Kapitel 8 Vorbereitungen für ein Labyrinth

Lernziele:

Wiederholung zentraler Inhalte der Kapitel 1-5 an einem neuen Beispiel

8.1 Typischer Spielfeldaufbau vieler Brettspiele

Ein weiterer wichtiger Schritt für die Realisierung unseres Spiels ist es, ein Labyrinth wie in der Abbildung 1 zu modellieren und zu programmieren. Dort können dann Krümel verstreut werden und Mampfi und die Monster haben die Möglichkeit sich zu bewegen.

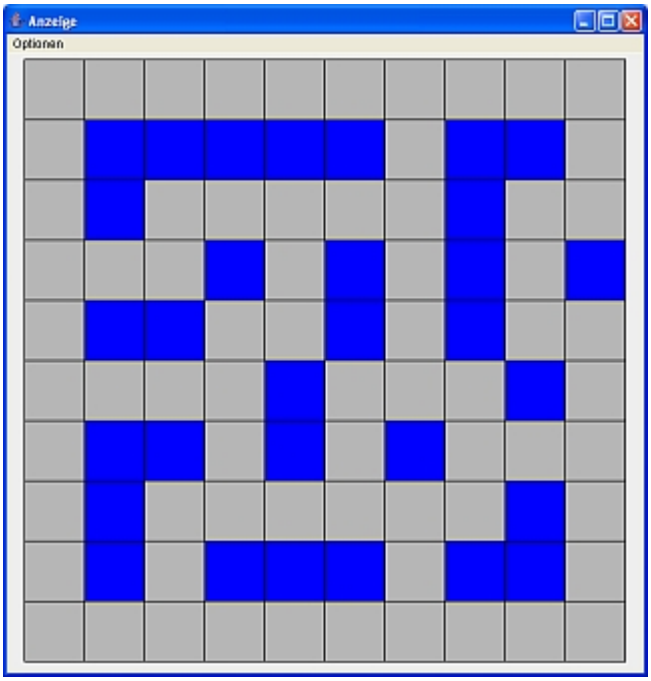


Abbildung 1: Ein Labyrinth

Aufgabe 8.1

a) Wie ist das Spielfeld von Schach bzw. Dame aufgebaut? Welche Gemeinsamkeiten bzw. Unterschiede bestehen zu dem Labyrinth in Abbildung 1?

b) Entwirf ein Klassendiagramm, das das LABYRINTH in Abbildung 1 modelliert. Pro Klasse reicht es, zwei bis drei wesentliche Attribute anzugeben. Methoden werden zunächst noch nicht betrachtet. Beschrifte alle Beziehungen!

c) Ergänze das Klassendiagramm aus b) um eine weitere Klasse, die für eine graphische Darstellung sorgt (ähnlich zur Klasse KREISSYMBOL bzw. PUNKTELISTEANZEIGE).

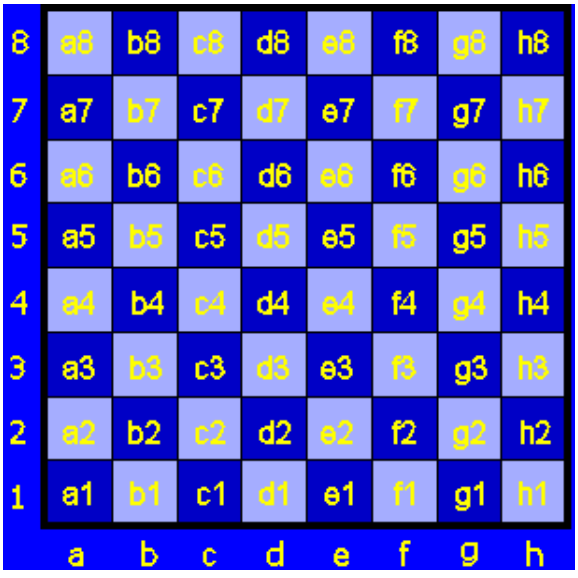


Abbildung 2:
Bezeichnungen beim Schachbrett

Viele Spielfelder sind aus einzelnen Zellen zusammengesetzt. Bei quadratischen Zellen eignen sich Koordinaten, um die Zellen eindeutig zu beschreiben. Ein sehr bekanntes Beispiel ist das Schachbrett, bei dem die Zeilen durch Nummern und die Spalten durch Buchstaben benannt sind. Eine Kombination aus einer Zahl und einem Buchstaben, d. h. eine Kombination der beiden Koordinaten legt eine Zelle eindeutig fest (Abbildung 2).

In gleicher Art und Weise ist es sinnvoll, das Spielfeld für Krümel & Monster aufzubauen. Im Gegensatz zu Schach ist es nützlich, für beide Koordinaten den gleichen Datentyp, ganze Zahlen, zu verwenden. Wie bei Computergraphiken üblich befindet sich der Koordinatenursprung links oben und die y-Achse zeigt nach unten (Abbildung 3).

