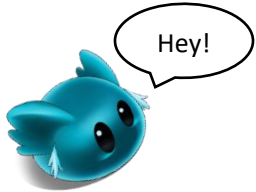

CAMP MINECRAFT MODS



- Programmierbasics -

Kapitel I – Einführung

Willkommen bei dem Camp Mods Programmieren mit Minecraft bei den ForscherFreunden! Wir freuen uns sehr, dass ihr den Weg hierhergeschafft habt – oder euch die Doku nach eurem Camp zuhause noch einmal durchlest. In dieser Dokumentation begleite ich euch dabei, alles möglichst einfach und gut zu verstehen, damit ihr ganz viel Spaß beim Lernen vom Coding haben könnt!

Ganz vergessen – Ich, das bin ich, Cloudy von den Forscherfreunden, und ich begleite euch durch die Doku und auf eurem Weg eure erste eigene Mod zu erstellen! Freut mich sehr!



Was wisst ihr denn so übers Programmieren allgemein – vielleicht habt ihr ja schonmal ein paar Sachen über diese Tätigkeit gehört. Vielleicht wisst ihr schon, dass Programmieren in vielen verschiedenen Berufsfeldern und Bereichen in der Welt zum Einsatz kommen. Ganz besonders bei den Leuten, die etwas mit Informatik studiert haben – also zum Beispiel Softwareengineers, Software Developers oder Systemadministratoren. Oh – aber was war das denn? Das waren ein paar englische Begriffe, aber was machen die hier in einer deutschen Doku? Einfach gesagt, findet die englische Sprache eine große Verwendung in der Informatik, vor allem im Bereich des Programmierens. Stellt euch also darauf ein, dass wir auch im Laufe der Woche eine ganze Menge englischer Begriffe benutzen und brauchen werden – ich hoffe ihr könnt alle gut Englisch!

Nun aber vielleicht etwas genauer zu dem, was wir überhaupt mit euch machen wollen – immerhin seid ihr ja nicht zum Lesen hier. Wir werden uns heute gemeinsam mal etwas genauer die Einführung zum Programmieren anschauen – immerhin arbeiten wir vermutlich mit Sachen, die ihr zuvor noch nie genutzt habt. So... die Einführung hätten wir dann geschafft! Wir sehen uns in Kapitel 2 wieder!



Kapitel 2 – Die Grundlagen

Nachdem wir jetzt die Einführung hinter uns haben, wollen wir uns jetzt doch einmal wirklich mit Programmieren beschäftigen – oder?!

Nicht ganz – wir fangen mit etwas anderem an, nämlich dem Umgang mit der sogenannten **Entwicklungsumgebung**... aber was ist das?

Eine **Entwicklungsumgebung** (Auch IDE genannt – kurz für Integrated Development Environment – sagte ja, wir brauchen noch etwas Englisch ^^) wird von Entwicklern genutzt, um sich die Arbeit und das Schreiben, Lesen und Anpassen von Code einfacher zu machen. Also kurzgesagt, ist das eine Möglichkeit sich weniger Arbeit zu machen! Jedoch muss man dafür die Funktionen der IDE wirklich verstehen, um all die Vorteile ausnutzen zu können. Alternativ kann man sogar in jedem beliebigen Texteditor seinen Java code schreiben, allerdings ist das eine unnötige Herausforderung, die kaum ein Programmierer wirklich machen würde....

Wir werden in den nächsten Tagen die Programmiersprache **Java** verwenden, dafür gibt es eine großartige Entwicklungsumgebung, die extra für diese Sprache entwickelt wurde – nämlich **IntelliJ IDEA**.

Beim Programmieren unterscheidet man zwischen zwei großen Arten an **Programmiersprachen**.

- *Textbasierte* Kontextsprachen (z.B. JavaScript, TypeScript, C, C#, ...)
- *Objektorientierte* Sprachen (z.B. Java, Python, GDScript, ...)

Die große Unterscheidung zwischen den Sprachen ist der *Syntax*, also das wie die Sprache geschrieben werden muss, damit der *Compiler* (nur ein fancy englisches Wort für das Bauteil, was den Code dann versteht und umsetzt) versteht, was er mit dem Code anfangen soll.

Zum Beispiel vergleichen wir verschiedene Sprachen ganz häufig über die Wortlänge des Codes um herauszufinden, wie komplex die Syntax der jeweiligen Sprache ist – und somit auch wie einfach oder schwer Verständlich für einen Menschen zu lesen.

Als Vergleich:

In einer Sprache wie **Python**, kann man z.B. ein einfaches Programm, welches „Hello World“ ausgibt, in etwa so schreiben:

```
print ("Hello World")
```

So viele verschiedene **Programmiersprachen**...
wusstet ihr, dass es über 1500 verschiedene
Programmiersprachen gibt? Ne ganze Menge...



Wohingegen wir in **Java**, deutlich mehr code brauchen, um dasselbe zu bewerkstelligen:

```
new *
public class HelloWorld {

    1 usage
    public static String message = "Hello World";
    new *
    public static void main(String[] args) {
        System.out.println(message);
    }
}
```

In **Java** brauchen wir nämlich nicht nur den **Methodenaufruf**, sondern auch direkt eine eigene **Klasse**, ein **Objekt** und dann eine **Instanz** des **Objektes**. Was das alles genau ist, schauen wir uns gleich in Ruhe nochmal an, nur so viel vorweg: In **Java** brauchen wir immer ein **Objekt** und eine **Klasse**, derer das **Objekt** angehört – also zum Beispiel einen roten Ferrari, als **Objekt**, und die **Klasse** wäre dann „Auto“.

