TECHNISCHE UNIVERSITÄT DARMSTADT

Kapitel 01d:
Arrays mit FopBot

Karsten Weihe

TECHNISCHE UNIVERSITÄT DARMSTADT

Array von Robot

Oracle Java Tutorials: Arrays

Wir gehen jetzt einen Schritt weiter und verwenden nicht einen oder zwei einzelne Roboter, sondern ein ganzes *Array* von Robotern. Das sind mehrere Roboter auf einmal, die nummeriert sind, also mit unterschiedlichen Indizes angesprochen werden.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import fopbot.*;
import static Direction.*;
public class Array implements Directions {
    public static void main(String[] args) {
        World.setVisible(true);
        ..........
    }
}
```

Die Datei für unser erstes Beispiel ist Array.java.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

Das sind die ersten vier Anweisungen des Beispiels, das Sie in Array.java finden.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

Dem Array geben wir den Namen robots, also Plural, um schon im Namen anzudeuten, dass es sich um mehrere Roboter handelt.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

Der Typ der Variablen robots soll jetzt nicht Robot sein, sondern *Array* von Robot. Robot ist dann nur der Typ der einzelnen *Komponenten* des Arrays. Dafür müssen eine öffnende und eine schließende eckige Klammer hinter den Namen des Komponententyps geschrieben werden.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

Auch ein Objekt von einem Arraytyp muss mit Operator new erzeugt werden, wie wir es schon mehrfach bei der Klasse Robot gesehen haben.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

Hinter „new“ geben wir bei einem Array nicht nur den Komponententyp an, sondern danach in eckigen Klammern noch die Anzahl der Komponenten, die das Array haben soll. Hier wird das Array also drei Roboter enthalten.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

Wie eingangs gesagt, kann man auf die drei Roboter im Array robots mittels ihrer Nummern zugreifen. Diese Nummern werden *Indizes* genannt, so nennen wir sie auch im Weiteren. Hinter dem Namen des Arrays steht der Index des einzelnen Roboters wieder in eckigen Klammern.

Dringend zu beachten ist, dass der erste Index eines Arrays in Java niemals 1, sondern immer 0 ist. Bei einem Array mit drei Komponenten gibt es also die Indizes 0, 1 und 2, aber es gibt *nicht* den Index 3.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

Jeden einzelnen Roboter müssen wir erst einmal wieder mit Operator new erzeugen, wie wir es bisher immer schon gemacht haben.

Array von Robot

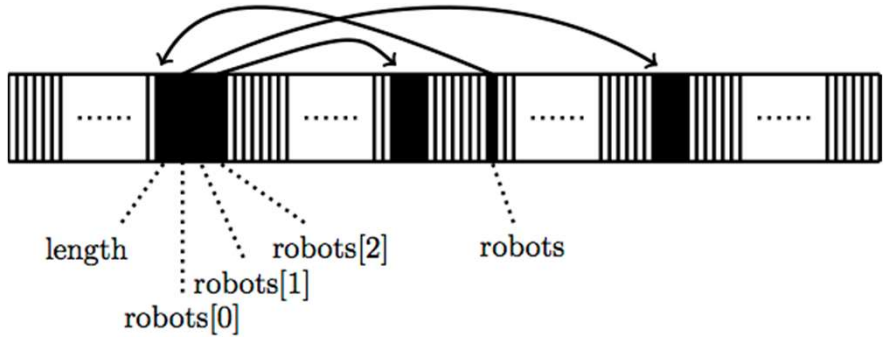
TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] robots = new Robot [ 3 ];
robots[0] = new Robot ( 0, 0, RIGHT, 4 );
robots[1] = new Robot ( 2, 5, DOWN, 4 );
robots[2] = new Robot ( 6, 2, UP, 4 );
```

In diesem kleinen Beispiel werden also drei Roboter erzeugt, die auf verschiedene Felder gestellt werden, in verschiedene Richtungen schauen und jeweils vier Münzen haben.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

