Inhalt

6 Referenzdatentypen - Klassen

Eigenschaften von Feldern

- Ein Feld fasst Daten zusammen, die für ein Programm eine Einheit bilden.
- **Alle Komponenten eines Feldes besitzen den *gleichen* Typ.**

Der letzte Punkt ist eine Einschränkung, unterschiedliche Typen sind oft wünschenswert. Im Gegensatz zu Feldern sind **Strukturen** Sammlungen von Daten *verschiedenen Typs*.

Definition Struktur

Struktur $=_{df}$ **Tupel über einer Menge von Daten (*verschiedenen* Typs)**

***Strukturen* sind *strukturierte* Datentypen, die für ein Programm eine Einheit bilden und Daten *verschiedenen* Typs zusammenfassen.**
Die zusammengefassten Daten sind die *Strukturelemente*.

Strukturen werden generell in *zwei* Schritten festgelegt:
Zunächst wird ein **Strukturtyp** festgelegt und anschließend können **Strukturen** von diesem *Strukturtyp* deklariert werden.

6.1 Deklaration und Instanziierung

In Java verwendet man eine **Klasse** zur Modellierung eines *Strukturtyps*. Die in einer Klasse zusammengefassten Variablen *verschiedenen Typs* werden als **Attribute** bzw. **Instanzvariablen** bezeichnet. Sie nehmen die *Daten (Eigenschaften) einer Struktur* auf.
Im Rahmen der Objektorientierung werden Klassen, neben den Attributen, später noch **Methoden** zur Manipulation dieser Daten enthalten.

6.1.1 Festlegen eines Strukturtyps als Klasse

Elementklassen, sogenannte **Top-Level-Klassen,** werden *außerhalb von anderen Klassen* in einer *eigenen Datei* abgespeichert. Diese Datei muss den Namen der Elementklasse tragen. Elementklassen dienen der *Übersichtlichkeit*, können weitestgehend *unabhängig vom Gesamtprojekt* entwickelt und getestet werden und haben den Vorteil, dass sie, getrennt vom Programm, als *Softwarekomponente* weitergegeben werden können.

Innere Klassen sind Klassen, die innerhalb bestehender Klassen vereinbart werden. Sie sind nur dann sinnvoll, wenn sie so spezifisch sind, dass deren Anwendung außerhalb des Programms nicht zu erwarten ist.

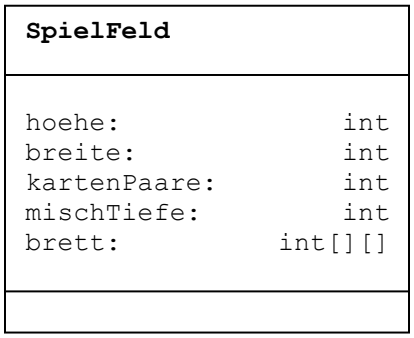
Wir werden hier *nur* mit *Elementklassen* arbeiten.

Deklaration einer Elementklasse als Strukturtyp
public class *Klassenname* **{** *Deklaration* **...** **}**

- **public** legt fest, dass die Klasse eine öffentliche, eine für jeden verfügbare Klasse ist.
- Die Deklarationen vereinbaren die Strukturelemente, also die *Attribute* bzw. *Instanzvariable*.

Beispiel „Memory“
Zunächst betrachten wir das **Spielfeld** eines Memoryspiels. Von diesem sollen die Größe (Höhe und Breite), die Anzahl der im Spiel befindlichen Kartenpaare, die Mischtiefe (wie oft die Karten vor Spielbeginn gemischt werden) und die aktuelle Verteilung der Karten auf dem Spielbrett notiert werden. Das sind Daten verschiedenen Typs, für Höhe, Breite, Kartenpaare und Mischtiefe verwenden wir ganze Zahlen vom Typ `int` und für das Spielbrett eine Matrix von ganzen Zahlen `int[][]`. Diese Daten sollen in einer *Elementklasse* `SpielFeld` zusammengefasst werden.

