TECHNISCHE UNIVERSITÄT DARMSTADT

Kapitel 01e:
Klassen und Methoden mit FopBot

Karsten Weihe

Wir nähern uns dem Thema Klassen und Methoden in Java, indem wir die Klasse, die wir schon intensiv kennen, nicht vollständig, aber zumindest teilweise nachbauen: die Klasse Robot aus FopBot.

TECHNISCHE UNIVERSITÄT DARMSTADT

Erste eigene Enumeration

Oracle Java Tutorials: Enum Types

Dafür brauchen wir aber eine kleine Vorarbeit, nämlich Direction nachzubauen. Den Begriff Enumeration in dieser Abschnittsüberschrift werden wir gleich genauer sehen.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public enum MyDirection {
    UP,
    DOWN,
    LEFT,
    RIGHT
}
```

Dies ist unser Nachbau von Direction.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public enum MyDirection {
    UP,
    DOWN,
    LEFT,
    RIGHT
}
```

Um klarzustellen, dass dies nicht das Original, sondern eben ein Nachbau ist, nennen wir ihn leicht anders.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public enum MyDirection {
   UP,
   DOWN,
   LEFT,
   RIGHT
}
```

Dieses Java-Schlüsselwort zeigt an, worum es geht. Man nennt dies im Englischen wie im Deutschen eine *Enumeration*.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public enum MyDirection {
   UP,
   DOWN,
   LEFT,
   RIGHT
}
```

Der Begriff Enumeration trifft die Sachlage gut, denn tatsächlich wird ja etwas aufgezählt, in diesem Fall die vier Richtungen.

Dies ist ein weiteres Beispiel für Konstanten: Eine Enumeration ist eine Zusammenfassung mehrerer Konstanten zu einer Einheit. Die einzelnen Konstanten in einer Enumeration werden durch Kommas separiert.

***Erinnerung*: Wir hatten schon im Kapitel 01a, Abschnitt zu Identifiern, gesagt, dass es bei Namen von Konstanten Konvention ist, alle Buchstaben großzuschreiben.**

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public enum MyDirection {
    UP,
    DOWN,
    LEFT,
    RIGHT
}
```

Die Konstanten sind in geschweifte Klammern zu setzen, und zwar auch dann, wenn es nicht mehr als eine Konstante gibt.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public enum MyDirection {
    UP,
    DOWN,
    LEFT,
    RIGHT
}
```

Das Schlüsselwort public am Beginn der Definition dieser Enumeration sorgt dafür, dass diese Enumeration auch außerhalb dieser Quelltextdatei verwendbar ist. Das ist in diesem Fall notwendig, denn wir wollen MyDirection ab der nächsten Folie in unserer eigenen Roboterklasse benutzen, die wir in eine andere Quelldatei schreiben.

Nur eine einzige Definition in einer Quelldatei darf public sein, und dann muss der Name der Quelldatei identisch mit dem Namen der definierten Entität sein. Die Enumeration MyDirection muss also in einer Quelldatei namens MyDirection.java sein, die durch den Java-Compiler dann in Java Bytecode in einer Datei namens MyDirection.class übersetzt wird.

***Vorgriff*: Solche Zugriffsrechte werden in Kapitel 01f im Abschnitt „Zugriffsrechte und Packages“ systematisch behandelt.**

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MyDirection dir = MyDirection.DOWN;
final MyDirection DIR = MyDirection.LEFT;
.....
dir = MyDirection.RIGHT;
DIR = MyDirection.UP;
.....
if ( dir == MyDirection.LEFT )
```

So sieht die Verwendung von MyDirection beispielhaft aus.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MyDirection dir = MyDirection.DOWN;
final MyDirection DIR = MyDirection.LEFT;
.....
dir = MyDirection.RIGHT;
DIR = MyDirection.UP;
.....
if ( dir == MyDirection.LEFT )
```

Von einem Enumerationstyp kann man genauso Variable einreichten wie von jedem anderen Typ.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MyDirection dir = MyDirection.DOWN;
final MyDirection DIR = MyDirection.LEFT;
.....
dir = MyDirection.RIGHT;
DIR = MyDirection.UP;
.....
if ( dir == MyDirection.LEFT )
```

Der Unterschied zu anderen Typen ist, dass ein Enumerationstyp nur eine vordefinierte Menge von Objekten hat und man keine eigenen Objekte von einem Enumerationstyp erzeugen kann. Die Namen, die wir im Enumerationstyp aufzählen, sind Referenzen auf jeweils ein Objekt.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MyDirection dir = MyDirection.DOWN;
final MyDirection DIR = MyDirection.LEFT;
.....
dir = MyDirection.RIGHT;
DIR = MyDirection.UP;
.....
if ( dir == MyDirection.LEFT )
```

Wie üblich, darf der Inhalt einer Konstanten nicht verändert werden.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MyDirection dir = MyDirection.DOWN;
final MyDirection DIR = MyDirection.LEFT;
.....
dir = MyDirection.RIGHT;
DIR = MyDirection.UP;
.....
if ( dir == MyDirection.LEFT )
```

Wichtigste Funktionalität soweit ist der Test auf Gleichheit beziehungsweise Ungleichheit.

Erste eigene Enumeration

TECHNISCHE UNIVERSITÄT DARMSTADT

```
import static MyDirection.*;

.........

MyDirection dir = DOWN;
```

Eine Frage ist allerdings noch offen: Wieso konnten wir in unseren FopBot-Beispielen auf die einzelnen Richtungen zugreifen, ohne immer den Namen der Enumeration davorzuschreiben? Den Mechanismus dazu in der Sprache Java sehen Sie hier.

***Vorgriff*: Das Schlüsselwort static erscheint hier erst einmal unmotiviert, was soll hier, bitte schön, statisch sein? In Kapitel 03b und 03c werden wir unter Stichworten wie Klassenkonstanten und Klassenmethoden sehen, woher es kommt, dass dieses Schlüsselwort dafür verwendet wird.**

TECHNISCHE UNIVERSITÄT DARMSTADT

Die zu erstellende Klasse MyRobot als UML-Klassendiagramm

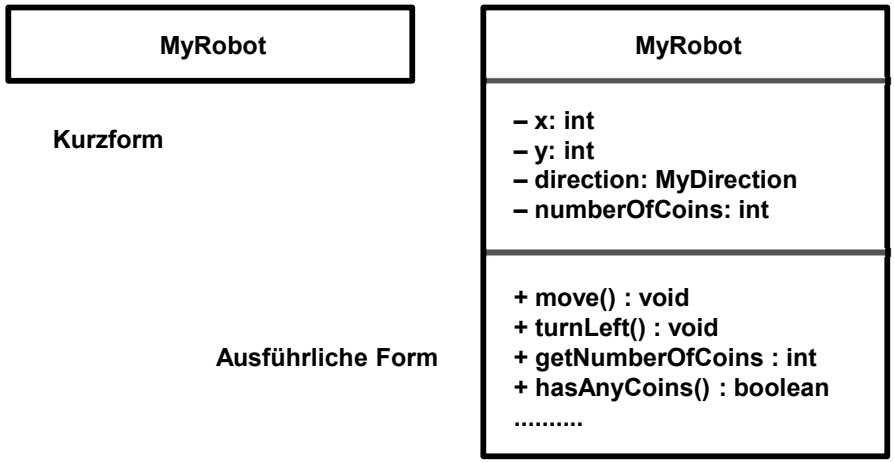
Nun kommen wir zum Nachbau der Klasse Robot. Dieser Nachbau wird sich über mehrere Abschnitte erstrecken, die verschiedene Aspekte einer Klassendefinition aufzeigen.

Der Einfachheit halber lassen wir alles aus, was benötigt wird, um einen Roboter in die FopBot-World zu zeichnen.

Bevor wir überhaupt an die Implementation selbst gehen, abstrahieren wir in diesem Abschnitt gleich schon davon. Die Abkürzung UML steht für *Unified Modelling Language*. UML ist ein De-facto-Standard für Modellierung im Bereich Software und IT. Unter anderem sind Klassendiagramme standardisiert, also graphische Darstellungen von Klassen u.ä. sowie deren Beziehungen untereinander.

Die Klasse MyRobot in UML

TECHNISCHE UNIVERSITÄT DARMSTADT



Links und rechts sehen Sie die Kurz- beziehungsweise Langform der graphischen Darstellung der Klasse MyRobot in UML. Die Kurzform bedarf wohl keiner Erläuterung. Die einzelnen Bestandteile der Langform schauen wir uns im Folgenden immer parallel im Java-Code und in UML an.

TECHNISCHE UNIVERSITÄT DARMSTADT

Der Kopf der Klassendefinition

Oracle Java Tutorials: Declaring Classes

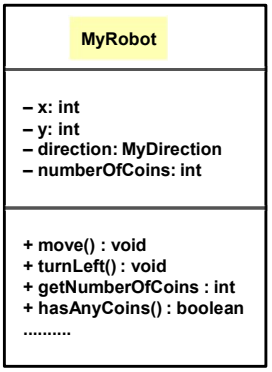
Wir betrachten also die verschiedenen Bestandteile einer Klasse der Reihe nach, beginnend mit dem Kopf.

Der Einfachheit halber lassen wir alles aus, was benötigt wird, um einen Roboter in die FopBot-World zu zeichnen.

Der Kopf der Klassendefinition

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   !! ausfuellen !!
}
```

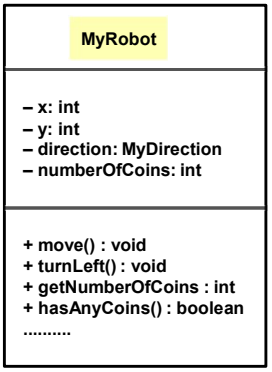


So wie links sieht in der einfachsten Form der Rahmen für eine Klassendefinition in Java aus. Rechts unten sehen Sie nochmals das UML-Klassendiagramm in Langform.

Der Kopf der Klassendefinition

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
    !! ausfuellen !!
}
```

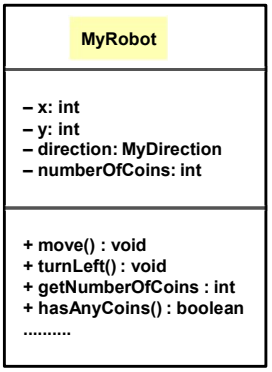


Der gelb unterlegte Text links ist der *Kopf* der Klasse. Sie sehen verschiedene Bestandteile des Kopfes in bestimmter Reihenfolge.

Der Kopf der Klassendefinition

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   !! ausfuellen !!
}
```



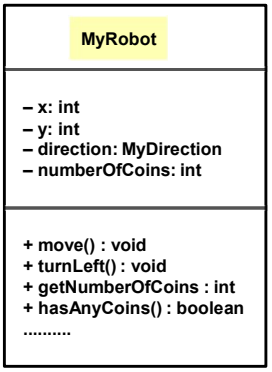
An dieser Stelle steht der Name der neuen Klasse.

***Vorgriff*: In Kapitel 01f und 01g werden wir sehen, dass zwischen dem Namen der Klasse und der darauffolgenden öffnenden Klammer noch andere Bestandteile des Kopfes kommen können.**

Der Kopf der Klassendefinition

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   !! ausfuellen !!
}
```



Unmittelbar vor dem Namen der Klasse, also wo soeben bei der Enumeration das Schlüsselwort enum stand, muss das Schlüsselwort class stehen, um erstens anzuzeigen, dass hier eine Klasse und nicht etwa eine Enumeration definiert wird, und zweitens, dass das allernächste Wort der Name der neuen Klasse ist.

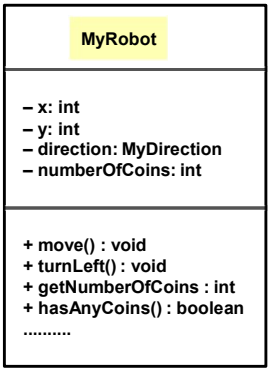
***Nebenbemerkung*: Es ist kein Zufall, dass das dieselbe Stelle ist, an der eben der Name MyDirection der Enumeration stand. In Programmiersprachen ist es zum besseren Verständnis üblich, Dinge, die in gewisser Weise parallel zueinander sind, auch sprachlich möglichst parallel zu gestalten.**

***Vorgriff*: In Kapitel 01g werden wir neben Enumerationen und Klassen noch eine dritte Art sehen, Interfaces, bei denen der Kopf genauso aufgebaut sein wird: erst das Schlüsselwort interface, dann der Name.**

Der Kopf der Klassendefinition

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
    !! ausfuellen !!
}
```

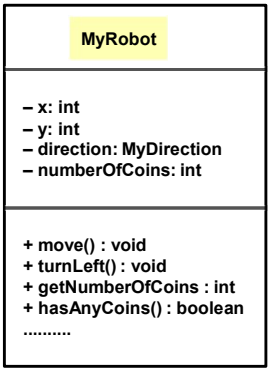


Auch hier setzen wir das Schlüsselwort public an den Anfang, um dafür zu sorgen, dass die Klasse MyRobot außerhalb dieser Quelldatei verwendbar ist. Der Name der Quelldatei ist also MyRobot.java, der Name der übersetzten Datei somit MyRobot.class.