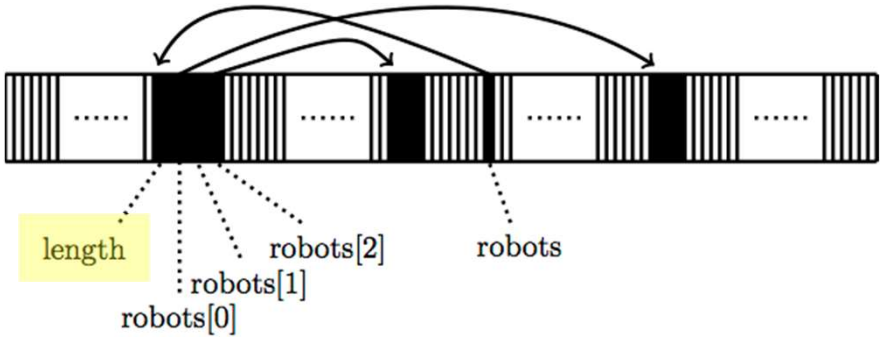



```
Robot[ ] robots = new Robot[3];
robot[0] = new Robot ( ..... );
robot[1] = null;
robot[2] = new Robot ( ..... );
```

So kann man sich ein solches Array im Computerspeicher vorstellen. Sie sehen, dass es dieselbe Unterscheidung zwischen Referenz und Objekt wie bei Klassen gibt und dass für jede der drei Komponenten des Arrays ein bestimmter Bereich im Objekt reserviert ist.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT




```
Robot[ ] robots = new Robot[3];
robot[0] = new Robot ( ..... );
robot[1] = null;
robot[2] = new Robot ( ..... );
```

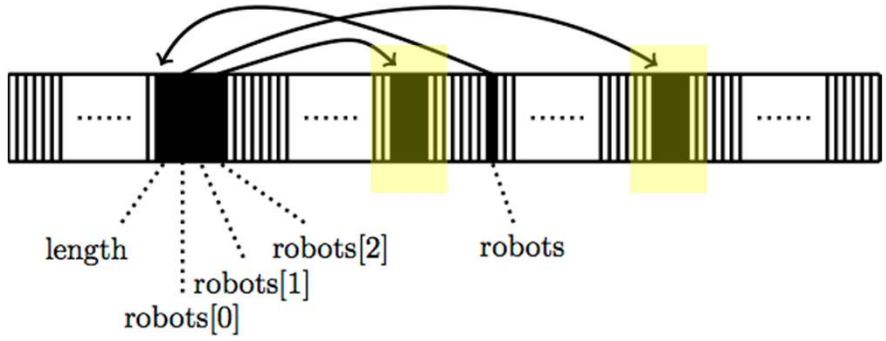
Zusätzlich richtet der Compiler ein Attribut namens length vom Typ int im Objekt ein, in dem die Anzahl der Komponenten gespeichert wird. In diesem Beispiel wird also 3 in length gespeichert.

Das Attribut length ist eine Konstante, was natürlich sinnvoll ist, denn die darin gespeicherte Zahl soll ja immer die Anzahl Arraykomponenten enthalten, und diese ändert sich nicht.

***Nebenbemerkung*: Aus historischen Gründen besteht der Name dieser Konstanten nicht aus Großbuchstaben, wie es die Konvention eigentlich fordert.**

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

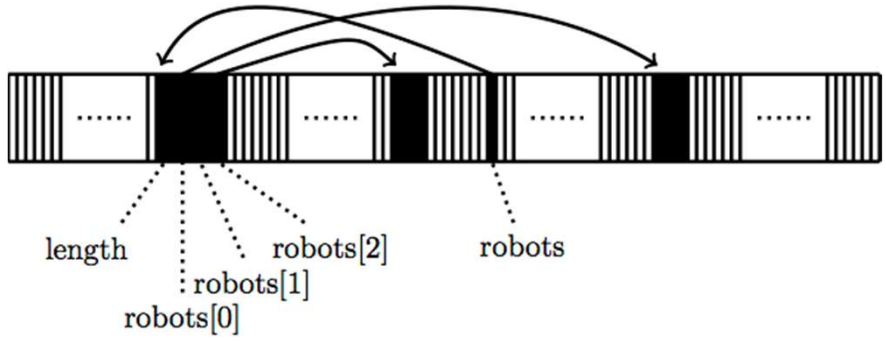



```
Robot[ ] robots = new Robot[3];
robot[0] = new Robot ( ..... );
robot[1] = null;
robot[2] = new Robot ( ..... );
```