Des Weiteren benötigen wir noch ein paar weitere Begriffe, um nachher gut mit **Java** arbeiten zu können. Hier kommt einmal eine Liste von allen wichtigen Begriffen, wir gehen auf diese dann in den folgenden Kapiteln genauer ein, immer wenn wir welche davon brauchen:

Name	Funktion
Datentypen	Ein Datentyp ist eine Definition eines Wertes, z.B.: <i>String</i> als Datentyp eines Schriftstückes, <i>Integer</i> als Datentyp für ganze Zahlen usw. Die Datentypen referenzieren in Java immer eine Klasse – also z.B. auch unsere Auto Klasse
Identifizier	Der Identifizier wird zur Erkennung, oder auch zur Wiedererkennung eines Datentyps genutzt. Man kann den Identifizier auch als einmaligen Namen verstehen.
Attribute	Attribute sind Eigenschaften, die für eine Klasse relevant sind – also z.B.: die Anzahl der Reifen eines Autos = 4, oder die Farbe = „rot“. Sie sind immer einem Datentyp zugeordnet, also schreiben wir z.B. für die Anzahl der Reifen: int Reifen = 4;
Wiedergabewert	Der Wiedergabewert ist für Funktionen wichtig, und setzt sich aus dem Schlüsselwort „return“ und dem Attribut zusammen, welches zurückgegeben wird.
Funktion	Eine Funktion ist eine bestimmte Methode , welche eine bestimmte Aktion übernimmt. So ist z.B. bei einem Auto eine Funktion das „Fahren“. Eine Funktion wird immer durch „()“ deutlich gemacht. Also hier: „ fahren() “
Klasse	Eine Klasse setzt sich nun aus allen Attributen dieser Klasse und ihrer Funktionen zusammen, also z.B.: ein Auto – besitzt 4 Reifen und kann fahren. Eine Klasse ist quasi der Bauplan für Objekte dieser Klasse – also eine <i>Vorlage</i> .
Objekt / Instanz	Ein Objekt ist eine verwendbare <i>Referenz</i> auf eine Klasse – z.B.: ein roter Ferrari Eine Instanz ist die existierende <i>Referenz</i> auf ein Objekt – z.B. <u>der</u> Ferrari vor meinem Haus.

Eine **Instanz** ist also ein reelles – quasi anfassbares **Objekt**. So zum Beispiel der Laptop vor euch, welcher eine Instanz des generellen Objektes Laptop ist, welcher wiederum von der Klasse Computer abstammt.

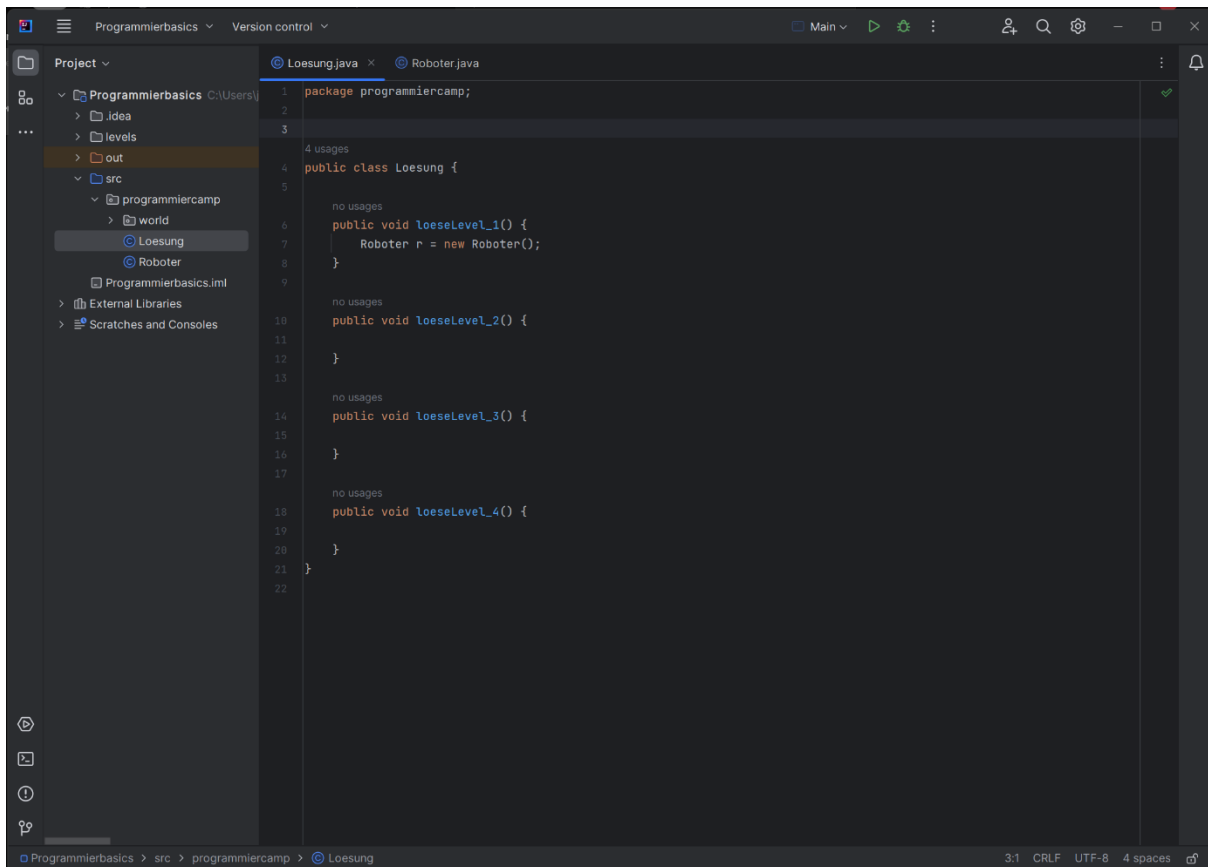
Euer Laptop ist also eine speziellere Version des generellen Objektes Laptop – und dieser ist eine speziellere Version eines Computers, nämlich eben tragbar und zu klappbar.