Der Kopf der Klassendefinition

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
    !! ausfuellen !!
}
```



Was eine Klasse inhaltlich ausmacht, findet sich alles zusammen im *Methodenrumpf*, das heißt, in geschweiften Klammern nach dem Methodenkopf. Der Methodenrumpf unserer Klasse wird verschiedenste Bestandteile haben, die Sie überblicksweise schon einmal rechts im UML-Klassendiagramm sehen. Aber analog zu Enumerationen: Auch in den Fällen, in denen der Methodenrumpf nur *einen* Bestandteil hat, dürfen die geschweiften Klammern nicht weggelassen werden (der Methodenrumpf könnte sogar leer sein, also ein leeres geschweiftes Klammerpaar).

TECHNISCHE UNIVERSITÄT DARMSTADT

Die Attribute der Klasse

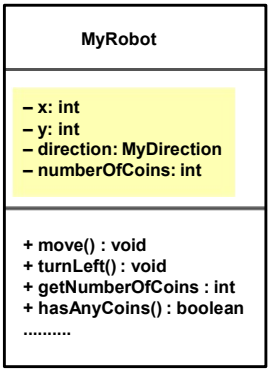
Oracle Java Tutorials:
Declaring Member Variables

In der Regel möchten wir, dass ein Objekt einer Klasse irgendwelche Daten enthält. Die einzelnen Datenelemente nennen wir die *Attribute* der Klasse.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   private int x;
   private int y;
   private MyDirection direction;
   private int numberOfCoins;

   !!! ausfuellen !!!
}
```

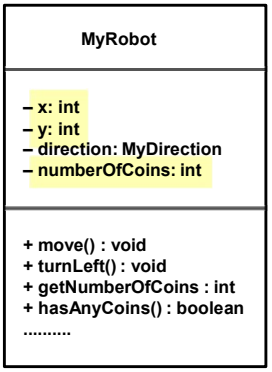


Und das sind die Attribute, die wir jedem Objekt der neuen Klasse mitgeben wollen. Für Attribute, die nicht final sind, gelten dieselben Namenskonventionen wie für Variable: Der erste Buchstabe ist kleingeschrieben, Wortanfänge im Innern sind großgeschrieben.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   private int x;
   private int y;
   private MyDirection direction;
   private int numberOfCoins;

   !!! ausfuellen !!!
}
```



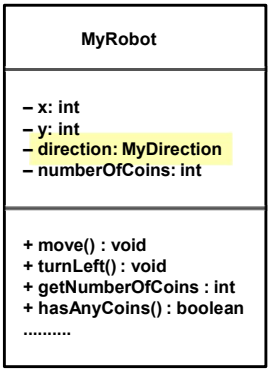
Das sind die uns schon bekannten Attribute vom primitiven Datentyp int.

Dabei enthält das Attribut x die horizontale Koordinate des Feldes, in dem der Roboter steht, und y enthält die vertikale Koordinate. Das heißt, der Roboter steht in der Spalte Nummer x und in der Zeile Nummer y. Da die Spalten und Zeilen beginnend mit 0 durchnummeriert sind, steht der Roboter also in der (x+1)-ten Spalte und (y+1)-ten Zeile.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   private int x;
   private int y;
   private MyDirection direction;
   private int numberOfCoins;

   !!! ausfuellen !!!
}
```



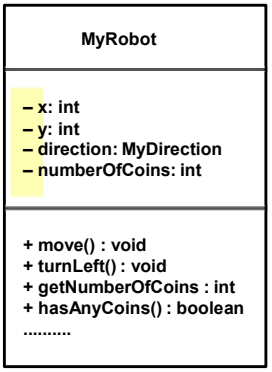
Das Attribut direction kann nur vier Werte annehmen, nämlich die vier Konstanten, die wir soeben in der Enumeration MyDirection definiert hatten.

Wie schon gesagt: Die Klasse MyDirection ist in MyDirection.java beziehungsweise übersetzt in MyDirection.class zu finden. Da MyDirection public definiert ist, kann MyDirection außerhalb von MyDirection.java – etwa hier in MyRobot.java – verwendet werden.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

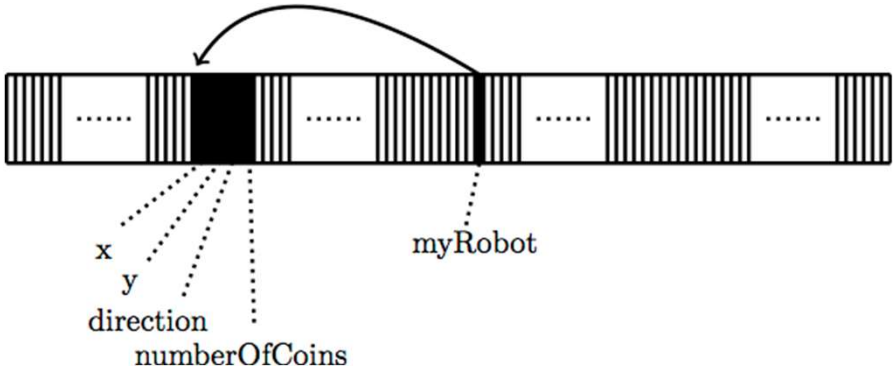
```
public class MyRobot {
   private int x;
   private int y;
   private MyDirection direction;
   private int numberOfCoins;

   !!! ausfuellen !!!
}
```



Die Schlüsselwörter public und private gehören zum Thema Zugriffsrechte, das wir uns später in Kapitel 01f, Abschnitt zu Zugriffsrechten und Packages, systematischer ansehen werden. Hier soll es erst einmal reichen, kurz zu erwähnen, dass die Werte dieser vier Attribute dank private nicht direkt von außerhalb der Klasse MyRobot gelesen oder mit neuen Werten überschrieben werden können. Das geht nur indirekt, über Methoden der Klasse wie move und turnLeft, die wir ebenfalls gleich nachbauen werden. Private Bestandteile einer Klasse werden in UML mit einem Minus gekennzeichnet.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

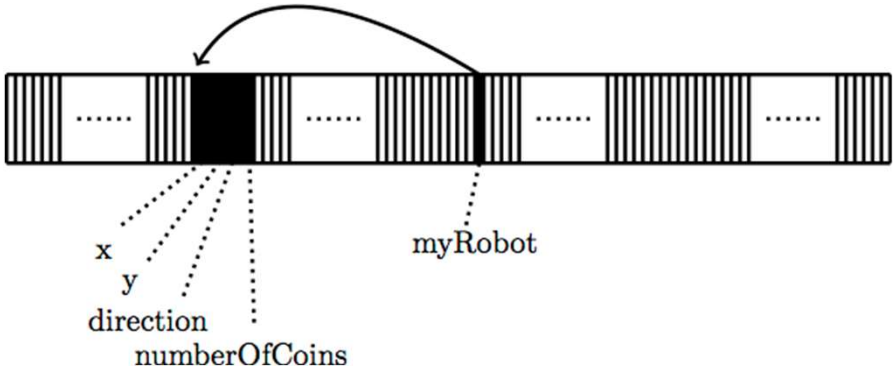


MyRobot myRobot = new MyRobot (3, 1, UP, 0);

So wie wir uns Objekte und Referenzen der Klasse Robot im Speicher veranschaulicht hatten, genau so können wir uns auch Objekte unseres Nachbaus vorstellen.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT



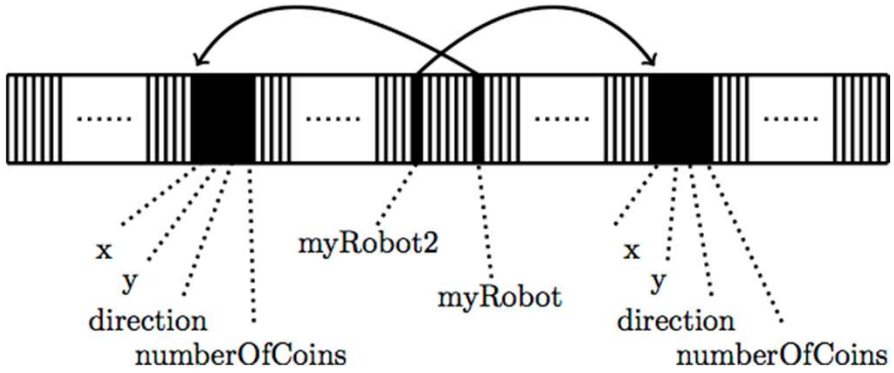

```
MyRobot myRobot = new MyRobot ( 3, 1, UP, 0 );
```

Hier wird eine spezielle Methode aufgerufen, dir wir in einem späteren Abschnitt dieses Kapitels ebenfalls nachzubauen haben.

***Vorgriff*: Diese spezielle Methode, die offenbar unter anderem dazu da ist, die Attribute eines Objektes einer Klasse zu initialisieren, heißt *Konstruktor*.**

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

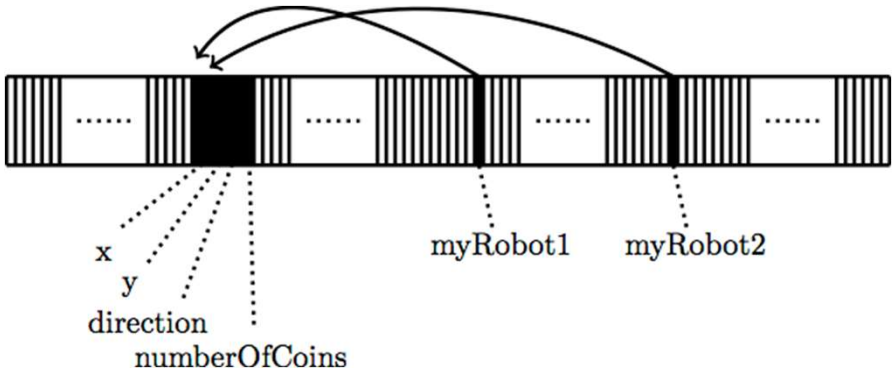


MyRobot myRobot = new MyRobot (3, 1, UP, 0);
MyRobot myRobot2 = new MyRobot (2, 1, RIGHT, 5);

Schon weit vorn in Kapitel 01a, haben wir diese Folie gesehen: So kann man sich das im Speicher vorstellen, wenn zwei Referenzen und zu jeder Referenz jeweils ein eigenes Objekt eingerichtet werden. Alle Attribute gibt es dann zweimal, jeweils mit unterschiedlichen Werten.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

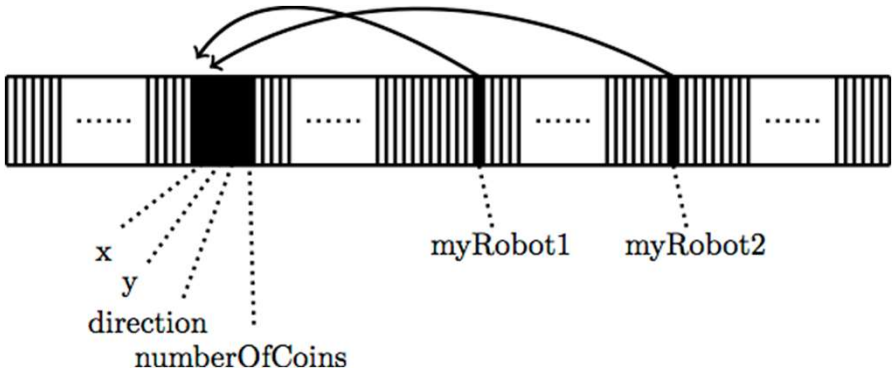



```
MyRobot myRobot1 = new MyRobot ( ..... );
MyRobot myRobot2 = myRobot1;
```

Diese Situation ist etwas anders gelagert: Wieder werden zwei Referenzen eingerichtet, aber diesmal nur ein Objekt. Durch die Zuweisung in der zweiten Zeile wird die Adresse, die in myRobot1 gespeichert ist, in myRobot2 kopiert. Mit anderen Worten: Beide Referenzen verweisen nun auf dasselbe Objekt.

Die Attribute der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT




```
myRobot1.x = 1;     // myRobot2.x == 1
myRobot2.y = 5;     // myRobot1.y == 5
```

Die Attribute, die über myRobot1 beziehungsweise myRobot2 so angesprochen werden könnten (wenn sie denn public wären), sind nun identisch. Wird der Wert eines Attributs über die eine Referenz gesetzt, dann bekommt man über die andere Referenz natürlich denselben Wert.

TECHNISCHE UNIVERSITÄT DARMSTADT

Die void-Methoden der Klasse

Oracle Java Tutorials: Defining Methods

Nun geht es an die erste Art von Methoden. Wir werden gleich sehen, was void-Methode bedeutet: Das ist eine Methode, die nichts zurückliefert, also etwa die Methoden move, turnLeft, pickCoin und putCoin.

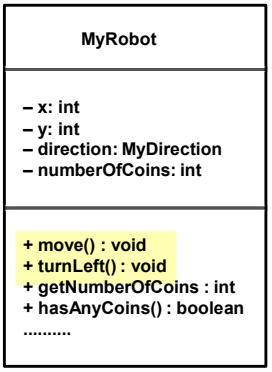
Im Gegensatz dazu liefern beispielsweise die Methoden getX und getY jeweils einen int-Wert und die Methode hasAnyCoins einen booleschen Wert zurück. Methoden wie diese, die einen Wert zurückliefern, sehen wir uns gleich nach den void-Methoden in einem eigenen Abschnitt an.

Wir werden für Klasse MyRobot nicht alle Methoden nachbauen, die die Klasse Robot hat. Aber diejenigen, die wir nachbauen, sollten alles Grundlegende zur Implementation von Methoden in Java zeigen.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   private int x;
   private int y;
   private MyDirection direction;
   private int numberOfCoins;

   !!! ausfuellen !!!
}
```



Hier, hinter die Attribute schreiben wir dann die Methoden. Die Reihenfolge ist eigentlich egal, Attribute und Methoden dürfen auch beliebig gemischt auftreten, Aber es ist üblich, wie in UML rechts erst die Attribute und dann die Methoden zu definieren.

Das Plus bei den Methoden besagt, dass diese Methoden public sind.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

Als erstes nehmen wir uns die Methode move vor.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

Was Sie hier oben sehen, ist der *Kopf* der Methode move. Sie sehen, dass er aus verschiedenen Bestandteilen in vorgegebener Reihenfolge besteht.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

An dieser Stelle sehen Sie den Namen der Methode, ...

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

... gefolgt von der Parameterliste der Methode. Wir haben schon beim Aufruf der Methode move ein leeres Klammerpaar wie hier gesehen. Das bedeutet, dass move keine Parameter hat.

***Nebenbemerkung*: Der Konstruktor, den wir zur Einrichtung von Objekten der Klasse Robot verwendet hatten und den wir ebenfalls noch gleich nachbauen werden, haben Sie schon als Beispiel dafür gesehen, wie der Aufruf einer Methode mit mehreren Parametern aussieht.**

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

Das Schlüsselwort public hier besagt im Gegensatz zu private, dass die Methode von außerhalb der Klasse Robot aufgerufen werden kann, wie wir es ja immer gemacht haben. Wir haben schon gesehen, dass deshalb im UML-Klassendiagramm ein Pluszeichen vor dieser Methode steht.

***Vorgriff*: Würde hier statt dessen private stehen, dürften nur die Methoden der Klasse MyRobot diese Methode move aufrufen.**

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

Das Schlüsselwort void unmittelbar vor dem Namen der Methode besagt, dass diese Methode keinen Wert zurückliefert. Das haben wir ja beim Aufruf von move bisher auch immer gesehen.

Eine void-Methode ist also eine Methode, bei der an dieser Stelle das Schlüsselwort void steht. Der Sinn von void-Methoden ist, dass sie eben andere Effekte als eine Rückgabe haben. Zum Beispiel die Methode move hat offenbar den Effekt, ein Attribut des Robot-Objektes zu ändern, nämlich das Attribut x für die Zeile oder das Attribut y für die Spalte, je nachdem, ob der Roboter gerade vertikal oder horizontal ausgerichtet ist.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

Die eigentlichen Anweisungen, die beim Aufruf einer Methode auszuführen sind, stehen im *Methodenrumpf*, das heißt, in geschweiften Klammern hinter dem Methodenkopf. Auch in dem Fall, dass eine Methode nur eine einzige Anweisung enthält, dürfen die geschweiften Klammern nicht weggelassen werden.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else     //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

So wie wir Methode move hier implementiert haben, enthält der Methodenrumpf tatsächlich nur eine einzige Anweisung, die allerdings sehr komplex ist: Der else-Teil der ersten Verzweigung besteht wieder aus einer Verzweigung mit if und else, und dieser letztere else-Teil besteht ebenfalls aus einer Verzweigung mit if und else.

Damit ergeben sich insgesamt vier Fälle, nämlich die vier Richtungen, in die der Roboter momentan zeigen könnte. In jedem dieser Fälle muss die Änderung spezifisch sein: Entweder wird die Zeilennummer um 1 erhöht oder um 1 vermindert, oder die Spaltennummer wird um 1 erhöht oder um 1 vermindert.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void move () {
      if ( direction == MyDirection.UP )
         y++;
      else if ( direction == MyDirection.RIGHT )
         x++;
      else if ( direction == MyDirection.DOWN )
         y – –;
      else    //  i.e., direction == MyDirection.LEFT
         x – –;
   }
   ..........
}
```

Für die letzte der vier Richtungen muss natürlich nichts mehr getestet werden, denn wenn der Roboter nicht nach oben, rechts oder unten zeigt, muss er ja zwangsläufig nach links zeigen. Zur besseren Verständlichkeit ist es in solchen komplexen Verzweigungskaskaden aber sinnvoll, den letzten Fall als Kommentar explizit dazuzuschreiben.