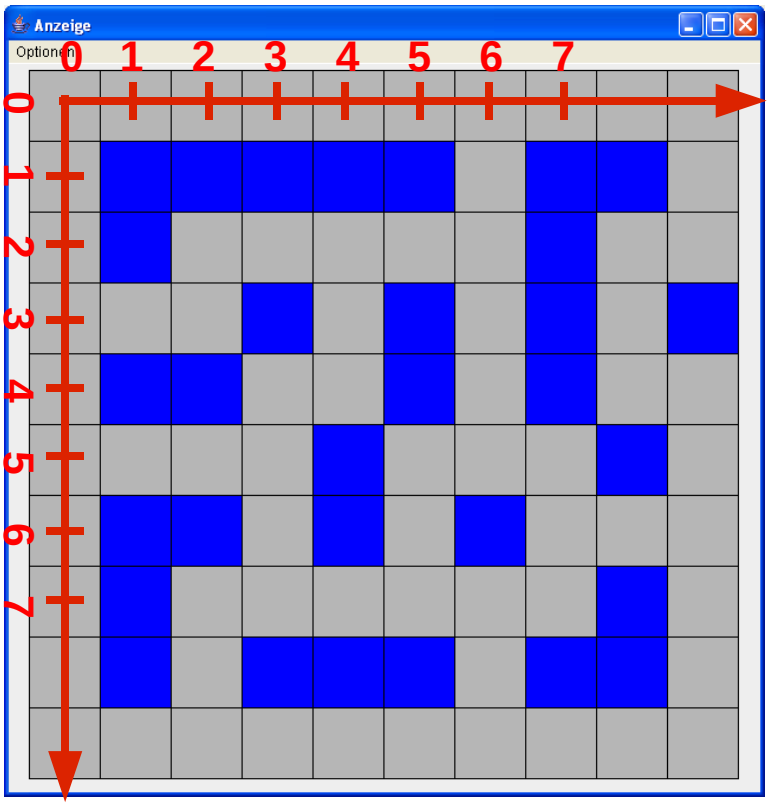


Abbildung 3:
Koordinatensystem für das Spielfeld von Krümel & Monster

Aufgabe 8.2
Schreibe eine Klasse ZELLE in Java. Beschränke dich zunächst auf drei Attribute. Beziehungen sollen noch nicht umgesetzt werden.

Hinweise:
- Vermutlich muss es nicht mehr erwähnt werden, aber weil es so wichtig ist, sei nochmal gesagt: Selbstverständlich sollte vor der Umsetzung in Java ein erweitertes Klassendiagramm zur Planung erstellt werden und nach der Umsetzung die Klasse getestet werden.
- Als Hilfe bei Schwierigkeiten bzw. zum Überprüfen deiner eigenen Lösung findest du im Anhang Tipps.

8.2 Die Klasse ZELLENSYMBOL

Um Objekte der Klasse ZELLE in dem Zeichenfenster darstellen zu können, benötigst du Objekte einer „graphischen“ Klasse. Dir wird dafür eine Klasse ZELLENSYMBOL zur Verfügung gestellt. Den Zusammenhang veranschaulicht das Klassendiagramm in Abbildung 4:

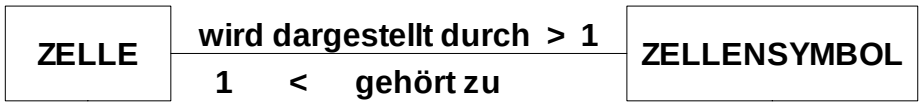
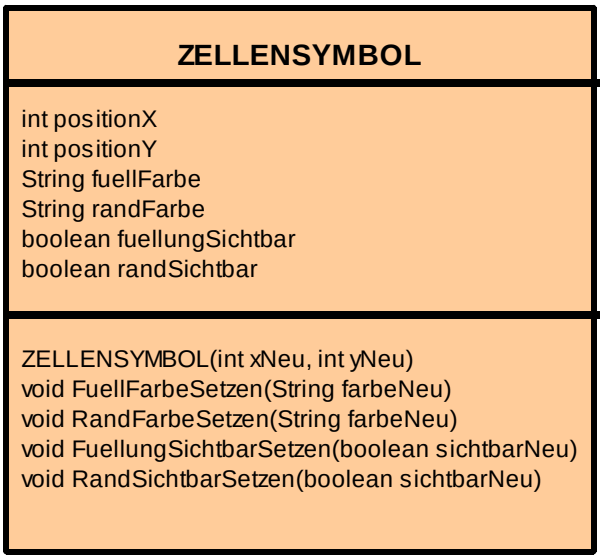


Abbildung 4:
Jedes Objekt der Klasse ZELLE wird genau durch ein Objekt der Klasse ZELLENSYMBOL dargestellt.



Wie schon bei Objekten der Klasse KREISFORM und PUNKTELISTE-ANZEIGE ist es wichtig, die Schnittstelle der Objekte der Klasse ZELLENSYMBOL zu kennen, um sie zielgerichtet einsetzen zu können. Einen Überblick über die Attribute und Methoden gibt das erweiterte Klassendiagramm links. An dieser Klasse soll nichts verändert werden, sondern es sollen nur die Methoden als Schnittstelle verwendet werden.

Abbildung 5: Attribute und Methoden der Klasse ZELLENSYMBOL – wichtige Informationen, um Objekte dieser Klasse zur Darstellung von Zellobjekten nutzen zu können

Bewusst heißt die Klasse ZELLEN**SYMBOL** und nicht ZELLEN**FORM** oder RECHTECKS**FORM**. Sie kann mehr als nur ein Rechteck in der Anzeige zeichnen. Beispielsweise ist mit Hilfe des Backends (siehe Kapitel 7.0) möglich, über das Menü „Optionen“ die Koordinaten der Zelle anzuzeigen (Abbildung 6).

