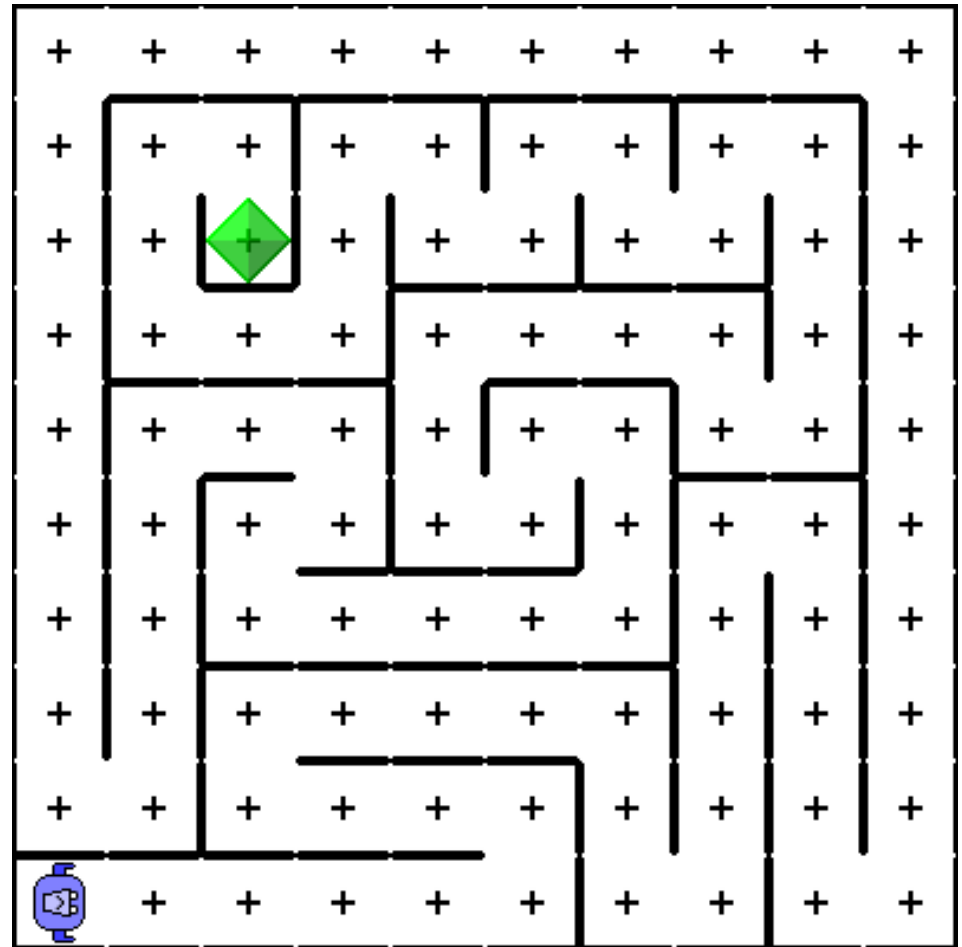
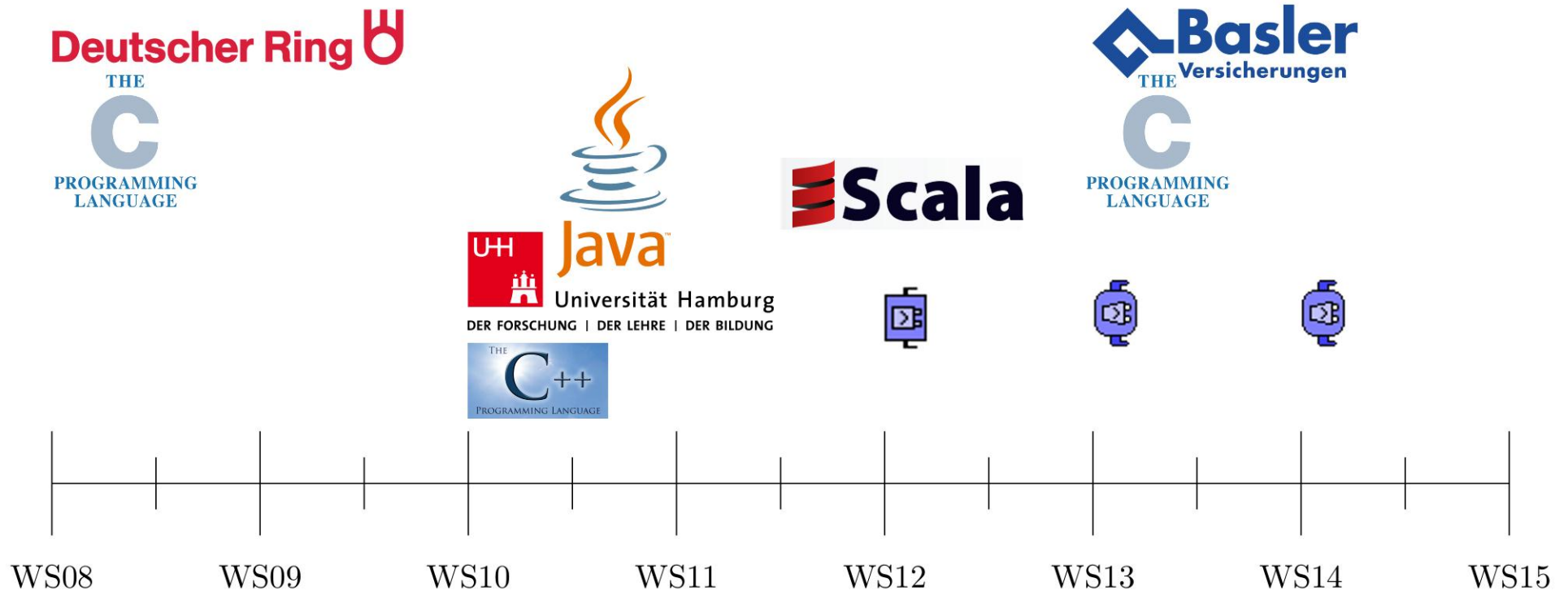


