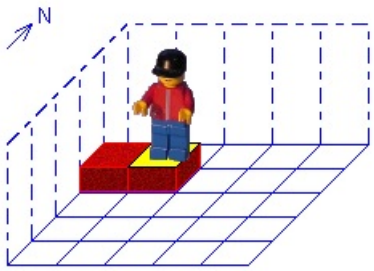
8 Vertiefung der Algorithmik

8.1 Java Karol

In den vorherigen Abschnitten haben Sie schon einzelne Kontrollstrukturen kennengelernt. Diese sollen nun mithilfe von Java Karol vertieft werden. Hierfür finden Sie im Klassenlaufwerk eine Eclipse-Projekt-Vorlage.

8.1.1 Einstieg

1. Erstellen Sie in der Klasse `Main` ein neues Objekt der Klasse `WeltMitRoboter`, so dass Länge, Breite und Höhe der Welt 5 beträgt. Rufen Sie anschließend die Methoden der Klasse `WeltMitRoboter` so auf, dass nach Programmstart folgendes Bild entsteht:



2. Implementieren Sie in der Klasse `WeltMitRoboter` eine Methode `void roboterSchrittVorWandGehen()`, dass wenn der Roboter vor einer Wand steht, er sich entsprechend dreht und anschließend einen Schritt vorwärts geht.
   Hinweis: Methode `boolean IstWand() in der Klasse Roboter`

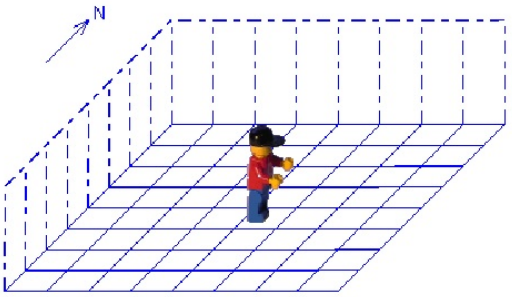
8.1.2 Wiederholung mit Bedingung

1. Es soll nun in der Klasse `WeltMitRoboter` eine Methode `void roboterBisWandgehen()` implementiert werden, mit welcher der Roboter mit der aktuellen Blickrichtung bis zur Wand läuft (unabhängig von der aktuellen Weltgröße).
   (a) Hierfür benötigen Sie eine neue Kontrollstruktur, die so genannte **while-Schleife**. Informieren Sie sich über diese Kontrollstruktur und gehen Sie dabei auf den Unterschied zur **do-while-Schleife** ein. Wie können Bedingungen in Java logisch verknüpft oder negiert werden?

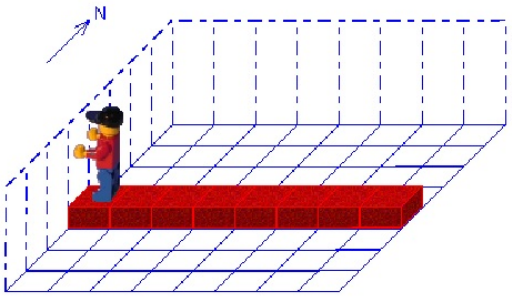
   (b) Implementieren Sie nun mithilfe einer while-Schleife in der Klasse `WeltMitRoboter` die Methode `void roboterBisWandgehen()`.

2. Implementieren Sie nun eine Methode `void roboterInStartPosition()`, welche den Roboter von seiner aktuellen Position in der Welt zurück in die linke obere Ecke gehen lässt.
   Hinweis: Methode `IstBlickNorden` in der Klasse `Roboter`.

3. Implementieren Sie eine Methode `void roboterReiheZieglsteineLegen()`, so dass der Roboter in seiner Reihe mit der aktuellen Blickrichtung eine Reihe Ziegelsteine hinlegt.