***Nebenbemerkung*: Die Abkürzung „i.e.“ steht für das lateinische „id est“. Sie wird im Englischen häufig und im Deutschen bisweilen für „das heißt“ verwendet und im Englischen üblicherweise „that is“ ausgesprochen.**

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
if ( direction == MyDirection.UP )
    y++;
else if ( direction == MyDirection.RIGHT )
    x++;
else if ( direction == MyDirection.DOWN )
    y – –;
else
    x – –;
```

```
if ( direction == MyDirection.UP )
    y++;
else
    if ( direction == MyDirection.RIGHT )
        x++;
    else
        if ( direction == MyDirection.DOWN )
          y – –;
        else
          x – –;
```

Ein Wort noch zur Formatierung dieser Verzweigungskaskade in Methode move: Allgemein üblich ist eher eine Einrückung der einzelnen Zeilen wie rechts oben auf dieser Folie gezeigt. Diese standardmäßige Einrückung unterstreicht die programmiersprachliche Struktur dieser komplexen Anweisung.

Speziell bei solchen Verzweigungskaskaden, in denen die einzelnen Fälle eigentlich inhaltlich auf derselben Ebene sind, ist es nicht unüblich, wie unten links – also wie auf der vorhergehenden Folie – einzurücken, denn dadurch wird die inhaltliche Gleichartigkeit der einzelnen Fälle unterstrichen.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public void turnLeft () {
      if ( direction == MyDirection.UP )
         direction = MyDirection.LEFT;
      else if ( direction == MyDirection.LEFT )
         direction = MyDirection.DOWN;
      else if ( direction == MyDirection.DOWN )
         direction = MyDirection.RIGHT;
      else     //  i.e., direction == MyDirection.RIGHT
         direction = MyDirection.UP;
   }
   ..........
}
```

Die Methode turnLeft bietet nichts konzeptionell Neues gegenüber move. Als Übung können Sie die Methode turnLeft für sich selbst Schritt für Schritt so nachvollziehen, wie wir es uns soeben bei Methode move angesehen haben.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MyRobot robot = new MyRobot ( !! ausfuellen !! );
robot.move();
robot.turnLeft();
for ( ..... 4 ..... ) {
    robot.move();
}
```

Völlig analog zur Klasse Robot können Sie Objekte der Klasse MyRobot einrichten und die soeben implementierten Methoden anwenden. Die Methoden hießen gleich wie bei Klasse Robot und hatten auch dieselbe Parameterliste, also sehen die Aufrufe exakt identisch aus.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

```
MyRobot robot = new MyRobot ( !! ausfuellen !! );
robot.move();
robot.turnLeft();
for ( ..... 4 ..... ) {
    robot.move();
}
```

Etwas phantasielos, nennen wir unseren Roboter auch wieder robot. Das ist ja kein Problem, denn der Name robot ist ja nicht an die Klasse Robot gebunden, sondern wir sind im Prinzip frei darin, wie wir den Roboter benennen wollen, solange wir die Regeln zur Definition von Identifiern aus Kapitel 01a, Abschnitt zu Identifiern, einhalten und außerdem nicht zwei verschiedenen Entitäten im selben Kontext denselben Namen geben.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

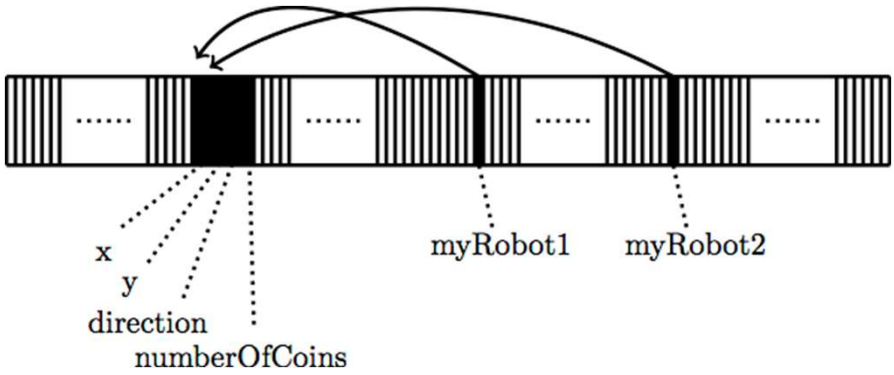
```
MyRobot robot = new MyRobot ( !! ausfuellen !! );
robot.move();
robot.turnLeft();
for ( ..... 4 ..... ) {
    robot.move();
}
```

Den Inhalt der Klammer, also die Position und Ausrichtung des Roboters, lassen wir hier aus, der ist unwichtig für die Botschaft dieser Folie.

***Vorgriff*: Wir haben die Methode, die hier aufgerufen wird und *Konstruktor* heißt, noch gleich zu definieren.**

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT

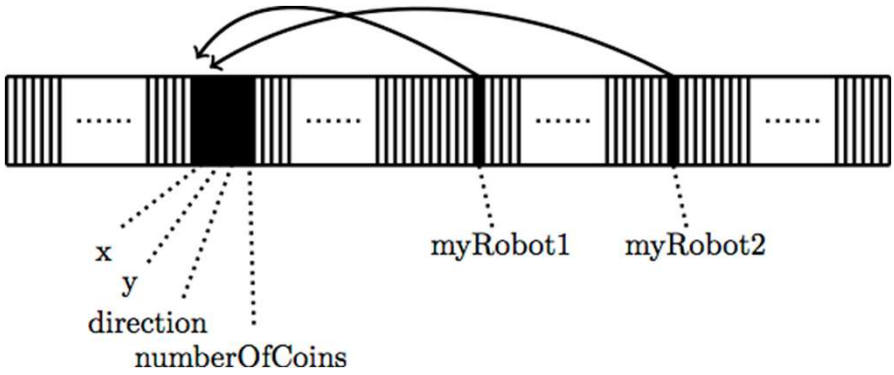


MyRobot myRobot1 = new MyRobot (.....);
MyRobot myRobot2 = myRobot1;

Diese Situation haben wir soeben bei Attributen schon einmal gesehen: Beide Referenzen verweisen auf dasselbe Objekt.

Die void-Methoden der Klasse

TECHNISCHE UNIVERSITÄT DARMSTADT




```
myRobot1.move();
myRobot2.move();     // One Robot makes two steps
```

Egal ob wir jetzt Methode move auf myRobot1 oder myRobot2 anwenden: Methode move wird beide Male auf dasselbe Objekt angewendet, das heißt, beide Male wird x beziehungsweise y desselben Objekts durch move geändert.

TECHNISCHE UNIVERSITÄT DARMSTADT

Ausführung von Methoden

Oracle Java Spezifikation: 15.12.4
Run Time Evaluation of Method Invocation

Wir haben Methoden der Klasse Robot ausgeführt und jetzt auch zwei Methoden der Klasse Robot selbst implementiert. Damit haben wir alles beisammen, um uns genauer anzuschauen, was eigentlich passiert, wenn eine Methode ausgeführt wird.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Springe zum Code von Robot.move mit Referenz robot
3. Führe diesen Code aus
4. Springe zurück und eins weiter zur nächsten Anweisung
5. Führe diese Anweisung aus (i++)
6. Spring zum Code von Robot.move mit Referenz robot2
7. Führe diesen Code aus
8. Springe zurück und eins weiter zur nächsten Anweisung
9. Führe diese Anweisung aus (i – –)

Wir benutzen wieder das alternative Ablaufmodell und schauen uns auf der rechten Seite auf dieser Folie erst einmal nur sehr grob und oberflächlich an, was passiert, wenn wir Java-Code wie den auf der linken Seite ausführen.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Springe zum Code von Robot.move mit Referenz robot
3. Führe diesen Code aus
4. Springe zurück und eins weiter zur nächsten Anweisung
5. Führe diese Anweisung aus (i++)
6. Spring zum Code von Robot.move mit Referenz robot2
7. Führe diesen Code aus
8. Springe zurück und eins weiter zur nächsten Anweisung
9. Führe diese Anweisung aus (i – –)

Rechts sehen Sie die Anweisungen im alternativen Modell, die dem Aufruf von Methode move links entsprechen.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Springe zum Code von Robot.move mit Referenz robot
3. Führe diesen Code aus
4. Springe zurück und eins weiter zur nächsten Anweisung
5. Führe diese Anweisung aus (i++)
6. Spring zum Code von Robot.move mit Referenz robot2
7. Führe diesen Code aus
8. Springe zurück und eins weiter zur nächsten Anweisung
9. Führe diese Anweisung aus (i – –)

Wir haben ja schon gesehen – und sehen es auch hier noch einmal –, dass wir durchaus mehrere Roboter zugleich haben können. Daher muss irgendwie die Information mitgegeben werden, dass Methode move mit einem bestimmten Roboter und nicht mit einem anderen Roboter aufgerufen wird. Wie diese Information in Methode move hineinkommt, werden wir sogleich sehen, wenn wir das alternative Modell etwas verfeinern.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Springe zum Code von Robot.move mit Referenz robot
3. Führe diesen Code aus
4. Springe zurück und eins weiter zur nächsten Anweisung
5. Führe diese Anweisung aus (i++)
6. Spring zum Code von Robot.move mit Referenz robot2
7. Führe diesen Code aus
8. Springe zurück und eins weiter zur nächsten Anweisung
9. Führe diese Anweisung aus (i – –)

Das nun ist natürlich der Code, den wir soeben für Methode move implementiert haben, einmal mit Referenz robot und einmal mit Referenz robot2.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

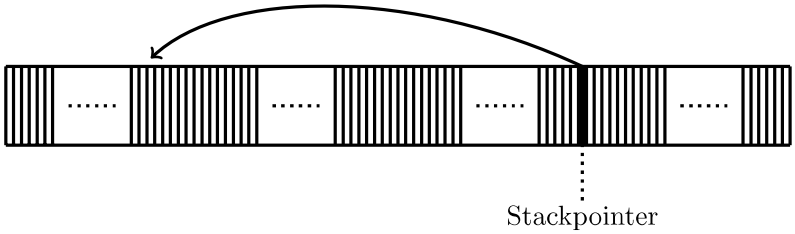
1. Richte int-Variable i ein und setze i auf 1
2. Springe zum Code von Robot.move mit Referenz robot
3. Führe diesen Code aus
4. Springe zurück und eins weiter zur nächsten Anweisung
5. Führe diese Anweisung aus (i++)
6. Spring zum Code von Robot.move mit Referenz robot2
7. Führe diesen Code aus
8. Springe zurück und eins weiter zur nächsten Anweisung
9. Führe diese Anweisung aus (i – –)

Und nach Ausführung der Methode muss der Prozess wieder dahin zurückspringen, wo die Methode aufgerufen wurde, genauer gesagt zur nächsten Anweisung *nach* dem Aufruf. Irgendwie muss also die Information gespeichert werden, von wo aus die Methode aufgerufen wurde, damit der Prozess an die richtige Stelle zurückspringt.

Da eine Methode wie hier an verschiedenen Stellen aufgerufen werden kann, ist die Rücksprungadresse bei jedem Aufruf potentiell eine andere. Die Rücksprungadresse kann also nicht durch den Compiler fest in der Übersetzung der Methode eingesetzt werden, sondern muss bei jedem Aufruf jeweils passend berechnet und gespeichert werden.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT



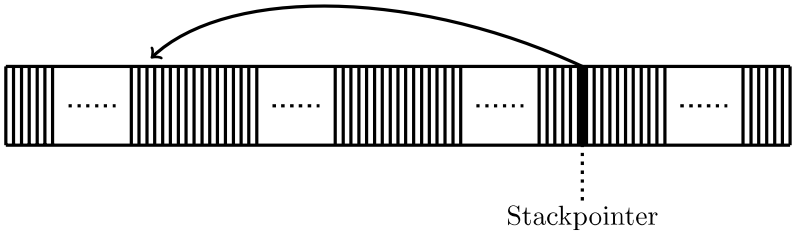
Call-Stack

Wenn ein Programm aufgerufen wird, wird vor Beginn der ersten Methode unter anderem eine Variable eingerichtet, die eine Adresse speichert. Diese Variable heißt *Stackpointer*. In unserem vereinfachten Speichermodell ist das einfach eine Variable wie jede andere auch.

Der zweite Namensbestandteil, Pointer, hebt darauf ab, dass eine Adresse gespeichert wird, der Stackpointer also eine Adresse enthält. Der erste Namensbestandteil, Stack, heißt auf deutsch Stapel, und wir werden gleich sehen, was es damit auf sich hat. Dieser Stack oder Stapel heißt genauer *Call-Stack*.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

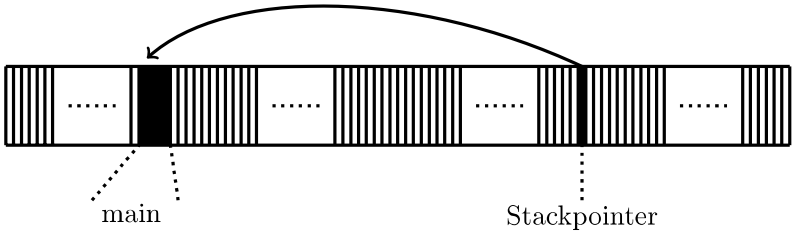


Call-Stack

***Nebenbemerkung*: In der Regel sieht es in der Realität so aus, dass die Hardware ein Register für den Stackpointer des momentan ablaufenden Programms bereitstellt. Register sind spezielle Speicherplätze separat vom eigentlichen Speicher. Wird ein Programm unterbrochen, weil andere Programme auf diesem Prozessor eine Zeitlang laufen sollen, dann wird der Stackpointer des unterbrochenen Programms in der Regel in einer ganz normalen Variable zwischengespeichert, bis das Programm wieder fortgesetzt und der Stackpointer dafür wieder ins Register geladen wird.**

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

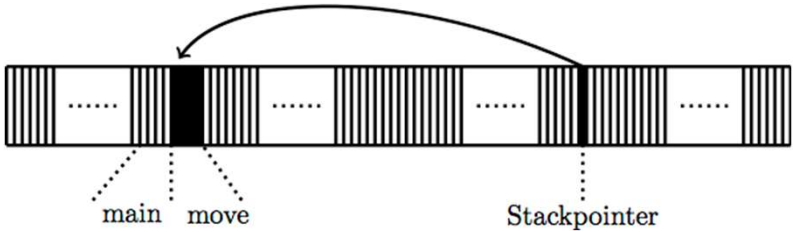



