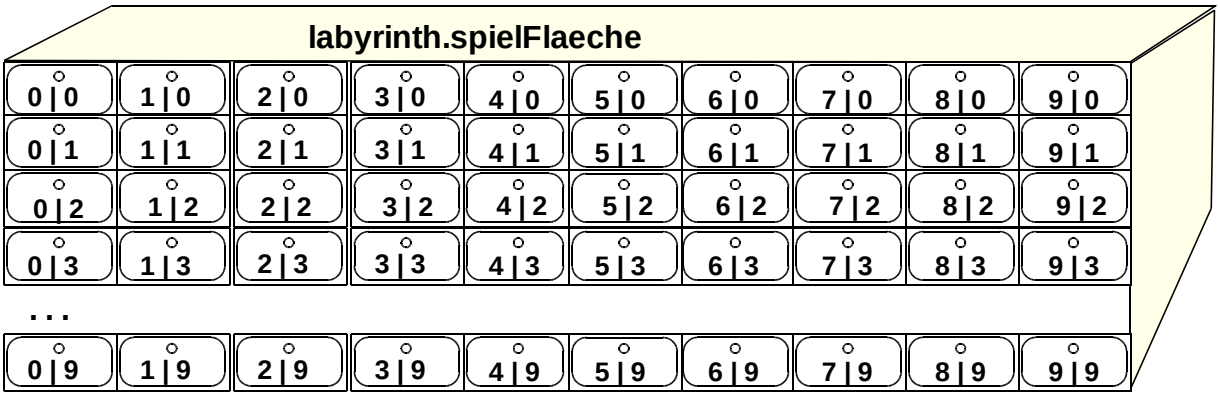


Abbildung 15: Schrankmodell für ein zweidimensionales Feld

Um die "Schubladen" des in Abbildung 14 erzeugten Feldes mit Referenzen auf Zellen zu füllen, sind folgende Zuweisungen nötig:

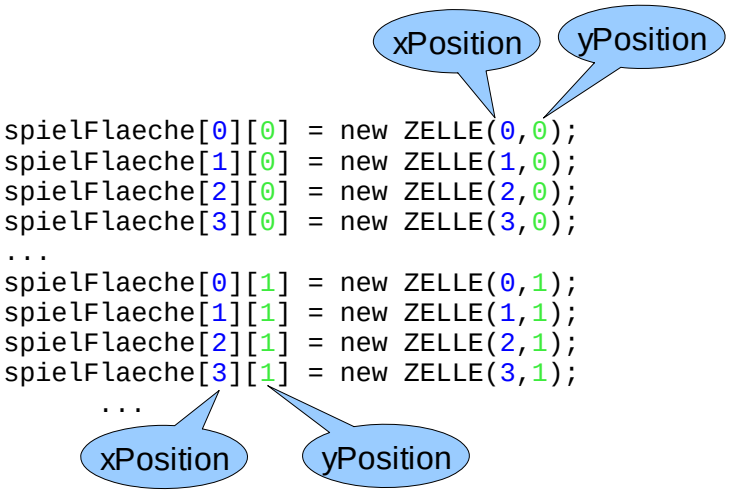


Abbildung 16: Quelltextauszug der Initialisierung der Feldelemente des Feldes zellFeld

Hinweis:
Beachte, dass wir entsprechend unseren Erfahrungen bei der Angabe von Punkten im Koordinatensystem bzw. bei der bisherigen Programmierung der Klasse ZELLE nun bei dem zweidimensionalen Feld spielFlaeche die erste Dimension für die positionX und die zweite Dimension für die positionY wählen.

Aufgabe 9.7
Wie lautet der Quelltext für die Erzeugung der Zellen der 5. Zeile und die entsprechende Zuweisung für das Array spielFlaeche?

Aufgabe 9.8
Wie könnte man den Quelltext aus Abbildung 16 viel kürzer formulieren?

Selbstverständlich notieren wir nicht wie in Abbildung 16 angedeutet 100 Zeilen, sondern nutzen wieder die Wiederholung mit fester Anzahl, um den Schreibaufwand zu reduzieren.

Aufgabe 9.9
Versuche einen Quelltext zu formulieren, der Wiederholungsanweisungen mit fester Anzahl nutzt. Welche Ergänzungen sind im Vergleich zu Kapitel 9.1 nötig? Gerne kannst du deinen Versuch schon am Rechner umsetzen. Solltest du Hilfe benötigen, findest du sie auf der nächsten Seite.

