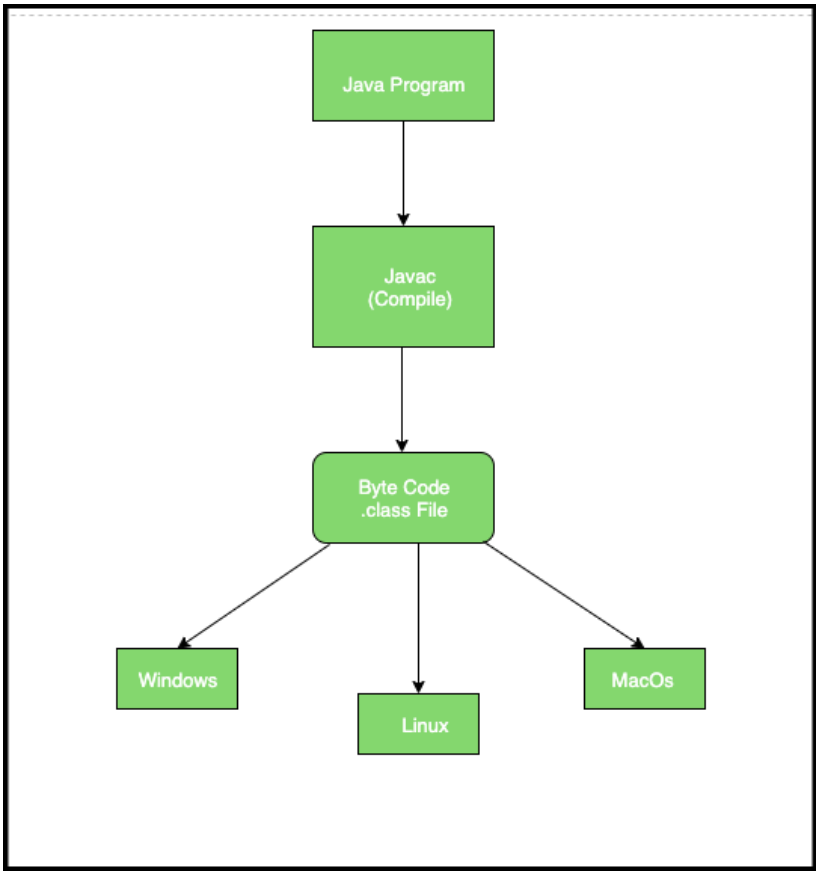
JAVA-HAMSTER HANDBUCH

NIKLAS
METZEN
Iuk-Gb LF02

Inhaltsverzeichnis:

Was ist Java?

Java ist eine Hochsprache, dass bedeutet Sie ist für Menschen verständlich. Das Gegenteil einer Hochsprache ist die Maschinensprache, die Einsen und Nullen, die eine Maschine verstehen kann. Java wurde von Sun Microsystems, Inc im Jahr 1995 entwickelt. Sun Microsystems wurde von Oracle Corporation übernommen. Java wird nun durch Oracle verwaltet. Mehr als 3 Milliarden Geräte benutzen Java für Programme. Java wird für Android Apps, Desktop Anwendungen, Internet Anwendungen, Spiele und vieles Mehr verwendet. Java funktioniert auf allen Plattformen. In traditionellen Programmiersprachen, wenn der Code kompiliert wird, wird der Code in Maschinensprache umgewandelt, die die bestimmte Hardware verstehen kann. Maschinensprache ist nicht universell und jede Hardware versteht unterschiedliche Befehle gar nicht oder anders. Auf anderer Hardware muss der Code wieder kompiliert werden bevor er für die bestimmte Hardware verständlich ist. In Java wird das Programm erst in Java Bytecode umgewandelt. Java Bytecode kann von einer Java Virtual Machine (JVM) in den bestimmten Maschinencode der Hardware umgewandelt werden. Das heißt solange JVM, was ein Teil von der Java Runtime Environment (JRE) ist, auf der bestimmten Maschine installiert ist kann jeder Java Code.



Warum sollte man Java verwenden?

1. Java funktioniert, wie schon erwähnt, auf allen Plattformen.
2. Java ist einfach zu lernen.
    a. Sehr ähnlich der englischen Sprache
3. Es gibt viele Open-Source Java Bibliotheken und Software Development Kits.
    a. Bekannteste JDKs sind Amazon Corretto oder AdoptOpenJDK
4. Java ist relativ ähnlich zu C++ und C#
    a. Einfaches wechseln der Sprachen.