```
public static void main ( String [ ] args ) {
   ..........
   robot.move();
   i++;
   robot2.move();
   ..........
}
```

Im Computerspeicher wird ein Bereich initialisiert, um die Daten zu speichern für alle Methoden, die momentan schon begonnen, aber noch nicht beendet sind, die also momentan mitten in ihrer Ausführung stehen. Der Bereich, der für einen einzelnen Methodenaufruf reserviert wird, heißt *Frame*. Wenn als erstes main aufgerufen wird, dann wird ein Frame für diesen Aufruf von main angelegt.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT




```
public static void main ( String [ ] args ) {
   ..........
   robot.move();
   i++;
   robot2.move();
   ..........
}
```

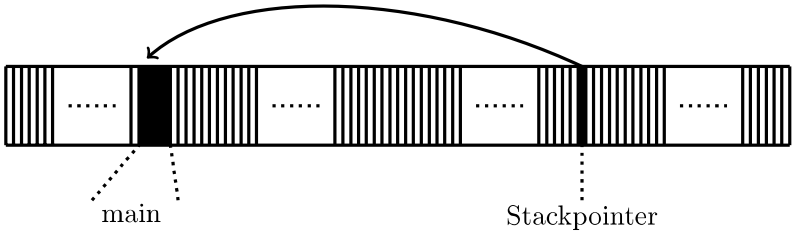
Wenn die Methode move aufgerufen wird, wird ein weiterer Frame auf dem Call-Stack abgelegt. Dieser Frame ist so groß, dass er alle Informationen zu diesem Aufruf enthalten kann. Für verschiedene Informationen werden im Frame jeweils passend große Speicherplätze reserviert. Neben diversen anderen Informationen sind das die Rücksprungadresse sowie die Referenz, mit der die Methode aufgerufen wird, hier also die Adresse in robot.

Der Inhalt des Stackpointers wird mit der Anfangsadresse des neuen Frames überschrieben. Jede Information im Frame hat einen festen Offset innerhalb des Frames. Soll beispielsweise auf die Rücksprungadresse zugegriffen werden, dann wird ihr Offset auf den momentanen Wert des Stackpointers draufaddiert, und das ist dann die Adresse, unter der die Rücksprungadresse zu finden ist.

Wir bekommen jetzt einen Eindruck davon, warum das Wort Stack in Stackpointer vorkommt: Die Frames werden sozusagen aufeinandergestapelt, was in unserer bildlichen Darstellungsweise allerdings eher horizontal statt vertikal zu sehen ist.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT



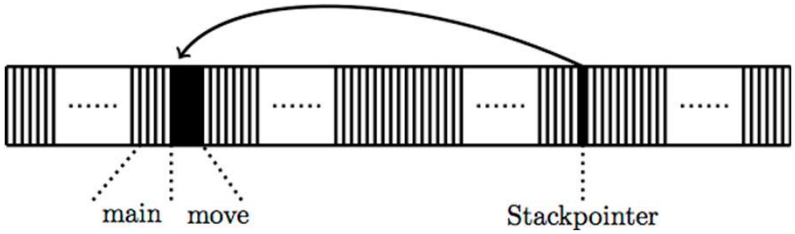

```
public static void main ( String [ ] args ) {
   ..........
   robot.move();
   i++;
   robot2.move();
   ..........
}
```

Irgendwann ist die Ausführung der Methode move zu Ende. Der Stackpointer wird wieder zurückgesetzt auf die Anfangsadresse des vorherigen Frames, was man auch metaphorisch so sagen könnte, dass der Frame vom Stack heruntergenommen wird.

Wie gesagt, springt das Programm an die im Frame bereitgehaltene Rücksprungadresse, also die nächste Anweisung nach dem soeben beendeten Aufruf.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

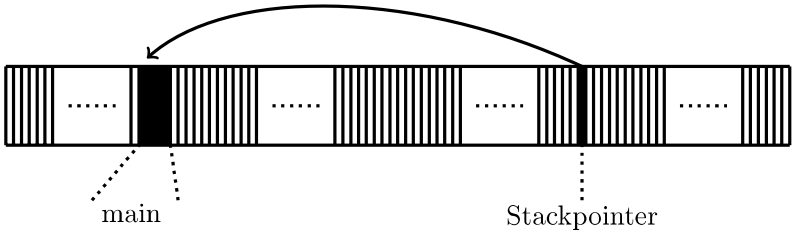



```
public static void main ( String [ ] args ) {
   ..........
   robot.move();
   i++;
   robot2.move();
   ..........
}
```

Nun wird ein weiteres Mal Methode move aufgerufen und daher ein zugehöriger Frame auf dem Call-Stack abgelegt. An die Position im Frame, die für die Referenz vorgesehen ist, wird nun der Inhalt von robot2 geschrieben, nicht wie beim ersten Aufruf der Inhalt von robot. Und an die Position im Frame, die für die Rücksprungadresse reserviert ist, wird die Adresse der nächsten Anweisung nach dem zweiten Aufruf von move geschrieben.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

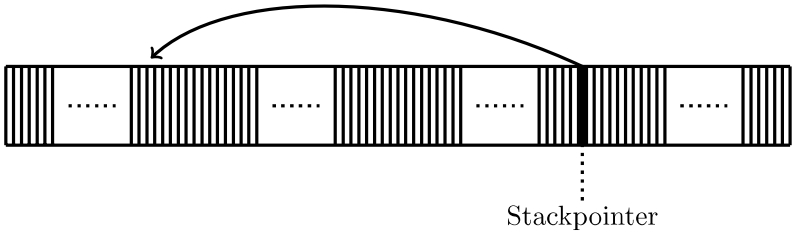



```
public static void main ( String [ ] args ) {
    ..........
    robot.move();
    i++;
    robot2.move();
    ..........
}
```

Nach Beendigung des zweiten Aufrufs von move wird wieder der zugehörige Frame vom Call-Stack geholt, ...

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT



... und nach Beendigung von main auch der Frame, der für diesen Aufruf von main abgelegt war. Damit ist der Prozess beendet.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um den geeigneten Wert
3. Berechne die Adresse der nächsten Anweisung (also von i++)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere die Adresse in robot an die dafür vorgesehene Stelle im neuen Frame
6. Springe zum Code von Robot.move
7. Führe diesen Code aus
8. Vermindere den Stackpointer um den geeigneten Wert
9. Springe nach R
10. Führe die Anweisung an R aus (i++)
11. .....

Mit unserem nun gewonnenen Verständnis des Call-Stacks als Basis können wir jetzt ein verfeinertes alternatives Modell verwenden, das genauer zeigt, was bei Methodenaufrufen so alles im Hintergrund passiert.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

Code von move mit Referenz robot und Frame F ausführen:

1. **Lies die Adresse des Robot-Objektes aus F und addiere den Offset von Attribut direction darauf**
2. **Speichere diese Adresse (also die von robot.direction) in A ab**
3. **Falls nicht Inhalt von A == Direction.UP, springe zu 7**
4. **Lies die Adresse des Objektes aus F und addiere den Offset von Attribut y darauf**
5. **Erhöhe den Wert an dieser Adresse um 1**
6. **<<Anweisungen zum Beenden der Methode>>**
7. **Falls nicht Inhalt von A == .....**

Hier sehen wir jetzt ausschnittsweise die erste Ausführung von move, also mit demjenigen Objekt der Klasse Robot, auf das die Referenz robot verweist.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um den geeigneten Wert
3. Berechne die Adresse der nächsten Anweisung (also von i++)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere die Adresse in robot an die dafür vorgesehene Stelle im neuen Frame
6. Springe zum Code von Robot.move
7. Führe diesen Code aus
8. Vermindere den Stackpointer um den geeigneten Wert
9. Springe nach R
10. Führe die Anweisung an R aus (i++)
11. .....

Wir haben soeben im Bild gesehen, dass vor Ausführung der Methode ein entsprechend großer Frame auf den Call-Stack gelegt und am Ende wieder vom Call-Stack heruntergenommen wird, und dass dies einfach durch Änderung des Stackpointers geschieht.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um den geeigneten Wert
3. Berechne die Adresse der nächsten Anweisung (also von i++)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere die Adresse in robot an die dafür vorgesehene Stelle im neuen Frame
6. Springe zum Code von Robot.move
7. Führe diesen Code aus
8. Vermindere den Stackpointer um den geeigneten Wert
9. Springe nach R
10. Führe die Anweisung an R aus (i++)
11. .....

Hier sehen Sie, wie das mit der Rücksprungadresse organisiert ist, dass eben nach dem ersten Aufruf von move im Beispiel links woandershin zurückgesprungen werden muss als nach dem zweiten Aufruf.

In jedem Frame gibt es eine bestimmte Adresse, die für die Speicherung der Rücksprungadresse vorgesehen ist. *Vor* Ausführung der Methode wird die Rücksprungadresse berechnet und an dieser Adresse abgelegt; *nach* Ausführung der Methoden springt der Prozess genau dorthin.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um den geeigneten Wert
3. Berechne die Adresse der nächsten Anweisung (also von i++)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere die Adresse in robot an die dafür vorgesehene Stelle im neuen Frame
6. Springe zum Code von Robot.move
7. Führe diesen Code aus
8. Vermindere den Stackpointer um den geeigneten Wert
9. Springe nach R
10. Führe die Anweisung an R aus (i++)
11. .....

Ebenso hatten wir schon gesagt, dass auch mitgespeichert werden muss, mit welchem Roboter die Methode aufgerufen wird, denn die Methode soll ja auf die Attribute *dieses* Roboters zugreifen und nicht auf die Attribute eines anderen Roboters.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

Code von move mit Referenz robot und Frame F ausführen:

1. Lies die Adresse des Robot-Objektes aus F und addiere den Offset von Attribut direction darauf
2. Speichere diese Adresse (also die von robot.direction) in A ab
3. Falls nicht Inhalt von A == Direction.UP, springe zu 7
4. Lies die Adresse des Objektes aus F und addiere den Offset von Attribut y darauf
5. Erhöhe den Wert an dieser Adresse um 1
6. <<Anweisungen zum Beenden der Methode>>
7. Falls nicht Inhalt von A == .....

Ein Attribut einer Klasse steht in jedem Objekt dieser Klasse immer an derselben Position. In der Informatik sagt man üblicherweise: Das Attribut hat einen festen Offset von der Anfangsadresse des Objektes. Daher kann die Adresse eines Attributs einfach dadurch berechnet werden, dass dieser feste Offset zur Anfangsadresse des Objektes hinzuaddiert wird. Und die Anfangsadresse des Objektes ist ja gerade der Inhalt der Referenz robot und somit ihrerseits an einem festen Offset im Frame von move zu finden.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

Code von move mit Referenz robot und Frame F ausführen:

1. Lies die Adresse des Robot-Objektes aus F und addiere den Offset von Attribut direction darauf
2. Speichere diese Adresse (also die von robot.direction) in A ab
3. Falls nicht Inhalt von A == Direction.UP, springe zu 7
4. Lies die Adresse des Objektes aus F und addiere den Offset von Attribut y darauf
5. Erhöhe den Wert an dieser Adresse um 1
6. <<Anweisungen zum Beenden der Methode>>
7. Falls nicht Inhalt von A == .....

An dieser Stelle passiert der eigentliche move-Schritt im Falle, dass der Roboter gerade nach oben schaut.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

Code von move mit Referenz robot und Frame F ausführen:

1. Lies die Adresse des Robot-Objektes aus F und addiere den Offset von Attribut direction darauf
2. Speichere diese Adresse (also die von robot.direction) in A ab
3. Falls nicht Inhalt von A == Direction.UP, springe zu 7
4. Lies die Adresse des Objektes aus F und addiere den Offset von Attribut y darauf
5. Erhöhe den Wert an dieser Adresse um 1
6. <<Anweisungen zum Beenden der Methode>>
7. Falls nicht Inhalt von A == .....

Die Details hiervon lassen wir aus; wir haben sie im Prinzip auf der letzten Folie gesehen.

Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

```
int i = 1;
robot.move();
i++;
robot2.move();
i – –;
```

Code von move mit Referenz robot und Frame F ausführen:

1. Lies die Adresse des Robot-Objektes aus F und addiere den Offset von Attribut direction darauf
2. Speichere diese Adresse (also die von robot.direction) in A ab
3. Falls nicht Inhalt von A == Direction.UP, springe zu 7
4. Lies die Adresse des Objektes aus F und addiere den Offset von Attribut y darauf
5. Erhöhe den Wert an dieser Adresse um 1
6. <<Anweisungen zum Beenden der Methode>>
7. Falls nicht Inhalt von A == .....

Die anderen drei Fälle – also wenn direction nicht UP ist – lassen wir ebenfalls aus.

TECHNISCHE UNIVERSITÄT DARMSTADT

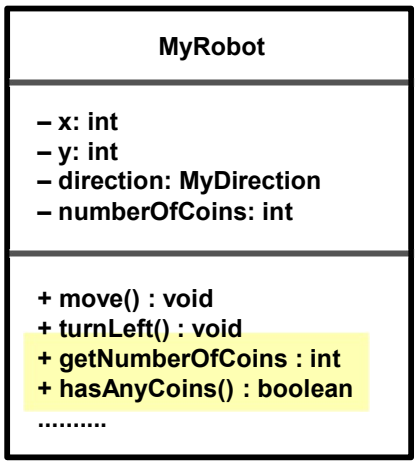
Die Methoden mit Rückgabe

Oracle Java Tutorials: Defining Methods

Bisher hatten wir nur Methoden *ohne* Rückgabe betrachtet, jetzt kommen die Methoden *mit* Rückgabe. Das sind dann Methoden, die im Gegensatz zu move und turnLeft einen Wert zurückliefern.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT



Ein paar dieser Methoden haben wir in dieses UML-Klassendiagramm beispielhaft hineingeschrieben, die anderen sind auf den Folien aus Platzgründen ausgelassen und wie üblich nur durch Punkte angedeutet.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {

   .........

   public int getX () {
      return x;
   }

   public int getY () {
      return y;
   }

   public MyDirection getDirection () {
      return direction;
   }

   public int getNumberOfCoins () {
      return numberOfCoins;

   .........
}
```

Und dies sind die Methoden, die wir im Folgenden implementieren.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {

   .........

   public int getX () {
      return x;
   }

   public int getY () {
      return y;
   }

   public MyDirection getDirection () {
      return direction;
   }

   public int getNumberOfCoins () {
      return numberOfCoins;

   .........
}
```

An der Stelle im Methodenkopf, wo void in der Definition der Methoden move und turnLeft steht, steht jetzt der Typ des Rückgabewertes, der dann auch entsprechend *Rückgabetyp* heißt.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {

   .........

   public int getX () {
      return x;
   }

   public int getY () {
      return y;
   }

   public MyDirection getDirection () {
      return direction;
   }

   public int getNumberOfCoins () {
      return numberOfCoins;

   .........
}
```

Das Schlüsselwort return gibt an, dass die Ausführung der Methode mit dieser Anweisung zu beenden und der Wert hinter dem return zurückzuliefern ist.