Die einzelnen Arraykomponenten eines Array von Robot sind dann wieder ganz normal Referenzen, die auf Objekte von Klasse Robot verweisen.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT




```
Robot[ ] robots = new Robot[3];
robot[0] = new Robot ( ..... );
robot[1] = null;
robot[2] = new Robot ( ..... );
```

Das ist jetzt neu: Man kann einer Referenz anstelle der Anfangsadresse eines Objektes auch den symbolischen Wert null zuweisen. Dann verweist diese Referenz eben nicht auf ein Objekt, sondern quasi auf Nichts. Im Bild oben sehen Sie daher nur von zweien der drei Arraykomponenten jeweils einen Pfeil auf ein Objekt ausgehen, nämlich von den Komponenten mit Index 0 beziehungsweise 2.

TECHNISCHE UNIVERSITÄT DARMSTADT

Das Literal null

Bevor wir mit Array von Robot weitermachen, rekapitulieren wir noch einmal den Sonderfall, den wir soeben erstmals gesehen haben.

Das Literal null

TECHNISCHE UNIVERSITÄT DARMSTADT

```
X a = null;
int [ ]  b = null;
```

Man kann eine Variable oder Konstante von einem Referenztyp entweder auf die Adresse eines Objekts setzen, oder man kann sie alternativ auf den symbolischen Wert null setzen, eine dritte Möglichkeit gibt es nicht. In den Variablen a und b steht nach Setzung auf null ein bestimmter, fester Wert, der nicht die Adresse irgendeiner Speicherstelle ist.

Greift man bei a auf Attribute der Klasse X lesend oder schreibend zu oder ruft man mit a eine Methode von X auf oder greift man lesend oder schreibend auf eine Komponente von b zu, dann bricht das Programm mit einem Fehler ab.

***Vorgriff*: Referenzen von Interface-Typen (Kapitel 01g) können ebenfalls auf null gesetzt werden.**

***Vorgriff*: Im Kapitel 05 zur Fehlerbehandlung werden wir sehen, das dieser Programmabbruch auch verhindert werden kann.**

Nun setzen wir das eigentliche Thema fort.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( ... 4 ... ) {
    robots[0].putCoin();
    robots[0].move();
    robots[1].putCoin();
    robots[1].move();
    robots[2].putCoin();
    robots[2].move();
}
```

In unserem kleinen Beispiel lassen wir jetzt viermal jeden der drei Roboter eine Münze ablegen und dann einen Schritt vorwärts machen.

Hier darf keine der drei Arraykomponenten den symbolischen Wert null haben, sondern alle drei Arraykomponenten müssen auf Objekte verweisen. Der Grund ist, dass der Aufruf einer Methode mit null zu einem Abbruch der Ausführung des Programms führt, was ja auch logisch ist, denn ohne ein Objekt existieren auch nicht die Attribute, die mit Methoden wie move und putCoin geändert werden.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < 3; j++ ) {
        robots[j].putCoin();
        robots[j].move();
    }
```

Da mit allen drei Robotern exakt dasselbe gemacht wird, kann man statt dessen auch eine Schleife über die Indizes des Arrays laufen lassen.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < 3; j++ ) {
        robots[j].putCoin();
        robots[j].move();
    }
```

Die innere Schleife durchläuft alle Indizes des Arrays robots, also 0, 1 und 2. In jedem Durchlauf der inneren Schleife wird ein anderer der drei Roboter angesprochen.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < 3; j++ ) {
        robots[j].putCoin();
        robots[j].move();
    }
```

Mit der int-Variablen j, die in der inneren Schleife alle Indizes des Arrays nacheinander durchläuft, kann nun in den Anweisungen auf den Roboter am aktuellen Index j zugegriffen werden.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < 3; j++ ) {
        robots[j].putCoin();
        robots[j].move();
    }
```

De innere Schleife benötigt geschweifte Klammern, da zwei Anweisungen zu ihr gehören, Ablegen einer Münze und Vorwärtsschritt.

Die gesamte innere Schleife ist eine einzige Anweisung, und zwar die einzige Anweisung der äußeren Schleife. Da die äußere Schleife also nur eine Anweisung hat, benötigt sie *keine* geschweiften Klammern.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < robots.length; j++ ) {
        robots[j].putCoin();
        robots[j].move();
    }
```

