Aufgabe: KARA steht vor einer freien Strecke, an deren Ende ein Baum steht. Er soll bis zum Baum laufen und sich dort umdrehen.
Dieses Problem ist mit einer `while`-Schleife sehr einfach zu lösen. Darum geht es hier nicht.

Das Neue: KARA soll ohne eine `while`-Schleife den Weg bis zum Baum finden, weil dies KARA ohne Java auch schon konnte. Damals wurde ein Zustand *zumBaum* benutzt, der einen Übergang in den selben Zustand und einen Übergang in den Zustand *stop* hatte. Dieses Vorgehen soll hier mit Methoden nachgebildet werden.

Lösung:

```
1  import javakara.JavaKaraProgram;
2  public class BisZumBaum extends JavaKaraProgram
3  {
4    void zumBaum()
5    {
6     if (!kara.treeFront())
7           {  kara.move();
8              zumBaum(); // rekursiver Aufruf
9           }
10    else { kara.turnLeft();
11           kara.turnLeft();
12          }
13   }
14
15   public void myProgram()
16   { zumBaum();
17   }
18 }  // Ende von BisZumBaum
```

Erläuterungen:

1. Die Methode `zumBaum` ruft sich, solange KARA nicht vor einem Baum steht, immer wieder nach einem `kara.move()` selbst auf. Erst wenn KARA den Baum erreicht hat, erfolgt kein Selbstaufruf mehr und die Methode ist beendet, nachdem KARA sich umgedreht hat. Der Aufruf von `zumBaum` in `myProgram` entspricht dem Setzen des Zustands *zumBaum* als Startzustand in KARA.

2. Andere Sichtweise: Wenn KARA keinen Baum vor sich hat, geht er vor und hat damit das Problem, den Baum zu finden, um einen Schritt vereinfacht und ruft die gleiche Methode auf.

3. Nach dem rekursiven Aufruf können weitere Methoden aufgerufen werden. Dazu ein Beispiel: KARA soll bis zum Baum laufen und zu seiner Startposition **zurückkehren**. *Rekursion* bedeutet *Zurückkehren*!

```
1  import javakara.JavaKaraProgram;
2
3  public class BisZumBaumUndZurueck extends JavaKaraProgram
4  {
5    void zumBaum()
6    {
7     if (!kara.treeFront())
8           {  kara.move();
9              zumBaum(); // rekursiver Aufruf
10             kara.move();
11          }
12    else { kara.turnLeft();
13           kara.turnLeft();
14          }
15   }
16
17   public void myProgram()
18   { zumBaum();
19   }
20 } // Ende von BisZumBaumUndZurueck
```

Den Ablauf der rekursiven Aufrufe macht man sich am besten mit der folgenden Übersicht klar.

```
void zumBaum()
{
if (!kara.treeFront())
{
kara.move();
zumBaum();
kara.move();
}
else { // entfaellt }
} // Ende von zumBaum
```

Man kann es sich so vorstellen, dass sich der Computer bei jedem Aufruf der Methode merkt, wo er die Methode verlassen hat, um die gleiche Methode noch einmal abzuarbeiten. Nach der Abarbeitung macht der Computer dann jeweils an der Stelle weiter, die er sich beim Aufruf gemerkt hat.

4. Rekursive Methoden benötigen **unbedingt** an einer Stelle eine Abbruchbedingung. Wenn nicht, dann merkt sich der Computer so viele Stellen, an denen er eine Methode rekursiv verlassen hat, bis der Speicher voll ist.

5. Viele Probleme lassen sich rekursiv und nicht-rekursiv lösen. Ein rekursiver Ansatz bietet sich immer dann, wenn man ein gegebenes Problem schrittweise vereinfachen kann und nach einem Schritt im Prinzip das gleiche Problem (aber um einen Schritt einfacher) vorliegen hat.

6. Manchmal sind nicht-rekursive schneller als rekursive Lösungen. Oft sind sie aber nur sehr viel komplizierter zu programmieren. Meistens sind rekursive Lösungen eleganter formulierbar.