Jedes Objekt der Klasse ZELLE wird durch ein Objekt der Klasse ZELLENSYMBOL dargestellt. Die Beziehungen zwischen den Objekten zeigt folgende Abbildung am Beispiel von vier Objekten:

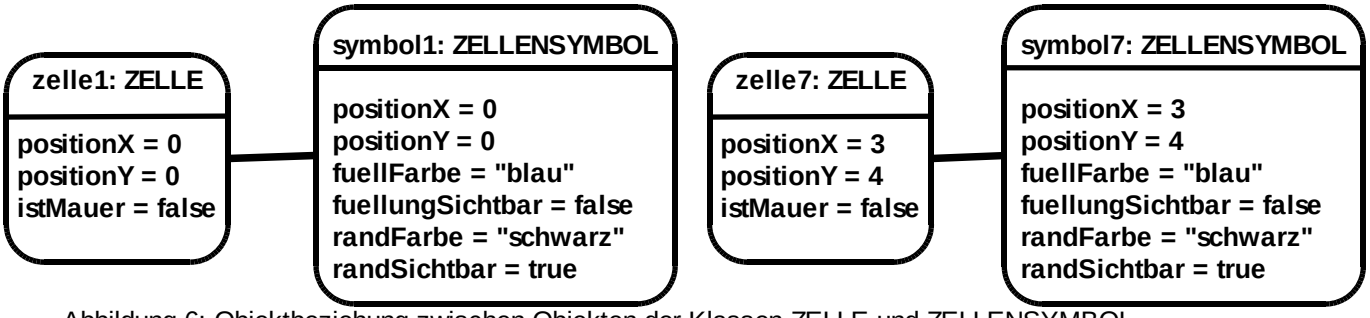


Abbildung 6: Objektbeziehung zwischen Objekten der Klassen ZELLE und ZELLENSYMBOL

Umgesetzt wird die Objektreferenz jeweils durch das Referenzattribut zSymbol bei Objekten der Klasse ZELLE. Dies ist in der Abbildung 7 dargestellt.

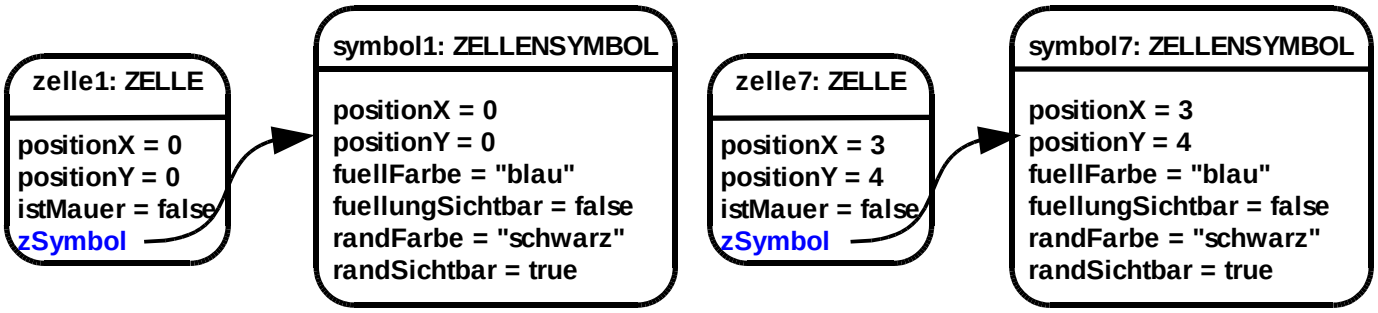


Abbildung 7: Umsetzung der Objektbeziehung aus Abbildung 6 durch Referenzattribute

Diese Objektreferenzen sind analog zu denen zwischen den Objekten mampfi und symbol (Kapitel 3).

Aufgabe 8.3
Was ist im Referenzattribut zSymbol der Objekte zelle1 bzw. zelle7 (Abbildung 7) gespeichert?

Aufgabe 8.4
Setze die Beziehung zwischen ZELLE und ZELLENSYMBOL entsprechend Abbildung 4 bzw. den Erklärungen oben in Java um.
Teste dein Programm durch das Erzeugen mehrerer Zellenobjekte.

8.3 Methoden der Klasse ZELLE

Aufgabe 8.5
Überlege dir sinnvolle Methoden für die Klasse ZELLE.

Bisher enthält die Klasse ZELLE außer dem Konstruktor keine Methoden. Eine wichtige Fähigkeit, die diese Klasse haben muss, ist das Setzen des Werts des Attributs istMauer. In diesem Kapitel soll zunächst nur diese eine Methode umgesetzt werden. Weitere folgen später.

Aufgabe 8.6
Ergänze in deinem erweiterten Klassendiagramm die Methode IstMauerSetzen. Welche Eingangsparameter sind erforderlich? Welche Anweisungen müssen im Methodenrumpf durchgeführt werden?

Im Methodenrumpf von *IstMauerSetzen* muss einerseits der Wert des Attributs istMauer auf den Eingabewert (istMauerNeu) gesetzt werden, andererseits muss abhängig vom Attributwert die Füllfarbe des zugehörigen Zellensymbolobjekts sichtbar bzw., unsichtbar gemacht werden. Der zweite Schritt ist nötig, da in der Graphik eine Mauer durch ein farbig ausgefülltes Rechteck angezeigt werden soll (Abbildung 1).

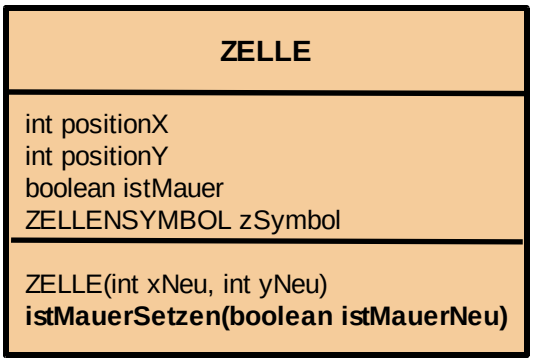


Abbildung 8: erweitertes Klassendiagramm der Klasse ZELLE

Aufgabe 8.7
Welche besondere Anweisung ist nötig, um wie im Text oben beschrieben das Ändern der Füllfarbe (je nach Wert des Attributs istMauer) umsetzen zu können?