Anstelle der expliziten Zahl sollten wir von jetzt an immer die Länge des Arrays mit dem eingangs erwähnten Attribut length abfragen. Erstens hatten wir in Kapitel 01b schon festgestellt, dass Konstanten mit sprechenden Namen die Verständlichkeit verbessern und bei Änderungen der Zahlenwerte Fehler vermeiden helfen. Zweitens schreibt man in der Regel Methoden, die Arrays als Parameter bekommen, und von Aufruf zu Aufruf der Methode ist das übergebene Array verschieden lang, so dass die Länge des Arrays also gar nicht anders als mit Konstante length überhaupt angegeben werden kann.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < robots.length; j++ ) {
        robots[j].putCoin();
        robots[j].move();
    }
```

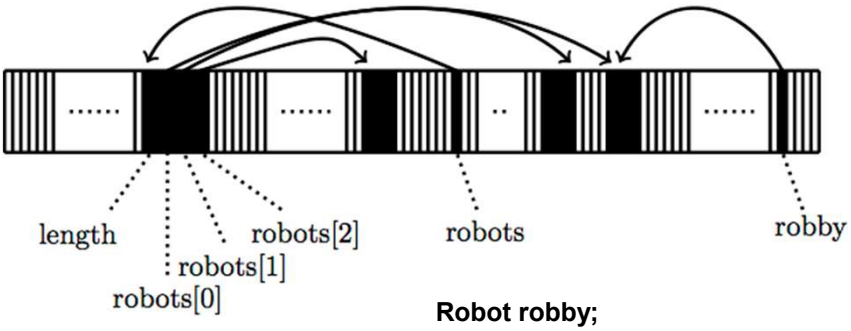
```
Robot robby;
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < robots.length; j++ ) {
        robby = robots[j];
        robby.putCoin();
        robby.move();
    }
```


Der Fall, dass alle Komponenten eines Arrays in aufsteigender Reihenfolge ihrer Indizes in einer for-Schleife durchlaufen werden sollen, ist so häufig und wichtig, dass man sich bei der Weiterentwicklung von Java irgendwann entschieden hat, für diesen Fall eine Kurzform der for-Schleife in Java einzurichten, die genau dasselbe tut wie die Normalform der for-Schleife auf der letzten Folie.

Auf dieser Folie sehen wir diese Kurzform noch nicht, sondern machen einen ersten Schritt dahin. Oben links sehen Sie dieselbe Schleife wie auf der vorhergehenden Folie und unten rechts eine äquivalente Schleife, in der die einzelnen Komponenten des Arrays robots nichts mehr direkt, sondern durch die zusätzlich definierte Referenz robby angesprochen werden.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT




```
Robot[ ] robots = new Robot[3];
robots[0] = new Robot ( ..... );
robots[1] = new Robot ( ..... );;
robots[2] = new Robot ( ..... );
```

```
Robot robby;
for ( int i = 0; i < 4; i++ )
   for ( int j = 0; j < robots.length; j++ ) {
       robby = robots[j];
       robby.putCoin()
       robby.move();
   }
```

So können Sie sich das im Computerspeicher im ersten Durchlauf durch die Schleife vorstellen: Referenz robby enthält dieselbe Adresse wie robots[0], spricht also dasselbe Robot-Objekt an.

Wie gesagt, darf keine der Arraykomponenten jetzt gleich null sein, da mit jeder Arraykomponente Methoden aufgerufen werden.

Array von Robot

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot robby;
for ( int i = 0; i < 4; i++ )
    for ( int j = 0; j < robots.length; j++ ) {
        robby = robots[j];
        robby.putCoin();
        robby.move();
    }
```

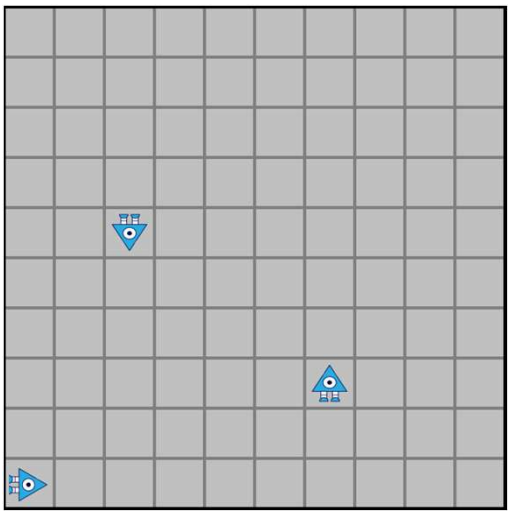
```
for ( int i = 0; i < 4; i++ )
    for ( Robot robby : robots ) {
        robby.putCoin();
        robby.move();
    }
```

Oben links sehen Sie noch einmal die modifizierte Schleife von eben, unten rechts nun die angekündigte Kurzform für for-Schleifen über Arrays. Beide Codefragmente sind wieder äquivalent.