Wie genau das Ganze für uns relevant wird und wie wir damit umgehen, werden wir in den kommenden Kapiteln uns genauer anschauen.

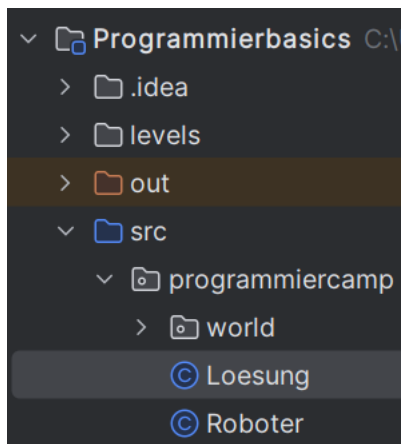


In Java brauchen wir – wie wir oben im *Hello World* Beispiel bereits gesehen haben also immer eigene **Klassen, Methoden, Objekte** und **Attribute**. In der Vorlage „*Programmierbasics*“ findet ihr bereits eine Vorlage, mit der Ihr die Grundlagen des Programmierens und Code Schreibens etwas üben könnt. Ich nehme euch dabei natürlich mit an die Hand!

Wenn ihr die Datei „**Programmierbasics**“ in IntelliJ öffnet, dann dürftet ihr in etwa dieses Fenster vorfinden:



Gehen wir im Folgenden mal auf die einzelnen Bestandteile des Fensters genauer ein. Zunächst haben wir da einmal die Ordnerstruktur, in der ihr alle Dateien und Verzeichnisse findet:



Hier seht ihr einmal den Hauptordner „Programmierbasics“

Dann den wichtigsten Ordner „src“ – in diesem Ordner findet ihr alle Dateien, die für das Programm wichtig sind.

Im Ordner „**Programmiercamp**“ findet ihr dann einen weiteren Ordner namens „**world**“ – dieser Ordner ist für uns irrelevant – hier steht nur der Code drin, damit das Programm überhaupt läuft, jedoch ist das noch zu viel für den Einstieg, als dass ihr euch damit auseinandersetzen müsstet.

Außerdem findet ihr zwei Dateien – keine Ordner – namens „**Loesung**“ und „**Roboter**“. Diese beiden Dateien schauen wir uns gleich gemeinsam einmal an, darin dürft und sollt ihr nämlich gleich arbeiten. Die beiden Dateien haben ein blaues C davor, in einem blauen Kreis. Das lässt auf den Typen der Datei schließen – hier haben wir nämlich **Klassen** Dateien. erinnert ihr euch, was wir in den Grundlagen gelernt haben? In Java brauchen wir immer **Klassen** und **Objekte**, um überhaupt programmieren zu können. Das hier ist also unsere erste Klasse, die wir uns mal genauer anschauen wollen!

Die Klasse „**Loesung**“, welche auch schon geöffnet sein sollte (falls nicht, einfach mal doppelt mit der linken Maustaste draufklicken), sieht nun wie folgt aus:

Hier sehen wir einmal das „package“ – also den Ordner, in dem die Klasse zuhause ist (hier: **programmiercamp**),

```
© Loesung.java x © Roboter.java
1 package programmiercamp;
2
3
4 4 usages
public class Loesung {
5
6     no usages
    public void loeseLevel_1() {
7         Roboter Erny = new Roboter();
8     }
9
10    no usages
    public void loeseLevel_2() {
11
12    }
13
14    no usages
    public void loeseLevel_3() {
15
16    }
17
18    no usages
    public void loeseLevel_4() {
19
20    }
21 }
22
```

und darunter finden wir in Zeile 4 genau das, was wir hier auch erwarten – nämlich eine neue Klasse. Sehen können wir das an dem Schlüsselwort „**class**“ (engl. Für Klasse). Davor steht nun noch ein englisches Wort: „**public**“. Dieses Schlüsselwort gibt an, dass die Klasse auch von anderen Klassen in dem Programm gesehen und verwendet werden darf – also ein „Visibility State“ (engl. Für Sicht Zustand). In Java, und auch anderen Programmiersprachen, gibt es typischerweise 2 wichtige visibility States: *Public* (öffentlich sichtbar), *Protected* (also geschützt Sichtbar). Wir brauchen fürs Erste nur *Public*, um den Code einfach zu halten.

Hinter dem Class statement finden wir nun noch den Namen der Klasse: „**Loesung**“, gefolgt von einer geschweiften Klammer {. Weiter Unten, in Zeile 21, wird die Klammer dann auch wieder geschlossen, }. Diese Klammern sind in der Informatik, vor allem in Java sehr wichtig. Denn nur was zwischen den Klammern steht, wird vom Computer auch als Code erkannt und ausgeführt. Ihr dürft auch festgestellt haben, dass die nächsten Zeilen komisch

verschoben sind. Das nennen wir „*Einrücken*“. IntelliJ wird für euch Code immer passend einrücken, jedoch nur, wenn ihr auch alle Klammern ordentlich setzt – Also passt immer gut darauf auf, dass ihr jede Klammer, die ihr öffnet, auch wieder schließt.


