

Karels Freund kommt demnächst zu Besuch und er möchte seinem Freund eine aufgeräumte Welt präsentieren. Wenn nur nicht überall diese Beeper rumliegen würden! Diese Unordnung! Deshalb formuliert er für sich folgende Anweisung.

Ich laufe und **wenn** ich auf einem Beeper stehe, **dann** *hebe ich ihn auf*.

Aufgabe 1: So wie Karel seine *Handlung* von einer Bedingung abhängig macht, so ergeht es uns Menschen auch täglich.

- Formulieren Sie eine Bedingung und was Sie tun würden, wenn diese erfüllt ist.
- Wiederholen Sie a) noch zwei Mal für andere Beispiele.

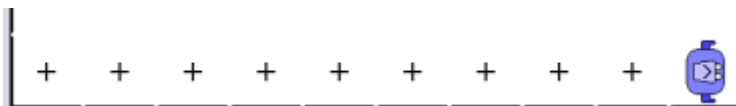
Beispiel: Wenn ich Hunger habe, dann mache ich mir ein Sandwich.

Zurück zu Karel. Wir wollen ihm helfen diese Unordnung in Ordnung zu bringen. So sieht es im Moment bei ihm aus.



```
void hebeBeeperAuf()
{
    repeat(9)
    {
        moveForward();
        if (onBeeper())
        {
            pickBeeper();
        }
    }
}
```

Und so sollte es danach aussehen.



Aufgabe 2: Im benutzten Quellcode tauchen neue Methoden auf.

- Unterstreichen Sie die Methode, die prüft, ob ein Beeper an der Stelle liegt, auf der Karel gerade steht.
- Markern Sie die Zeilen, mit denen eine **einseitige Bedingung (wenn ...dann)** implementiert wird. Zukunft sagen wird dazu kurz: **if-Anweisung**.

```
void hebeBeeperAuf()
{
    repeat(9)
    {
        moveForward();
        if (onBeeper())
        {
            pickBeeper();
        }
    }
}
```

Aufgabe 3:

Leider kommt Karel sein kleiner Bruder Karli zuvor. Er will Karel ärgern und legt überall dort, wo noch kein Beeper liegt, einen hin.



Wie macht er das? Hinweis: Karli kennt die Methode **!onBeeper()**, damit kann er prüfen, ob er nicht auf einem Beeper steht. Das ! negiert die Methode onBeeper().

Passen Sie den Quellcode aus Aufgabe 2 nun für Karli an.

```
void legeBeeperAb()
{
    repeat(9)
    {
        moveForward();
    }
}
```

Eine weitere Methode der Klasse Roboter heißt **frontIsClear()**. Sie prüft, ob ein Roboter **vor keiner Mauer steht**. Die Methode **!frontIsClear()** prüft demzufolge, ob sich vor einem Roboter **eine Mauer** befindet.

Aufgabe 4: Ergänzen Sie **die Klassendokumentation** für die Klasse Roboter. Überlegen Sie sich sinnvolle Beschreibungen auch für die Methoden, die bisher noch nicht besprochen worden sind.

Klassendokumentation Roboter

onBeeper()

Karel prüft, ob sich ein Beeper auf dem Platz befindet, auf dem er derzeit steht.

frontIsClear()

Karel prüft, ob

beeperAhead()

Karel prüft, ob

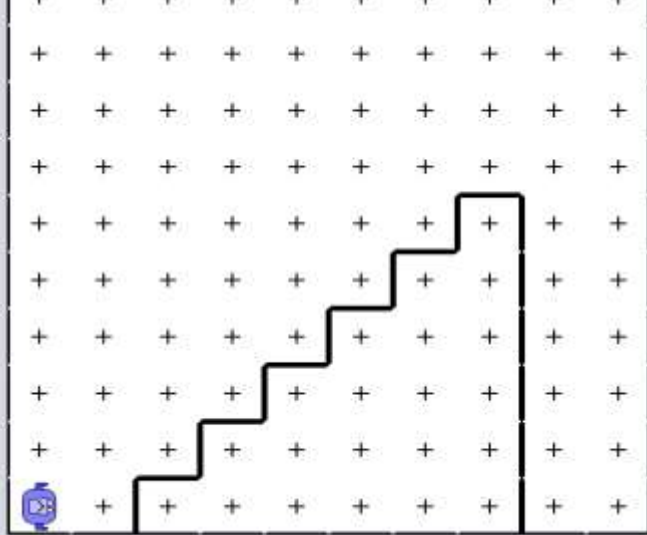
leftIsClear()

Karel prüft, ob

rightIsClear()

Karel prüft, ob

Zur Erinnerung: In einer **Methode** wird immer das gewünschte Verhalten des Objektes programmiert.



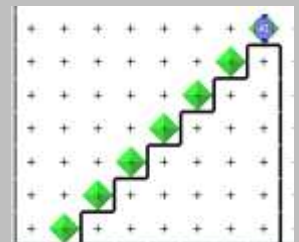
```

15 void climbTheStairs()
16 {
17     repeat(9)
18     {
19         moveForward();
20         if (!frontIsClear())
21         {
22             turnLeft();
23             moveForward();
24             turnRight();
25         }
26     }
27 }
28
29
    
```

Aufgabe 5: Karel's Treppensteig-Problem ist nicht neu für uns. Nur wurde es diesmal mit einer **if-Anweisung** etwas gepimpt.

a) Leider geht Karel zu weit. Wo landet er, wenn das Programm fertig ist? Kennzeichnen Sie es im obigen Bild ein und geben Sie auch die Blickrichtung von Karel an.

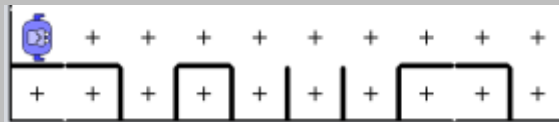
b) Verändern Sie das Programm so, dass Karel auf der obersten Treppenstufe mit Blick nach Osten stehen bleibt und zudem auf jeder Treppenstufe einen Beeper abgelegt hat (siehe Bild rechts).



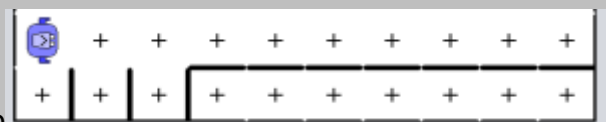
Aufgabe 6:

Karel wird noch verrückt. Überall sind Baustellen, aber die Straße zum Bahnhof ist voller Löcher und keiner kümmert sich drum! Da muss er selber ran. Er wird in jedes Loch einfach einen Beeper ablegen.

Ob die Straßen nun so

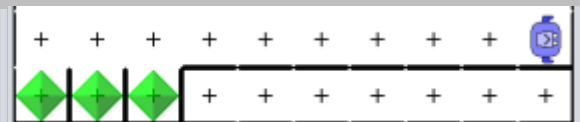
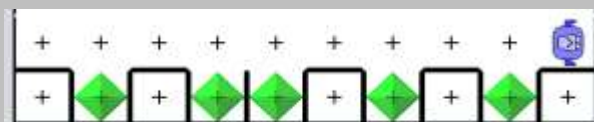


oder so



aussehen. Schreiben Sie ein Programm für Karel, welches Probleme dieser Art löst.

(Allgemeingültigkeit eines Algorithmus)



`void repairTheStreet()`