In dieser Kurzform unten rechts gibt man den Komponententyp und einen neuen Namen an, nach dem Doppelpunkt dann noch den Namen des Arrays. Es bleibt dabei, dass die Schleife so viele Durchläufe hat wie das Array Komponenten, einen für jeden der Indizes, und dass die Indizes aufsteigend durchlaufen werden.

Array von Robot

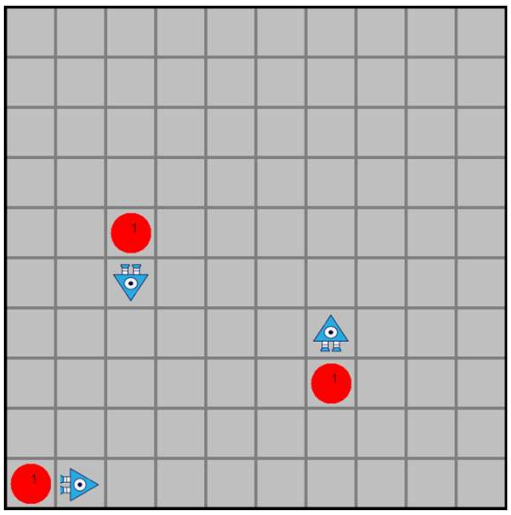
TECHNISCHE UNIVERSITÄT DARMSTADT



So sieht die Situation unmittelbar nach Einrichtung der drei Roboter aus, also unmittelbar vor Beginn der äußeren for-Schleife.

Array von Robot

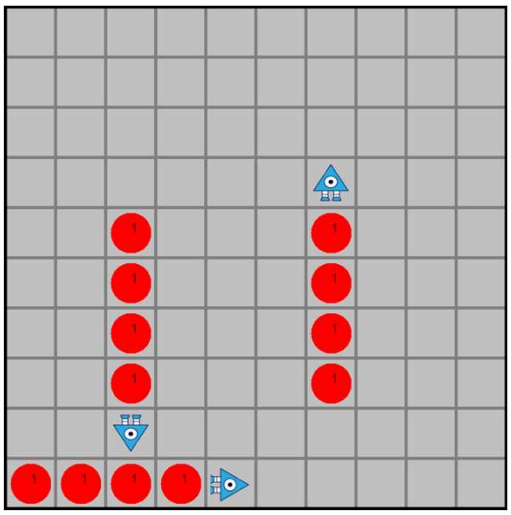
TECHNISCHE UNIVERSITÄT DARMSTADT



Und so sieht die Situation nach einem Durchlauf durch die äußere for-Schleife aus. Jeder der drei Roboter hat eine Münze auf dem Feld abgelegt, auf das er ursprünglich platziert wurde, und ist dann einen Schritt in seiner jeweiligen Richtung gegangen.

Array von Robot

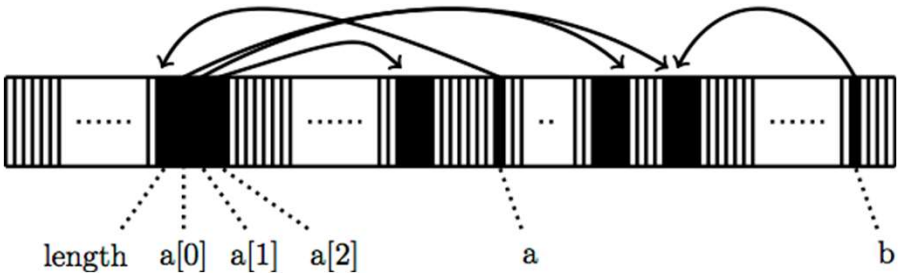
TECHNISCHE UNIVERSITÄT DARMSTADT



Das ist die finale Situation nach dem Ende der äußeren for-Schleife, also nach vier Durchläufen. Viermal hat jeder Roboter eine Münze abgelegt und einen Schritt vorwärts gemacht.