Karel The Robot – Lektion I

- Zusammengesetzte Befehle
- Zählschleifen
- Fallunterscheidung



Womit hat Fred sich in den letzten 7 Jahren beschäftigt?



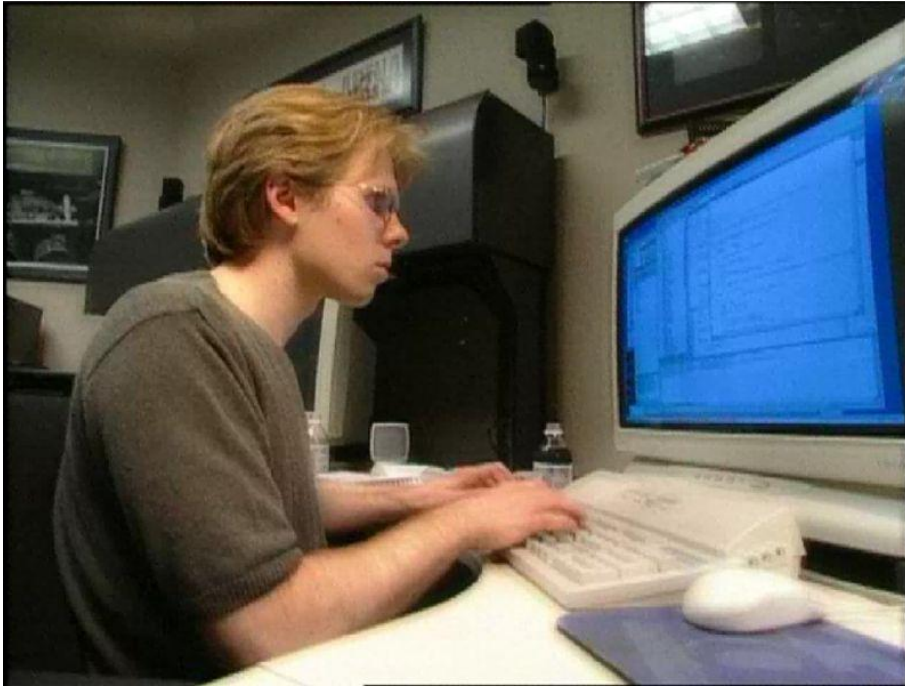
Wer seid ihr?

- In den Übungsgruppen werden wir uns namentlich kennenlernen.
- Wer hat noch nie programmiert?
 - Keine Angst, Ihr seid unser primäres Zielpublikum!
- Wer hat schon erste Erfahrungen im Programmieren gesammelt?
 - Für Euch wird der Einstieg etwas leichter sein.
- Wer hat schon mehr als 1000 Zeilen Code geschrieben?
 - Eure Kommilitonen können von Eurem Wissen profitieren.

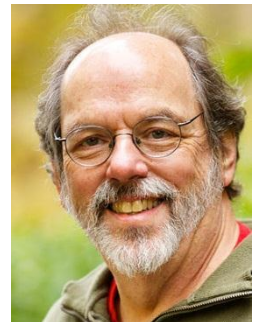
Der betreute Laborbetrieb an der Universität Hamburg



Programmieren, was ist das eigentlich?



- *“If you don't think carefully, you might believe that programming is just typing statements in a programming language.”* Ward Cunningham



Eine einfache Programmiersprache: Karel the Robot

- “In the 1970s, a Stanford graduate student named Rich Pattis decided that it would be easier to teach **the fundamentals of programming** if students could somehow learn the basic ideas in a simple environment free from the complexities that characterize most programming languages.”