Nun finden wir noch 4 weitere Codeblöcke in der Datei, die wir uns anschauen wollen. Hier sprechen wir, wie ihr vielleicht schon erkennen konntet von **Funktionen**. Funktionen erkennen wir – wie bereits gelernt, an den Klammern hinter dem Namen der Funktion – also hier z.B.: **loeselevel_1()**.

Vor dem Funktionsnamen finden wir wieder das Schlüsselwort **public**, aber diesmal noch ein weiteres Wort – **void**. Dieses Wort ist der sogenannte Rückgabewert der Funktion, also der Datentyp, den die Funktion am ende in einem **return** statement übermittelt. In unserem Fall ist der Rückgabewert **void** – das bedeutet, dass wir keinen **return** Wert erwarten, und somit auch kein **reutrn** Statement überhaupt in der Funktion finden.

Auch hier finden wir wieder geschweifte Klammern, und auch hier sind diese für den Code wichtig. Diesmal sind die geschweiften Klammern aber in ihrer Funktion etwas anders als zuvor – sie definieren nun den Methodenrumpf – also den Bereich des Codes, in welchem die Funktionalität der Methode beschrieben ist im Code.

Also quasi, was die Funktion überhaupt machen soll...



Hinter der geschweiften Klammer können wir nun einmal die Taste „Enter“ drücken (das ist die große Taste mit einem Pfeil  drauf, rechts auf der Tastatur). Jetzt sollten wir eine neue Zeile innerhalb des Funktionsrumpfes haben, bereits richtig eingerückt und fertig, um der Funktion einen Sinn zu geben.

Kapitel 2.1 – Der Roboter Erny und seine Lieblingsbälle...

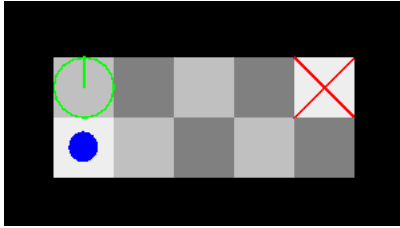
Im Folgenden schauen wir uns einmal eure Aufgabe in dieser Klasse genauer an. Wir haben jetzt bereits eine Menge über die Grundlagen des Programmierens gelernt, jetzt wollen wir eure neuen Fähigkeiten auch direkt mal anwenden und testen.

Starten wir mit der Mission: **Roboter Erny und seine Lieblingsbälle!**

Ja okay, der Name ist vielleicht nicht meine beste Erfindung... aber darum geht es ja auch gar nicht! In dieser Mission geht es um folgendes – Mein guter Freund Erny hier, ist ein Roboter. Das ist also sein Datentyp, also die Klasse. Erny ist eine Instanz, also ein Objekt der Klasse Roboter. Erny's Fähigkeiten, also die Methoden seiner Klasse Roboter, sind wie folgt:

- laufen()
- linksdrehen()
- (Ball) aufheben()
- (Ball) ablegen()

Mit diesen Methoden sollt ihr nun einen Code schreiben, welcher Erny dabei hilft seinen Ball zu finden, und nach Hause zu transportieren. Dabei muss Erny durch ein Level laufen und seinen Ball aufheben, um ihn dann zuhause wieder abzulegen. Das erste Level sieht wie folgt aus:



Erny startet immer oben links in der Ecke des Feldes, und schaut nach oben – also in Richtung der grünen Linie. Erny muss also seinen Lieblingsball unter ihm einsammeln, und auf das rote X bringen, sein Zuhause.

Um ein Level zu lösen, müssen wir also nun in die jeweilige Funktion zum Level – natürlich hinter die geschweifte Klammer auf – unseren Code schreiben. Das könnte dann zum Beispiel wie folgt aussehen:

```

4      4 usages
      public class Loesung {
5
6          no usages
          public void loeseLevel_1() {
7              Roboter Erny = new Roboter();
8              Erny.linksdrehen();
9              Erny.laufen();
10             Erny.aufheben();
11             Erny.laufen();
12             Erny.ablegen();
13         }
14     }
15

```

Gehen wir den Beispielcode mal gemeinsam durch.

Dies ist der Methodenrumpf der Funktion, also der Bereich, wo wir die Funktionalität reinschreiben.

Unser **Roboter** wird hier als Klasse genannt, seinen **Namen** Erny bekommt er **hier**, das ist jetzt sein Identifier. Hinter dem = Zeichen, wird unser Roboter dann **instanciiert** – das bedeutet, dass ein neues Objekt der Klasse Roboter – mit Namen Erny – erschaffen wird. Das Schlüsselwort hier ist „**new**“, dahinter finden wir wieder die Klasse – aber diesmal

sieht das doch mehr wie eine Funktion aus – mit Klammern und so? Ja, ganz genau. Das ist eine ganz spezielle Form einer Funktion – nämlich der **Konstruktor**. Watt’n Ding fragt ihr? Das ist die Funktion, die aufgerufen wird, sobald ein neues Objekt der Klasse erstellt wird. In dieser Funktion werden alle **Klassenattribute** – wie z.B. bei einem Auto die Anzahl der Reifen, die Farbe und die Geschwindigkeit usw. festgelegt. Diese Attribute bekommen also erst durch den Klassen Konstruktor ihre **Startwerte** zugewiesen.

Die Funktion müssen wir nicht genau verstehen – oder wissen, was die jetzt genau im Code macht. Wichtig ist nur, dass wir immer ein Objekt mit diesem Aufruf erzeugen – und das brauchen wir, um damit weiter programmieren zu können.