Bei diesen vier Methoden wird jeweils der Wert eines der Attribute der Klasse Robot zurückgegeben. Das ist natürlich der Wert des Attributs in demjenigen Robot-Objekt, mit dem die Methode aufgerufen wird.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   .........
   public int getX () {
      return x;
   }
   .........
}
```

```
int i = 1;
i = robot.getX();
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um die Größe der Frames für Robot.getX
3. Berechne die Adresse der nächsten Operation (Zuweisung an i)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere den Inhalt von robot an die dafür vorgesehene Stelle S im neuen Frame
6. Nimm die Adresse an S und addiere den Offset von x in Robot
7. Kopiere den dortigen Wert in die Adresse W, die für den Rückgabewert von Methoden vorgesehen ist
8. Kopiere den Inhalt von W nach i

Mit einer dieser vier Methoden schauen wir uns den Ablauf im alternativen Modell an. Oben links sehen Sie die Definition der Methode getX nochmals wie auf der letzten Folie, unten links sehen Sie einen beispielhaften Aufruf der Methode getX, bei dem der zurückgelieferte Wert in einer int-Variable gespeichert ist.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public int getX () {
      return x;
   }
   ..........
}
```

```
int i = 1;
i = robot.getX();
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um die Größe der Frames für Robot.getX
3. Berechne die Adresse der nächsten Operation (Zuweisung an i)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere den Inhalt von robot an die dafür vorgesehene Stelle S im neuen Frame
6. Nimm die Adresse an S und addiere den Offset von x in Robot
7. Kopiere den dortigen Wert in die Adresse W, die für den Rückgabewert von Methoden vorgesehen ist
8. Kopiere den Inhalt von W nach i

Das kennen wir schon: Wir berechnen die Adresse des Wertes, auf den wir zugreifen wollen.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public int getX () {
      return x;
   }
   ..........
}
```

```
int i = 1;
i = robot.getX();
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um die Größe der Frames für Robot.getX
3. Berechne die Adresse der nächsten Operation (Zuweisung an i)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere den Inhalt von robot an die dafür vorgesehene Stelle S im neuen Frame
6. Nimm die Adresse an S und addiere den Offset von x in Robot
7. Kopiere den dortigen Wert in die Adresse W, die für den Rückgabewert von Methoden vorgesehen ist
8. Kopiere den Inhalt von W nach i

Sie können sich das so vorstellen, dass es eine feste Stelle im Speicher gibt, die wir hier W nennen und die für das kurzzeitige Zwischenspeichern von Rückgabewerten reserviert ist. Das heißt, der Rückgabewert, den ein Methodenaufruf zurückliefert, muss recht bald weiterverarbeitet werden, bevor W durch die Beendigung eines anderen Methodenaufrufs überschrieben wird.

***Nebenbemerkung*: In realitätsnäheren Maschinenmodellen würde man von W als einem *Register* sprechen.**

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public int getX () {
      return x;
   }
   ..........
}
```

```
int i = 1;
i = robot.getX();
```

1. Richte int-Variable i ein und setze i auf 1
2. Erhöhe den Stackpointer um die Größe der Frames für Robot.getX
3. Berechne die Adresse der nächsten Operation (Zuweisung an i)
4. Schreibe diese an die Position R der Rücksprungadresse im neuen Frame
5. Kopiere den Inhalt von robot an die dafür vorgesehene Stelle S im neuen Frame
6. Nimm die Adresse an S und addiere den Offset von x in Robot
7. Kopiere den dortigen Wert in die Adresse W, die für den Rückgabewert von Methoden vorgesehen ist
8. Kopiere den Inhalt von W nach i

Sowohl links im Java-Code als auch rechts im alternativen Modell finden Sie diese Weiterverarbeitung jeweils in der letzten Zeile: Dieser Wert wird in i gespeichert, so dass W folgenlos jederzeit überschrieben werden kann.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public int getNumberOfCoins () {
      return numberOfCoins;
   }

   public boolean hasAnyCoins () {
      return numberOfCoins > 0;
   }
   ..........
}
```

Oben sehen Sie nochmals eine der vier bisher betrachteten Methoden. Unten ist eine Methode implementiert, die wir schon verwendet, bisher aber nicht nachgebaut hatten.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public int getNumberOfCoins () {
      return numberOfCoins;
   }

   public boolean hasAnyCoins () {
      return numberOfCoins > 0;
   }
   ..........
}
```

Die Anzahl der Beeper selbst ist vom Typ int, also eine Zahl. Aber der Größenvergleich zweier Zahlen ist true oder false, also boolean, so dass der tatsächliche Typ des zurückgelieferten Wertes auch hier mit dem deklarierten Rückgabetyp übereinstimmt.

Die Methoden mit Rückgabe

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public int getNumberOfCoins () {
      return numberOfCoins;
   }

   public boolean hasAnyCoins () {
      return getNumberOfCoins() > 0;
   }
   ..........
}
```

Dieselbe Methode zur Veranschaulichung nochmals, aber leicht anders implementiert: Der Zugriff auf das Attribut geschieht jetzt nicht mehr nicht direkt, sondern wird an die obige Methode delegiert. Wenn in einer Methode also eine andere Methode so wie hier aufgerufen wird – also ohne Angabe eines Objektes über eine Referenz vorneweg mit Punkt getrennt –, dann wird letztere Methode mit demselben Objekt wie erstere Methode aufgerufen. Das ist hier ja offensichtlich auch genau das, was man will.

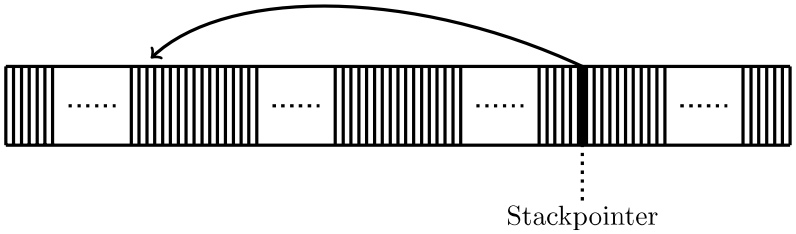
TECHNISCHE UNIVERSITÄT DARMSTADT

Weiter Ausführung von Methoden

Wir sehen uns jetzt die zuletzt definierten Methoden auf dem Call-Stack an, ...

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

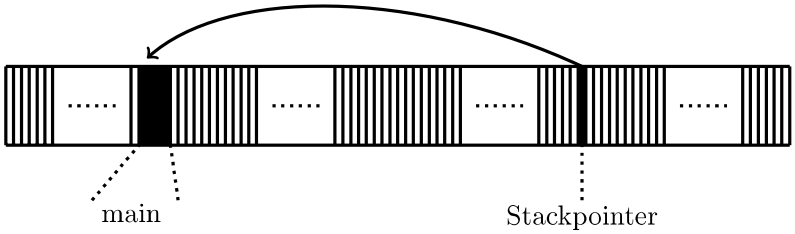



```
public boolean hasAnyCoins () {
    return getNumberOfCoins() > 0;
}
```

... konkret als erstes das letzte Beispiel von eben, noch einmal rechts unten hingeschrieben: eine Methode, die ihrerseits eine Methode mit demselben Robot-Objekt aufruft.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

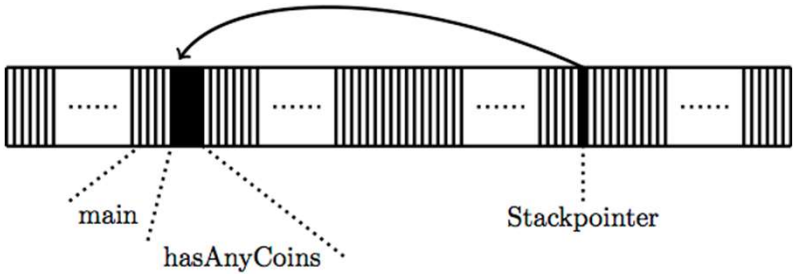



```
public boolean hasAnyCoins () {
    return getNumberOfCoins() > 0;
}
```

Wie üblich ist Methode main der Einstiegspunkt für den Prozess.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

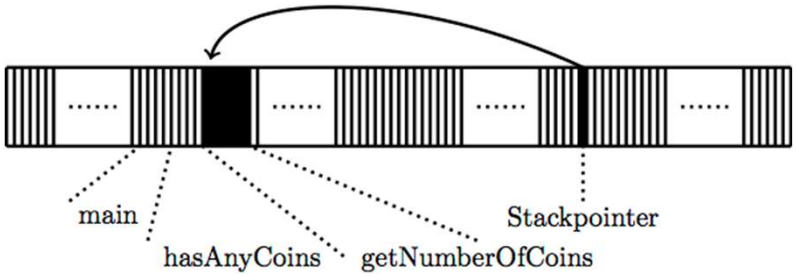



```
public boolean hasAnyCoins () {
    return getNumberOfCoins() > 0;
}
```

Wir nehmen an, dass die Methode hasAnyCoins direkt in der Methode main aufgerufen wird. Vor diesem Aufruf von hasAnyCoins passiert sicherlich noch einiges mehr in main, was sich im Call-Stack niederschlägt. Aber das ignorieren wir und spulen die Betrachtung sozusagen vor bis zum Aufruf von hasAnyCoins, also bis zur Einrichtung eines Frames dafür.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

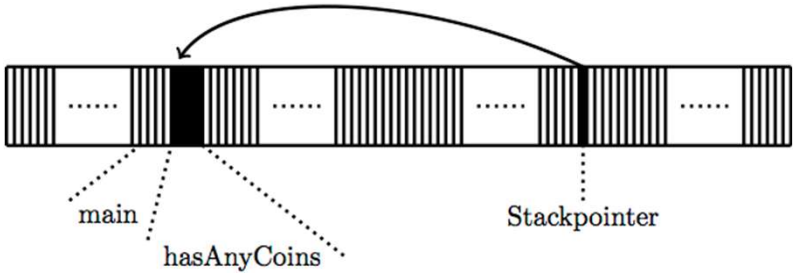



```
public boolean hasAnyCoins () {
    return getNumberOfCoins() > 0;
}
```

Innerhalb der Methode hasAnyCoins wird ziemlich sofort die Methode getNumberOfCoins aufgerufen. In diesem Moment wird für letztere Methode natürlich ebenfalls ein Frame angelegt, zusätzlich zu den bisherigen. Wie immer, wird der Stackpointer entsprechend auf den Anfang dieses neuen Frames umgesetzt.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

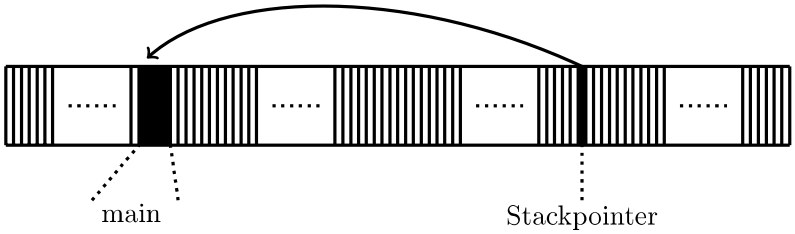



```
public boolean hasAnyCoins () {
    return getNumberOfCoins() > 0;
}
```

Die Methode getNumberOfCoins ist dann auch bald wieder zu Ende, und der zugehörige Frame ist wieder abgebaut, das heißt, der Stackpointer ist auf die Anfangsadresse des vorherigen Frames zurückgesetzt.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

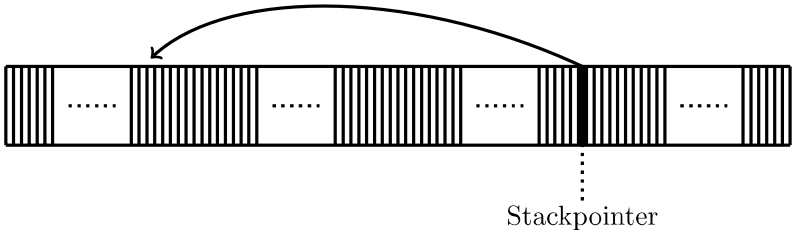



```
public boolean hasAnyCoins () {
    return getNumberOfCoins() > 0;
}
```

Durch das return wird auch die Ausführung der Methode hasAnyCoins beendet und der zugehörige Frame entfernt.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

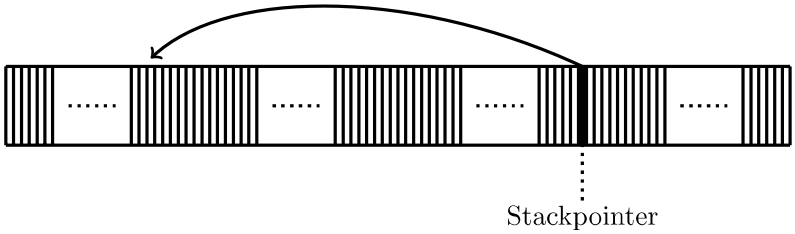



```
public boolean hasAnyCoins () {
    return getNumberOfCoins() > 0;
}
```