Da das Feld nun zweidimensional ist, werden auch zwei Wiederholungen mit fester Anzahl benötigt. Diese müssen ineinander geschachtelt sein. Der folgende Quelltext initialisiert die Feldelemente des Arrays spielFlaeche:

```
for(int zaehlerY = 0; zaehlerY < 10; zaehlerY = zaehlerY+1)
{
	for(int zaehlerX = 0; zaehlerX < 10; zaehlerX = zaehlerX+1)
	{
		spielFlaeche[zaehlerX][zaehlerY] = new ZELLE(zaehlerX,zaehlerY);
	}
}
```

Den Ablauf der geschachtelten Wiederholungen mit fester Anzahl veranschaulicht folgende Tabelle :

zaehlerX	zaehlerY	
	0	// Die Zählvariable zaehlerY der äußeren Wiederholung wird initialisiert.
0	0	// Die Zählvariable zaehlerX der inneren Wiederholung wird initialisiert.
0	0	`spielFlaeche[0][0] = new ZELLE(0,0);`
1	0	`spielFlaeche[1][0] = new ZELLE(1,0);`
2	0	`spielFlaeche[2][0] = new ZELLE(2,0);`
3	0	`spielFlaeche[3][0] = new ZELLE(3,0);`
...	0	
9	0	`spielFlaeche[9][0] = new ZELLE(9,0);`
	0	// Die Zählvariable zaehlerX der inneren Wiederholung hat mit dem Wert 9 die // obere Grenze erreicht, die innere Wiederholungsanweisung wird damit // beendet. Es wurden alle Zellen der ersten Zeile (y-Koordinate hat den Wert 0) // erzeugt und die Referenzen den Feldelementen des Arrays spielFlaeche // passend zugewiesen.
	1	// Die Zählvariable zaehlerY der äußeren Wiederholung wird um eins erhöht
0	1	// Die Zählvariable zaehlerX der inneren Wiederholung wird initialisiert.
0	1	`spielFlaeche[0][1] = new ZELLE(0,1);`
1	1	`spielFlaeche[1][1] = new ZELLE(1,1);`
2	1	`spielFlaeche[2][1] = new ZELLE(2,1);`
3	1	`spielFlaeche[3][1] = new ZELLE(3,1);`
...	1	
9	1	`spielFlaeche[9][1] = new ZELLE(9,1);`
	1	// Die Zählvariable zaehlerX der inneren Wiederholung hat mit dem Wert 9 die // obere Grenze erreicht, die innere Wiederholungsanweisung wird damit // beendet. Es wurden alle Zellen der zweiten Zeile (y-Koordinate hat den Wert // 1) erzeugt und die Referenzen den Feldelementen des Arrays spielFlaeche // passend zugewiesen.
	2	// Die Zählvariable zaehlerY der äußeren Wiederholung wird um eins erhöht
0	2	// Die Zählvariable zaehlerX der inneren Wiederholung wird initialisiert.
0	2	`spielFlaeche[0][2] = new ZELLE(0,2);`
1	2	`spielFlaeche[1][2] = new ZELLE(1,2);`
2	2	`spielFlaeche[2][2] = new ZELLE(2,2);`
3	2	`spielFlaeche[3][2] = new ZELLE(3,2);`
...	...	...

Abbildung 17: Ablauf der geschachtelten gezählten Wiederholungen

Aufgabe 9.10
Wie lautet die letzte Zeile des Ablaufes aus Abbildung 17, wenn das Feld mit der Größe entsprechend Abbildung 14 erzeugt wurde?

Aufgabe 9.11

a) Ergänze die Klasse LABYRINTH (soweit noch nicht in Aufgabe 9.9 geschehen) entsprechend den Erklärungen so, dass alle 100 Zellen durch geschachtelte Wiederholungsanweisungen erzeugt und dem Feld spielFlaeche zugeordnet werden.
b) Erzeuge ein Objekt der Klasse LABYRINTH. Überprüfe über den Objektinspektor bzw. die Anzeige, ob alle Zellen vorhanden sind. Falls nicht, kann dir in der Fehleranalyse das Menü Optionen --> Zelldaten im Anzeigefenster helfen. Ist diese Option ausgewählt, werden die Koordinaten von allen Zellen angezeigt.
c) Bisher gibt es keine Zelle, die eine Mauer ist, d.h. auf der Spielfläche gibt es keine Gänge. Ändere dies, indem du in der Klasse LABYRINTH eine Methode *GaengeErstellen* schreibst, in der von Zellen deiner Wahl die Methode *IstMauerSetzen* mit dem Eingabewert `true` aufgerufen wird. Teste diese Methode.