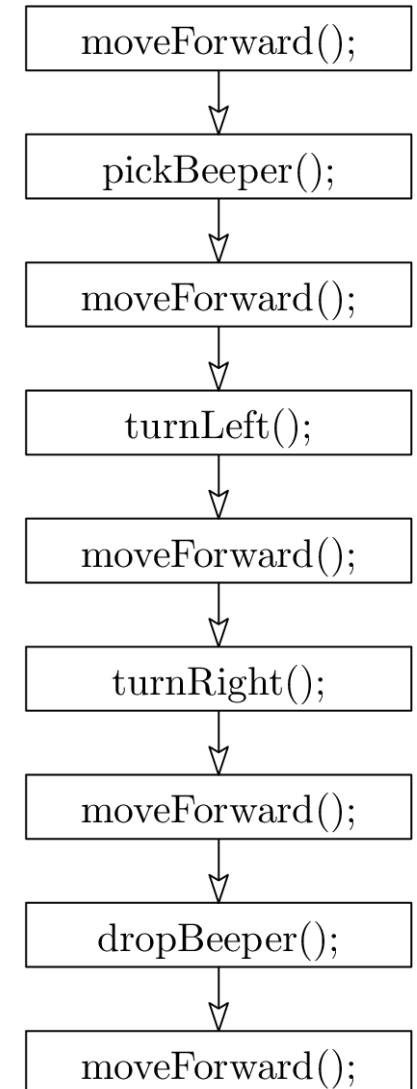
- “In sophisticated languages like Java, there are so many details that learning these details often becomes the focus of the course. When that happens, the much more critical issues of **problem solving** tend to get lost in the shuffle. By starting with Karel, you can concentrate on solving problems from the very beginning. And because Karel encourages imagination and creativity, you can have quite a lot of fun along the way.”

Kontrollstrukturen

- In welcher Reihenfolge werden Karels elementare Befehle abgearbeitet?
- Die Abarbeitungsreihenfolge wird durch 5 **Kontrollstrukturen** beeinflusst:
 1. Sequenz
 2. Aufruf zusammengesetzter Befehle
 3. Zählschleife
 4. Fallunterscheidung
 5. Wird noch nicht verraten, kommt nächste Woche ;)

Die einfachste Kontrollstruktur: Sequenz

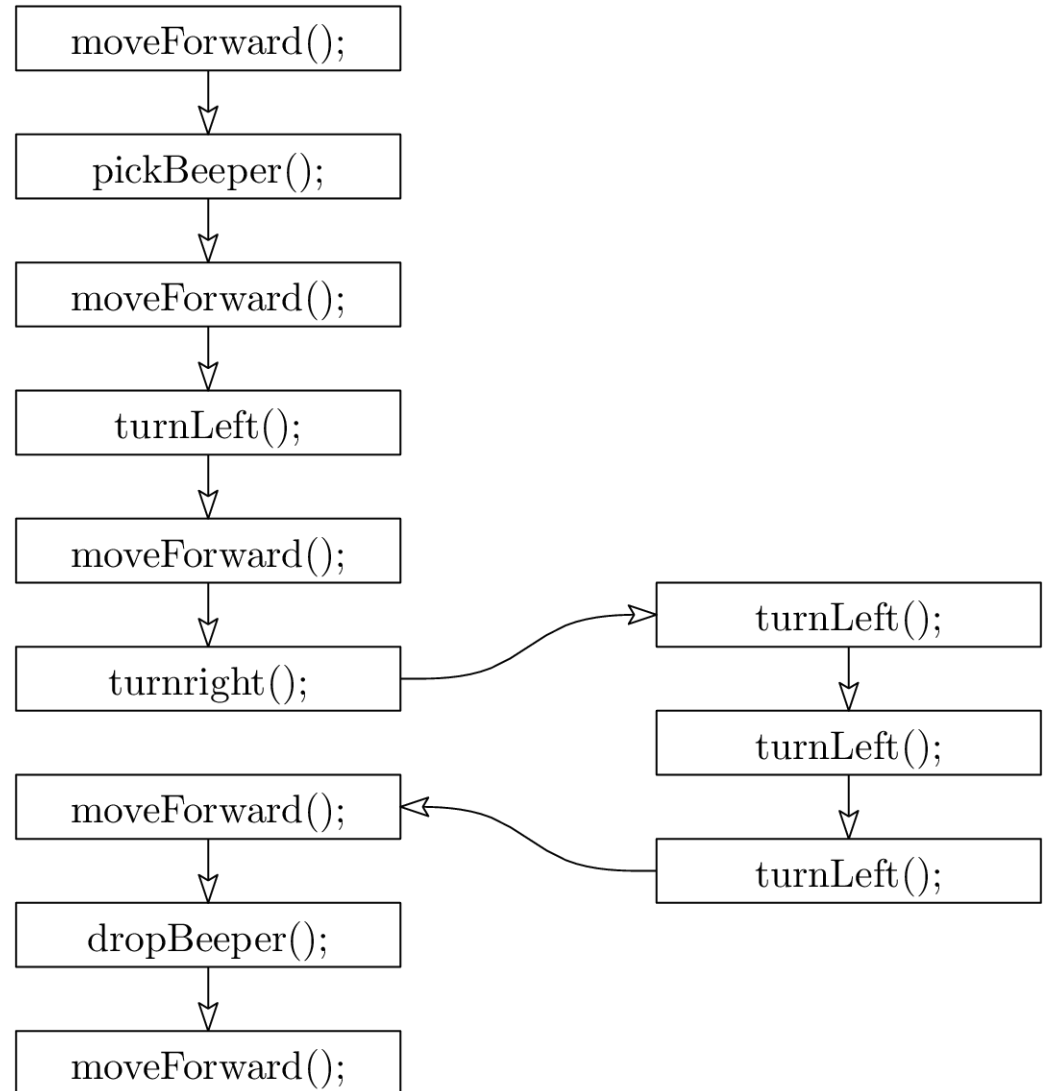
```
void karelsFirstProgram()  
{  
    moveForward();  
    pickBeeper();  
    moveForward();  
    turnLeft();  
    moveForward();  
    turnRight();  
    moveForward();  
    dropBeeper();  
    moveForward();  
}
```