Wir spulen die Betrachtung wieder vorwärts, diesmal bis zur Beendigung von main, und überspringen alles, was vielleicht noch dazwischen kommt. Wenn main beendet wird, ist der Prozess beendet.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT



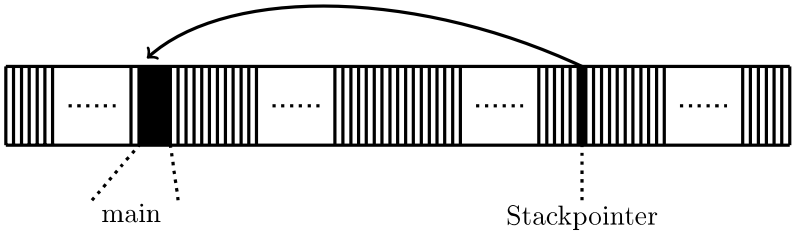

```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Jetzt noch ein weiteres Beispiel. In diesem Beispiel werden wir uns einen wichtigen Punkt vor Augen führen, der in den bisherigen Beispielen noch nicht vorkam: dass der Frame zu einer Methode an verschiedenen Stellen auf dem Call-Stack platziert sein kann, je nachdem, wie die momentane Aufrufhierarchie jeweils aussieht.

Analog zu eben gehen wir davon aus, dass diese fünf Zeilen irgendwo in main stehen.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

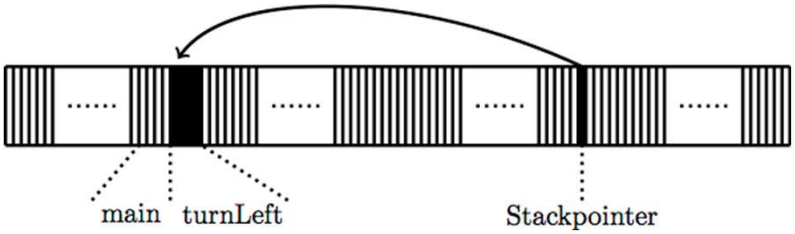



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Natürlich beginnt es wieder mit main.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

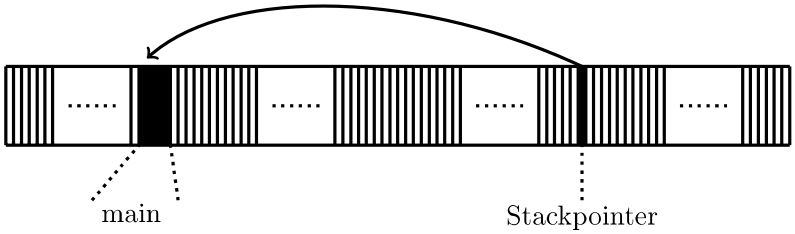



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Wir überspringen wieder alles vor diesen fünf Zeilen, das heißt, wir spulen unsere Betrachtung bis zu diesem Aufruf von turnLeft vor.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

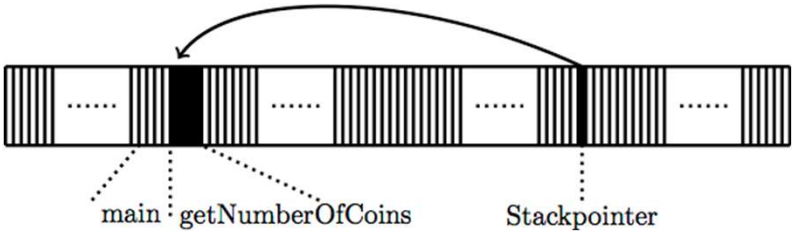



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Nach Beendigung von turnLeft sieht der Call-Stack dann wieder so aus wie zuvor, ...

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

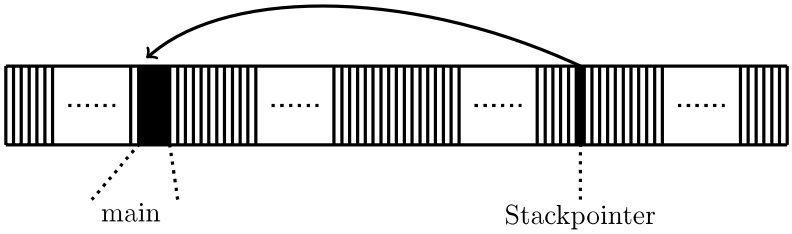



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

... bis dann als nächstes getNumberOfCoins aufgerufen wird.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

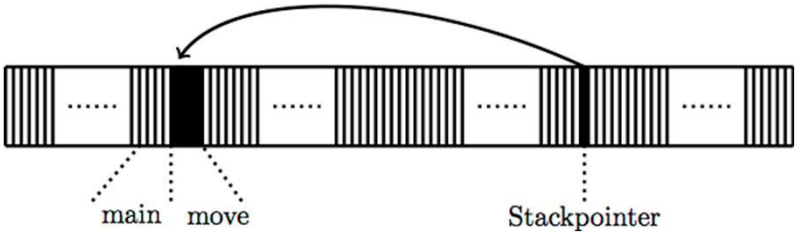



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Die Situation zwischen zwei Methodenaufrufen.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

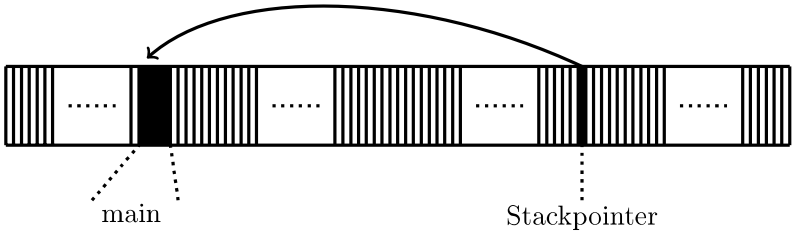



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
    robot.putCoin();
```

Der nächste Methodenaufruf ...

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

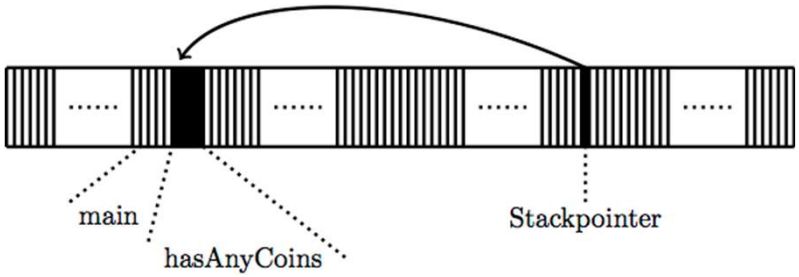



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

... und wieder die Situation unmittelbar danach.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

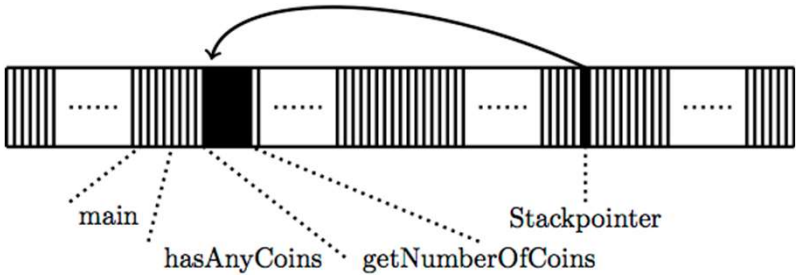



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Auch wenn ein Methodenaufruf an einer solchen Stelle steht, sieht die Situation selbstverständlich analog aus.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

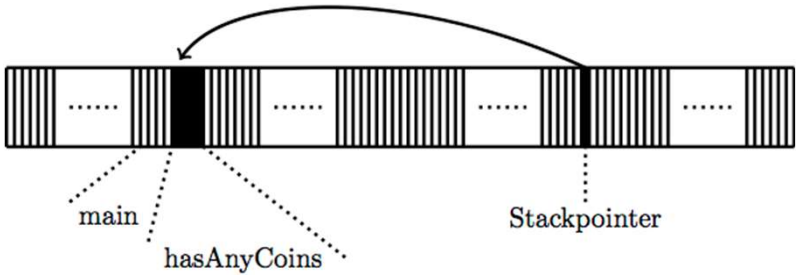



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Während der Ausführung von hasAnyCoins wird ja getNumberOfCoins aufgerufen, also auch dafür ein Frame.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

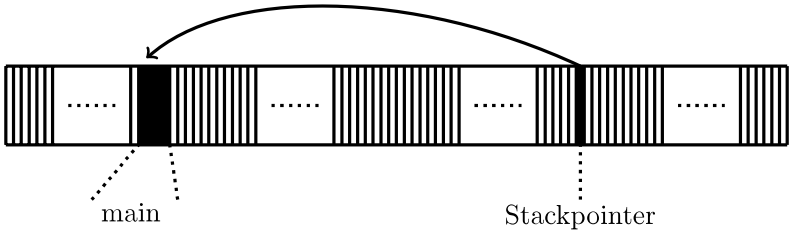



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Und nach Beendigung von getNumberOfCoins sieht die Situation wieder wie unmittelbar davor aus.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

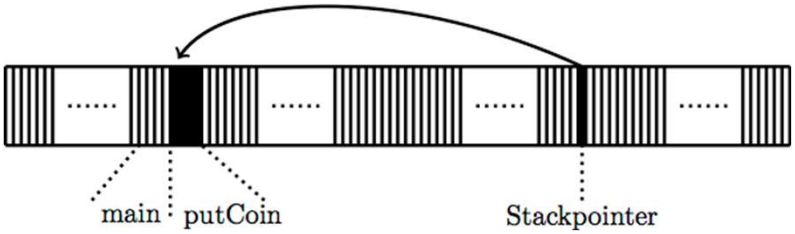



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Wieder die Situation zwischen zwei Methodenaufrufen.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

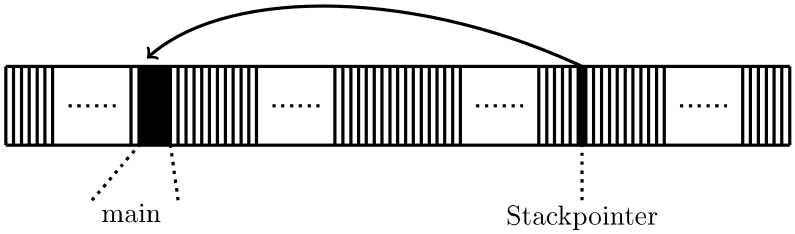



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Und schließlich noch der letzte Methodenaufruf, wobei wir hier annehmen, dass hasAnyCoins den Wert true zurückgeliefert hat, so dass putCoin tatsächlich aufgerufen wird.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

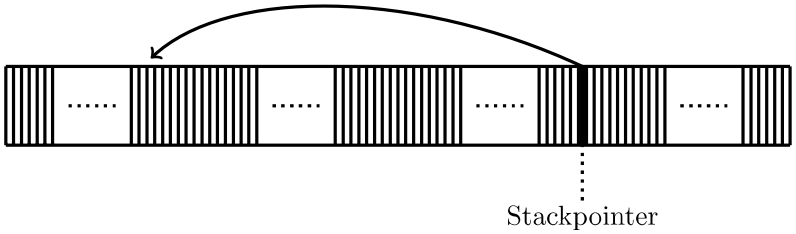



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

Wieder die schon bekannte Situation, ...

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT

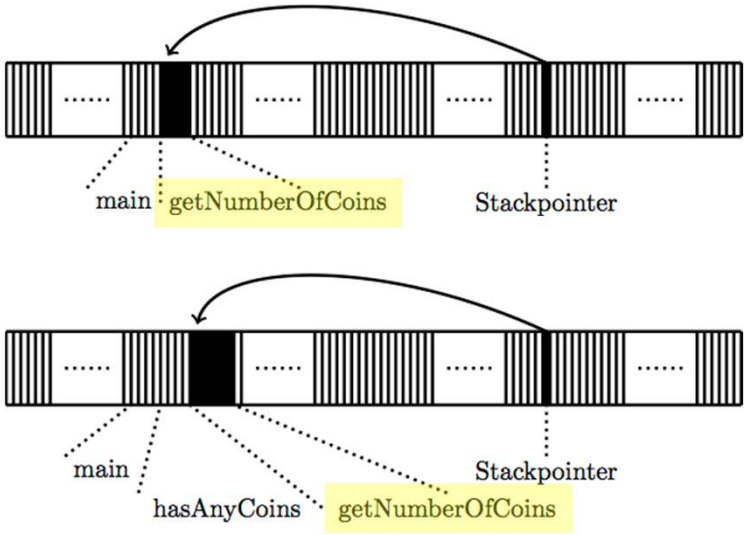



```
robot.turnLeft();
int i = robot.getNumberOfCoins();
robot.move();
if ( robot.hasAnyCoins() )
   robot.putCoin();
```

... und wir spulen wieder vor bis zum Ende der Abarbeitung von main.

Weiter Ausführung von Methoden

TECHNISCHE UNIVERSITÄT DARMSTADT



Die Gegenüberstellung von zwei Schnappschüssen aus dem Beispiel zeigt jetzt den entscheidenden Punkt: In beiden Situationen wird gerade die Methode getNumberOfCoins ausgeführt, aber an verschiedenen Stellen in der Aufrufhierarchie, einmal direkt von main aus, einmal von hasAnyCoins aus. Daher ist der Frame an unterschiedlichen Stellen auf dem Call-Stack platziert.

Aber der Stackpointer verweist ja immer auf den Anfang des Frames, egal wo der Frame platziert ist. Und jede einzelne Information im Frame hat einen festen Offset im Frame, so dass der Zugriff auf eine Information immer gleich funktioniert: Wert des Stackpointers plus Offset der einzelnen Information gleich Adresse dieser einzelnen Information. Die Auswertung muss nicht berücksichtigen, wo der Frame auf dem Call-Stack liegt, denn diese Position ist im Stackpointer gespeichert und kann einfach abgerufen werden.