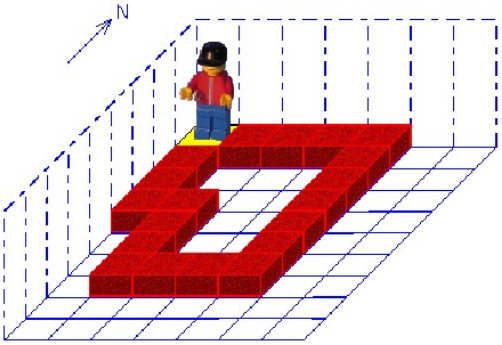


Vor Ausführung

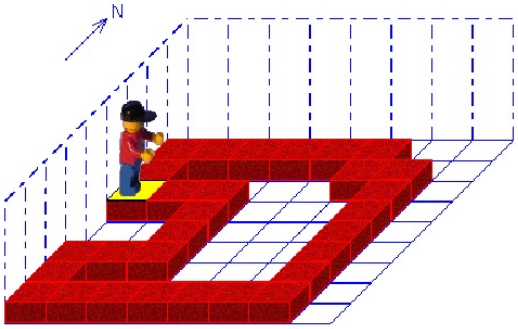


Nach Ausführung

4. (a) Implementieren Sie eine Methode `void roboterDurchSumpfLaufen()`, welche den Roboter von seiner Startposition (markiert durch eine Markierung) auf einem Weg durch Ziegelsteine durch den Sumpf laufen lässt und wieder zum Startpunkt zurückkehrt. Sie können davon ausgehen, dass an jeder Ecke des Weges es nur eine Fortsetzung gibt. Die Pfade könnten dabei wie folgt aussehen:



Sumpf 1



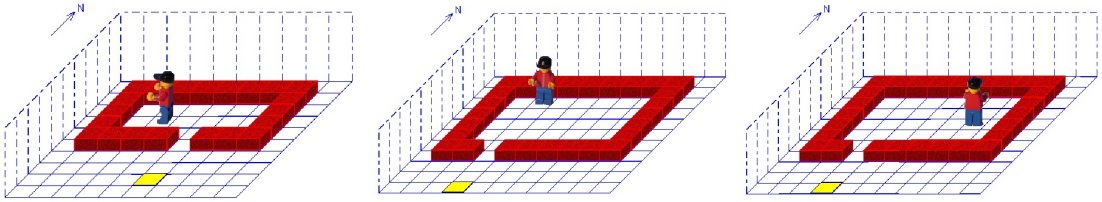
Sumpf 2

(b) Im Ordner `files` gibt es die Dateien `sumpf1` und `sumpf2`. Ihre Methode können Sie damit wie folgt in einer `main`-Methode testen:

```
//Laden der Dateien mit einer relativen Pfadangabe
WeltMitRoboter welt = new WeltMitRoboter(".\\files\\sumpf1.kdw");
welt.roboterDurchSumpfLaufen();
welt = new WeltMitRoboter(".\\files\\sumpf2.kdw");
welt.roboterDurchSumpfLaufen();
```

Für Experten:

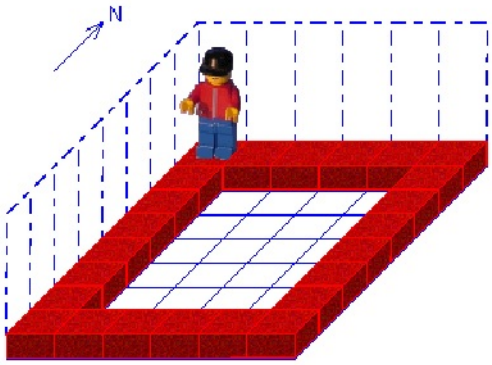
5. Implementieren Sie eine Methode `void roboterZeitungHolen()`, so dass der Roboter von einer beliebigen Position aus seinem Haus (Ausgang ist im Süden) und zur Zeitung geht (repräsentiert durch die Markierung). Die Ausgangssituation könnte dabei wie folgt aussehen:



Im Ordner `files` finden Sie die Dateien `zeitung1`, `zeitung2` und `zeitung3`. Sie können ihre Methode analog zur vorheriger Aufgabe testen.

8.1.3 Wiederholung mit fester Anzahl

1. Nun soll eine Methode `void roboterRandPflastern()` implementiert werden, die ausgehend von einer leeren Welt folgendes Muster in einer beliebig großen Welt erzeugt:



Hierfür muss unter anderem folgender Code vier Mal wiederholt werden:

```
while(!roboter.IstWand()) {
  roboter.Hinlegen();
  roboter.Schritt();
}
```

(a) Die Wiederholung mit fester Anzahl kann ebenfalls mit einer while-Schleife realisiert werden, jedoch verwendet man für solche Wiederholungen in Java häufig eine so genannte **for-Schleife**. Informieren Sie sich über den Aufbau einer for-Schleife in Java. Wieso gibt es in der Programmierung for-Schleifen, wenn diese auch durch entsprechende while-Schleifen realisiert werden können?