Definition und Aufruf zusammengesetzter Befehle

```
void karelsFirstProgram()
{
    moveForward();
    pickBeeper();
    moveForward();
    turnLeft();
    moveForward();
    turnright();
    moveForward();
    dropBeeper();
    moveForward();
}
```

```
void turnright()
{
    turnLeft();
    turnLeft();
    turnLeft();
}
```

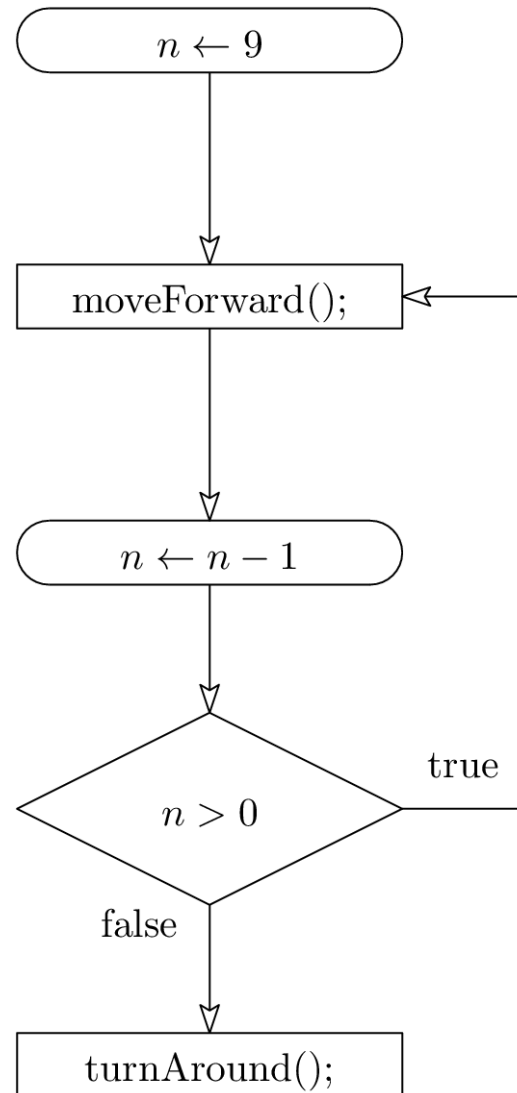


Begriffsschärfe am Beispiel Zusammengesetzter Befehl

- Den Begriff „Zusammengesetzter Befehl“ gibt es so nur in Karel.
- Teilnehmer mit Programmiererfahrung werden wahrscheinlich andere Begriffe verwenden, was ich am Anfang überhaupt nicht schlimm finde.
- Folgende Synonyme sind ebenfalls akzeptabel:
 - Benutzerdefinierter Befehl
 - Eigener Befehl
 - Neuer Befehl
 - Rezept
 - Methode
 - Prozedur
 - Funktion
- Ab Woche 3 werden wir Begriffe schärfer fassen und strenger abfragen!