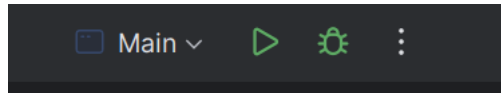
Darunter seht ihr jetzt in den Zeilen 8 – 12 Code stehen. Falls ihr euch erinnert, haben wir oben gesagt, dass Erny nur 4 Fähigkeiten zurzeit hat. Nämlich laufen, sich nach links drehen, einen Ball aufheben und diesen wieder ablegen. Diese Funktionen seht ihr hier einmal demonstriert. Um auf die Funktionen zuzugreifen, brauchen wir den Identifier – also den Namen, den wir eben beim Erzeugen des Roboters festgelegt haben. Hier also unseren guten Freund Erny, und dann folgt der sogenannte „**Punktoperator**“. Das ist einfach nur ein Ausdruck dafür, dass ihr mit *Identifier.MethodenNamen()*; auf die jeweiligen Funktionen und Methoden und Attribute zugreifen könnt. Der Identifier ist hier der Name – also Erny, dann der „.“ Und dann der Methodenname, also „linksdrehen“ gefolgt von den Charakteristischen Klammern einer Funktion „()“ und zu guter Letzt der Abschluss einer jeden Zeile Code in Java: das Semikolon „;“. Wunderbar! Wir haben soeben Erny dazu gebracht sich auf dem Feld zu bewegen! (Beim Linksdrehen dreht sich Erny immer nur um 90° nach Links...)

Den Code, den ihr oben im Bild seht, ist nicht die Lösung zum Level, versucht doch mal selbst den Code so zu schreiben mit den Methoden, dass Erny am Ende auf dem Roten „X“ Endet und seinen Ball ablegt!

Denkt daran, dass Erny am Anfang
nach oben schaut...!



Um das Level dann abspielen zu lassen, müsst ihr einfach oben rechts auf den grünen Pfeil drücken,



Und anschließend in der Konsole, die sich unten im Fenster öffnet hinter die „>>>“ dann die Zahl des jeweiligen Levels, welches ihr bearbeitet habt, eintragen und Enter drücken. Also erst mal für das erste Level eine 1.

```
Run Main x
" C:\Program Files\Microsoft\jdk-11.0.12.7-hotspot\bin\java.exe"
Bitte wähle ein Level. Gebe dafür die Zahl des Levels ein.
01: level_1
02: level_2
03: level_3
04: level_4
>>> 1
```

Danach müsste sich ein neues Fenster in der Mitte des Bildschirms öffnen, in welchem Erny das Level abläuft – und hoffentlich seinen Ball mit nach Hause kriegt. Am Ende, sobald der Code all eure Zeilen durchgegangen ist, beendet er das Programm und teilt euch mit, ob Erny es erfolgreich nach Hause geschafft hat oder nicht.

Falls ihr das Level nicht direkt beim ersten Versuch geschafft habt, verzweifelt nicht daran, sondern versucht einfach euren Code noch einmal anzupassen. Überlegt euch im Vorhinein schon einmal, welche Schritte Erny gehen muss, und wie er sich drehen muss, damit er am Ende auch richtig steht.

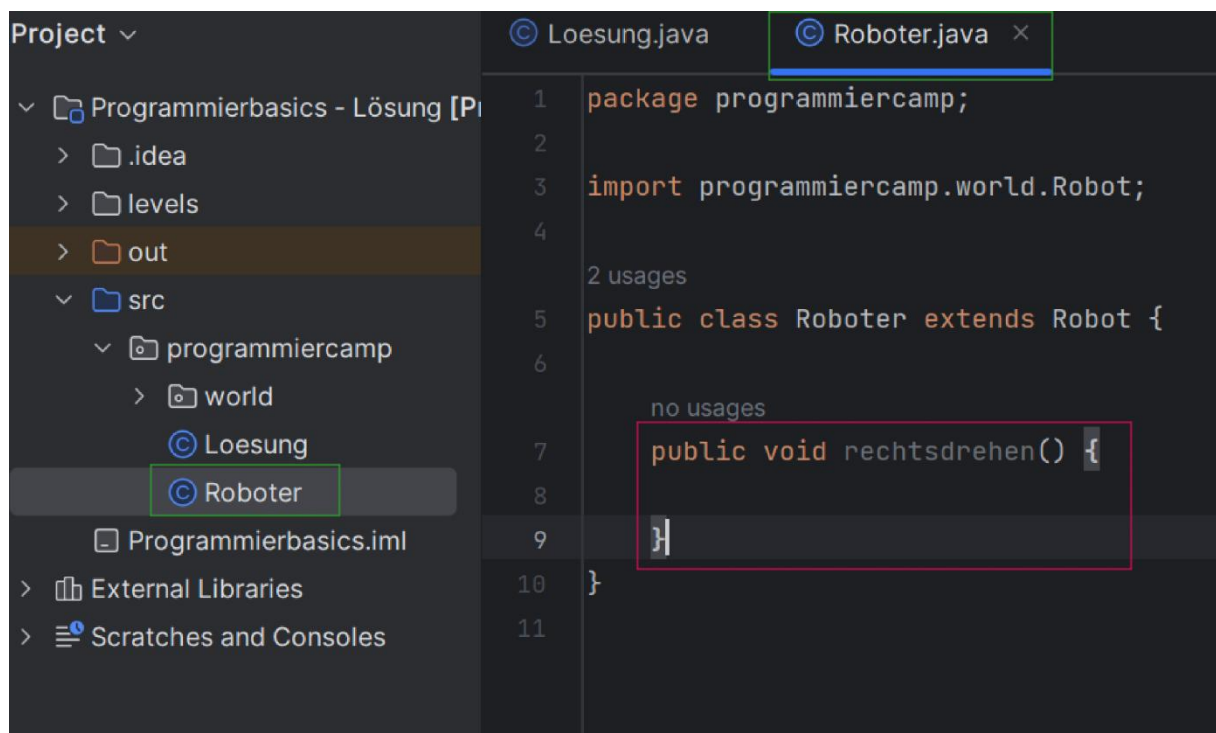


Jetzt ist Erny glücklich mit seinen
Bällen zuhause! Danke ^^

Kapitel 2.2 – Mehr Bälle für unsere Roboterfreunde – und eine eigene Funktion?