   Beispiel:
   Der Methodenaufruf

   ```
   spielFlaeche[3][4].IstMauerSetzen(true);
   ```

   setzt in der Zelle mit den Koordinaten (3/4) den Wert des Attributs istMauer auf true. Bei richtiger Programmierung der Klasse ZELLE müsste dann im zugehörigen Zellensymbol die Füllfarbe sichtbar werden.
d) Zeichne das Sequenzdiagramm zu dem Methodenaufruf unter c).

9.3 Unterschiedliche Größen des Labyrinths

Um an ein Labyrinth mit anderen Maßen als 10 x 10 zu erstellen, gibt es die Attribute breite und hoehe.

Aufgabe 9.12
Wie muss sich der Methodenkopf und -rumpf des Konstruktors ändern, damit du beim Erzeugen des Labyrinths die Breite und Höhe (ohne Eingriffe in den Quelltext) festlegen kannst.

Über Eingangsparameter beim Konstruktor lassen sich die Breite und Höhe des Labyrinths festlegen. Der Methodenkopf muss entsprechend dem erweiterten Klassendiagramm in Abbildung 18 geändert werden.

Im Methodenrumpf des Konstruktors muss man die Eingabewerte zunächst den Attributen zuweisen und dann die weiteren konkreten Werte 10 durch die Platzhalter (Attributbezeichner) breite bzw. hoehe ersetzen.

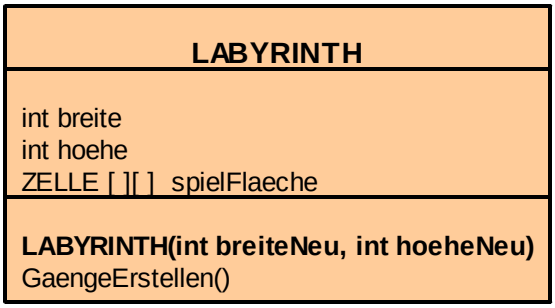


Abbildung 18: Ergänzung von Eingangsparametern im Konstruktor

Aufgabe 9.13
Ändere den Konstruktor der Klasse LABYRINTH entsprechend den angestellten Vorüberlegungen.
Teste die optimierte Klasse LABYRINTH, indem du beispielsweis ein Labyrinth mit der Breite 8 und Höhe 5 erzeugst.

Hinweise:
- Achte darauf, dass bei einer Breite und Höhe des Labyrinths von 10 der Index der Felder bei 0 beginnt und bei 9 (und nicht 10) endet! Der maximale Index ist somit eins weniger als die Breite bzw. Höhe.
- Aktuell können maximal 10 Zeilen bzw. 10 Spalten dargestellt werden.
- Solltest du eine weitere Hilfe benötigen, findest du ein Struktogramm im Anhang.

9.4 Zusammenfassung

Aufgabe 9.14
a) Fasse die wesentlichen Inhalte dieses Kapitels in deinem Heft zusammen. Der neue Fachbegriff in diesem Kapitel ist zweidimensionales Feld.
Sinnvoll ist es auch, sich die Schritte zu aufzuschreiben, die bei der Deklaration und Initialisierung eines Feldes von Referenzattributen ausgeführt werden müssen
b) Im Lehrtext oder/und bei der praktischen Umsetzung kommen auch folgende Begriffe aus den früheren Kapiteln vor: Feld, Index, Feldelement, Referenzattribut, Konstruktor, Zuweisung, new-Operator, Sequenzdiagramm. Solltest du dich bei dem ein oder anderen Begriff nicht sicher fühlen, oder etwas Neues verstanden haben, dann ergänze in deinen Aufzeichnungen.

Aufgabe 9.15
Erstelle eine Methode GaengeErstellen2, die ein alternatives Spielfeld mit anderen Mauerpositionen als in Aufgabe 9.10 herstellt. Versuche an der ein oder anderen Stelle die Mauern durch eine Wiederholungsanweisung zu erzeugen, um den Quelltext zu reduzieren. (Ob das wirklich geht, hängt aber von der Position deiner Mauern ab.)

Anhang: Tipps zu den Aufgaben

Tipps zu Aufgabe 9.13

LABYRINTH(int breiteNeu, int hoeheNeu)

```
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)
    }
}
```