Eine Möglichkeit, den Methodenrumpf umzusetzen, zeigt das folgende Struktogramm. Verwendet wird dabei eine bedingte Anweisung (vgl. Kapitel 5):

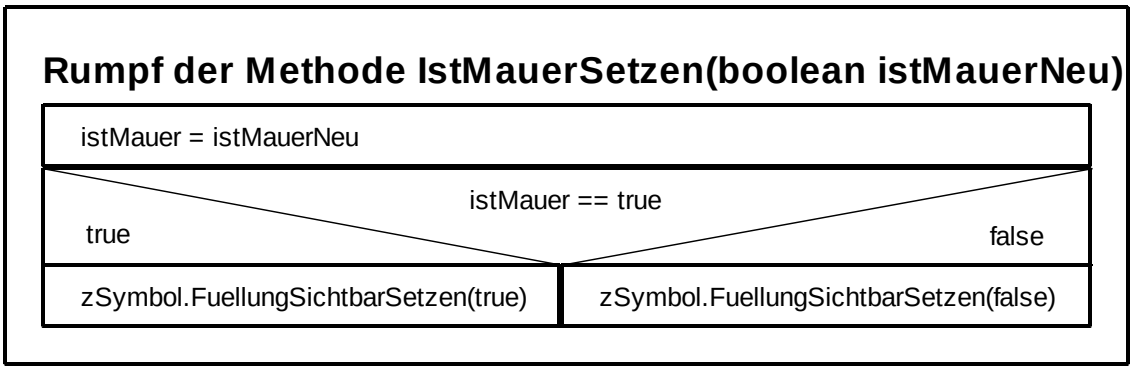


Abbildung 9: Struktogramm zum Methodenrumpf der Methode *IstMauerSetzen* – nach einer Zuweisung folgt ein Methodenaufruf an das Objekt der Klasse ZELLENSYMBOL. Abhängig vom Wert des Attributs istMauer (Bedingung) ist der Eingabewert beim Methodenaufruf unterschiedlich.

Aufgabe 8.8
Ergänze den Quellcode deiner Klasse ZELLE entsprechend dem Struktogramm um die Methode *IstMauerSetzen*. Teste die Methode.

8.4 Methodenaufrufe – Ersetzen des Parameters durch konkrete Eingabewerte

Folgende alternativen Möglichkeiten gibt es den Methodenrumpf zu formulieren. Bei den Varianten A und B wurde nur die Bedingung der bedingten Anweisung geändert, bei der Variante C konnte durch eine elegante Formulierung auf die bedingte Anweisung verzichtet werden.

Variante A:

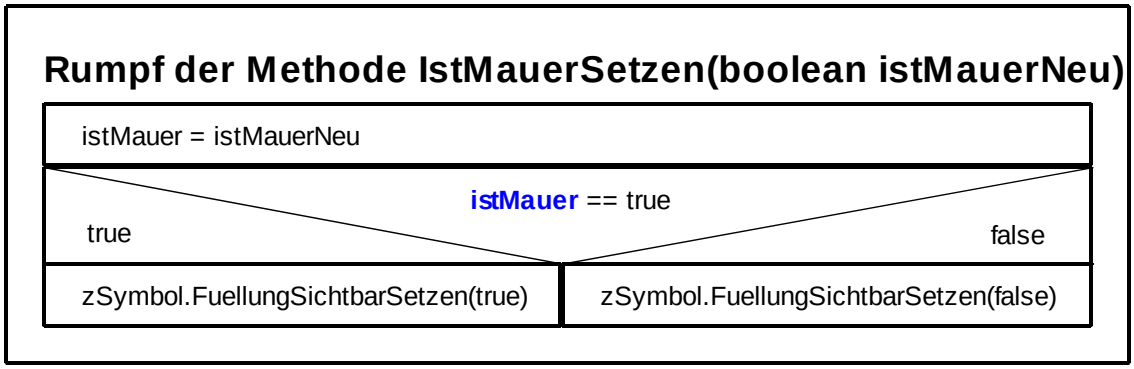


Abbildung 10: Struktogramm zur Variante A des Methodenrumpfs der Methode *IstMauersSetzen*

Variante B:

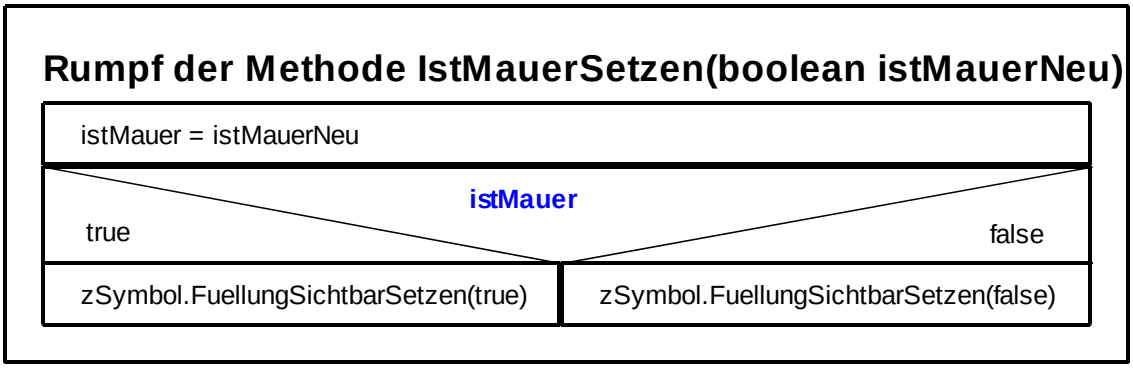


Abbildung 11: Struktogramm zur Variante B des Methodenrumpfs der Methode *IstMauersSetzen*