Das erste Level dürften wir jetzt alle geschafft haben – Super! Jetzt wollen wir uns etwas schwerere Level anschauen und auch direkt mal etwas Neues lernen. Schauen wir uns doch nochmal die Funktionalitäten unseres Freundes Erny an. Fällt euch etwas Komisches auf? Er dreht sich aktuell nur nach links... aber er könnte sich doch bestimmt auch nach rechts drehen? Dann müssten wir nicht immer 3-mal *linksdrehen()* schreiben, nur damit er sich einmal nach rechts dreht... oder?

Ganz genau. Aber die **Funktion** – wenn ihr einmal die Klasse „Roboter“ öffnet – ist leer... also passiert da gerade noch GOARNIX!



```
1 package programmiercamp;
2
3 import programmiercamp.world.Robot;
4
5 2 usages
6 public class Roboter extends Robot {
7
8     no usages
9     public void rechtsdrehen() {
10
11 }
```

Also überlegen wir uns mal, wie wir mit unseren bereits bekannten Methoden eine Rechtsdrehung schreiben könnten... Vermutlich ist das Laufen nicht so wichtig... und den Ball aufzuheben oder abzulegen auch nicht so wirklich. Aber die Linksdrehung könnte uns helfen – was ist denn überhaupt der Unterschied zwischen einer links- und einer Rechtsdrehung? Naja... weiter oben habe ich euch bereits einen Hinweis gegeben.

Um jetzt die Funktion mit Sinn und Code zu füllen, braucht ihr keinen Roboter – und keine Instanz mehr, immerhin verändern wir den Code von ALLEN Robotern – also von der Klasse Roboter selber. Das heißt, wir können auf die Funktionen einfach so zugreifen – ohne irgendwelche weiteren Schlüsselworte oder so. Also z.B.: einfach *laufen();* oder *aufheben();*

Versucht mal so jetzt die Funktionalität der Rechtsdrehung hinzukriegen, die Lösung findet ihr sonst zum Vergleich auf der nächsten Seite!

--- Lösung ---

Euer Code sollte jetzt genauso aussehen:

```
© Loesung.java  © Roboter.java x
1 package programmiercamp;
2
3 import programmiercamp.world.Robot;
4
5 2 usages
6 public class Roboter extends Robot {
7
8     no usages
9     public void rechtsdrehen() {
10         linksdrehen();
11         linksdrehen();
12         linksdrehen();
13     }
14 }
```

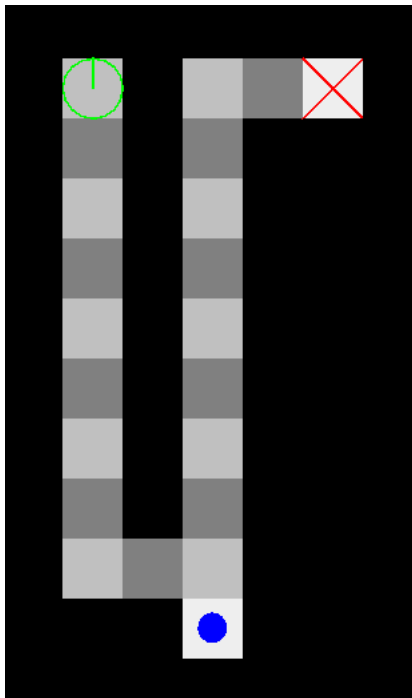
Ja – eine Rechtsdrehung ist vereinfacht nichts anderes, als sich dreimal um 90° nach links zu drehen.

Super, unsere Roboter können sich jetzt auch nach rechts drehen. Aber warum haben wir das überhaupt gemacht?

Solche Funktionen haben beim Programmieren einen gewaltigen Vorteil und Nutzen – nämlich die Vereinfachung und Reduktion vom Code. Bedeutet: statt mehrmals im Code 3-mal linksdrehen zu verwenden um sich nach rechts zu drehen, müssen wir jetzt vielmehr nur eine Zeile Code schreiben – *rechtsdrehen()*;

Das spart uns eine Menge Arbeit, wie ihr in den nächsten Leveln noch sehen werdet.

Wo wir schon beim Thema sind – hier kommt Level 2:



Das sieht doch schon nach einer Menge mehr Arbeit aus – also auch nach einer Menge mehr Code!

Versucht doch mal das Level zu lösen, wir sehen uns dann bei Level 3 wieder.

Ach, eine Sache noch, ihr dürft euren Roboter natürlich auch so nennen wie ihr wollt, ihr müsst nicht Erny helfen.

Dafür müsst ihr einfach den Identifier anpassen – also statt

Roboter Erny = new Roboter();

Schreiben:

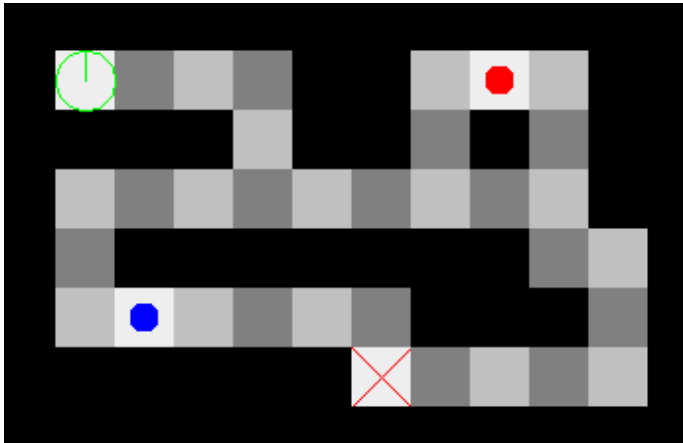
Roboter Ruffy = new Roboter();

Oder wie auch immer ihr euren Freund nennen wollt...