(b) Zum Verständnis: Was wird bei folgenden Codeausschnitten auf der Konsole ausgegeben?

```
for(int i = 0; i<4;i++){
  System.out.println(i);
}
```

```
for(int i = 0; i<4;i=i+2){
  System.out.println(i);
}
```

```
for(int i = 5; i>4;i--){
  System.out.println(i);
}
```

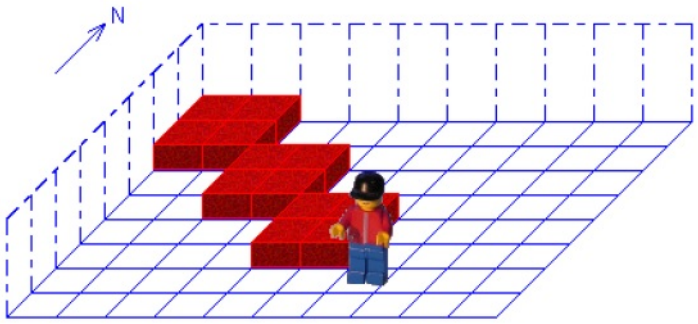
```
for(int i = 5; i<4;i=i+2){
  System.out.println(i);
}
```

```
for(int i = 0; i<2;i++){
  for(int j = 0;j<=2;j++){
     System.out.println
      (i + ", " + j);
  }
}
```

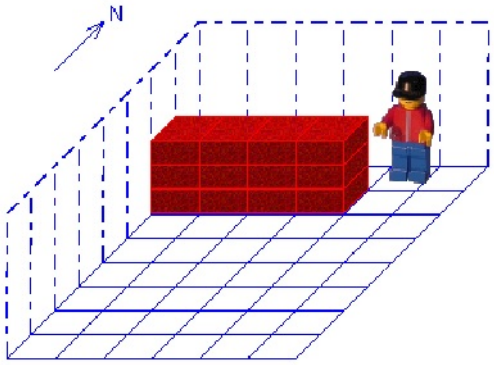
(c) Implementieren Sie nun die Methode `void roboterRandPflastern()`, so dass der Roboter das obige Muster legt.

2. Implementieren Sie eine Methode `void roboterZiegelReiheLegen(int anzahl)`, so dass der Roboter ausgehend von seiner aktuellen Position in Abhängigkeit von der übergebenen Anzahl abwechselnd einen Ziegelstein hinlegt und dann einen Schritt vorwärts geht. Achten Sie darauf, dass sobald er gegen eine Wand läuft, sich entsprechend dreht und damit weiterlaufen und Ziegelsteine hinlegen kann.

3. Implementieren Sie mithilfe einer for-Schleife eine Methode `void roboterDreiQuadrateLegen()`, die den Roboter ausgehend von der linken oberen Ecke folgendes Muster legen lässt:



4. Erstellen Sie eine Methode `roboterMauerBauen(int breite, int hoehe)`, so dass der Roboter ausgehend von der linken oberen Ecke eine Mauer mit der übergebenen Breite und Höhe baut.

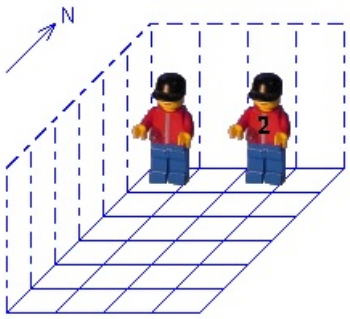


Ergebnis nach `roboterMauerBauen(4,3)`

8.1.4 Fortgeschrittene Aufgaben

1. Implementieren Sie eine Methode `void roboterWettrennen(int anzahlRunden)`, so dass sich zwei Roboter wie folgt ein Wettrennen liefern:

   - Ausgangssituation:

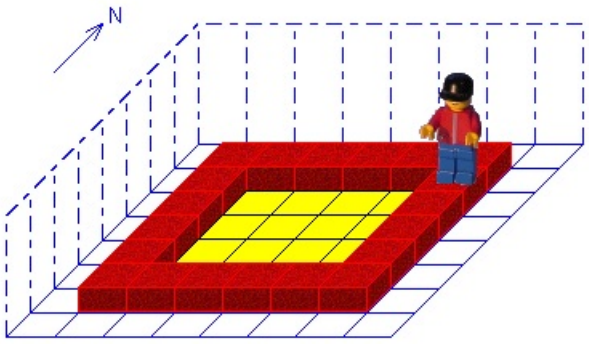
     

     Sie können den zweiten Roboter in dieser Methode beispielsweise wie folgt erzeugen:

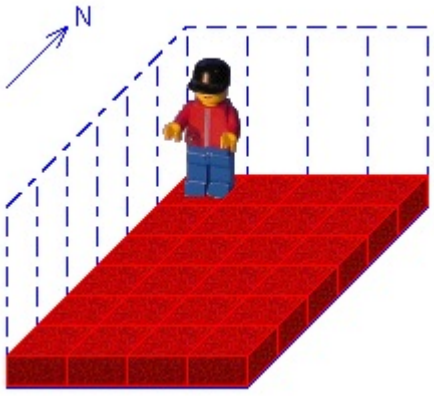
     ```
     Roboter roboter2 = new Roboter(3, 1, 'S', welt);
     ```

   - Anschließend soll abwechselnd gewürfelt werden, wie viele Schritte ein Roboter machen darf.
   - Sobald ein Roboter die Wand erreicht hat, hat dieser die Runde gewonnen. Wenn beide Roboter noch nicht gleich oft gewürfelt haben, hat der andere noch die Möglichkeit auf ein Unentschieden.
   - Nach einer Runde soll der aktuelle Spielstand mithilfe der Methode `MeldungAusgeben(String was)` in der Klasse `Roboter` ausgegeben werden und wieder die Ausgangssituation hergestellt werden.
   - Nachdem alle Runden vorbei sind, soll der Gewinner ebenfalls mit der Methode `MeldungAusgeben` ausgegeben werden.

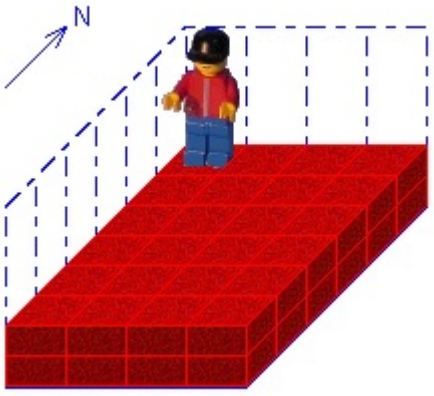
2. Erstellen Sie eine möglichst effektive Methode `roboterSandkastenBauen()`, so dass der Roboter in einer leeren beliebig großen Welt folgendes Muster legt:



3. Implementieren Sie eine Methode `void roboterEbenePflastern()`, so dass der Roboter ausgehend von der linken oberen Ecke einer beliebig großen Welt eine Ebene Ziegelsteine hinlegt. Diese Methode soll auch mehrmals hintereinander aufgerufen werden können, so dass eine weitere Ebene Ziegelsteine auf die bisherige Ebene gelegt wird.

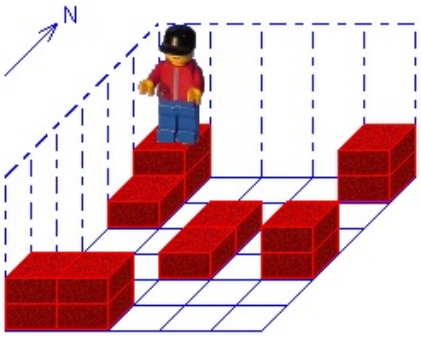


Nach einem einmaligen Aufruf von `roboterEbenePflastern()`



Nach einem doppelten Aufruf von `roboterEbenePflastern()`

4. Implementieren Sie eine Methode `void roboterWeltAufräumen()`, mit der der Roboter alle Ziegelsteine in der Welt aufhebt und anschließend mithilfe der Methode `MeldungAusgeben` die Anzahl der aufgesammelten Ziegelsteine ausgibt. Sie können Ihre Methode mit der Datei `aufraeumen.kdw` im Ordner `files` testen.



Welt aus der Datei `aufraeumen`