Variante C:

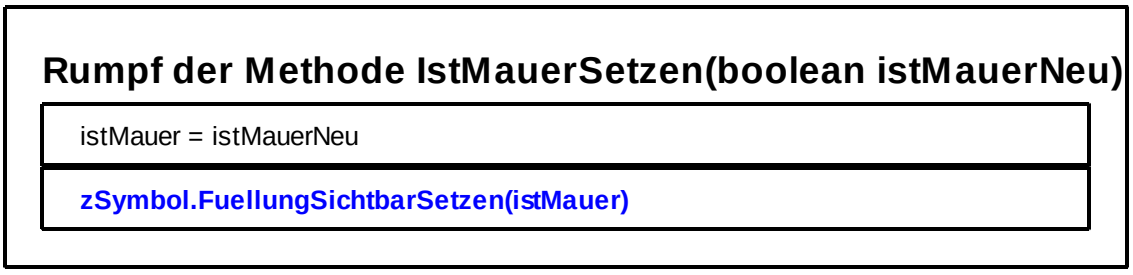


Abbildung 12: Struktogramm zur Variante C des Methodenrumpfs der Methode *IstMauersSetzen*

Warum alle diese Varianten der Methode zum gleichen Ergebnis führen, kann man sich überlegen, indem man die Methodenaufrufe mit verschiedenen Eingabewerten "durchspielt".

Bei der Variante A führt ein Methodenaufruf mit dem Eingabewert `true` zu folgendem Ablauf:

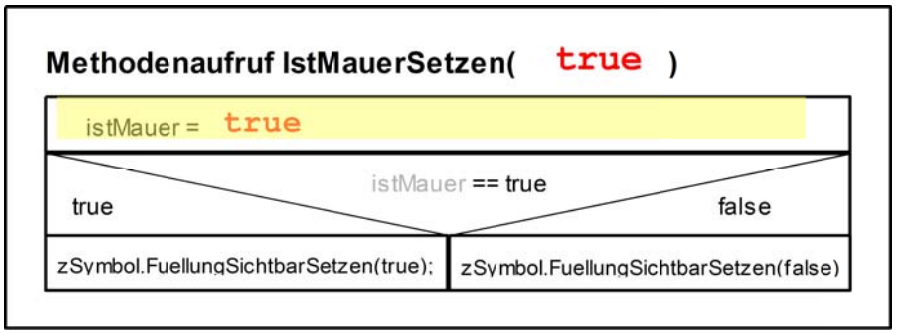


Abbildung 13:
Aufruf der Methode *IstMauerSetzen* mit dem Eingabewert true –
1. Schritt:
Dem Attribut istMauer wird der Wert true zugewiesen

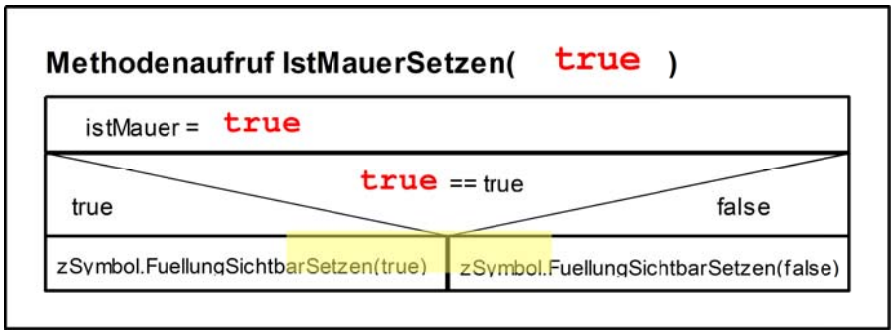


Abbildung 14:
Aufruf der Methode *IstMauerSetzen* mit dem Eingabewert true –
2. Schritt:
Auswertung der Bedingung: Da der Wert von istMauer true ist, wird der Vergleich
true == true
mit true ausgewertet

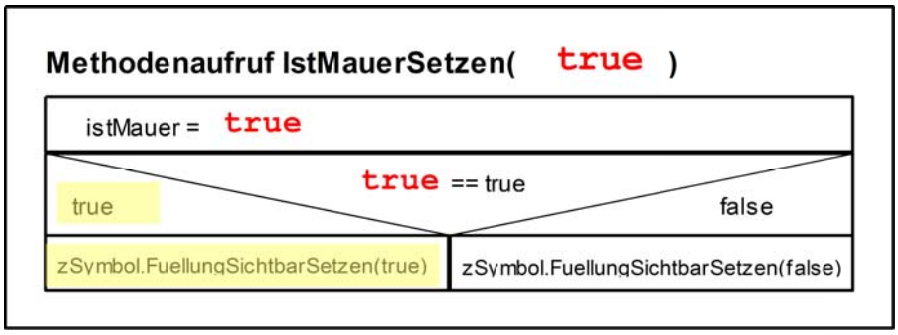


Abbildung 15:
Aufruf der Methode *IstMauerSetzen* mit dem Eingabewert true –
3. Schritt:
In der bedingten Anweisung wird dann der linke Zweig ("dann-Zweig" im Struktogrammm) ausgeführt: Die Methode *FuellungSichtbarSetzen* wird mit dem Wert true aufgerufen.