Als **Klassendiagramm** dargestellt hat die Klasse in Java folgende Form:



Die Deklaration als Elementarklasse sieht wie folgt aus:

SpielFeld.java

```
public class SpielFeld
{
  int hoehe = 4;
  int breite = 4;
  int kartenPaare = 8;
  int mischTiefe = 100;
  int[][] brett;
}
```

Die Initialisierung der Instanzvariablen einer Klasse ist, wie bei Deklarationen generell, möglich, aber nicht notwendig.

6.1.2 Festlegen einer Struktur als Objekt einer Klasse

In der `main`-Methode der Klasse `SpielFeld` wird ein Spielfeld erzeugt, anschließend die Spielkarten auf dem Spielbrett aufgelegt, gemischt und die Verteilung ausgegeben. Zuerst muss zur Klasse `SpielFeld` ein *Objekt* erzeugt werden. Da man Objekte auch als *Instanzen* bezeichnet, spricht man von der **Instanziierung**.

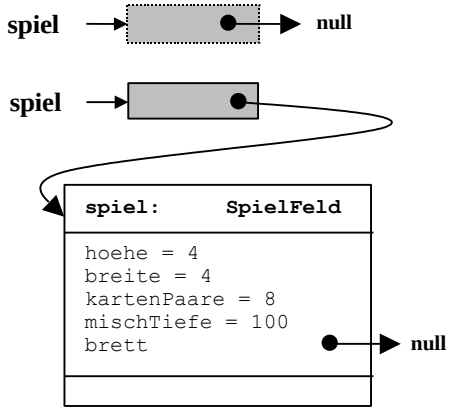
Analog den Feldern werden Objekte in zwei Schritten erzeugt:

Deklaration eines Objektes einer Klasse
Dem Compiler wird mitgeteilt, dass es sich um eine Referenzvariable handelt und Speicher für eine Adresse benötigt wird.

Klassenname Objektname **;**
`SpielFeld spiel;`

Definition eines Objektes einer Klasse
Der notwendige Speicherplatz wird mittels **new**-Operator bereitgestellt.

Objektname **=** **new** *Klassenname* **( ) ;**
`spiel = new SpielFeld();`



Deklaration und Definition eines Objektes einer Klasse

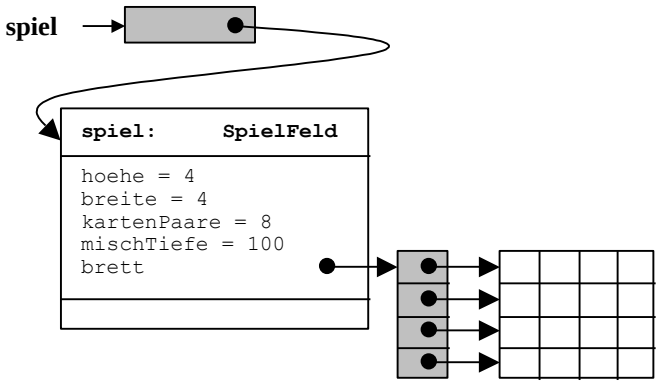
Klassenname Objektname **=** **new** *Klassenname* **( ) ;**
`SpielFeld spiel = new SpielFeld();`

Da die Instanzvariable `brett` eine Referenz im Sinne von Feldern ist, muss entsprechend Speicher bereitgestellt werden.

Zugriff auf die Attribute
Der **Zugriff** auf die einzelnen Instanzvariablen erfolgt durch den **Punkt-Operator**.

Objektname **.** *Instanzvariablenname*

```
spiel.brett = new int[ spiel.hoehe][ spiel.breite];
```



Das Memory-Spielfeld wird nun vorbereitet: Die Karten werden aufgelegt und anschließend durch Vertauschen gemischt. Die Mischtiefe gibt die Anzahl der Vertauschungen an. Zur Kontrolle wird das so vorbereitete Spielfeld ausgegeben.