Grundelemente von Java

Klassen

Java ist eine Objekt orientierte Programmiersprache. Alles in Java hängt mit einer Klasse zusammen. In Klassen sind Methoden und Eigenschaften eines Objekts definiert. Eine Klasse dient als Vorlage, aus dieser Vorlage können beliebig viele Objekte erzeugt werden. Ähnlich zu einer Blaupause.

Objekte

Objekte sind einzelne Instanzen von Klassen, denen Eigenschaften der bestimmten Klassen hinzugefügt werden können. Objekte erleichtern es, Daten aus dem echten Leben digital darzustellen. Mit einer Klasse, die bestimmte Eigenschaften hat, können unendlich viele Objekte erstellt werden, die alle eine gewisse Ähnlichkeit haben jedoch auch bestimmte Unterschiede. Erklären lässt sich dies mit dem Beispiel, dass alle BMWs, Fahrzeuge sind, aber nicht alle Fahrzeuge sind BMWs. Ein Fahrzeug kann somit als Klasse funktionieren, das bestimmte Eigenschaften beschreibt die ein Objekt der Klasse haben könnte (Räder, Lenkrad, Motor, etc.). Die Fahrzeuge, die daraus entstehen, haben gewisse Ähnlichkeit zu anderen Fahrzeugen unterscheiden sich jedoch unter den selbst definierten Aspekten.

Void

Void wird deklariert, wenn die genutzte Methode kein Ergebnis wiedergeben soll.

Private

Programmelemente, die mit „Private“ deklariert werden, sind nur in Ihrer eigenen Klasse sichtbar, dass heißt nur innerhalb der Klasse kann auf das Programmelement zugegriffen werden. Private-Elemente sollten definiert werden, wenn wichtige Details zu verbergen sind.

Public

Programmelemente, die mit „Public“ deklariert werden, sind nur in Ihrer eigenen Klasse sichtbar, dass heißt die Elemente können von jeder anderen Klasse verwendet werden. Eine Klasse muss in der Quelldatei als „Public“ deklariert werden, damit das Programm ausgeführt werden kann. Diese Klasse ist meistens die sogenannte „Main“-Klasse.

Default

Wenn Programmelemente nicht deklariert werden bekommen sie automatisch die „default“ Deklarierung. Die Deklarierung „default“ bedeutet, dass dieses Element nur sichtbar innerhalb der eigenen Klasse ist.

Protected

Programmelemente, die mit „protected” deklariert werden, sind in ihrer eigenen Klasse und innerhalb aller abgeleiteten Klassen verwendbar. Für Klassen außerhalb der Deklarierung sind die „protected“ Elemente nicht sichtbar.

Main

Die main()-Methode ist der Startpunkt jeder Java-Anwendung, jede Java-Anwendung besitzt eine Methode main(). Main ist der Hauptort an dem der Java-Code ausgeführt wird, alle im Programm verwendeten Klassen und Elemente haben einen Zusammenhang mit der Main Methode.

Operatoren

In Programmiersprachen werden Operatoren genutzt um Operationen mit Variablen und Werten zu tätigen. In Java werden gängige Operatoren in folgende Gruppen kategorisiert:

Arithmetische Operatoren:

Arithmetische Operatoren werden genutzt, um mathematische Operationen mit Variablen zu tätigen.

1. Das Symbol für den Addition Operator ist „+“. Mit diesem Operator werden Variablen addiert.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x - y);
  }
}

//Die Variable "x" und "y" werden mit zwei verschiedenen Werten definiert. In der System.out.println() Funktion
//werden diese beiden Operatoren addiert, bevor das Ergebnis im Terminal erscheint.
```

2. Das Symbol für den Subtraktion Operator ist „-“. Mit diesem Operator werden Variablen subtrahiert.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x - y);
  }
}

//Die Variable "x" und "y" werden mit zwei verschiedenen Werten definiert. In der System.out.println() Funktion
//werden diese beiden Operatoren subtrahiert, bevor das Ergebnis im Terminal erscheint.
```

3. Das Symbol für den Multiplikation Operator ist „*“. Mit diesem Operator werden Variablen multipliziert.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x * y);
  }
}

//Die Variable "x" und "y" werden mit zwei verschiedenen Werten definiert. In der System.out.println() Funktion
//werden diese beiden Operatoren multipliziert, bevor das Ergebnis im Terminal erscheint.
```

4. Das Symbol für den Division Operator ist „/“. Mit diesem Operator werden Variablen dividiert.