Bei der Variante A führt ein Methodenaufruf mit dem Eingabewert `false` zu folgendem Ablauf:

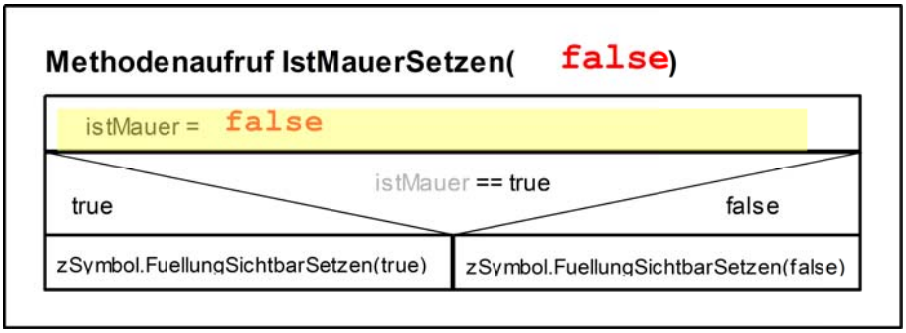


Abbildung 16:
Aufruf der Methode *IstMauerSetzen* mit dem Eingabewert false –
1. Schritt:
Dem Attribut istMauer wird der Wert false zugewiesen

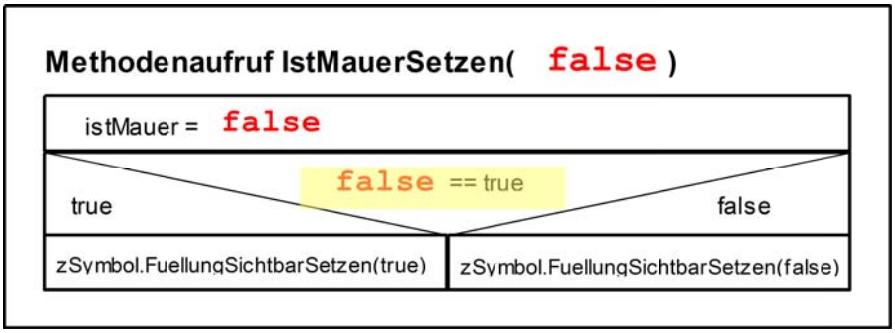


Abbildung 17:
Aufruf der Methode *IstMauerSetzen* mit dem Eingabewert false – 2. Schritt:
Auswertung der Bedingung: Da der Wert von istMauer false ist, wird der Vergleich
false == true
mit false ausgewertet

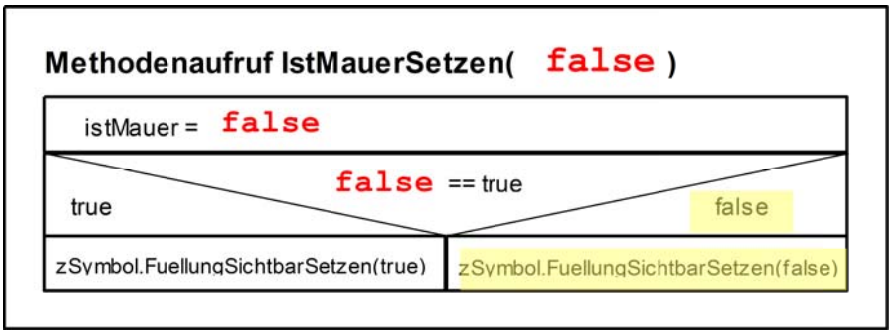


Abbildung 18:
Aufruf der Methode *IstMauerSetzen* mit dem Eingabewert true – 3. Schritt:
In der bedingten Anweisung wird der rechte Zweig ("sonst-Zweig" im Struktogrammm) ausgeführt: Die Methode FuellungSichtbarSetzen wird mit dem Wert false aufgerufen.

Aufgabe 8.9
Erkläre in gleicher Art und Weise, warum beim Aufruf der Methode *IstMauerSetzen* in den Varianten B und C genau das gleiche geschieht wie z.B. bei der eben besprochenen Variante. Du findest dazu im Anhang Struktogramme als Vorlage, in denen du Attribute durch Attributwerte ersetzen kannst. Verwende dazu einen Farbstift und überschreibe damit analog zu den Abbildungen 13 bis 18 Parameternamen durch konkrete boolsche Werte.

Aufgabe 8.10
a) Ersetze den Methodenrumpf aus Aufgabe 8.8 durch den Methodenrumpf der Variante C.
b) Teste die Methode!

Aufgabe 8.11
Vorschau auf Kapitel 9:
Schreibe eine Klasse LABYRINTH, die mindestens aus einer Zeile von ZELLEN besteht.

8.5 Zusammenfassung

Aufgabe 8.12
In diesem Kapitel kommen keine neuen Inhalte vor, sondern bereits Bekanntes wird in einem neuen Kontext angewendet. Hattest du an der ein oder anderen Stellen Schwierigkeiten oder ein "Aha-Erlebnis"? Optimiere bei Bedarf deine Zusammenfassungen aus den vorherigen Kapiteln.

Aufgabe 8.13
Auch die Objektkommunikation spielt in diesem Kapitel eine wichtige Rolle. Erstelle ein Sequenzdiagramm von der Objektkommunikation, beim Aufruf der Methode *IstMauerSetzen* durch einen externen Aufrufer.