SpielFeld.java (Grobstruktur)

```
public class SpielFeld
{
// Attribute der Klasse Spiefeld

  public static void main( String[] args)
  {
// Erzeugen eines Objekts der Klasse SpielFeld
// Erzeugen des Spielbretts
// Anordnen der Kartenpaare auf dem Spielbrett
// Mischen der Karten
// Kontrollausgabe des Spielbretts
  }
}
```

SpielFeld.java

```
// SpielFeld.java                                        MM 2014

/**
 * Memoryspiel: Brett und Kartenpaare.
 */
public class SpielFeld
{
/* ------------------------------------------------------ */
                                                  // Attribute
/**
 * Bretthoehe.
 */
  int hoehe = 4;
```

```
/**
 * Brettbreite.
 */
  int breite = 4;

/**
 * Anzahl der Kartenpaare.
 */
  int kartenPaare = 8;

/**
 * Mischparameter.
 */
  int mischTiefe = 100;

/**
 * Spielbrett fuer die Karten.
 */
  int[][] brett;

/* ------------------------------------------------------ */
                                                  // Programm
/**
 * Test der Klasse SpielFeld
 */
  public static void main( String[] args)
  {
              // Erzeugen eines Objekts der Klasse SpielFeld
    SpielFeld spiel = new SpielFeld();

                                    // Erzeugen des Spielbretts
    spiel.brett = new int[ spiel.hoehe][ spiel.breite];

              // Anordnen der Kartenpaare auf dem Spielbrett
    int karte = 1;
    for( int z = 0; z < spiel.hoehe; z++)
      for( int s = 0; s < spiel.breite; s++)
        spiel.brett[ z][ j] = ++karte / 2;

                                           // Mischen der Karten
    int m1, n1, m2, n2, dummy;

    for( int i = 0; i < spiel.mischTiefe; i++)
    {
      m1 = (int)(( spiel.hoehe) * Math.random());
      n1 = (int)(( spiel.breite) * Math.random());
      m2 = (int)(( spiel.hoehe) * Math.random());
      n2 = (int)(( spiel.breite) * Math.random());

      dummy = spiel.brett[ m1][ n1];
      spiel.brett[ m1][ n1] = spiel.brett[ m2][ n2];
      spiel.brett[ m2][ n2] = dummy;
```

```
      }

                              // Kontrollausgabe des Spielbretts
      for( int z = 0; z < spiel.hoehe; z++)
      {
        for( int s = 0; s < spiel.breite; s++)
        {
          System.out.print( "    ");

          if( spiel.brett[ z][ s] < 10)
            System.out.print( " ");

          System.out.print
          ( "" + spiel.brett[ z][ s] + " |");
        }
        System.out.println();
      }
    }
  }
```

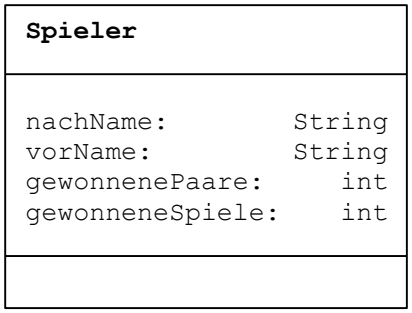
6.1.3 Kopieren von Objekten

Klassen sind analog Feldern *Referenzdatentypen*. Deren Objekte haben als Wert eine Adresse, die auf Speicherbereiche für die Werte der Instanzvariablen verweist. Folglich hat man beim **Kopieren** von Objekten analog den Feldern zunächst die Datenstruktur und dann die Daten zu übertragen.

6.2 Felder von Klassen

Von jedem Mitspieler eines Memoryspiels soll der Name, die Anzahl der im aktuellen Spiel bereits gewonnenen Kartenpaare und die Anzahl der insgesamt gewonnenen Spiele gemerkt werden. Die Klasse `Spieler` fasst alle diese Daten zusammen:

```
public class Spieler
{
  String[1] nachName = "";
  String vorName = "";
  int gewonnenePaare = 0;
  int gewonneneSpiele = 0;
}
```