```
public class Main {
  public static void main(String[] args) {
    int x = 12;
    int y = 3;
    System.out.println(x / y);
  }
}

//Die Variable "x" und "y" werden mit zwei verschiedenen Werten definiert. In der System.out.println() Funktion
//werden diese beiden Operatoren dividiert, bevor das Ergebnis im Terminal erscheint.
```

5. Das Symbol für den Modulus Operator ist „%“. Mit diesem Operator werden Variablen dividiert und ein Rest kann ermittelt werden.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 2;
    System.out.println(x % y);
  }
}

//Die Variable "x" und "y" werden mit zwei verschiedenen Werten definiert. In der System.out.println() Funktion
//werden diese beiden Operatoren mit der Modulus Operator dividiert, im Terminal erscheint der Rest der
//division.
```

6. Das Symbol für den Inkrementellen Additions Operator ist „++“. Mit diesem Operator werden Variablen um Eins erweitert.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    ++x;
    System.out.println(x);
  }
}

//Die Variable "x" wird definiert. Darauffolgend in der nächsten Funktion wird "x" um Eins erweitert. In der
//System.out.println() funktion wird der "x" Wert, der um Eins erweitert wurde, als 6 ausgegeben.
```

7. Das Symbol für den Inkrementellen Subtraktions Operator ist „++“. Mit diesem Operator werden Variablen um Eins verkleinert.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    --x;
    System.out.println(x);
  }
}

//Die Variable "x" wird definiert. Darauffolgend in der nächsten Funktion wird "x" um Eins verkleinert. In der
//System.out.println() funktion wird der "x" Wert, der um Eins verkleinert wurde, als 4 ausgegeben.
```

Zuordnungs Operatoren:

Zuordnungs Operatoren werden genutzt, um Variablen bestimmte Werte zu zuweisen.

1. Um einer Variable einen Wert zuzuweisen, benutzt man das Symbol „=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(x);
  }
}

//Die Variable x wird gleich die Zahl 5 gesetzt. Im Terminal wird die Zahl 5 ausgegeben.
```

2. Um zu einer Variable einen anderen Wert zu addieren, benutzt man das Symbol „+=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x += 3;
    System.out.println(x);
  }
}

//Die Variable "x" wird gleich 5 gesetzt. Die Variable "x" wird mit der Zahl 3 addiert. Im Terminal wird die
//Zahl 8 ausgegeben.
```

3. Um zu einer Variable einen anderen Wert zu subtrahieren, benutzt man das Symbol „-=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x -= 3;
    System.out.println(x);
  }
}

//Die Variable "x" wird gleich 5 gesetzt. Die Variable "x" wird mit der Zahl 3 subtrahiert. Im Terminal wird die
//Zahl 2 ausgegeben.
```

4. Um zu einer Variable einen anderen Wert zu multiplizieren, benutzt man das Symbol „*=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    x *= 3;
    System.out.println(x);
  }
}

//Die Variable "x" wird gleich 5 gesetzt. Die Variable "x" wird mit der Zahl 3 multipliziert. Im Terminal wird
// die Zahl 2 ausgegeben.
```

5. Um zu einer Variable einen anderen Wert zu dividieren, benutzt man das Symbol „/=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    double x = 5;
    x /= 3;
    System.out.println(x);
  }
}

//Die Variable "x" wird gleich 5 gesetzt. Die Variable "x" wird mit der Zahl 3 diviert. Im Terminal wird die
//Zahl 1.6666 ausgegeben.
```

Vergleichs Operatoren

Mit Vergleichs Operatoren können Variablen verglichen werden.

1. Um eine Variable gleich eine andere Variable zu setzen, benutzt man das Symbol „==“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 5;
    System.out.println(x == y);
  }
}

//Die Variable "x" und "y" wird definiert. In der System.out.println() Funktion wird "x" gleich "y" gesetzt
//Im Terminal erscheint "true" als Ausgabe, da "x" und "y" tatsächlich gleich ist.
```

2. Um eine Variable nicht gleich eine andere Variable zu setzen, benutzt man das Symbol „!=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 5;
    System.out.println(x != y);
  }
}