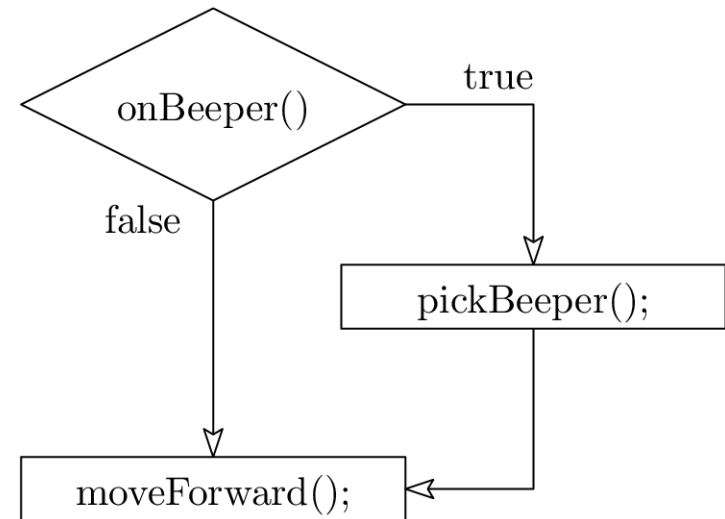
Die Zählschleife

```
repeat (9)
{
    moveForward();
}
turnAround();
```



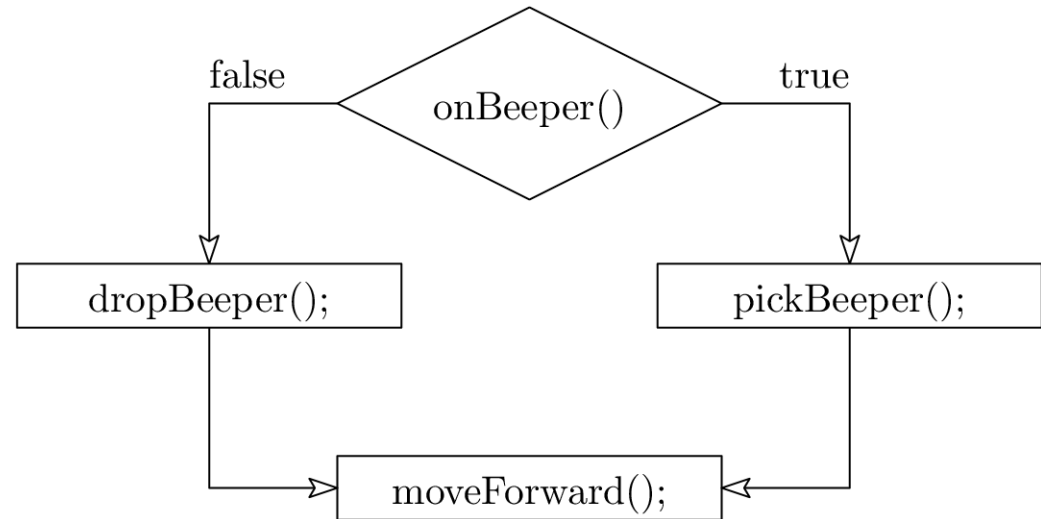
Die einfache Fallunterscheidung

```
if (onBeeper())  
{  
    pickBeeper();  
}  
moveForward();
```



Die Fallunterscheidung mit Alternative

```
if (onBeeper())  
{  
    pickBeeper();  
}  
else  
{  
    dropBeeper();  
}  
moveForward();
```



Bedingungen negieren

```
if (frontIsClear())  
{  
}  
else  
{  
    turnLeft();  
}
```

Ein if/else mit einem leeren then-Block kann durch ein if ohne else ersetzt werden, indem man die Bedingung **negiert**:

```
if (!frontIsClear())  
{  
    turnLeft();  
}
```