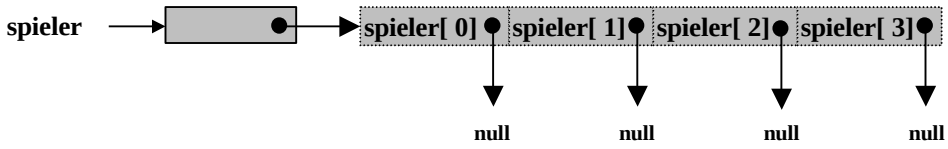


Von der Klasse `Spieler` benötigen wir mindestens zwei Objekte. Es können aber auch mehr als zwei Spieler mitspielen. Diese werden in einem *Feld von Spielern* verwaltet, also ein Feld, dessen *Komponenten Objekte einer Klasse* sind.
Dieses Feld von Spielern wird nun mit dem **new**-Operator bereitgestellt. Zuvor muss die Anzahl der mitzuspielenden Personen bekannt sein.

```
int spielerAnzahl
= IOTools.readInteger( "Wie viel Spieler gibt es? ");
```

Definition und Deklaration des Spielerfeldes lauten dann wie folgt:

```
Spieler[] spieler = new Spieler[ spielerAnzahl];
```

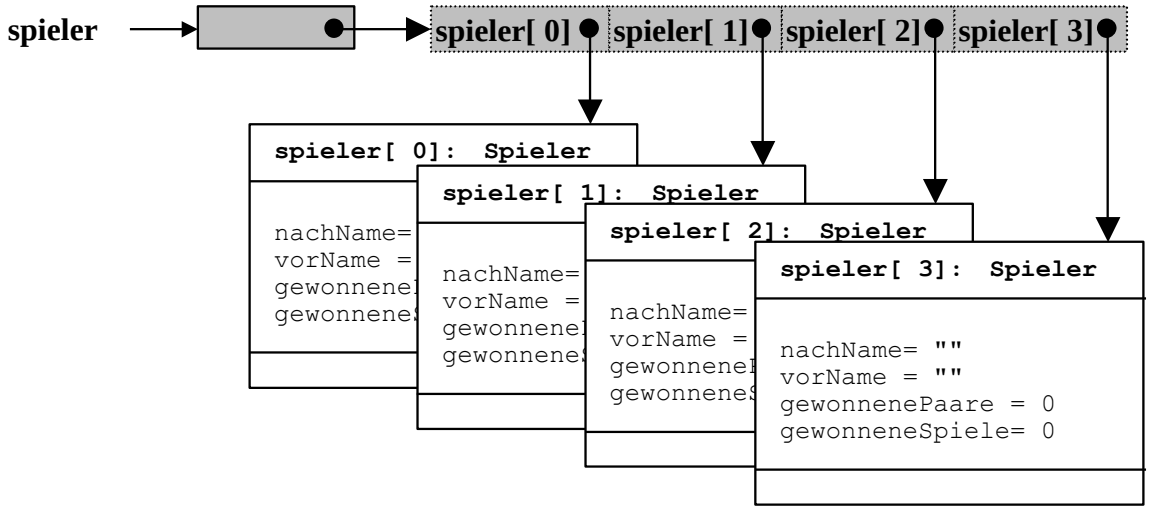


Die Variable `spieler` fasst hier insgesamt vier Einträge zusammen und verweist damit auf vier Komponenten `spieler[ 0]` bis `spieler[ 3]`, die alle von Typ `Spieler` sind und anfangs eine **null**-Referenz beinhalten.
Anschließend muss noch jedem Spieler einzeln Speicherplatz für seine individuellen Daten bereitgestellt werden:

```
for( int i = 0; i < spieler.length; i++)
  spieler[ i] = new Spieler();
```

Die nachfolgende Abbildung zeigt den Zustand des Feldes `spieler` nach dem Erzeugen der Feldkomponenten und der Datenbereiche im Speicher für jeden einzelnen Spieler.

[1] Die Klasse `String` im Paket `java.lang` stellt Methoden für Operationen mit Zeichenketten bereit.



Schließlich sind noch die Daten der Spieler zu aktualisieren.

