Aufgabe 1: Schiebeparkplatz

Team-ID: 00501

Team: MWG go brrrrr

Bearbeiter/-innen dieser Aufgabe:
Noah Degner

22. November 2021

Inhaltsverzeichnis

Lösungsidee

Das Programm geht von Anfang bis Ende die Parkreihe durch und ermittelt dabei für jedes Auto (bzw. jeden Parkplatz), ob und wie die quer stehenden Wagen verschoben werden können, sodass das geparkte Auto hinausfahren kann und möglichst wenige Wagen verschoben werden müssen.

Dafür wird erst geprüft, ob der Weg aus dem Parkplatz überhaupt versperrt ist. Falls nicht, muss nichts verschoben werden. Falls doch, wird geprüft ob der Wagen, welcher die Ausfahrt versperrt, nach links oder nach rechts verschoben werden kann. Falls dieser dann auf einem weiteren quer stehenden Wagen landen würde, wird für diesen geprüft ob dieser in Schieberichtung des ersten quer stehenden Wagens aus dem weg geschoben werden kann. Dies geschieht so lange bis entweder alle verschobenen Wagen sich nicht mehr überlappen (aber noch auf dem Parkplatz bleiben) oder ein Wagen vom Platz geschoben werden müsste. Das ganze wird einmal in Richtung links und einmal in Richtung rechts für den die Ausfahrt versperrenden Wagen geprüft.

Falls das Verschieben in beide Richtungen möglich ist, werden die nötigen Verschiebungen für die Richtung für welche weniger Wagen verschoben werden müssen ausgegeben. Falls das Verschieben in nur eine Richtung möglich ist, werden die nötigen Verschiebungen für diese Richtung angegeben und falls in keine Richtung verschoben werden kann wird Ausgegeben, dass es keinen Weg gibt, den Weg für den geparkten Wagen frei zu machen.

Umsetzung

Implementiert wurde dies, indem zuerst ein Array mit Länge der Parkreihe an den Stellen, an welchen vor der Parkreihe ein Auto quer stehen würde, mit den Bezeichnung dieser quer stehenden Autos befüllt. Stellen ohne quer stehenden Wagen bleiben dabei leer.

Dann wird mittels for-Schleife folgendes für jeden Parkplatz wiederholt:

Zuerst wird mittels des Arrays geprüft, ob die Ausfahrt überhaupt blockiert ist. Falls nein wird dies ausgegeben und der Rest der for-Schleife mittels else-Statement übersprungen.

Falls der Weg jedoch blockiert wird, wird für den rechten und den linken Weg jeweils rekursiv geprüft, ob der blockierende Wagen verschoben werden kann. Um zu erfahren, ob das verschieben scheiter werden beide Richtungen in jeweils einem try-catch-Statement ausgeführt, sodass falls eine ArrayOutOfBoundsException geworfen wird (sprich das Programm einen Wagen über den Rand der Reihe verschieben müsste) diese gefangen wird und das Programm weiß, dass die Verschiebung in diese Richtung nicht geht.

Danach wird einfach die "Anleitung" zu den Verschiebungen ausgegeben (sie Kriterien in der Lösungsidee).

Ausführen des Programms

Um das Programm auszuführen, müssen alle Klassen (List, QuerstehendesAuto & Schiebeparkplatz; sprich das gesamte Programm) über die Anwendung BlueJ geöffnet werden. Dort muss dann die Klasse Schiebeparkplatz mit der rechten Maustaste angeklickt werden und auf *new Schiebeparkplatz* gedrückt werden. Mit dem dann gefragten *int* Wert geben sie an, welches Beispiel zu testen ist (es werden 0-6 als Eingabe akzeptiert). Die Ausgabe erfolgt dann von selbst.

Um eigene Beispiele zu testen, müssen diese jeweils in einem *else if* Statement nach dem Schema der schon vorhandenen Beispiele eingefügt werden.