Array von X

TECHNISCHE UNIVERSITÄT DARMSTADT




```
X[ ] a = new X[3];
a[0] = new X();
a[2] = new X();
```

```
for ( X b : a ) {
    .........
}
```

Um den Blick zu weiten, sehen wir uns dasselbe wieder bei einer Klasse X anstelle von Klasse Robot an. Sie sehen, es ist alles analog.

TECHNISCHE UNIVERSITÄT DARMSTADT

Schachbrett mit Array von Robotern

***Erinnerung*: In Kapitel 01b, Abschnitt „Rechteck nicht mehr vollständig, sondern schachbrettartig füllen“, hatten wir einen Roboter durch einen rechteckigen Bereich der World laufen und ein Schachbrett aus Münzen erzeugen lassen.**

Wir zeichnen dasselbe Schachbrett nochmals, aber diesmal mit einem separaten Roboter für jede Zeile. Diese Roboter sind wieder die Komponenten eines Arrays von Klasse Robot.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

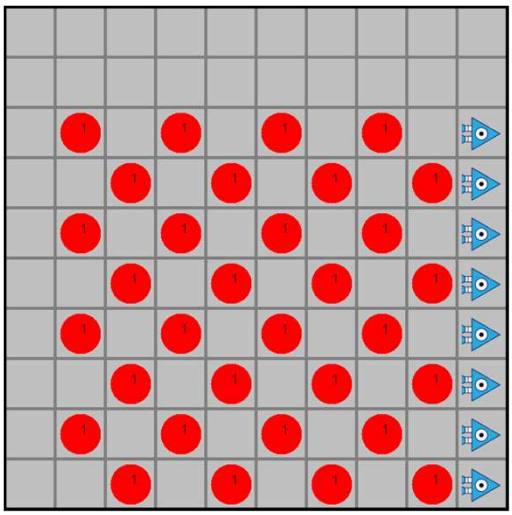
```
import fopbot.*;
import static Direction.*;

public class ChessArray implements Directions {
    public static void main ( String[] args ) {
        World.setVisible(true);
        .....
    }
}
```

Die Datei ist ChessArray.java.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

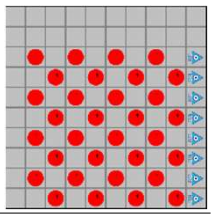


Und das wird das Ergebnis sein: wieder das Schachbrett wie in Kapitel 01b, aber jetzt acht Roboter, je einer für jede Zeile. Jeder Roboter geht achtmal von links nach rechts vorwärts und legt dabei insgesamt vier Münzen ab.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    boolean coinToBePut = ( i % 2 == 0 );
    for ( ... 8 ... ) {
       if ( coinToBePut )
          lineRobots[i].putCoin();
       coinToBePut = ! coinToBePut;
       lineRobots[i].move();
    }
}
```

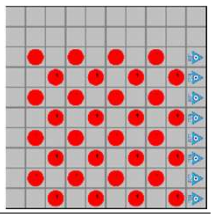


Wie angekündigt, re-implementieren wir unser Schachbrettzeichenprogramm also nun mit einem Array von Robotern. Wir löschen den hier gezeigten Code auf der nächsten Folie weitgehend und bauen ihn schrittweise wieder auf. Sie kennen das ja von früheren Beispielen.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
!!! ausfuellen !!!
```

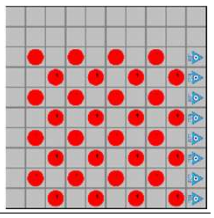


Jede Zeile soll von einem separaten Roboter bearbeitet werden. Wir brauchen also ein Array von insgesamt acht Robotern.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ] lineRobots = new Robot [ 8 ];
!!! ausfuellen !!!
```

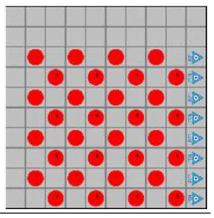


Dass jeder Roboter für eine Zeile zuständig ist, wird hier im Namen des Arrays schon angedeutet.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    !!! ausfuellen !!!
}
```

