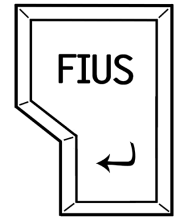




Universität Stuttgart



## Übungsblatt extra

Java Vorkurs(2022)

### Aufgabe 1 - Neo im Labyrinth

Die Agenten haben es geschafft, Neo in einem Labyrinth einzusperren. Um zu entkommen, muss er die einzige Telefonzelle finden. Neo erinnert sich, dass man aus jedem Labyrinth entkommen kann, wenn man immer an der rechten Wand entlangläuft.

Implementiere die Operation `checkSideWall()` der Klasse `MySmartNeo`. Sie soll `true` zurückgeben, wenn sich auf Neos rechter Seite eine Wand befindet. Ansonsten soll sie `false` zurückgeben.

Implementiere die Operation `moveSmart()` der Klasse `MySmartNeo`. In dieser soll Neo zuerst überprüfen, ob sich zu seiner rechten eine Wand befindet. Ist dies nicht der Fall, soll er sich nach rechts drehen und einen Schritt in diese Richtung gehen. Anderenfalls soll er überprüfen, ob er geradeaus laufen kann. Ist dies der Fall, soll er einen Schritt nach vorne machen. Sollte sich sowohl rechts, als auch geradeaus eine Wand befinden, soll Neo sich nach links drehen und stehen bleiben.

Die Klasse `OptionalLabyrinthTask` enthält ein Attribut vom Typ `SmartNeo`. Weise diesem Attribut in der `prepare`-Operation ein neues `MySmartNeo`-Objekt zu und platziere es auf Position  $[0, 0]$ . Vervollständige weiterhin die `solve`-Operation mithilfe der Operationen, die du gerade geschrieben hast, sodass Neo das Telefon erreicht und darauf stehen bleibt.

## Aufgabe 2 - Neo im Labyrinth Teil 2

Nachdem Neo aus dem Labyrinth entkommen ist, fragt er sich, ob er nicht schneller gewesen wäre, wenn er der linken Wand gefolgt wäre.

Implementiere die Operationen `checkSideWall()` und `moveSmart()` der Klasse `MyLefthandedSmartNeo` entsprechend. Wie musst du die `OptionalLabyrinthTask` Klasse verändern, damit Neo an der linken Wand entlang läuft?

## Aufgabe 3 - Rekursion

### Rekursion

Ein Kommando welches sich selbst aufruft wird als rekursiv bezeichnet. Die Fibonacci Reihe zum Beispiel (1,1,2,3,5,8,13,21,...) kann beschrieben werden durch  $F(0) = 1$   $F(1) = 1$   $F(n) = F(n-2) + F(n-1)$ . wobei  $F(n)$  die n-te Fibonacci Zahl ist. bei  $F(n) = F(n-2) + F(n-1)$  wird erneut mit einem kleineren Argument verwendet. Dadurch wird das Argument in allen Schritten kleiner bis es bei  $F(0)$  oder  $F(1)$  ankommt welche einen festen Wert haben.

- a) versuche (rekursiv) ein Fibonacci Kommando zu implementieren welches einen Integer n erhält und dann  $F(n)$  Münzen ablegt.
- b) lasse Neo die ersten sechs Elemente der Fibonacci Folge nacheinander auf das Spielfeld legen.
- c) (bonus) implementiere das Fibonacci Kommando aus a) iterativ (ohne Rekursion).

## Aufgabe 4 - Rekursion 2 - Füllen.

- a) Implementiere das `moveTo(Position position)` Kommando in der Klasse `neo`, welches `neo` zu der übergebenen Position laufen lässt. Um das Kommando zu testen, lasse `neo` zu (2,4) laufen (Tipp: In dieser Aufgabe gibt es keine Wände auf dem Spielfeld. Du musst daher keinen komplexen Wegfindungsalgorithmus implementieren.)
- b) `Neo` ist umgeben von Münzen. Wir wollen auch nun das innere dieser Form mit Münzen füllen. Dafür verwenden wir einen beliebigen (rekursiven) Füllalgorithmus der funktioniert wie folgt:
- `Fill(Position pos(x,y))`:
- lege eine Münze auf `pos`
  - falls `(x-1,y)` leer ist: `Fill(x-1,y)`
  - falls `(x+1,y)` leer ist: `Fill(x+1,y)`
  - falls `(x,y-1)` leer ist: `Fill(x,y-1)`
  - falls `(x,y+1)` leer ist: `Fill(x,y+1)`

implementieren diesen Füllalgorithmus, und lasse `neo` die Form mit Münzen füllen