Spieler.java (Grobstruktur)

```
public class Spieler
{
// Attribute

  public static void main( String[] args)
  {
// Einlesen der Spieleranzahl
// Erzeugen eines Feldes von Spielern
// Bereitstellen des Datenspeichers für jeden Spieler
// Einlesen der Spielerdaten
// Auflisten aller eingelesenen Spielerdaten
  }
}
```

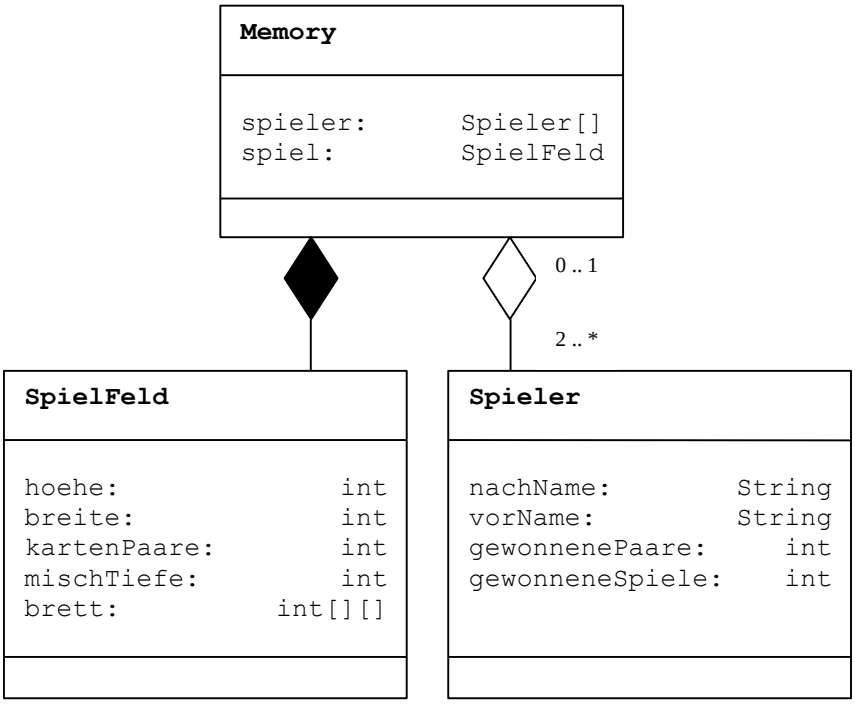
Spieler.java

```
// Spieler.java                                          MM 2014
import Tools.IO.*;                                    // Eingaben

/**
 * Spieler fuer ein Memoryspiel.
 */
public class Spieler
{
/* ------------------------------------------------------ */
                                                    // Attribute
/**
 * Nachname des Spielers.
 */
  String nachName = "";
```

```
  /**
   * Vorname des Spielers.
   */
    String vorName = "";

  /**
   * Anzahl der gewonnenen Kartenpaare im aktuellen Spiel.
   */
    int gewonnenePaare = 0;

  /**
   * Anzahl der gewonnenen Spiele.
   */
    int gewonneneSpiele = 0;

  /* ------------------------------------------------------ */
                                                  // Programm
  /**
   * Test der Klasse Spieler
   */
    public static void main( String[] args)
    {
                                    // Einlesen der Spieleranzahl
      int spielerAnzahl
      = IOTools.readInteger( "Wieviel Spieler gibt es? ");

                          // Erzeugen eines Feldes von Spielern
      Spieler[] spieler = new Spieler[ spielerAnzahl];

        // Bereitstellen des Datenspeichers für jeden Spieler
      for( int i = 0; i < spieler.length; i++)
        spieler[ i] = new Spieler();

                                    // Einlesen der Spielerdaten
      for( int i = 0; i < spieler.length; i++)
      {
        System.out.println( "Spieler "+ (i + 1));
        spieler[ i].vorName
        = IOTools.readLine( "Wie ist Dein Vorname? ");
        spieler[ i].nachName
        = IOTools.readLine( "Wie ist Dein Nachname? ");
      }

                  // Auflisten aller eingelesenen Spielerdaten
      System.out.println( "Es spielen mit: ");
      for( int i = 0; i < spieler.length; i++)
        System.out.println ( "Spieler "+ (i + 1) + " " +
                             spieler[ i].vorName + " " +
                             spieler[ i].nachName);
    }
  }
```