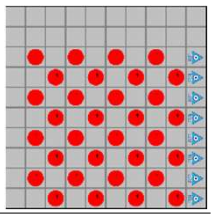


Mit dieser Schleife durchlaufen wir alle acht Zeilen, wobei wir aufpassen müssen, da die Indizes eines Arrays ja immer mit 0 beginnen. Daher muss der Roboter an Arrayindex 0 für die erste Zeile zuständig sein, der Roboter an Arrayindex 1 für die zweite Zeile und so weiter.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    !!! ausfuellen !!!
}
```

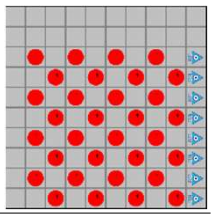


Für jede Zeile richten wir also einen Roboter ein, der in seiner Zeile natürlich genau vier Münzen ablegt. Er steht initial in der zweiten Spalte des Schachbretts, also in der Spalte mit Nummer 1, und er schaut nach rechts, um seine Zeile von links nach rechts zu durchlaufen.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    !!! ausfuellen !!!
}
```

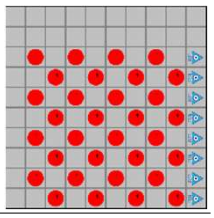


Da die Zeilennummern genauso wie die Arrayindizes mit 0 beginnen, passen hier beide Zahlen zusammen.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    !!! ausfuellen? !!!
    for ( ... 8 ... ) {
        !!! ausfuellen !!!
    }
}
```

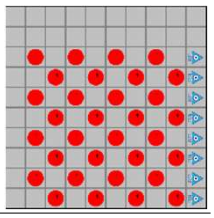


Jeder Roboter soll achtmal vorwärtsgehen und dabei an jeder einzelnen Spalte entscheiden, ob er eine Münze ablegt oder nicht.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    !!! ausfuellen? !!!
    for ( ... 8 ... ) {
        !!! ausfuellen !!!
    }
}
```

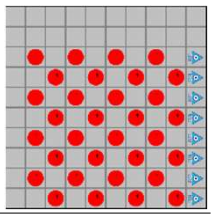


In diesem Moment können wir noch nicht sicher sagen, ob hier wirklich noch etwas hin muss, daher das Fragezeichen.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    !!! ausfuellen? !!!
    for ( ... 8 ... ) {
        !!! ausfuellen !!!
        lineRobots[i].move();
    }
}
```

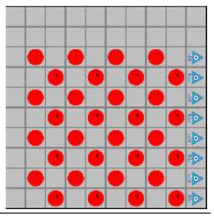


Da die Roboter schon auf der ersten Spalte des Schachbretts platziert werden, soll der Roboter zuerst gegebenenfalls eine Münze ablegen, und erst dann soll er weitergehen. Also setzen wir die Vorwärtsbewegung an das Ende des Schleifenrumpfs.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    !!! ausfuellen? !!!
    for ( ... 8 ... ) {
       if ( coinToBePut )
          !!! ausfuellen !!!
       lineRobots[i].move();
    }
}
```

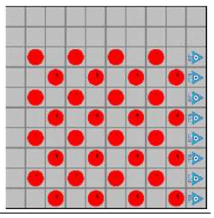


Wir machen es uns an dieser Stelle einfach und sagen, der Wahrheitswert, ob eine Münze abzulegen ist, soll in einer eigenen Variable gespeichert werden. Natürlich geben wir dieser Variable einen sprechenden Namen.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    boolean coinToBePut = !!! ausfuellen !!!
    for ( ... 8 ... ) {
        if ( coinToBePut )
        !!! ausfuellen !!!
        lineRobots[i].move();
    }
}
```

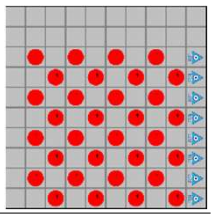


Offensichtlich ist diese Variable boolesch und muss außerhalb der inneren for-Schleife definiert und initialisiert werden.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    boolean coinToBePut = !!! ausfuellen !!!
    for ( ... 8 ... ) {
       if ( coinToBePut )
          lineRobots[i].putCoin();
       !!! ausfuellen !!!
       lineRobots[i].move();
    }
}
```

