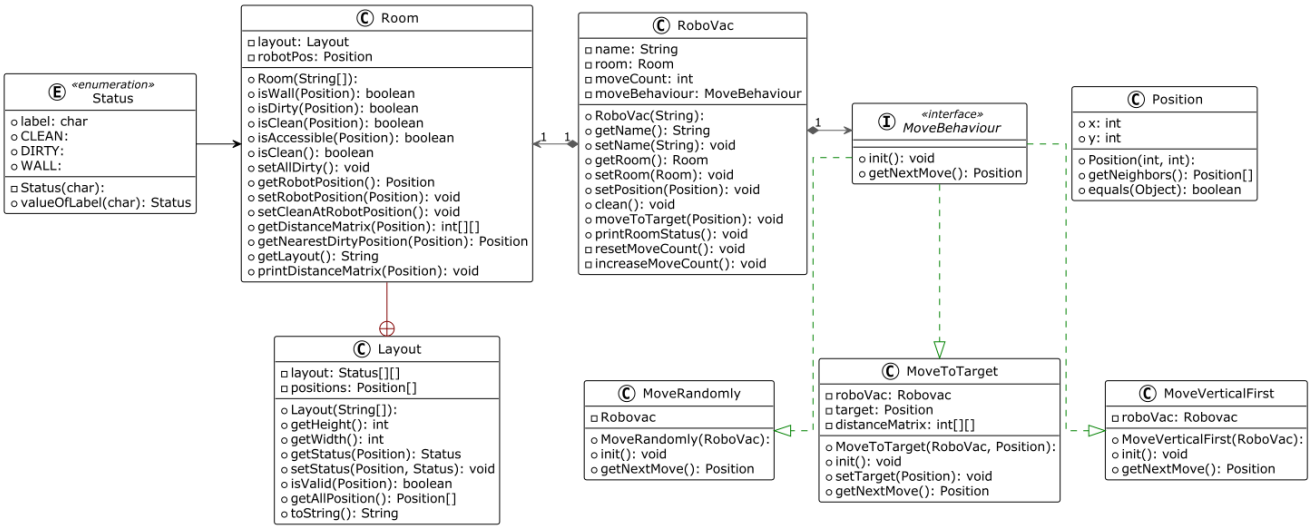
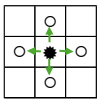
# RoboVac 2.0

Überarbeiten Sie Ihre Implementierung des Staubsauger-Roboters. Orientieren Sie sich dabei an folgendem Klassendiagramm:




Beachten Sie dabei folgende Änderungen:

- Es wurde eine neue Klasse *Position* eingeführt, die eine Position im Raum beschreibt. Interessant sind hier insbesondere
  - die Methode *getNeighbors()*, die die Positionen aller 4 Nachbarfelder liefert.
  - die Methode *equals()*, die die Gleichheit von 2 Positionen prüft.

  Viele Methoden in der *Raum*- bzw. *RoboVac*-Klasse arbeiten nun mit Positionsobjekten, statt mit int-Koordinaten. Diese müssen entsprechend geändert werden.
- Die Klasse *Room* wurde grundlegend verändert:
  - Das Room-Layout wurde in eine innere Klasse *Layout* ausgelagert. Dies hat den Vorteil, dass die *Room*-Klasse ausschließlich mit *Position*-Objekten arbeitet und die Umsetzung von Positionen auf Raum-Koordinaten konzentriert in der *Layout*-Klasse stattfindet.
  - Die *Raum*-Klasse wurde mit zusätzlichen Methoden ausgestattet, z. B.
    - *isValid()*: Prüfung, ob eine gegebene Position innerhalb des Raumlayouts liegt
    - *isAccessible()*: Prüfung, ob eine gegebene Position vom Roboter erreichbar ist, d. h. es ist eine valide Position und keine Wand.
    - *getNearestDirtyPosition()*: Liefert zu einer gegebenen Position die am nächsten liegende schmutzige Position.
- Der Roboter wurde um eine Funktion erweitert, um auf dem kürzesten Weg eine bestimmte Zielposition im Raum zu erreichen. Dazu wurden
  - in der Klasse *RoboVac* die Methode *moveToTarget()* eingeführt,
  - eine neue Bewegungsstrategie *MoveToTarget()* hinzugefügt und
  - in der *Room* Klasse eine Methode *getDistanceMatrix()* zur Berechnung einer Distanzmatrix vorgesehen.
- Die *move*-Methode im *MoveBehaviour*-Interface wurde zu einer *getNextMove*-Methode umgearbeitet. Damit liegt es nicht mehr in der Verantwortung dieser Methode, den Roboter auch tatsächlich zu bewegen, sondern nur die nächste Position vorzuschlagen.
  Hintergrund: Der Roboter kann nun diesen Vorschlag bewerten und entscheiden, ob er diese Bewegung tatsächlich ausführt oder eventuell auf eine andere Bewegungsstrategie umschaltet.
  Beispielsituation:
  - Der Roboter befindet sich in einer Ecke.
  - Der Vorschlag der aktiven *MoveVerticalFirst*-Strategie führt ihn auf ein Feld, das bereits gereinigt ist, was in der weiteren Folge zu unnötigen Umwegen führen kann.