Beispiele

Ausgaben zu den Beispielen auf BwInf Webseiten:

parkplatz0:

A: benötigt keine Verschiebungen!
B: benötigt keine Verschiebungen!
C:, H 1 rechts
D:, H 1 links
E: benötigt keine Verschiebungen!
F:, H 1 links, I 2 links
G:, I 1 links

parkplatz1:

A: benötigt keine Verschiebungen!
B:, P 1 rechts, O 1 rechts
C:, O 1 links
D:, P 1 rechts
E:, Q 1 rechts, P 2 rechts
F: benötigt keine Verschiebungen!
G:, Q 1 rechts
H:, Q 2 rechts
I: benötigt keine Verschiebungen!
J: benötigt keine Verschiebungen!
K:, R 1 rechts
L:, R 2 rechts
M: benötigt keine Verschiebungen!
N: benötigt keine Verschiebungen!

parkplatz2:

A: benötigt keine Verschiebungen!
B: benötigt keine Verschiebungen!
C:, O 1 rechts
D:, O 1 links
E: benötigt keine Verschiebungen!
F:, O 1 links, P 2 links
G:, P 1 links
H:, R 1 rechts, Q 1 rechts
I:, P 1 links, Q 1 links
J:, R 1 rechts
K:, P 1 links, Q 1 links, R 1 links
L: benötigt keine Verschiebungen!
M:, P 1 links, Q 1 links, R 1 links, S 2 links
N:, S 1 links

parkplatz3:

A: benötigt keine Verschiebungen!
B:, O 1 rechts
C:, O 1 links
D: benötigt keine Verschiebungen!
E:, P 1 rechts
F:, P 2 rechts
G: benötigt keine Verschiebungen!
H: benötigt keine Verschiebungen!

I:, Q 2 links
J:, Q 1 links
K:, Q 2 links, R 2 links
L:, Q 1 links, R 1 links
M:, Q 2 links, R 2 links, S 2 links
N:, Q 1 links, R 1 links, S 1 links

parkplatz4:

A:, R 1 rechts, Q 1 rechts
B:, R 2 rechts, Q 2 rechts
C:, R 1 rechts
D:, R 2 rechts
E: benötigt keine Verschiebungen!
F: benötigt keine Verschiebungen!
G:, S 1 rechts
H:, S 2 rechts
I: benötigt keine Verschiebungen!
J: benötigt keine Verschiebungen!
K:, T 1 rechts
L:, T 1 links
M: benötigt keine Verschiebungen!
N:, U 1 rechts
O:, U 1 links
P: benötigt keine Verschiebungen!

parkplatz5:

A: benötigt keine Verschiebungen!
B: benötigt keine Verschiebungen!
C:, P 2 links
D:, P 1 links
E:, Q 1 rechts
F:, Q 2 rechts
G: benötigt keine Verschiebungen!
H: benötigt keine Verschiebungen!
I:, R 1 rechts
J:, R 2 rechts
K: benötigt keine Verschiebungen!
L: benötigt keine Verschiebungen!
M:, S 1 rechts
N:, S 1 links

O: benötigt keine Verschiebungen!

Eigene Beispiele:

parkplatz6:

Schiebeparkplatz	Ausgabe
A C 3 D 1	A: benötigt keine Verschiebungen! B: Leider gibt es keinen Weg, die quer stehenden Autos so zu verschieben, dass dieses Auto frei Kommt. C:, D 1 links