6.3 Klassen von Klassen

In Klassen selbst können auch Klassen als Instanzvariablen vereinbart werden. Das nächste Beispiel soll das Memoryspiel vervollständigen. Es wird eine Klasse `Memory` deklariert, welche die Spieler und das Spielfeld zusammenführt.




```
public class Memory
{
  Spieler[] spieler;
  SpielFeld spiel;
}
```

Für ein ***Memoryspiel*** ist nun ein Objekt der Klasse `Memory` zu vereinbaren:

```
Memory memory = new Memory();
```

Außerdem ist das ***Spielfeld*** zu installieren:

```
memory.spiel = new SpielFeld();
memory.spiel.brett
= new int[memory.spiel.hoehe][memory.spiel.breite];
```

Der Zugriff auf die Instanzvariablen erfolgt, wie üblich, mit dem Punktoperator. Treten mehrere Klassen geschachtelt auf, so sind u.U. mehrere Punkte in den Bezeichnern notwendig.
Mit `memory.spiel.hoehe` und `memory.spiel.breite` wird auf die Instanzveriablen `hoehe` und `breite` der **Instanzvariablen** `spiel` des **Objekts** `memory` der **Klasse** `Memory` zugegriffen.

Und schließlich sind die ***Spieler*** bereitzustellen:

```
memory.spieler = new Spieler[ spielerAnzahl];
for( int i = 0; i < memory.spieler.length; i++)
  memory.spieler[ i] = new Spieler();
```

In der Klasse `Memory` soll nun das vollständige Spiel implementiert werden:

Memory.java

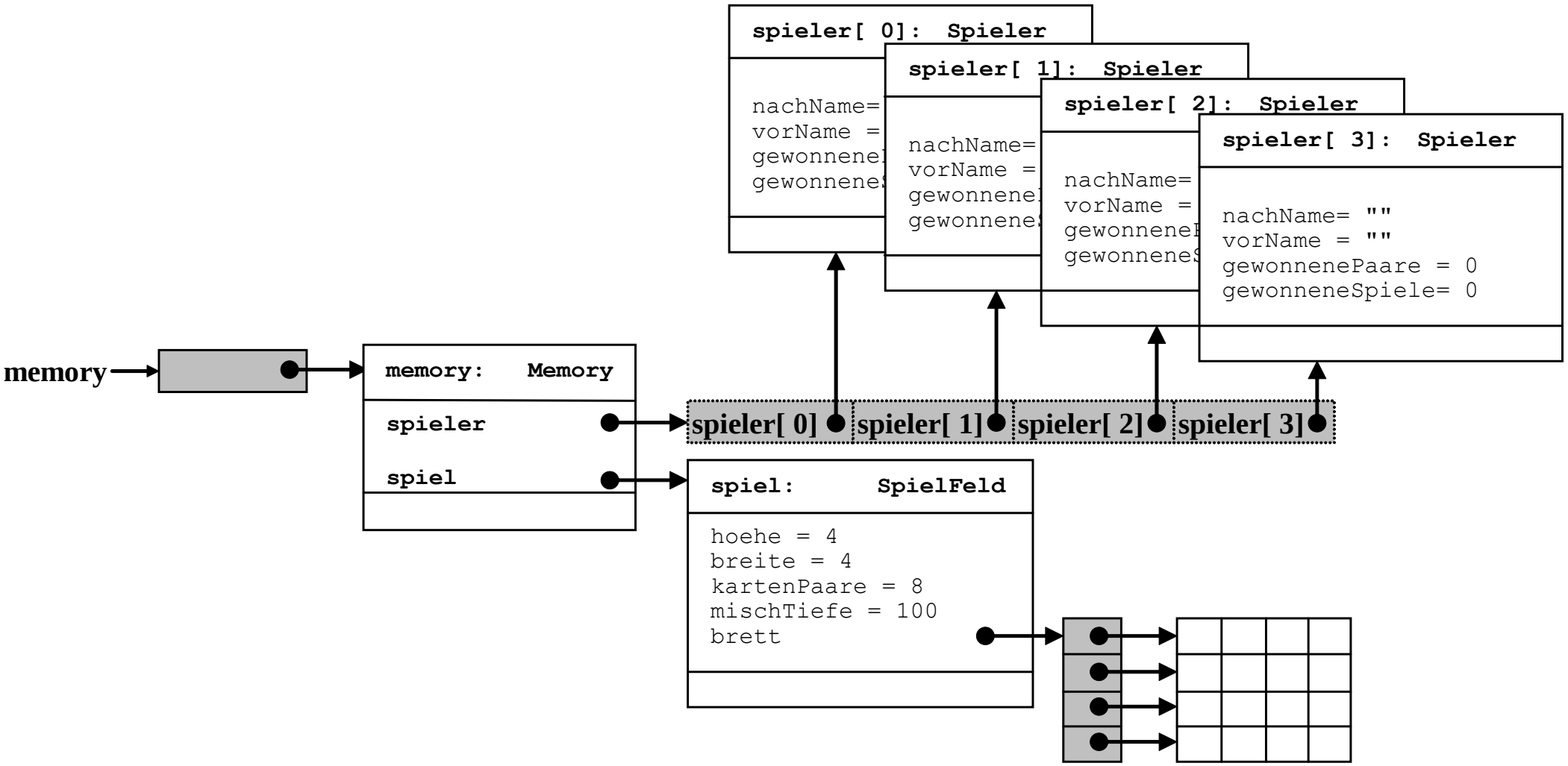
```
// Memory.java                                          MM 2014

import Tools.IO.*;                                  // Eingaben

/**
 * Memoryspiel.
 */
public class Memory
{
/* ------------------------------------------------------ */
                                                  // Attribute
/**
 * Alle Mitspieler, mindestens zwei.
 */
  Spieler[] spieler;

/**
 * Memoryspiel: Brett und Kartenpaare.
 */
  SpielFeld spiel;


/* ------------------------------------------------------ */
                                                  // Programm
/**
 * Spielen.
 */
  public static void main( String[] args)
  {
// Erzeugt ein Memoryspiel
    Memory memory = new Memory();


/* ------------------------------------------------------ */
// Spielfeld
                                  // Erzeugen des Spielfeldes
    memory.spiel = new SpielFeld();
                                 // Erzeugen des Spielbrettes
    memory.spiel.brett
    = new int[memory.spiel.hoehe][memory.spiel.breite];
```

```
            // Anordnen der Kartenpaare auf dem Spielbrett
    int karte = 1;
    for( int z = 0; z < memory.spiel.hoehe; z++)
      for( int s = 0; s < memory.spiel.breite; s++)
        memory.spiel.brett[ z][ s] = ++karte / 2;

                                        // Mischen der Karten
    int m1, n1, m2, n2, dummy;
    for( int i = 0; i < memory.spiel.mischTiefe; i++)
    {
      m1 = (int)(( memory.spiel.hoehe) * Math.random());
      n1 = (int)(( memory.spiel.breite) * Math.random());
      m2 = (int)(( memory.spiel.hoehe) * Math.random());
      n2 = (int)(( memory.spiel.breite) * Math.random());

      dummy = memory.spiel.brett[ m1][ n1];
      memory.spiel.brett[ m1][ n1]
      = memory.spiel.brett[ m2][ n2];
      memory.spiel.brett[ m2][ n2] = dummy;
    }


/* ------------------------------------------------- */
// Spieler
                                // Einlesen der Spieleranzahl
    int spielerAnzahl =
    IOTools.readInteger( "Wie viel Spieler gibt es? ");

                                      // Erzeugen der Spieler
    memory.spieler = new Spieler[ spielerAnzahl];
    for( int i = 0; i < memory.spieler.length; i++)
      memory.spieler[ i] = new Spieler();

                                 // Einlesen der Spielerdaten
    for( int s = 0; s < memory.spieler.length; s++)
    {
      System.out.println( "Spieler "+ (s + 1));
      memory.spieler[ s].vorName
      = IOTools.readLine( "Wie ist Dein Vorname? ");
      memory.spieler[ s].nachName
      = IOTools.readLine( "Wie ist Dein Nachname? ");
    }


/* ------------------------------------------------ */
// Spiel
    int dran = 0;
    while( memory.spiel.kartenPaare != 0)
    {
      System.out.println
      ( "Spieler "+  memory.spieler[ dran].vorName +
        ", du bist dran! Nimm zwei Karten!");
```

```
do                                                  // Karte 1
{
  m1 = IOTools.readInteger( "");
  n1 = IOTools.readInteger( "");
}
while( m1 < 0 || n1 < 0 ||
       m1 >= memory.spiel.hoehe ||
       n1 >= memory.spiel.breite ||
       memory.spiel.brett[ m1][ n1] == 0);

                                  // Zwischenstand Spielbrett
for( int z = 0; z < memory.spiel.hoehe; z++)
{
  for( int s = 0; s < memory.spiel.breite; s++)