Anhang: Tipps zu den Aufgaben

Tipps zu Aufgabe 8.2

Ein mögliches Klassendiagramm für die Klasse ZELLE ist in Abbildung 19 zu sehen. Neben den Positionsangaben ist ein wichtiges Attribut istMauer, in dem gespeichert wird, ob die Zelle betretbar ist oder nicht. (Zellen, deren Wert des Attributs istMauer true ist, sind in Abbildung 1 blau dargestellt.)
Wie schon bei der Klasse MAMPFI müssen als Nächstes Überlegungen zu den Datentypen der Attribute angestellt werden. Das Ergebnis ist im erweiterten Klassendiagramm in Abbildung 20 zu finden.

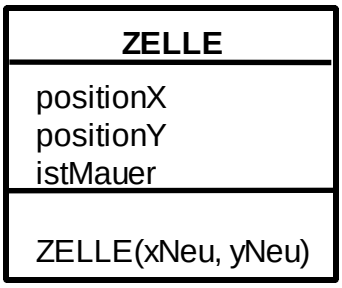


Abbildung 19:
Klassendiagramm der Klasse ZELLE ohne Methoden (außer dem Konstruktor)

Aufgabe A.8.1
Neu in Abbildung 19 ist, dass auch der Konstruktor Eingabeparameter hat. Warum ist dies sinnvoll?

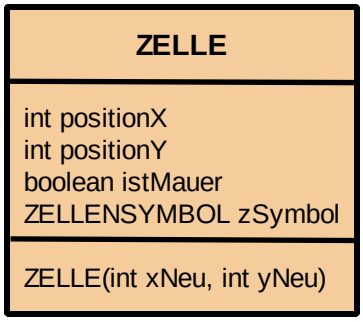


Abbildung 20:
erweitertes Klassendiagramm der Klasse ZELLE

Jedes Zellobjekt hat eine Position auf dem Spielfeld. Ist jedoch diese Position einmal festgelegt, macht es keinen Sinn, diese wieder zu ändern. Zellobjekte bewegen sich nicht. Es ist angebracht, die Position gleich bei der Objekterzeugung (also im Konstruktor) festzulegen. Deshalb benötigt der Konstruktor als Eingabewerte die x- und die y-Position (Abbildung 19 / 20).

Aufgabe A.8.2
Welche Aufgaben müssen vom Konstruktor erledigt werden? Denke auch daran, dass im Zusammenhang mit dem Referenzattribut einiges getan werden muss.

Der Konstruktor hat die Aufgabe, ein Objekt in einen gültigen Anfangszustand zu versetzen, d.h. allen Attributen müssen passende Werte zugewiesen werden. Der positionX und positionY sollen also die Eingabewerte des Konstruktors zugewiesen werden. Die Zelle soll zunächst keine Mauer sein.
Das zu referenzierende Objekt muss erzeugt und dem Referenzattribut zugewisen werden. Es ist durchaus sinnvoll, den Wert des Attributs fuellfarbe von dem Zellensymbolobjekt auf "blau" zu setzen. Das ist schon eine gute Vorbereitung für die Mauer. Wird das Attribut fuellfarbe dann über die Methode *FuellungSichtbarSetzen* (vgl. Klassendiagramm in Abbildung 5) auf true gesetzt, dann wird die blaue Füllfarbe sichtbar und das Zellensymbol stellt eine Mauer dar.
Dadurch ergibt sich folgender Konstruktor der Klasse ZELLE:

```
// Konstruktor für Objekte der Klasse ZELLE
ZELLE(int xNeu, int yNeu)
{
    positionX = xNeu;
    positionY = yNeu;
    istMauer = false;

    zSymbol = new ZELLENSYMBOL(positionX, positionY);
    zSymbol.FuellFarbeSetzen("blau");
    zSymbol.FuellungSichtbarSetzen(false);
}
```

Aufgabe A.8.3

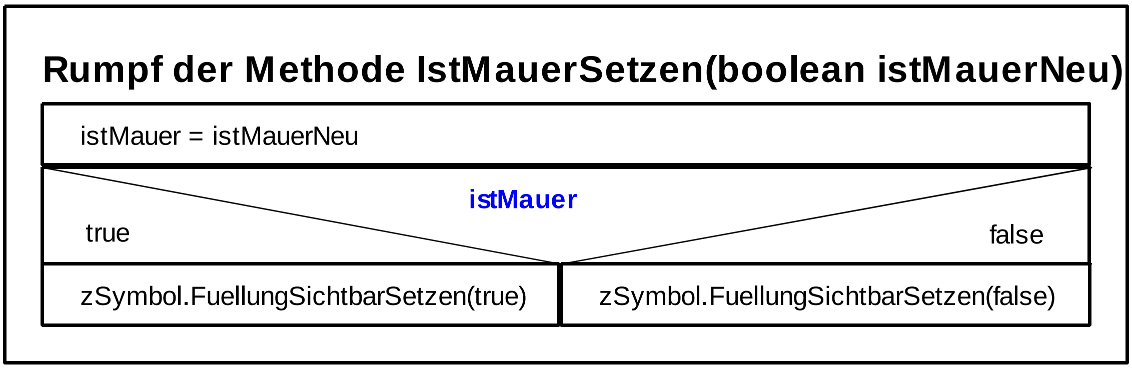
a) Setze die Klasse entsprechend den Überlegungen in diesem Kapitel mit deiner Entwicklungsumgebung um, falls noch nicht über Aufgabe 8.2, 8.4, 8.8 geschehen.