Kapitel 2.3 – Die Level 3 und 4, Die Abenteuer der Roboterfreunde

So, nachdem wir bereits 2 Level gemeinsam gelöst haben, und die generellen Verfahren euch jetzt bekannt sein sollten, wollen wir uns doch einmal die letzten zwei Level anschauen.

In Level 3, geht es einmal um folgendes:



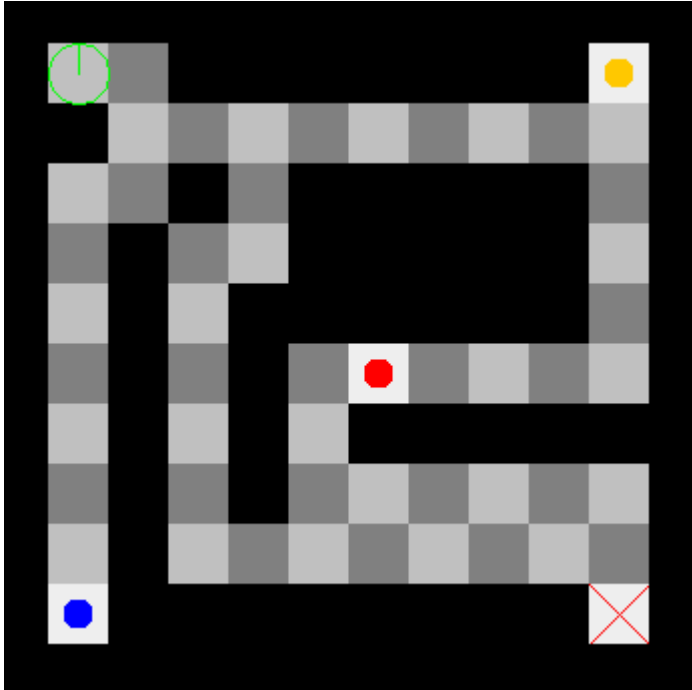
Es gibt nun zwei Bälle, die es einzusammeln gilt. Außerdem ist die Levelstruktur generell deutlich komplizierter geworden, sodass ihr eine Menge Code schreiben müsst. Versucht mal euren Code so kurz wie möglich zu schreiben – dazu einmal zwei kleine Hinweise oder Vorschläge:

1. Die Funktion `laufen()` besitzt zwei verschiedene aufrufe – das heißt es gibt zwei Möglichkeiten diese Funktion zu verwenden. Wir nennen so etwas „Überladen von Funktionen“. Heißt, wenn ich `laufen()` schreibe, dann läuft der Roboter ein Feld nach vorne, wenn ich aber `laufen(2)` schreibe, dann läuft er 2 Felder nach vorne. Wir können jetzt also innerhalb der Klammern sogenannte „Parameter“ angeben, also spezifische Angaben zur Funktionalität. In unserem Fall also können wir sowohl `laufen()` schreiben, als auch `laufen(int Anzahl)` – wobei „int Anzahl“ angibt, dass ihr eine ganze Zahl (z.B. 4 oder 5 usw.) angeben könnt, die der Roboter dann nach vorne läuft.
2. Unser Roboter kann entweder allein den Kurs ablaufen, ihr könnt aber auch z.B. zwei Roboter sich die Arbeit teilen lassen. Wenn ihr also die Code Zeile „Roboter *name* = new Roboter();“ zwei Mal schreibt, und dann beim zweiten Mal natürlich einen anderen Namen verwendet, könnt ihr einfach zwei Roboter verwenden. Ihr steuert dann eure Roboter wie davor auch, jeweils mit ihren Namen an, um ihnen eigene Aufträge zu geben. Leider laufen die Roboter nicht tatsächlich parallel, sondern bewegen sich immer nacheinander, aber das könnt ihr etwas umgehen, indem ihr einfach immer abwechselnd die beiden Roboter ansprecht. Also:

```
roboter1.laufen();  
roboter2.laufen();  
roboter1.rechtsdrehen();  
roboter2.linksdrehen();  
.  
.  
.  
Hurra!
```
3. Noch ein letzter Hinweis, eure Roboter können auch mehrere Bälle gleichzeitig tragen, also könnt ihr auch mit einem Roboter und wenig Code das Level lösen.
4. Noch eine Sache zum Abschluss – macht euch, bevor ihr den Code schreibt, z.B. auf einem Blatt Papier Gedanken, wie genau das Level zu lösen ist, also wie sich welcher Roboter wann drehen muss. Das machen wir Programmierer generell gerne und häufig – denn Vorbereitung ist der beste Weg, um guten und sauberen Code zu schreiben.

Im 4. Level geht's nun ums Anwenden aller bisher gelernter Grundlagen. Macht euch also zuvor gut Gedanken, am besten auf Papier, wie sich eure Roboter verhalten sollen, und schreibt dann möglichst effizienten Code, um das Level zu lösen.