Quellcode

```
public void verschieben(int anzahlPlaetze, List<QuerstehendesAuto> querAutoListe) {
    String[] belegtePlaetze = erkenneBelegtePlaetze(anzahlPlaetze, querAutoListe);
    for(int i = 0; i < anzahlPlaetze; i++) {
      //Überprüft, ob überhaupt ein quer stehendes Auto vor dem aktuellen Auto steht. Falls nein, kann direkt ausgegeben werden, dass gar keine Verschiebung nötig ist.
      if(belegtePlaetze[i] == null) {
        System.out.println(alphabet.charAt(i) + ": benötigt keine Verschiebungen!");
      }
      else {
        boolean rechtsMoeglich = true;
        boolean linksMoeglich = true;
        verschobeneAutosRechts = 0;
        verschobeneAutosLinks = 0;
        String anleitungRechts = new String();
        String anleitungLinks = new String();
        //Versucht, die Verschiebungen entweder nach rechts oder links zu ermitteln. Falls eine Richtung nicht möglich ist, erfährt dies das Programm, denn dann wird eine NullPointerException geworfen.
        try {
          anleitungRechts = ermittleVerschiebungRechts(i, belegtePlaetze);
        }
        catch(Exception e) {
          rechtsMoeglich = false;
```

```
            }
            try {
               anleitungLinks = ermittleVerschiebungLinks(i, belegtePlaetze);
            }
            catch(Exception e) {
               linksMoeglich = false;
            }
            //Falls sowohl der rechte als auch der linke Weg möglich sind, wird der Weg, welcher die
wenigsten Autos verschiebt ausgegeben.
            if(rechtsMoeglich && linksMoeglich) {
               if(verschobeneAutosRechts > verschobeneAutosLinks) {
                  System.out.println(alphabet.charAt(i) + ":" + anleitungLinks);
               }
               else {
                  System.out.println(alphabet.charAt(i) + ":" + anleitungRechts);
               }
            }
            //Falls nur ein Weg möglich ist, wird dieser ausgegeben.
            else if(rechtsMoeglich) {
               System.out.println(alphabet.charAt(i) + ":" + anleitungRechts);
            }
            else if(linksMoeglich) {
               System.out.println(alphabet.charAt(i) + ":" + anleitungLinks);
            }
            //Falls kein Weg möglich ist, wird man auch darüber in Kenntnis gesetzt
            else {
               System.out.println(alphabet.charAt(i) + ": Leider gibt es keinen Weg, die quer
stehenden Autos so zu verschieben, dass dieses Auto frei Kommt.");
            }

         }
      }
   }

private String[] erkenneBelegtePlaetze(int anzahlPlaetze, List<QuerstehendesAuto> querAutoListe)
{
      String[] blockierteWege = new String[anzahlPlaetze];
      if(querAutoListe != null) {
         querAutoListe.toFirst();
         while(querAutoListe.hasAccess()) {
            int position = querAutoListe.getContent().getPosition();
            blockierteWege[position] = querAutoListe.getContent().getBezeichnung();
```

```
            blockierteWege[position + 1] = querAutoListe.getContent().getBezeichnung();
            querAutoListe.next();
        }
    }
    return blockierteWege;
}

private String ermittleVerschiebungRechts(int autoPosition, String[] belegtePlaetze) {
    if(belegtePlaetze[autoPosition] == null) {
        return "";
    }
    verschobeneAutosRechts++; //Da ein weiteres Auto zu verschieben ist, wird die Anzahl der
verschobenen Autos erhöht.
    if(belegtePlaetze[autoPosition] == belegtePlaetze[autoPosition + 1]) { //Prüft, ob der Rest des
Autos rechts von der Autoposition ist.
        return ermittleVerschiebungRechts(autoPosition + 2, belegtePlaetze) + ", " +
belegtePlaetze[autoPosition] + " 1 rechts";
    }
    else { //Da der Rest des Autos nicht rechts von der Autoposition war, muss dieser links von der
Autoposition sein (es wird von einer Autolänge von 2 ausgegangen).
        return ermittleVerschiebungRechts(autoPosition + 2, belegtePlaetze) + ", " +
belegtePlaetze[autoPosition] + " 2 rechts";
    }
}

private String ermittleVerschiebungLinks(int autoPosition, String[] belegtePlaetze) {
    if(belegtePlaetze[autoPosition] == null) {
        return "";
    }
    verschobeneAutosLinks++; //Da ein weiteres Auto zu verschieben ist, wird die Anzahl der
verschobenen Autos erhöht.
    if(belegtePlaetze[autoPosition] == belegtePlaetze[autoPosition - 1]) { //Prüft, ob der Rest des
Autos links von der Autoposition ist.
        return ermittleVerschiebungLinks(autoPosition - 2, belegtePlaetze) + ", " +
belegtePlaetze[autoPosition] + " 1 links";
    }
    else { //Da der Rest des Autos nicht links von der Autoposition war, muss dieser rechts von der
Autoposition sein (es wird von einer Autolänge von 2 ausgegangen).
        return ermittleVerschiebungLinks(autoPosition - 2, belegtePlaetze) + ", " +
belegtePlaetze[autoPosition] + " 2 links";
    }
}
```