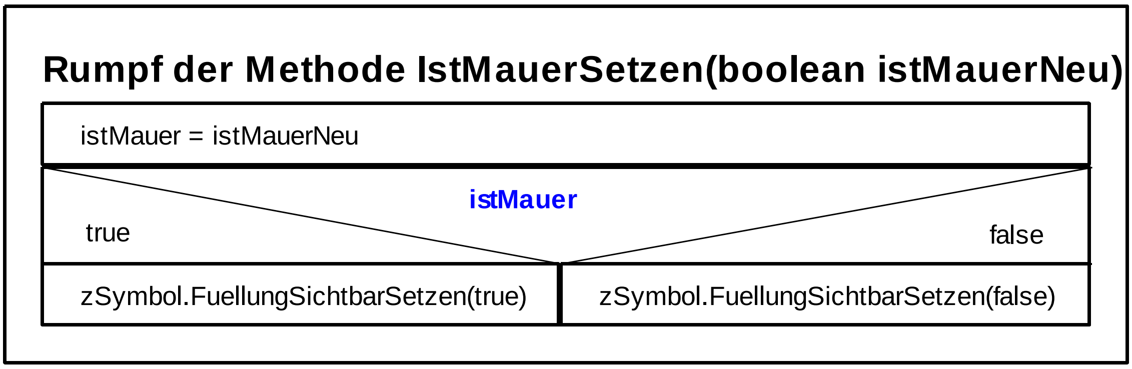
b) Teste die neue Klasse, indem du
- mehrere Objekte davon erzeugst.
- dir auch die Koordinaten - wie in der Abbildung auf Seite 3 dieses Kapitels beschrieben - anzeigen lässt.
- auch versuchst, ein Zellobjekt an einer Position zu erzeugen, an der bereits eine Zelle existiert.

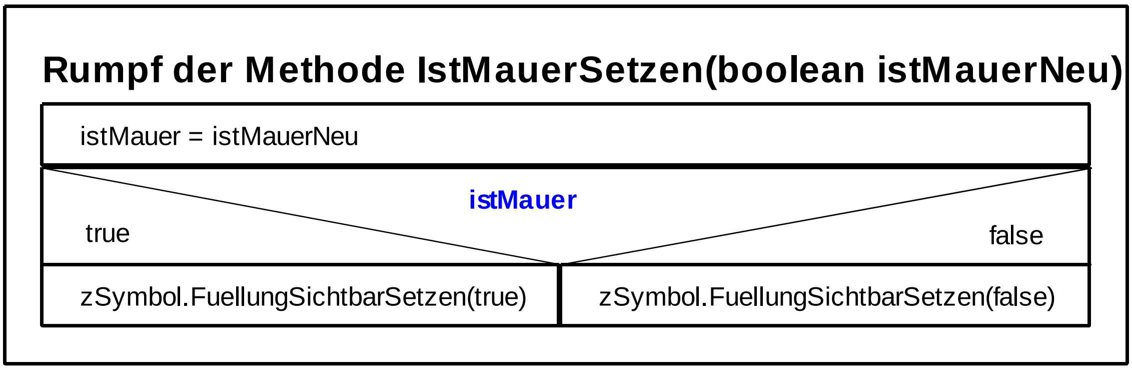
c) Betrachte mit Hilfe des Objektinspektor die Attributwerte einer der unter b) erzeugten Zellen. Betrachte auch die Attributwerte des zugehörigen Zellensymbolobjekts (Nur zu den Attributen, die in Abbildung 7 aufgelistet sind.)

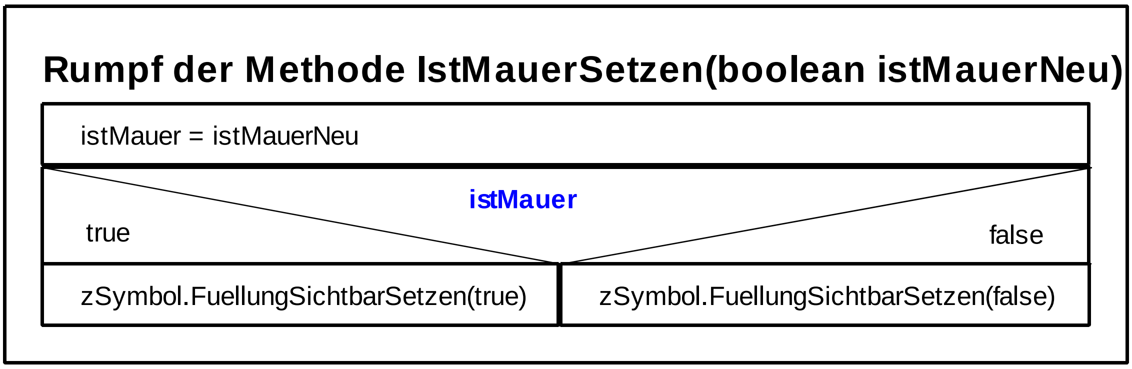
Anhang: Vorlagen zu Aufgabe 8.8

Variante B









Anhang: Vorlagen zu Aufgabe 8.8

Variante C

Rumpf der Methode IstMauerSetzen(boolean istMauerNeu)

```
istMauer = istMauerNeu
zSymbol.FuellungSichtbarSetzen(istMauer)
```

Rumpf der Methode IstMauerSetzen(boolean istMauerNeu)

```
istMauer = istMauerNeu
zSymbol.FuellungSichtbarSetzen(istMauer)
```

Rumpf der Methode IstMauerSetzen(boolean istMauerNeu)

```
istMauer = istMauerNeu
zSymbol.FuellungSichtbarSetzen(istMauer)
```

Rumpf der Methode IstMauerSetzen(boolean istMauerNeu)

```
istMauer = istMauerNeu
zSymbol.FuellungSichtbarSetzen(istMauer)
```