//Die Variable "x" und "y" wird definiert. In der System.out.println() Funktion wird "x" gleich "y" gesetzt
//Im Terminal erscheint "false" als Ausgabe, da "x" und "y" tatsächlich ungleich ist.
```

3. Um eine Variable kleiner als eine andere Variable zu setzen, benutzt man das Symbol „<“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x < y);
  }
}

//Die Variable "x" und "y" wird definiert. In der System.out.println() Funktion wird geprüft ob "x" größer als
//"y" ist. Im Terminal erscheint "false" als Ausgabe, da "y" kleiner als "x" ist.
```

4. Um eine Variable größer als eine andere Variable zu setzen, benutzt man das Symbol „>“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x > y);
  }
}

//Die Variable "x" und "y" wird definiert. In der System.out.println() Funktion wird geprüft ob "x" größer als
//"y" ist. Im Terminal erscheint "true" als Ausgabe, da "x" größer als "y" ist.
```

5. Um eine Variable kleiner oder gleich als eine andere Variable zu setzen, benutzt man das Symbol „<=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x <= y);
  }
}

//Die Variable "x" und "y" wird definiert. In der System.out.println() Funktion wird geprüft ob "x" kleiner oder
//gleich "y" ist. Im Terminal erscheint "false" als Ausgabe, da "y" kleiner oder gleich "x" ist.
```

6. Um eine Variable größer oder gleich als eine andere Variable zu setzen, benutzt man das Symbol „>=“ als Operator.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    int y = 3;
    System.out.println(x >= y);
  }
}

//Die Variable "x" und "y" wird definiert. In der System.out.println() Funktion wird geprüft ob "x" größer oder
//gleich "y" ist. Im Terminal erscheint "true" als Ausgabe, da "x" größer oder gleich "y" ist.
```

Logische Operatoren

Logische Operatoren werden genutzt um, logische Ausdrücke zwischen zwei Variablen zu ermitteln.

1. Das Symbol „&&“ wird für das Logische Und verwendet und ist wahr, wenn beide Bedingungen erfüllt sind.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(x > 3 && x < 10);
  }
}

//Die Variable "x" wird definiert. Im Terminal erscheint "true", da "x" sowohl größer als 3 ist und kleiner als
//10 ist.
```

2. Das Symbol „||“ wird für das Logische Oder verwendet und ist wahr, wenn eine der beiden Bedingungen erfüllt sind.

```
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(x > 3 || x > 10);
  }
}

//Die Variable "x" wird definiert. Im Terminal erscheint "true", da "x" sowohl größer als 3 ist jedoch nicht
//größer als 10 ist. Es muss jedoch nur einer der beiden Bedingungen erfüllt sein.
```

3. Das Symbol „!“ wird für das Logische nicht benutzt. Das Ergebnis der Abfrage wird verneint.

```java
public class Main {
  public static void main(String[] args) {
    int x = 5;
    System.out.println(!(x > 3 && x < 10)); // returns false because ! (not) is used to reverse the result
  }
}

//Die Variable "x" wird definiert. Im Terminal erscheint "false", das eigentliche Ergebnis der und Abfrage ist
//"true" und das ! Symbol dreht das Ergebnis um.
```

Datentypen

Was ist ein Datentyp?

Ein Datentyp gibt an, von welcher Art die Daten sind, die mit ihm beschrieben werden und welche Operationen auf diese ausgeführt werden können. Folgendes sind gängige Datentypen in Java:

Int

Mit der Int Klassifikation können Ganze Zahlen zwischen – 2147483648 und 2147483647 dargestellt werden.

```java
public class Main {
  public static void main(String[] args) {
    int myNum = 100000;
    System.out.println(myNum);
  }
}

//Im Terminal wird 100000 ausgegeben.
```

Char

Mit der Char Klassifikation können einzelne Buchstaben dargestellt werden. Eine Char Variable muss mit einem einzelnen Anführungszeichen deklariert werden.

```java
public class Main {
  public static void main(String[] args) {
    char myGrade = 'B';
    System.out.println(myGrade);
  }
}

// Im Terminal wird der einzelne Buchstaben "B" ausgegeben.
```

String

Mit der String Klassifikation können mehrere Buchstaben in einer Variable zusammengefasst und dargestellt werden.

```java
public class Main {
  public static void main(String[] args) {
    String str = "abc";
    System.out.println(str);
  }
}

//Im Terminal wird "abc" ausgegeben.
```