TECHNISCHE UNIVERSITÄT DARMSTADT

Methoden mit Parametern

Oracle Java Tutorials: Defining Methods

Alle bisher nachgebauten Methoden hatten noch leere Parameterlisten. Jetzt bauen wir drei Methoden von Klasse Robot nach, die jeweils *einen* Parameter haben, sowie eine, die *zwei* Parameter hat.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   .........
   public void setX ( int newX ) {
      x = newX;
   }
   public void setY ( int newY ) {
      y = newY;
   }
   .........
}
```

Hier sehen Sie die beiden Methoden mit jeweils einem Parameter, die wir nachbauen wollen. Mit diesen beiden Methoden lassen sich Roboter auf eine bestimmte Zeile beziehungsweise Spalte positionieren. Die anderen Attribute des Roboters werden dadurch nicht verändert.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   .........
   public void setX ( int newX )  {
      x = newX;
   }
   public void setY ( int newY )  {
      y = newY;
   }
   .........
}
```

Bei Methoden *ohne* Parameter stand hier nur das leere Klammerpaar, also eine leere Parameterliste. Wenn bei der Definition einer Methode ausgesagt werden soll, dass sie einen Parameter hat, dann schreiben wir wie bei Variablen zuerst den Typ hin und geben dem Parameter so wie einer Variablen auch einen Namen. Dieser Name ist ein Identifier und muss daher den Regeln für Identifier folgen.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   .........
   public void setX ( int newX ) {
      x = newX;
   }
   public void setY ( int newY ) {
      y = newY;
   }
   .........
}
```

Innerhalb der Methode kann der Parameter dann tatsächlich auch so wie eine Variable benutzt werden. Ein Parameter ist praktisch eine zusätzliche Variable in der Methode, nur mit dem Unterschied, dass sie nicht in der Methode, sondern beim Aufruf durch den übergebenen Wert initialisiert wird.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   .........
   public void setX ( int x )  {
      this.x = x;
   }
   public void setY ( int y )  {
      this.y = y;
   }
   .........
}
```

Eine kleine Variation: In der ursprünglichen Version auf der letzten Folie hatten wir den beiden Parametern Namen gegeben, die sich von den Namen der zugehörigen Attribute unterscheiden. Das ist im Prinzip auch notwendig, denn mit demselben Namen kann man im selben Kontext natürlich nicht zwei verschiedene Entitäten ansprechen. Aber in größeren Programmen ist es misslich, sich immer wieder neue Namen auszudenken für Entitäten, die eigentlich inhaltlich dasselbe sind; zwei Entitäten, die inhaltlich dasselbe sind, sollten zur besseren Verständlichkeit auch exakt gleich heißen.

Mit dem Schlüsselwort this kann man in einer Methode das Objekt ansprechen, mit dem die Methode aufgerufen wird. Das erlaubt die Unterscheidung zwischen Parameter und Attribut: *Mit* this ist es das Attribut, *ohne* this ist es der Parameter.

Damit ist auch die nicht gestellte, aber im Raum stehende Frage geklärt, ob bei Namenskollision zwischen Attribut und Parameter nun das Attribut oder der Parameter angesprochen wird: Es ist der Parameter.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {

   .........
   public void setPosition ( int newX, int newY )  {
      x = newX;
      y = newY;
   }

   .........
}
```

So sieht die Definition einer Methode mit zwei Parametern aus. Die wesentliche neue Erkenntnis ist, dass die einzelnen Parameter durch Kommas voneinander getrennt sind. Das ist natürlich nicht nur bei zwei Parametern so, sondern auch bei mehr als zweien.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   .....
   public void copy ( MyRobot otherRobot ) {
      x = otherRobot.x;
      y = otherRobot.y;
      direction = otherRobot.direction;
      numberOfCoins = otherRobot.numberOfCoins;
   }
   .....
}
```

Noch ein letztes Beispiel in diesem Abschnitt, diesmal wieder eine Methode mit nur einem Parameter. Die Attribute des Objektes, mit dem diese Methode aufgerufen wird, werden überschrieben, und zwar so, dass dieses Objekt eine Kopie des Objektes ist, auf das der Parameter verweist.

Wenn wir nicht nur die Adresse eines Objektes zuweisen, sondern das Objekt als Ganzes – also seine Attribute – kopieren wollen, dann müssen wir eine solche Methode schreiben.

***Nebenbemerkung*: In einem realen Fallbeispiel aus der Praxis würde man die Methode clone der Klasse Object dafür überschreiben; siehe speziell dazu Kapitel 03a, Abschnitt zur Klasse java.lang.Object, sowie das Oracle Java Tutorial „Object as a Superclass“.**

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   .........
   public void copy ( MyRobot otherRobot ) {
      x = otherRobot.x;
      y = otherRobot.y;
      direction = otherRobot.direction;
      numberOfCoins = otherRobot.numberOfCoins;
   }
   .........
}
```

Das Neue ist, dass der Parameter nun von einer Klasse ist, nicht mehr vom primitiven Datentyp int.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

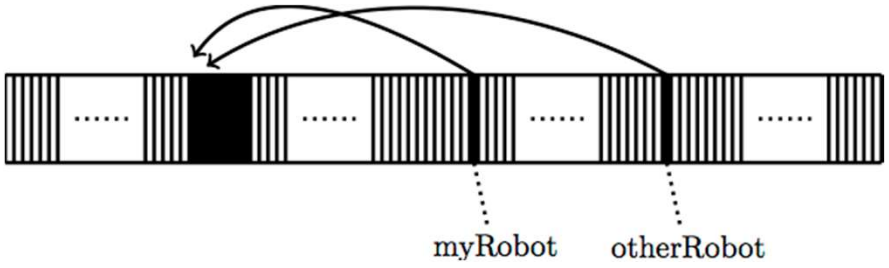
```
public class MyRobot {
   .....
   public void copy ( MyRobot otherRobot ) {
      x = otherRobot.x;
      y = otherRobot.y;
      direction = otherRobot.direction;
      numberOfCoins = otherRobot.numberOfCoins;
   }
   .....
}
```

Die Attribute des Objektes hinter dem Parameter lassen sich ansprechen, indem der Name des Parameters vorangestellt wird, durch Punkt getrennt.

***Achtung*: Ein wichtiger Aspekt ist, dass in dieser Methode überhaupt auf die Attribute des anderen Roboters zugegriffen werden *darf*. Die Attribute der Klasse MyRobot sind ja private, das heißt, nicht von außerhalb zugreifbar. Wir hatten offen gelassen, was „von außerhalb“ genau heißt, jetzt können wir diese Formulierung präziser fassen: Ist ein Attribut private, dann darf darauf nicht in den Methoden anderer Klassen zugegriffen werden; in den Methoden der eigenen Klasse darf hingegen frei auf die Attribute eines Objektes zugegriffen werden – auch wenn die Methode mit einem anderen Objekt aufgerufen wird.**

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT



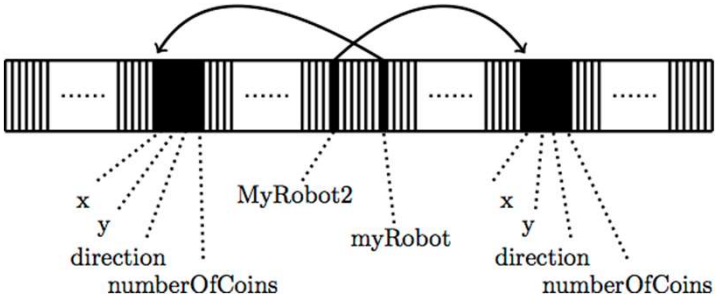

```
Robot myRobot = new Robot ( ..... ):
Robot myRobot2 = new Robot ( ..... );
.....
myRobot2.copy ( myRobot );
```

Es ist wichtig sich klarzumachen, dass der prinzipielle Unterschied zwischen Klassen und primitiven Datentypen auch bei der Parameterübergabe greift: während bei Parametern von primitiven Datentypen wie int einfach der Wert kopiert wird, wird bei Klassen nur die Referenz kopiert, nicht das Objekt.

Das hat Konsequenzen in dem Fall, dass wir auf die Attribute von otherRobot nicht nur lesend, sondern auch schreibend zugreifen. Dann werden die Werte dieser Attribute im Roboter myRobot geändert und eben *nicht* in einer Kopie des Roboters (die ja eben gar nicht erstellt wird).

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT




```
Robot myRobot = new Robot ( ..... ):
Robot myRobot2 = new Robot ( ..... );
.....
myRobot2.copy ( myRobot );
```

Das ist eben *Kopieren* anstelle von *Zuweisen*: Beim Zuweisen mit „myRobot2 = myRobot;“ würden beide Referenzen auf dasselbe Objekt verweisen. Beim Kopieren mit „myRobot2.copy(myRobot);“ bleibt es bei zwei verschiedenen Objekten, aber ihre Attribute haben identische Werte.

Methoden mit Parametern

TECHNISCHE UNIVERSITÄT DARMSTADT

```
/**
* @param  x  the new horizontal position, must be nonnegative
* @param  y  the new vertical position, must be nonnegative
*/
public void setPosition ( int x, int y ) { ..... }


/**
* @return the current horizontal position
*/
public int getX() { ..... }
```

***Erinnerung*: In Kapitel 01a, im Abschnitt zu Kommentaren, hatten wir schon beispielhaft gesehen, wie man eine ganze Quelldatei in JavaDoc dokumentiert. Mit den beiden Klauseln @param und @return können wir die Parameter und den Rückgabewert in JavaDoc beschreiben.**

TECHNISCHE UNIVERSITÄT DARMSTADT

Der Konstruktor

Oracle Java Tutorials:
Providing Constructors for Your Classes

Was noch fehlt, ist die Methode, die beim Einrichten eines neuen Robot-Objektes aufgerufen wird. Eine solche Methode einer Klasse heißt *Konstruktor*.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot lolek = new Robot ( 1, 2, MyDirection.UP, 0 );

Robot bolek = new Robot ( 4, 3, MyDirection.RIGHT, 8 );
```

Zur Erinnerung: Der Konstruktor ist die Methode, die wir immer schon mit „new Robot“ aufgerufen haben. Offensichtlich hat sie vier Parameter, die ersten beiden und der letzte vom Typ int, der dritte vom Typ MyDirection.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public MyRobot ( int initX, int initY,
                    MyDirection initDirection,
                    int initNumberOfCoins ) {
      x = initX;
      y = initY;
      direction = initDirection;
      numberOfCoins = initNumberOfCoins;
   }
   ..........
}
```

Und das ist auch schon der ganze Konstruktor.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public MyRobot ( int initX, int initY,
                    MyDirection initDirection,
                    int initNumberOfCoins ) {
      x = initX;
      y = initY;
      direction = initDirection;
      numberOfCoins = initNumberOfCoins;
   }
   ..........
}
```

Der Konstruktor hat denselben Namen wie die Klasse. Keine andere Methode einer Klasse kann denselben Namen wie die Klasse tragen.

Die Auslassung des Rückgabetyps beziehungsweise void vor dem Namen ist kein Versehen, sondern der Konstruktor hat keinen Rückgabetyp, auch nicht void. An der Stelle, an der bei anderen Methoden der Rückgabetyp beziehungsweise void steht, steht daher beim Konstruktor gar nichts.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
public class MyRobot {
   ..........
   public MyRobot ( int initX, int initY,
                    MyDirection initDirection,
                    int initNumberOfCoins ) {
      x = initX;
      y = initY;
      direction = initDirection;
      numberOfCoins = initNumberOfCoins;
   }
   ..........
}
```

Das sind die vier Parameter. Eigentlich sehen Sie hier nichts Neues: Für jeden Parameter wird zuerst sein Typ hingeschrieben, dann der Identifier, mit dem man den Parameter in der Methode ansprechen will. Die einzelnen Parameter sind durch Kommas voneinander getrennt.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

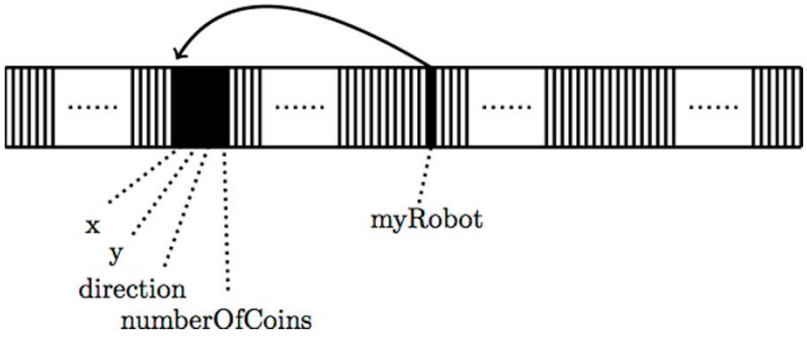
```
public class MyRobot {
   ..........
   public MyRobot ( int initX, int initY,
                    MyDirection initDirection,
                    int initNumberOfCoins ) {
      x = initX;
      y = initY;
      direction = initDirection;
      numberOfCoins = initNumberOfCoins;
   }
   ..........
}
```

Analog zur Methode clone werden auch hier die vier Attribute des Objektes gesetzt.

Initialisierung der Attribute ist ein sehr typischer Fall für die Aufgaben eines Konstruktors. Es gibt aber auch etliche Klassen, die noch viel mehr als das tun. Beispiele dafür werden wir mit der Zeit sehen.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT




```
x = initX;
y = initY;
direction = initDirection;
numberOfCoins = initNumberOfCoins;
```

Sie können sich das wieder anhand der schematischen Darstellung im Computerspeicher vor Augen führen.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot myRobot = new Robot ( 3, 1, MyDirection.UP, 0 );
```

1. Richte eine Referenz myRobot ein
2. Suche Speicherplatz, der groß genug für ein Objekt vom Typ Robot ist
3. Reserviere den gefundenen Speicherplatz
4. Rufe den Konstruktor mit der Anfangs-adresse dieses Speicherplatzes auf
5. Liefere diese Adresse zurück
6. Kopiere diese Adresse nach myRobot

Zum Abschluss dieses Abschnitts schauen wir uns die Einrichtung einer Referenz und eines Objektes sowie den Aufruf des Konstruktors im alternativen Ablaufmodell an. Die Reihenfolge der einzelnen Schritte muss nicht zwingend hundertprozentig die hier gezeigte sein, wird aber sicherlich nicht wesentlich anders sein.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot myRobot = new Robot ( 3, 1, MyDirection.UP, 0 );
```

1. Richte eine Referenz myRobot ein
2. Suche Speicherplatz, der groß genug für ein Objekt vom Typ Robot ist
3. Reserviere den gefundenen Speicherplatz
4. Rufe den Konstruktor mit der Anfangs-adresse dieses Speicherplatzes auf
5. Liefere diese Adresse zurück
6. Kopiere diese Adresse nach myRobot

Hier ist nichts Neues zu sehen.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot myRobot = new Robot ( 3, 1, MyDirection.UP, 0 );
```

1. Richte eine Referenz myRobot ein
2. Suche Speicherplatz, der groß genug für ein Objekt vom Typ Robot ist
3. Reserviere den gefundenen Speicherplatz
4. Rufe den Konstruktor mit der Anfangs-adresse dieses Speicherplatzes auf
5. Liefere diese Adresse zurück
6. Kopiere diese Adresse nach myRobot

Mit Operator new wird dem Laufzeitsystem der Auftrag gegeben, einen geeigneten, noch nicht reservierten Speicherplatz zu suchen und dann auch zu reservieren.