  {
    System.out.print( "    ");

    if( memory.spiel.brett[ z][ s] == 0)
      System.out.print( "  ");
    else
      if( z == m1 && s == n1)
      {
        if( memory.spiel.brett[ z][ s] < 10)
          System.out.print( " ");
        System.out.print
          ( "" + memory.spiel.brett[ z][ s]);
      }
      else System.out.print( " X");

    System.out.print( " |");
  }
  System.out.println();
}

do                                                  // Karte 2
{
  m2 = IOTools.readInteger( "");
  n2 = IOTools.readInteger( "");
}
while( m1 == m2 && n1 == n2 ||
       m2 < 0 || n2 < 0 ||
       m2 >= memory.spiel.hoehe ||
       n2 >= memory.spiel.breite ||
       memory.spiel.brett[ m2][ n2] == 0);

                                  // Zwischenstand Spielbrett
for( int z = 0; z < memory.spiel.hoehe; z++)
{
  for( int s = 0; s < memory.spiel.breite; s++)
  {
    System.out.print( "    ");
```

```
            if( memory.spiel.brett[ z][ s] == 0)
              System.out.print( "  ");
            else
              if( z == m1 && s == n1 ||
                  z == m2 && s == n2)
              {
                if( memory.spiel.brett[ z][ s] < 10)
                  System.out.print( " ");
                System.out.print
                  ( "" + memory.spiel.brett[ z][ s]);
              }
              else System.out.print( " X");

            System.out.print( " |");
          }
          System.out.println();
        }

                                                  // Auswertung
        if( memory.spiel.brett[ m1][ n1]
            == memory.spiel.brett[ m2][ n2])
        {
          System.out.println
          ( "Richtig, das Paar gehoehrt dir!");
          memory.spieler[ dran].gewonnenePaare++;
          memory.spiel.kartenPaare--;
          memory.spiel.brett[ m1][ n1] = 0;
          memory.spiel.brett[ m2][ n2] = 0;
        }
        else
        {                                    // Spielerwechsel
          System.out.println
          ( "Falsch, jetzt ist der Naechste dran!");
          dran++; dran %= memory.spieler.length;
        }

  /*
      // Kontrollausgabe des Spielbrettes fuer Testzwecke
        for( int z = 0; z < memory.spiel.hoehe; z++)
        {
          for( int s = 0; s < memory.spiel.breite; s++)
          {
            System.out.print( "    ");

            if( memory.spiel.brett[ z][ s] < 10)
              System.out.print( " ");

            System.out.print
            ( "" + memory.spiel.brett[ z][ s] + " |");
          }
        System.out.println();
```

```
        }
*/
    }


/* ------------------------------------------------ */
// Endstand
    System.out.println( "Liste unsortiert: ");
    for( int i = 0; i < memory.spieler.length; i++)
      System.out.println
      ( "Spieler "+ (i + 1) + ": " +
        memory.spieler[ i].vorName + " " +
        memory.spieler[ i].nachName + ", gewonnen: " +
        memory.spieler[ i].gewonnenePaare + " Paare.");
  }
}
```


spieler[0]: Spieler
nachName=
vorName =
spieler[1]: Spieler
spieler[2]: Spieler
spieler[3]: Spieler
nachName= ""
vorName = ""
gewonnenePaare = 0
gewonneneSpiele= 0
memory
memory: Memory
spieler
spiel
spieler[0]
spieler[1]
spieler[2]
spieler[3]
spiel: SpielFeld
hoehe = 4
breite = 4
kartenPaare = 8
mischTiefe = 100
brett