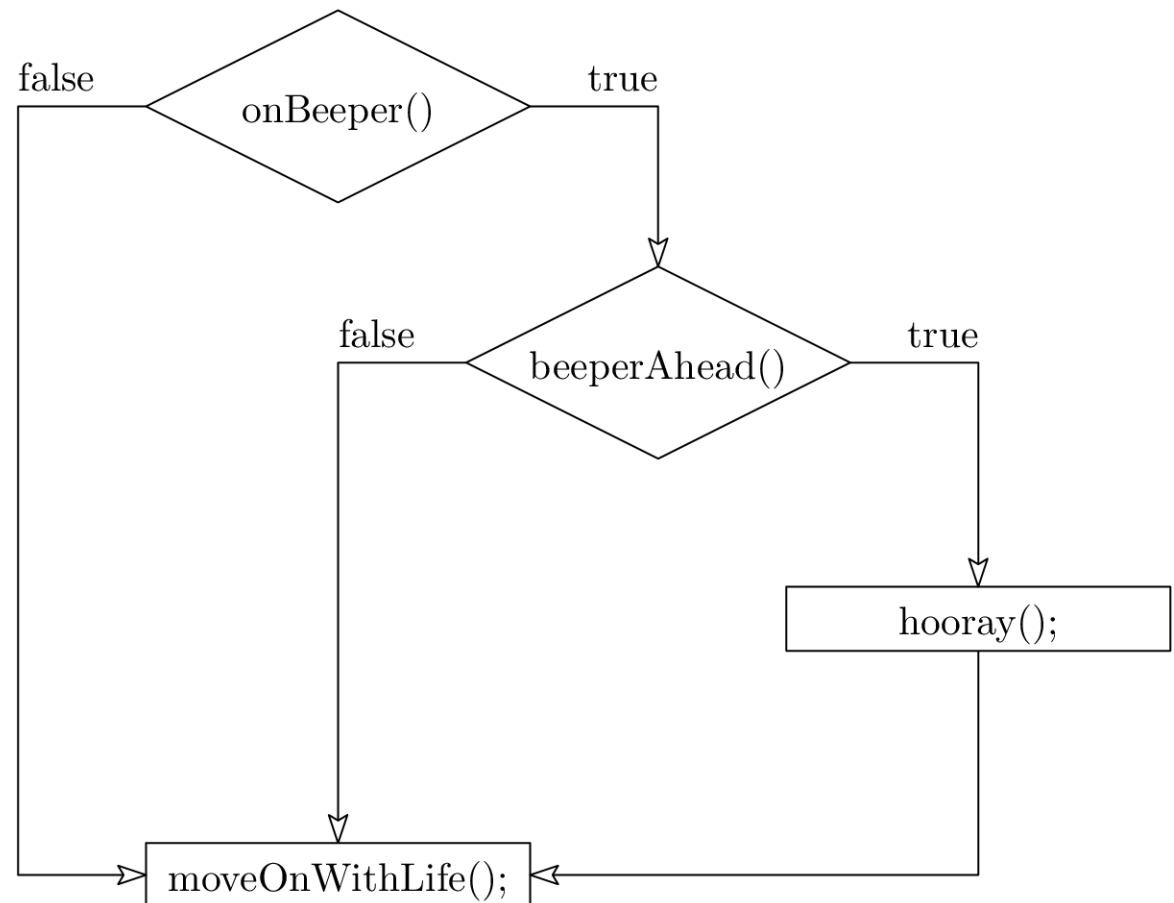
Bedingungen kombinieren

- Wenn man mehr als eine Bedingung prüfen möchte, stehen einem in Karel zu diesem Zweck 2 gängige Operatoren zur Verfügung:
- Konjunktion („**und**“)
 - `onBeeper()` **&&** `beeperAhead()`
 - **Beide** Bedingungen müssen gelten.
- Disjunktion („*inklusive* **oder**“)
 - `onBeeper()` **||** `beeperAhead()`
 - **Mindestens eine** der beiden Bedingungen muss gelten.

Die Konjunktion („und“)

```
if (onBeeper() && beeperAhead())  
{  
    hooray();  
}  
moveOnWithLife();
```

Gibt es ein Programm ohne &&
mit demselben Kontrollfluss?