Float

Mit der Float Klassifikation können Reelle Zahlen zwischen 3.4e-038 und 3.4e+038 dargestellt werden. Die Zahl, die in der Variable definiert wird, sollte mit einem „f“ beendet werden.

```
public class Main {
  public static void main(String[] args) {
    float myNum = 5.75f;
    System.out.println(myNum);
  }
}

//Im Terminal wird 5.75 ausgegeben.
```

Double

Mit der Double Klassifikation können Reelle Zahlen zwischen 1.7e-308 und 1.7e+308 dargestellt werden. Die Zahl, die in der Variable definiert wird, sollte mit einem „d“ beendet werden.

```
public class Main {
  public static void main(String[] args) {
    double myNum = 19.99d;
    System.out.println(myNum);
  }
}

//Im Terminal wird 19.99 ausgegeben.
```

Boolean:

Boolean sind eine der wichtigsten Bestandteile der Programmierung. Mit der Boolean Klassifikation können Variablen definiert werden, die immer nur einen von zwei Zuständen haben können. Die Variable ist entweder „true“ (Wahr) oder „false“ (Falsch). Die Variable kann nicht „true“ und „false“ gleichzeitig sein, Boolean ist binär.

```
public class Main {
  public static void main(String[] args) {
    boolean isJavaFun = true;
    boolean isFishTasty = false;
    System.out.println(isJavaFun);
    System.out.println(isFishTasty);
  }
}

//Zwei Boolische Ausdrücke werden definiert. Das Ergebnis der Boolische Ausdrücke wird im Terminal erscheinen.
```

Anweisungen

In der Programmierung sind Anweisungen Programmelemente, die bestimmte Variablen vergleichen können und basierend darauf, ob die Variable eine Bedingung erfüllt oder nicht, einen vordefinierten Block Code ausführen.

If-Abfrage

Das If Element führt einen bestimmten Code-Block aus, wenn die angegebene Bedingung erfüllt (auch „true“ genannt) ist.

Das Else Element wird ausgeführt, wenn die angegebene Bedingung nicht erfüllt (auch „false“ genannt) ist.

Das Else-if Element kann einen weiteren Code-Block ausführen, wenn die Bedingung des ersten If-Elements nicht erfüllt wird, aber die Bedingung des Else-If-Element erfüllt ist.

```
public class Main {
  public static void main(String[] args) {
    int time = 22;
    if (time < 10) {
      System.out.println("Good morning.");
    } else if (time < 20) {
      System.out.println("Good day.");
    }  else {
      System.out.println("Good evening.");
    }
  }
}

//Je nachdem wie groß die Variable "time" ist wird
//entweder der if, else if oder else Block ausgeführt.
```

For-Schleife

Die For-Schleife wiederholt den angegeben Code-Block solange, bis eine vorbestimmte Variable erreicht ist.

In den Argumenten der For-Schleife können mehrere Aussagen getroffen werden. Als erstes wird eine Variable mit einer bestimmten Zahl zugewiesen (meistens ein Integer), daraufhin wird mit einem Operator bestimmt, welche Bedingung erfüllt werden muss. Die For-Schleife wird solange ausgeführt bis die Bedingung nicht mehr erfüllt wird. In der darauffolgenden Aussage kann bestimmt werden ob die Variable erweitert wird. Dies wird meist inkrementell mit dem „++“ Operator getätigt, kann aber auch beliebig erweitert werden.

```
public class Main {
  public static void main(String[] args) {
    for (int i = 0; i < 5; i++) {
      System.out.println(i);
    }
  }
}

//Zuerst wird die Integer Variable "i" definiert. Die
//Variable hat die Größe 0 am Anfang. Die For-Schleife //wird solange ausgeführt wie "i" kleiner als 5 ist.
//Nach dem Ausführen der For-Schleife wird "i" um eins //inkrementell erweitert.
```

While-Schleife

Die While-Schleife wiederholt einen bestimmten Code-Block solange, wie die angegebene Bedingung erfüllt (true) ist, dass heißt eine Bedingung die immer erfüllt ist, lässt die Schleife für immer ausführen. Damit die While-Schleife nicht für immer läuft, kann im Code-Block die Variable inkrementell erweitert werden, bis die Bedingung nicht mehr erfüllt (false) ist. Die While-Schleife ist Kopfgesteuert, dass bedeutet der Code Block wird nur ausgeführt, wenn die Bedingung am Kopfteil erfüllt wird.

```
public class Main {
  public static void main(String[] args) {
    int i = 0;
    while (i < 5) {
      System.out.println(i);
      i++;
    }
  }
}

//Die Integer Variable "i" wird definiert. Solange "i" kleiner als 5 ist wird die While-Schleife ausgeführt.
//In dem Code-Block wird "i" um 1 jedes mal erweitert. Wenn "i" größer als 5 ist stoppt die While-Schleife.
```