Level 4 sieht nun wie folgt aus:



Versucht das Level einmal mit nur **einem** Roboter zu lösen, und einmal mithilfe **seiner Freunde**. Sagt dann, wenn ihr es geschafft habt, eurem Teamer Bescheid, dass ihr gemeinsam einmal die wichtigen Funktionen durchgehen könnt und ihr schauen könnt, ob ihr alles gut verstanden habt.

Nun, was solltest du nach dieser Einführung alles gelernt haben?

Erst mal natürlich den generellen Umgang mit **IntelliJ** und **Java** Code, darüber hinaus den Unterschied zwischen **Attributen** und **Funktionen**, **Funktionen** und **Konstruktoren**, wie man ein neues **Objekt** *instanciiert*, also eine **Instanz** erzeugt, und wie ihr auf die **Funktionen** und **Attribute** einer solchen **Instanz** dann *zugreifen* könnt. Außerdem habt ihr hoffentlich gelernt, wie man mehrere **Objekte** einer **Klasse** nebeneinander *erzeugen* und *Verwalten* kann.



Damit haben wir die **Grundlagen** bereits durch –
Glückwunsch! Ihr seid jetzt bereits wahre **Profis**
was **Roboter** und **Code** angeht.

Im nächsten **Kapitel** – in der nächsten **Doku**,
beschäftigen wir uns dann mit **Minecraft**!

Bis Morgen!

EXTRA – LEVELDESIGN

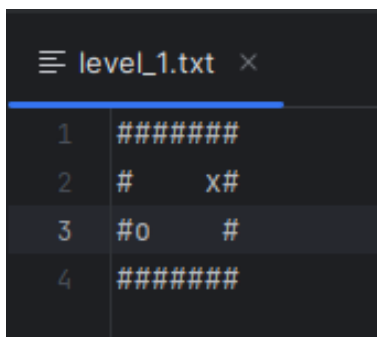
Für die Schnellen unter euch – die jetzt eventuell schon fertig sind und nichts mehr zu tun haben, ihr könnt jetzt einmal selbst ein Level für eure Roboterfreunde erstellen.

Dafür brauchen wir nur folgendes:

1. Eine neue Textdatei für das Level
2. Die Textdatei in dem Level Auswahlmenu
3. Den Code zum Löse

Also gehen wir die Punkte doch einmal durch!

Eine Leveldatei sieht wie folgt aus:



Wir haben hier das 1. Level, als Textdatei. Die Datei ist in einem bestimmten Code geschrieben, den ihr natürlich auch verstehen müsst.

Wir haben die # (Rauten, oder Hashtags), welche die **Wände** des Levels darstellen, das X, welches das **Zielfeld** darstellt, und die O's, welche die **Bälle** darstellen (**wichtig**, kleingeschriebenes O).

Um jetzt also ein eigenes Level zu entwickeln, müsst ihr eine solche Datei selbst schreiben. Die Leveldateien findet ihr in dem Ordner *levels*.

Auf den Ordner machen wir nun einmal Rechtsklick, und wählen in dem Menu dann *new -> File* aus, um eine neue Textdatei zu erzeugen. Diese nennen wir dann *level_5.txt*, und können dann in der Datei unser Level designen. Überlegt euch also zuvor einmal auf einem Papier, wie euer Level aussehen soll, und versucht es dann mit den Symbolen von oben, in die Datei zu übertragen. Leere Stellen, also Gänge für den Roboter macht ihr einfach mit einer Leertaste.

Wenn ihr fertig seid, könnt ihr eure Datei einmal Speichern, mit *ctrl + S* auf der Tastatur, und dann gehen wir einmal in die Datei *src -> programmiercamp -> world -> Main* und schauen uns die Zeile 21 an:



Hier sehen wir die Zeile: *private final static String[] levels = {„level_1“, „level_2“, „level_3“, „level_4“};*

In diese Liste an Levels innerhalb der geschweiften Klammern, fügen wir jetzt hinter das Level_4 noch unser Level 5 hinzu. Also schreiben wir hinter „level_4“ noch ein Komma, gefolgt von „level_5“. Nun sollte unser Level auch in der Auswahl gezeigt werden, wenn wir das Programm starten.

Als letztes brauchen wir jetzt noch Code – denn bis jetzt solltet ihr einen Fehler bekommen haben, wenn ihr das Programm gestartet habt. Wir müssen also in der Datei *Loesung* eine neue Funktion erstellen, für das Level 5. Das machen wir so:

```
24      no usages
25      public void loeseLevel_4() {
26          Roboter Erny = new Roboter();
27      }
28
29      no usages
30      public void loeseLevel_5() {
31          Roboter Erny = new Roboter();
32      }
33  }
```

Wir müssen also hinter die letzte Klammer von Zeile 26 eine neue Zeile machen, und dann schreiben:

```
public void loeseLevel_5() {};
```

innerhalb der geschweiften Klammern, die automatisch vervollständigt werden, machen wir eine neue Zeile, ihr könnt also nach dem schreiben der Klammer auf (also nach {) einfach Enter drücken, und die zweite wird automatisch platziert, und da dann eure Lösung, wie auch zuvor für den Code schreiben. Wenn ihr euch euer Level nur einmal anschauen wollt, dann schreibt einfach nur die Zeile Code:

```
Roboter Erny = new Roboter();
```

Und startet das Programm, wählt euer Level aus, und verschiebt das Fenster, dass ihr das Level nicht gelöst habt, einfach. Dann könnt ihr euch euer selbst erstelltes Level einmal anschauen und euch überlegen, wie ihr das am einfachsten lösen könnt!

Achtet darauf, dass die Methode richtig geschrieben ist, also kleines l vorne, und großes L in der Mitte (loeseLevel_5).

Versucht doch jetzt dann einmal euer eigenes Level zu lösen!