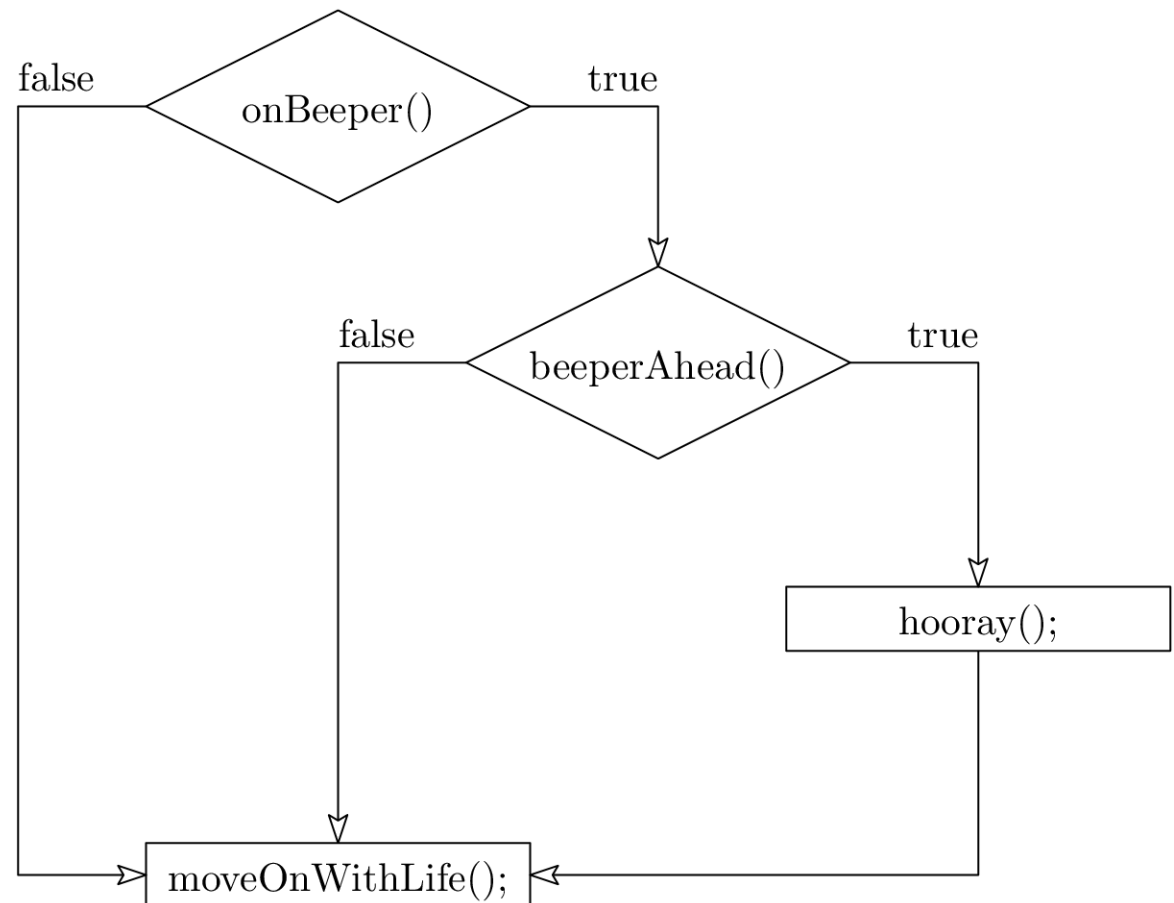


Die Konjunktion („und“)

```
if (onBeeper() && beeperAhead())  
{  
    hooray();  
}  
moveOnWithLife();
```

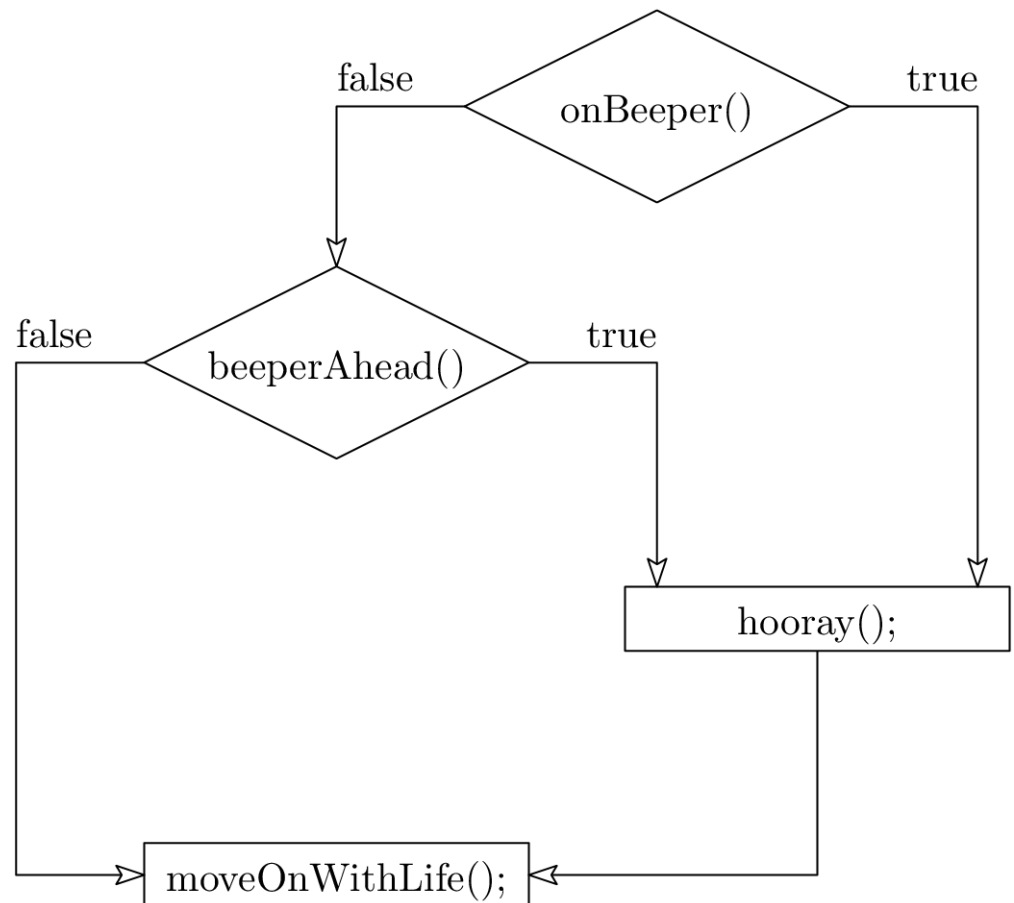
Gibt es ein Programm ohne &&
mit demselben Kontrollfluss? Ja!

```
if (onBeeper())  
{  
    if (beeperAhead())  
    {  
        hooray();  
    }  
}  
moveOnWithLife();
```



Die Disjunktion („oder“)

```
if (onBeeper() || beeperAhead())  
{  
    hooray();  
}  
moveOnWithLife();
```