Do-while-Schleife

Die Do-While-Schleife ist eine Variation der While-Schleife. In der Do-While-Schleife wird der angegebene Code-Block einmal ausgeführt und nur wenn die Bedingung erfüllt ist, wird der Code-Block wiederholt. Wenn die Bedingung nicht erfüllt ist, stoppt die Do-While-Schleife nach dem einmaligen ausführen. Deswegen wird die Do-While-Schleife auch Fußgesteuert genannt.

```
public class Main {
  public static void main(String[] args) {
    int i = 0;
    do {
      System.out.println(i);
      i++;
    }
    while (i < 5);
  }
}

//Die Variable "i" wird definiert. Die Do-While Schleife wird einmal ausgeführt. Die "i" Variable wird um Eins
//erweitert. Daraufhin wird überprüft ob "i" kleiner als Fünf ist. Solange "i" kleiner als Fünf ist wird die
//Schleife ausgeführt.
```

Switch Case:

Der Switch Case ist eine ähnliche Art der IF-Abfrage. Der Switch Case wählt anhand des gegebenen Arguments einen der spezifizierten Cases aus. Der Ablauf ist wie folgt:

Der Switch Ausdruck wird einmalig ausgewährtet. Die gegebene Variable wird mit jedem einzelnen Case verglichen. Wenn die Variable mit einer der definierten Cases übereinstimmt, wird der übereinstimmende Case ausgewählt und ausgeführt.

```
public class Main {
  public static void main(String[] args) {
    int tag = 4;
    switch (tag) {
      case 1:
        System.out.println("Montag");
        break;
      case 2:
        System.out.println("Dienstag");
        break;
      case 3:
        System.out.println("Mittwoch");
        break;
      case 4:
        System.out.println("Donnerstag");
        break;
      case 5:
        System.out.println("Freitag");
        break;
      case 6:
        System.out.println("Samstag");
        break;
      case 7:
        System.out.println("Sonntag");
        break;
    }
  }
}

//Zuerst wird die Integer Variable "tag" definiert
//Im Switch Case wird überprüft welcher Case zu der
//Variable passt. In diesem Fall wird im Terminal
//Donnerstag ausgegeben
```

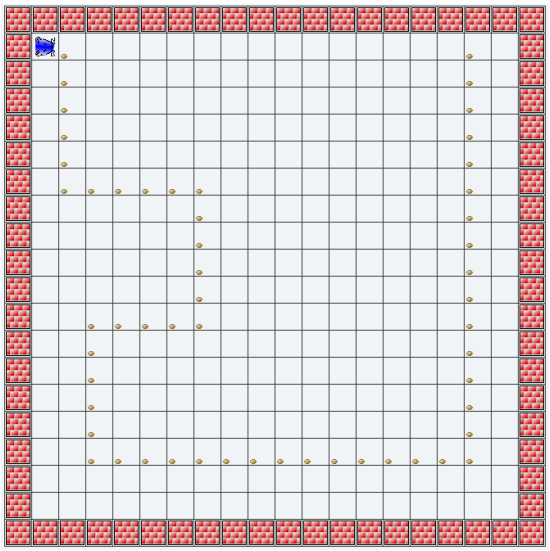
Was ist Java Hamster?

Das Java-Hamster-Modell ist ein Modell, mit dem Grundkonzepte der Programmierung auf spielerische Art und Weise erlernt werden sollen. Es werden Programme geschrieben, mit denen ein virtueller Hamster durch eine virtuelle Landschaft gesteuert wird. Das Hamster-Modell spezialisiert sich nicht nur auf Java, es kann auch in verschiedenen Sprachen Programmiert werden. Java-Hamster wird genutzt, da Java eine immer noch weit verbreitete Sprache ist. Java ist eine gute Sprache um Programmieren anzufangen.