- Daher ändert der Roboter sein Verhalten: Er sucht sich den nächstliegenden Punkt, welcher nicht gereinigt ist und fährt diesen mittels *MoveToTarget*-Strategie direkt an, um dann wieder in die *MoveVerticalFirst*-Strategie zu wechseln und so weiterzuarbeiten.

- Implementieren Sie eine verbesserte Umsetzung der *clean*-Methode:
  - Der Roboter soll zunächst nach der *moveVerticalFirst*-Strategie arbeiten. Er erhält hier bei jedem *getNextMove()* einen Vorschlag für den nächsten Schritt.
  - Führt ihn dieser Vorschlag auf ein schmutziges Feld, so vollzieht er den Schritt.
  - Würde ihn dieser Vorschlag auf ein gereinigtes Feld führen, so nutzt er die *moveToTarget*-Methode, um die nächstliegende schmutzige Position direkt anzufahren. Dabei wird ein Wechsel der Bewegungsstrategie durchgeführt.
  - Wenn er die Zielposition erreicht hat, so wird in der clean-Methode mit der *moveVertical-First*-Strategie fortgefahren.
  - Die Methode endet erst, wenn der gesamte Raum gereinigt wurde.

Sie können und sollen bei Bedarf weitere Klassenmember (Methoden, Eigenschaften) einführen.

**Überlegungen zur MoveToTarget-Bewegungsstrategie – Ermittlung der Distanzmatrix:**

Raum-Layout

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | # | # | # | # | # | # | # | # | # | # | # | # |
| 1 | # | D | | | | | | | | | | # |
| 2 | # | | | | | | | | | | | # |
| 3 | # | | | | | | | | | | | # |
| 4 | # | | | | # | # | # | # | | | | # |
| 5 | # | | | | # | | | # | | | | # |
| 6 | # | | | | # | | | | | | | # |
| 7 | # | | | | # | | R | | | | | # |
| 8 | # | # | # | # | # | # | # | # | # | # | # | # |

Distanzmatrix

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 1 | -1 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | -1 |
| 2 | -1 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | -1 |
| 3 | -1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 | 11 | -1 |
| 4 | -1 | 3 | 4 | 5 | -1 | -1 | -1 | -1 | 10 | 11 | 12 | -1 |
| 5 | -1 | 4 | 5 | 6 | -1 | 16 | 15 | -1 | 11 | 12 | 13 | -1 |
| 6 | -1 | 5 | 6 | 7 | -1 | 15 | 14 | 13 | 12 | 13 | 14 | -1 |
| 7 | -1 | 6 | 7 | 8 | -1 | 16 | 15 | 14 | 13 | 14 | 15 | -1 |
| 8 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

Das Raum-Layout zeigt die Positionen der Wände, des Roboters und weiterer wichtiger Objekte (z. B. Dockingstation).

Die Distanzmatrix zeigt für jede Position im Raum die geringste Anzahl an Schritten an, die benötigt werden, um zu einer bestimmten Zielposition zu gelangen.
Beispiel: Von der Position (6,7) sind 15 Schritte zur Zielposition (1,1) erforderlich.
Der kürzeste Weg wird dann zurückgelegt, wenn bei jedem Schritt die Distanz zum Ziel abnimmt.

**Algorithmus zur Berechnung der Distanzmatrix:**

Erstelle eine int-Matrix in der Größe des Raum-Layouts
Inititalisiere die Distanzmatrix mit -1 als Kennzeichnung noch nicht berechneter Distanzen
Setze die Distanz im Ziel auf 0
Setze ActPos auf die Zielposition
Solange ActPos definiert:
    Für alle Nachbarn von ActPos
        Wenn der Nachbar erreichbar ist und noch keine Distanz für den Nachbarn berechnet wurde:
            Setze die Distanz des Nachbarn auf (Distanz von ActPos + 1)
            Füge die Position des Nachbarn zu einer Bearbeitungs-Queue hinzu
    Setze ActPos auf das nächste Element in der Bearbeitungsqueue und entferne das Element aus der Queue

In die Bearbeitungsqueue werde ständig neue Nachbarn eingelagert, bzw. wieder entnommen
Queue: (2,1), (1,2), (3,1), (2,2), (1,3), ...

Tipp: Verwenden Sie zur Verwaltung der Bearbeitungsqueue geeignete Klassen des **Java Collections Frameworks** in **java.util**.