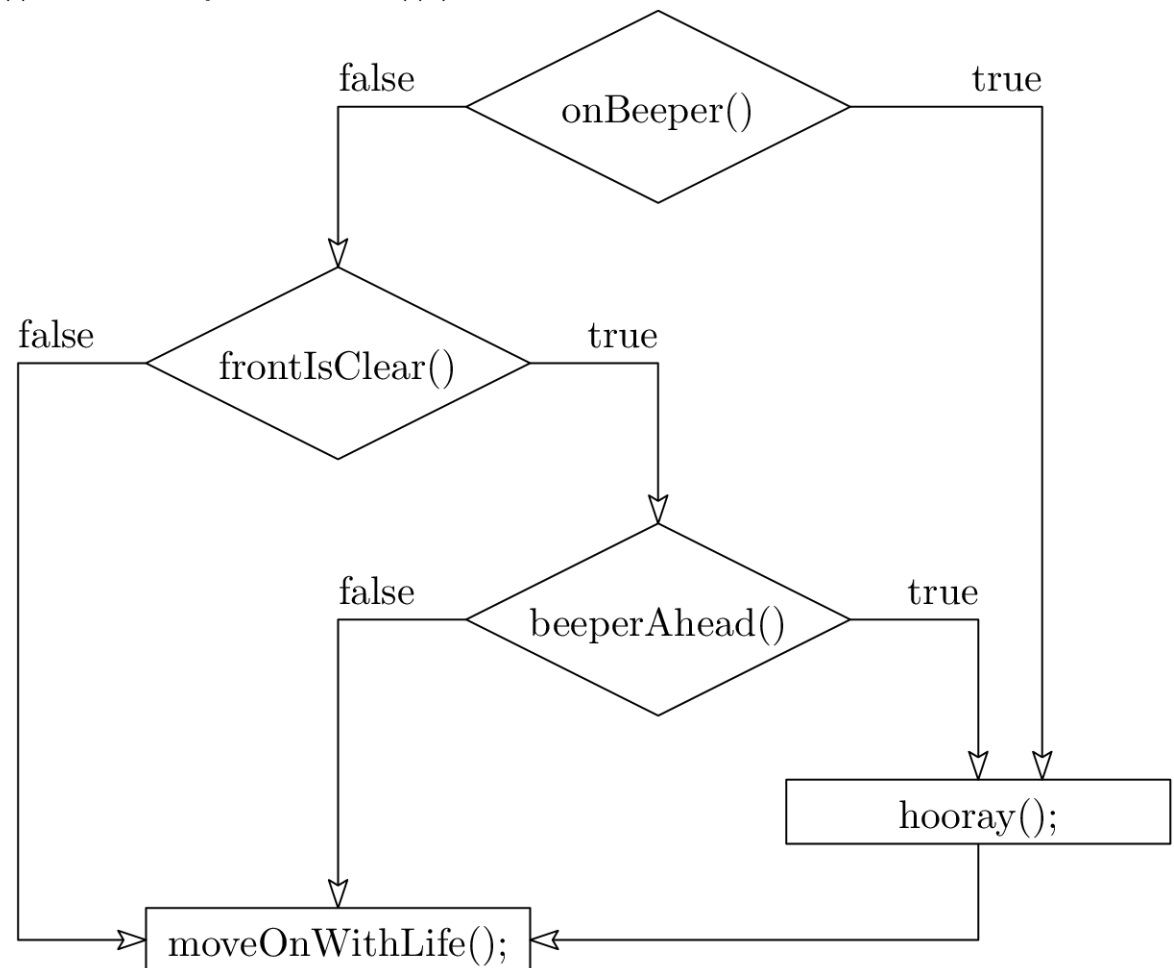


Bindungsstärke

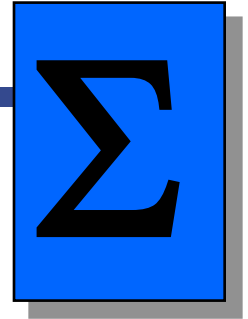
```
if (onBeeper() || frontIsClear() && beeperAhead())  
{  
    hooray();  
}  
moveOnWithLife();
```

Die Konjunktion (“und”) bindet stärker als die Disjunktion (“oder”).

Merksatz: “Und” vor Strichrechnung



Zusammenfassung



- Programmieren besteht in erster Linie darin, beliebig komplexe **Probleme** in immer kleinere Teilprobleme **zu zerlegen**.
- Hierzu kann man in Karel **zusammengesetzte Befehle** definieren.
- Irgendwann sind die Probleme so trivial, dass sie direkt von einem der **elementaren Befehle** gelöst werden können.
- Karel kann mit der **Fallunterscheidung** auf seine Umwelt reagieren.
- Elementare Bedingungen können mit den Operatoren **!**, **&&** und **||** zu beliebig komplexen Bedingungen zusammengesetzt werden.