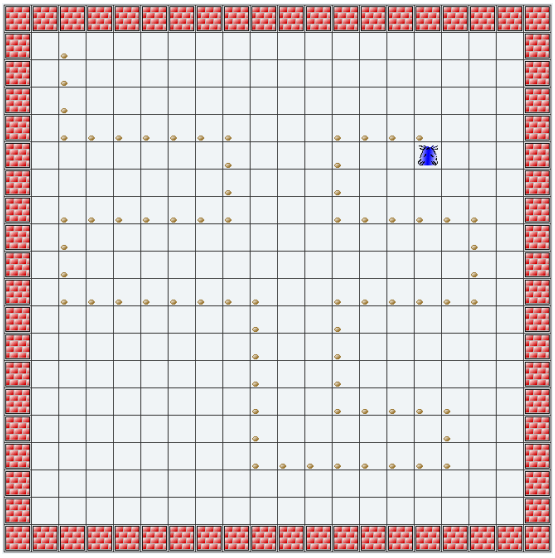
Aufgabe

Der Hamster soll einer Körnerspur folgen und anhalten, wenn er an einer Wand ankommt.

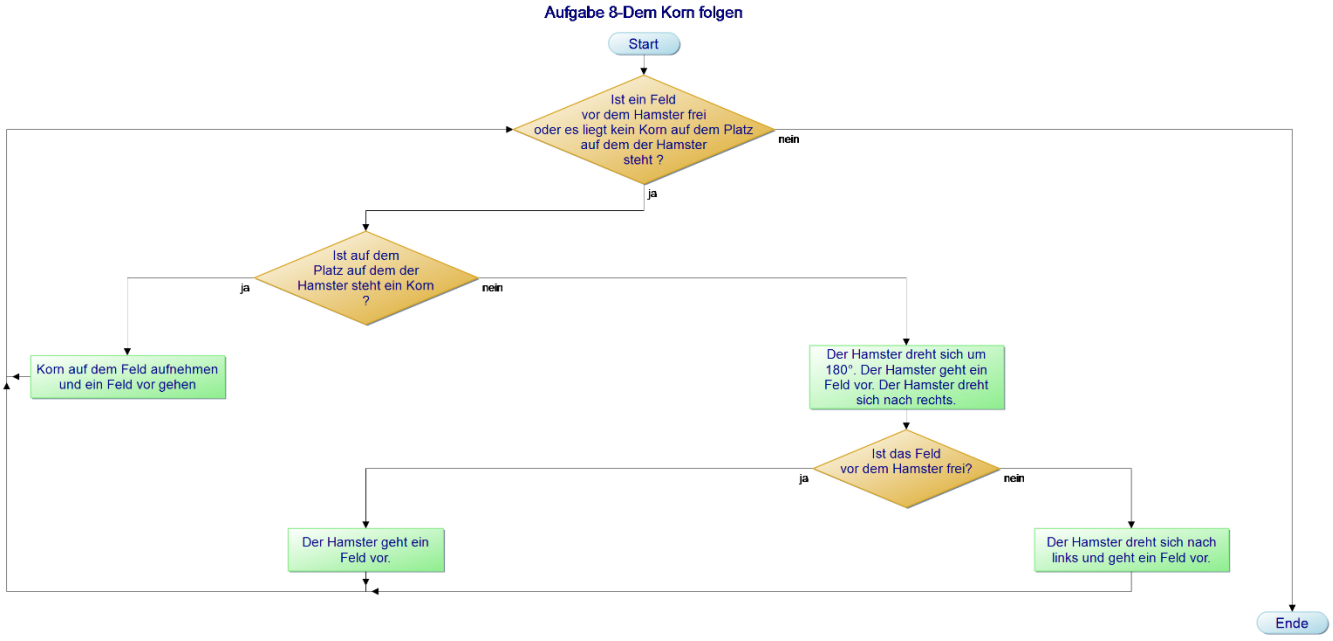
Territorium Vorlage der PFD:



Eigenes Territorium zum Vergleich:



Ablauf-Diagramm des Programms



Quellcode des Programms

```
void main() {
vor();
vorausgehen();
}

//Wenn ein Korn auf dem Feld ist soll das Korn aufgenommen werden und ein Schritt nach vorne gegangen werden.
//Wenn kein Korn auf dem Feld ist soll der Hamster sich um 180° drehen auf das vorherige Feld vor gehen.
//Auf dem Feld dreht der Hamster sich nach rechts. Wenn vor dem Hamster das Feld frei ist geht der Hamster
vor.
//Wenn das Feld nicht frei ist dreht der Hamster sich nach links und geht ein Feld vor.
//Daraufhin geht der Hamster wieder einen schritt vor.
void Wegfrei(){
if(kornDa()){
    nimm();
    vor();
    }
    else{
        umdrehen();
        vor();
        rechtsUm();
        if(!vornFrei()){
            linksUm();
            vor();
        }
        else{
            vor();
        }
    }
}

//Stätige Ausführung der Wegfindung. Wenn der Weg vor dem Hamster frei ist oder kein Korn auf dem Feld liegt.
void vorausgehen(){
do{
    Wegfrei();
}
    while(vornFrei() || !kornDa());
}

//Der Hamster dreht sich nach rechts.
void rechtsUm(){
umdrehen();
linksUm();
}

//Der Hamster dreht sich um 180°.
void umdrehen(){
linksUm();
linksUm();
}
```