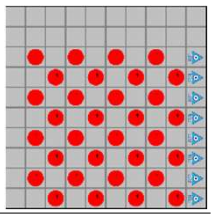


Dann ist auch klar, was im if-Teil passieren muss, nämlich eine Münze ablegen.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    boolean coinToBePut = !!! ausfuellen !!!
    for ( ... 8 ... ) {
       if ( coinToBePut )
          lineRobots[i].putCoin();
       coinToBePut = ! coinToBePut;
       lineRobots[i].move();
    }
}
```

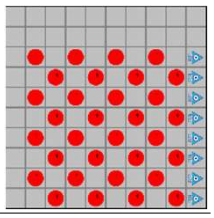


Die Variable, in der gespeichert ist, ob eine Münze zu setzen ist oder nicht, muss natürlich Spalte für Spalte jeweils ihren Wert umdrehen.

Schachbrett mit Array

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot [ ]  lineRobots = new Robot [ 8 ];
for ( int i = 0; i < lineRobots.length; i++ ) {
    lineRobots[i] = new Robot ( 1, i, RIGHT, 4 );
    boolean coinToBePut = ( i % 2 == 1 );
    for ( ... 8 ... ) {
       if ( coinToBePut )
          lineRobots[i].putCoin();
       coinToBePut = ! coinToBePut;
       lineRobots[i].move();
    }
}
```



Wir müssen noch die boolesche Variable coinToBePut für jede Zeile jeweils initialisieren. Bei den geraden Zeilennummern soll sie initial false sein, bei den ungeraden soll sie initial true sein.

TECHNISCHE UNIVERSITÄT DARMSTADT

Array von primitiven Datentypen

Bis jetzt hatten wir nur Arrays von Klassen. Arrays von primitiven Datentypen sind genauso möglich.

Arrays von primitiven Datent.

TECHNISCHE UNIVERSITÄT DARMSTADT

```
double[ ] vector = new double[123];

vector[0] = 3.14;

for ( int i = 1; i < vector.length; i++ )
   vector[i] = vector[i-1] + i;

double sum = 0;

for ( int i = 0; i < vector.length; i++ )
   sum += vector[i];
```

Sie sehen hier als einfaches Beispiel ein Array mit Komponententyp double. Abgesehen vom Komponententyp gibt es hier nichts Neues zu sehen.

Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```
double [ ] a = new double [ 5 ];
a[0] = 2.771;
a[1] = 5;
a[2] = -3.14;
a[3] = 0;
a[2*5-6] = 3.14 * 2.71 + 1.41;
a[-1] = 1;
a[5] = 2;
```

Wir spielen noch einmal kurz mit einem kleineren Array ein bisschen herum.

Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```
double [ ] a = new double [ 5 ];
a[0] = 2.771;
a[1] = 5;
a[2] = -3.14;
a[3] = 0;
a[2*5-6] = 3.14 * 2.71 + 1.41;
a[-1] = 1;
a[5] = 2;
```

Die einzelnen double-Werte mit Indizes 0 bis 4 lassen sich mit beliebigen double-Ausdrücken initialisieren.

Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```
double [ ] a = new double [ 5 ];
a[0] = 2.771;
a[1] = 5;
a[2] = -3.14;
a[3] = 0;
a[2*5-6] = 3.14 * 2.71 + 1.41;
a[-1] = 1;
a[5] = 2;
```

Der Index einer Arraykomponente kann durch einen beliebigen int-Ausdruck spezifiziert werden, der dann einen Wert im Bereich 0 bis length – 1 haben muss.

Arrays

TECHNISCHE UNIVERSITÄT DARMSTADT

```
double [ ] a = new double [ 5 ];
a[0] = 2.771;
a[1] = 5;
a[2] = -3.14;
a[3] = 0;
a[2*5-6] = 3.14 * 2.71 + 1.41;
a[-1] = 1;
a[5] = 2;
```

Wird ein Index außerhalb des Bereichs angegeben, dann wird der Programmlauf mit einer Fehlermeldung abgebrochen.

***Vorgriff*: Im Kapitel 05 zur Fehlerbehandlung werden wir sehen, dass dieser Programmabbruch auch vermieden und durch eine eigene Fehlerbehandlung ersetzt werden kann.**

Damit beenden wir diesen Abschnitt und das ganze Kapitel.