***Vorgriff*: Wenn man sehr viele Objekte in seinem Programm einrichtet, kann es durchaus passieren, dass kein Speicherplatz mehr gefunden wird. In diesem Fall gibt es zur Laufzeit einen Fehler. Im Kapitel 03a, Abschnitt zum Garbage Collector, schauen wir uns diese Art von Fehlern an.**

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot myRobot = new Robot ( 3, 1, MyDirection.UP, 0 );
```

1. Richte eine Referenz myRobot ein
2. Suche Speicherplatz, der groß genug für ein Objekt vom Typ Robot ist
3. Reserviere den gefundenen Speicherplatz
4. Rufe den Konstruktor mit der Anfangs-adresse dieses Speicherplatzes auf
5. Liefere diese Adresse zurück
6. Kopiere diese Adresse nach myRobot

Nachdem der Speicherplatz für das Objekt reserviert ist, kann der Konstruktor aufgerufen werden. Unser Konstruktor initialisiert die Attribute durch die Parameter, dafür muss der Speicherplatz für das Objekt natürlich vorher festgelegt worden sein. Daher kommt der Aufruf des Konstruktors erst hier.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

Robot myRobot = new Robot (3, 1, MyDirection.UP, 0);

1. **Richte eine Referenz myRobot ein**
2. **Suche Speicherplatz, der groß genug für ein Objekt vom Typ Robot ist**
3. **Reserviere den gefundenen Speicherplatz**
4. **Rufe den Konstruktor mit der Anfangs-adresse dieses Speicherplatzes auf**
5. **Liefere diese Adresse zurück**
6. **Kopiere diese Adresse nach myRobot**

Operator new hat einen Rückgabewert, nämlich die Anfangsadresse des eingerichteten Objekts.

Der Konstruktor

TECHNISCHE UNIVERSITÄT DARMSTADT

```
Robot myRobot = new Robot ( 3, 1, MyDirection.UP, 0 );
```

1. Richte eine Referenz myRobot ein
2. Suche Speicherplatz, der groß genug für ein Objekt vom Typ Robot ist
3. Reserviere den gefundenen Speicherplatz
4. Rufe den Konstruktor mit der Anfangs-adresse dieses Speicherplatzes auf
5. Liefere diese Adresse zurück
6. Kopiere diese Adresse nach myRobot

Und diese Anfangsadresse haben wir immer mittels Zuweisung in einer Referenz gespeichert.

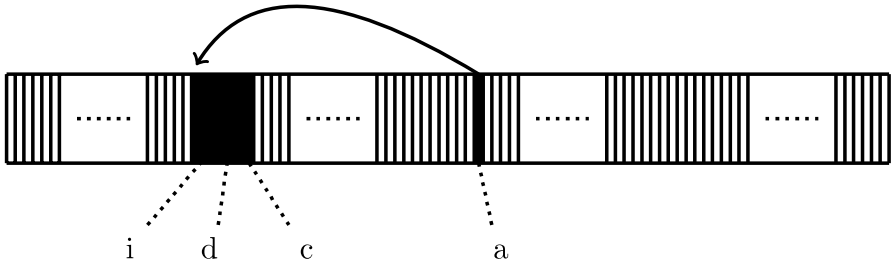
TECHNISCHE UNIVERSITÄT DARMSTADT

Rein illustrative Klasse X

Gegen Ende dieses Kapitels lösen wir uns von FopBot und schauen uns einige der in diesem Kapitel gesehenen Aspekte allgemein anhand einer rein illustrativen Klasse an. Mit „rein illustrativ“ ist gemeint, dass diese Klasse keinen sinnvollen Zweck erfüllt, aber gerade deshalb geeignet ist, die Themen quasi in Reinkultur zu illustrieren.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT



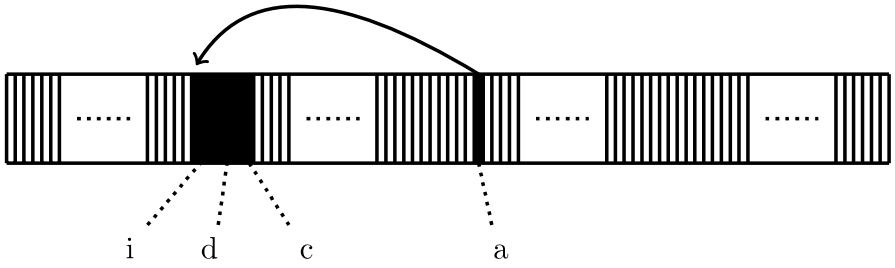

```
public class X {
  public int i;
  public double d;
  public char c;
}
```

```
X a = new X();
```

Wir definieren jetzt keine Roboterklasse mehr, sondern eine beliebige Klasse, die wir etwas phantasielos einfach X nennen.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT



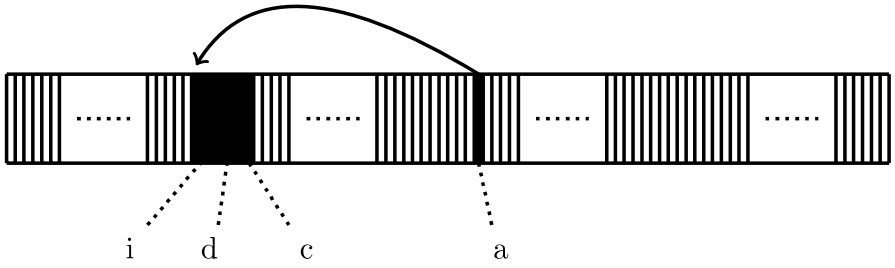

```
public class X {
  public int i;
  public double d;
  public char c;
}
```

```
X a = new X();
```

So wie Referenzen und Objekte von Klasse Robot, können wir Referenzen und Objekte von Klasse X einrichten.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT



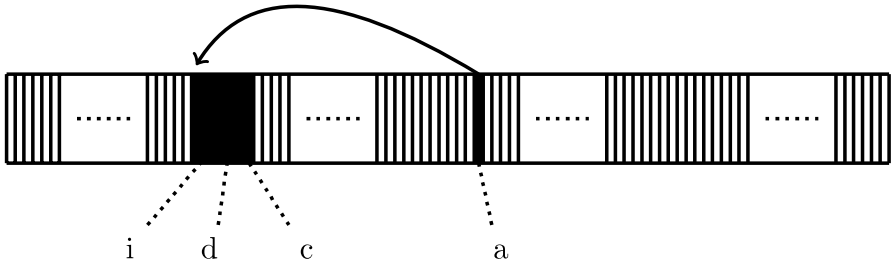

```
public class X {
  public int i;
  public double d;
  public char c;
}
```

```
X a = new X();
```

Offensichtlich haben wir in Klasse X keinen Konstruktor definiert. Dann muss man dennoch ein leeres Klammerpaar hinschreiben. Die Logik dahinter ist die: Wenn man einer Klasse keinen Konstruktor mitgibt, dann erzeugt der Compiler automatisch einen Konstruktor, der keine Parameter hat und auch keine Anweisungen enthält, also keinerlei Effekt hat. Der ist dann aber dennoch aufzurufen.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT



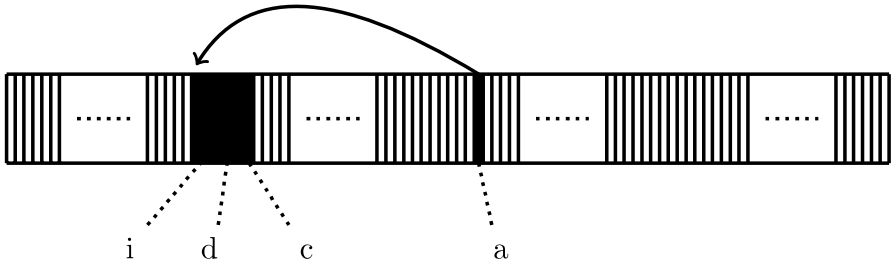

```
public class X {
  public int i = 1;
  public double d = 3.14159;
  public char c = ´z´;
}
```

```
X a = new X();
```

Anstatt einen Konstruktor zur Initialisierung der Attribute einzurichten, kann man auch gleich in der Definition der Klasse festlegen, dass einzelne oder wie hier alle Attribute mit vorgegebenen Werten initialisiert werden. Macht man das nicht, dann werden i und d jeweils auf die Zahl 0 gesetzt und c auf das Zeichen mit Unicode-Wert 0. Dieses Zeichen hat keine eigenständige Bedeutung, sondern steht sozusagen für die Abwesenheit eines echten Zeichens.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT



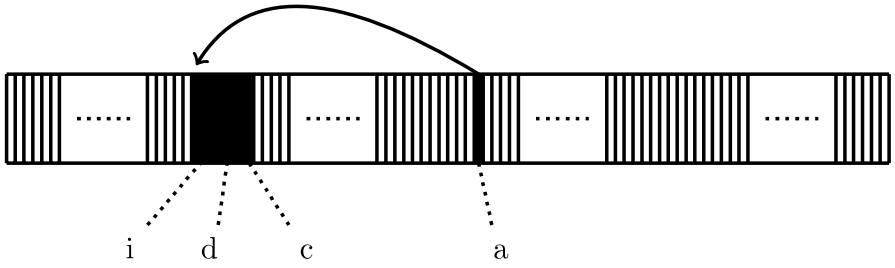

```
public class X {
  public int i;
  public double d;
  public char c;
}
```

```
X a = new X();
a.i = 1;
a.d = 3.14159;
a.c = ´z´;
```

Im Gegensatz zu allen unseren bisherigen Klassen definieren wir in X jetzt die Attribute public, damit wir ohne Umstände damit außerhalb der Klasse X umgehen können, so wie rechts zu sehen.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT




```
public class X {
  public int i;
  public double d;
  public char c;
}
```

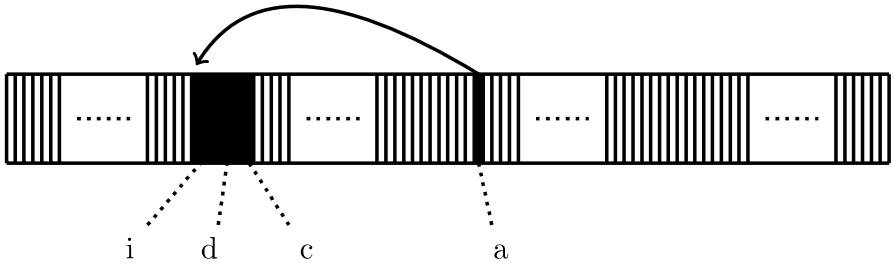
```
X a = new X();
a.i = 1;
a.d = 3.14159;
a.c = ´z´;
```

Wegen public können Sie außerhalb von X auf die einzelnen Attribute lesend und schreibend zugreifen.

Die Notation ist analog zum Aufruf von Methoden: zuerst der Name der Variable, dann der Name des Attributs, beides durch einen Punkt voneinander getrennt. In diesem ersten kleinen Beispiel sind die Zugriffe auf die Attribute i, d und c allesamt schreibend: Die Werte dieser Attribute werden durch Zuweisung auf die jeweils rechts vom Zuweisungszeichen stehenden Werte gesetzt.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT



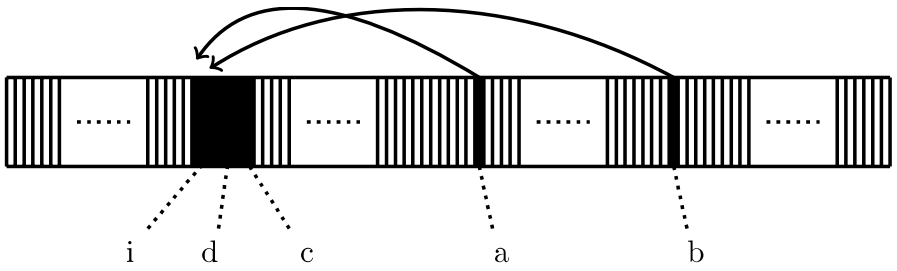

```
public class X {
  public int i;
  public double d;
  public char c;
}
```

```
a.i = a.i + 1;
a.d = a.i;
```

Hier sehen Sie zwei Beispiele für *lesenden* Zugriff auf das Attribut i: In der ersten Zeile wird auf das Attribut i des Objektes lesend, dann schreibend zugegriffen. Der Wert von i wird herausgelesen, die herausgelesene Kopie um 1 erhöht und der resultierende Wert wieder an die Stelle von i geschrieben. In der zweiten Zeile wird auf das Attribut i lesend und auf das Attribut d schreibend zugegriffen.

Zwei Referenzen auf ein Objekt

TECHNISCHE UNIVERSITÄT DARMSTADT



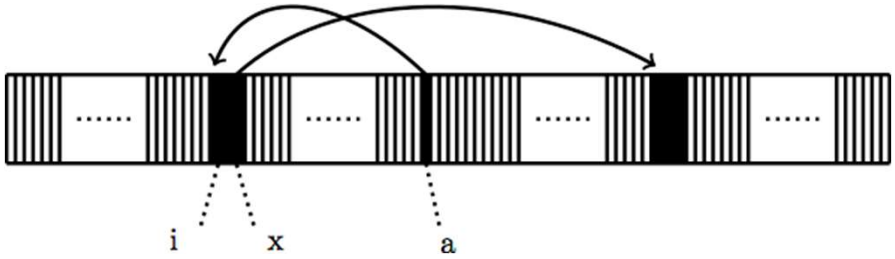

```
public class X {
  public int i;
  public double d;
  public char c;
}
```

```
X a = new X();
X b = a;
a.d = 3.14;    //  <=> b.d = 3.14;
int n = b.i;   //  <=> int n = a.i;
```

Wie bei Klasse MyRobot betrachten wir auch hier kurz den Fall, dass zwei verschiedene Referenzen auf dasselbe Objekt verweisen. Auch hier wird natürlich über beide Referenzen auf dasselbe Objekt zugegriffen.

Klasse X

TECHNISCHE UNIVERSITÄT DARMSTADT




```
public class X {
   .........
}
```

```
public class Y {
   public int i;
   public X x;
}
```

```
Y a = new Y();
a.x = new X();
```

Selbstverständlich müssen Attribute nicht von primitiven Datentypen sein. Genauso gut können Sie Attribute von Klassen haben. Dieses kleine Beispiel zeigt, wie man sich das vorstellen kann. Nach dem bisher Gesagten sollten sie auf dieser Folie eigentlich gar nichts Neues und schon gar nicht irgendetwas Überraschendes finden.