Probleme

In der „Vorausgehen“ Funktion wurden zuerst Körner in das Maul des Hamsters gelegt, somit wurde die Funktion ausgeführt. Das Problem besteht darin, dass in der Aufgabenstellung nichts davon steht, das Körner in das Maul des Hamsters gelegt werden dürfen und was viel wichtiger ist, dass Programm hat mit dem MauerDaException Fehler terminiert, somit hat der Hamster nicht gestoppt, weil der Weg zu Ende war, sondern er stoppte, weil er versuchte gegen die Wand am Ende des Pfades zu laufen.

```
void vorausgehen(){
do{
    Wegfrei();
}
    while(!maulLeer());
}
```

```
void vorausgehen(){
do{
        Wegfrei();
}
        while(vornFrei() || !kornDa());
}
```

Lösung

Mit der ODER Abfrage in der do-while-Schleife schaut der Hamster, ob er auf das Feld vor Ihm gehen könnte oder ob auf dem Feld, auf dem der Hamster steht kein Korn ist. Diese Bedingung wird nur erfüllt, wenn der Hamster am Ende des Pfades ist. Sprich, vor dem Hamster ist eine Mauer. Also ist die „vornFrei()“ Boolesche Abfrage „false“ und auf dem Platz, auf dem der Hamster steht ist ein Korn. Somit die die „!kornDa()“ Boolesche Abfrage auch „false“.

Reflektion

Das Java-Hamster-Modell ist ein gutes Modell um die Grundsätze von Java kennenzulernen. Ich wäre gerne noch ein wenig Tiefer in die Materie gegangen. Dinge wie Klassen habe ich nur oberflächlich verstanden. Das Java-Hamster-Modell oder beziehungsweise die IDE des Modells ist jedoch sehr instabil und beim Benutzen von Shortcuts kann es manchmal passieren, dass die IDE abstürzt. Deswegen werde ich nicht mit den Java-Hamster-Modell weitermachen. Ich werde jedoch aus eigenem Interesse noch weiter Java programmieren. Mir hat dieses kleine Projekt sehr gefallen.

Zeitlicher Ablaufplan

- 2 Stunden
    - Planung und Entwicklung
- 4 Stunden
    - Dokumentation und Nachforschung
- 1 Stunde
    - Aufarbeitung des Handbuchs

Quellen (Stand 20.12.2020)

What is Java? A Beginner's Guide to Java and its Evolution | Edureka (Was ist Java?)

Java-Tutorial für Einsteiger: Professionell von Beginn an! (jaxenter.de) (Grundelemente von Java)

Java Operators (w3schools.com) (Operatoren)

What is data type? - Definition from WhatIs.com (techtarget.com) (Was ist ein Datentyp?)

Java int Keyword (w3schools.com) (Int)

Java char Keyword (w3schools.com) (Char)

String (Java Platform SE 7) (oracle.com) (String)

Java float Keyword (w3schools.com) (Float)

Java double Keyword (w3schools.com) (Double)

Java boolean Keyword (w3schools.com) (Boolean)

Java if Keyword (w3schools.com) (If-Abfrage)

Java for Keyword (w3schools.com) (For-Schleife)

Java while Keyword (w3schools.com) (While-Schleife)

Java do Keyword (w3schools.com) (Do-While-Schleife)

inf-schule | Schleifen » Kopf- und fußgesteuerte Schleifen (inf-schule.de) (Kopf- und Fußgesteuerte Schleifen)

Java switch Keyword (w3schools.com) (Switch-Case)

Java-Hamster-Modell (java-hamster-modell.de) (Was ist das Java-Hamster-Modell?)

PapDesigner (friedrich-folkmann.de) (PAP-Editor)

GitHub - dCdUP/Java_Hamster_Projekt (Git-Hub Repository)