

# Funktionale und objektorientierte Programmierkonzepte

## Übungsblatt 01



TECHNISCHE  
UNIVERSITÄT  
DARMSTADT

Prof. Karsten Weihe

Wintersemester 23/24

Themen:

Relevante Foliensätze:

Abgabe der Hausübung:

v1.0.2

Programmieren in Java mit Hilfe von FopBot

01a und 01b

03.11.2023 bis 23:50 Uhr

### Hausübung 01

#### *Foreign Contaminants*

**Gesamt: 16 Punkte**

**Beachten Sie die Seite *Verbindliche Anforderungen für alle Abgaben im Moodle-Kurs*.**

Verstöße gegen verbindliche Anforderungen führen zu Punktabzügen und können die korrekte Bewertung Ihrer Abgabe beeinflussen. Sofern vorhanden, müssen die in der Vorlage mit TODO markierten crash-Aufrufe entfernt werden. Andernfalls wird die jeweilige Aufgabe nicht bewertet.

Die für diese Hausübung relevanten Verzeichnisse sind `src/main/java/h01` und ggf. `src/test/java/h01`.

#### **Verbindliche Anforderungen für die gesamte Hausübung:**

- Alle Zufallszahlen werden mit der Methode `Utils.getRandomInteger(int min, int max)` erzeugt.
- Ihre Implementierung muss auch dann funktionieren, wenn die Einstellungen in der `h01.properties`-Datei geändert werden.

#### **Hinweise (für die gesamte Hausübung):**

- Sie können das Spiel jederzeit über die `main`-Methode der Klasse `Main` starten. Hierbei werden automatisch die Einstellungen aus der `h01.properties`-Datei eingelesen und werden in der Datei `GameConstants` gespeichert.
- Alle Spieleinstellungen können Sie in der Datei `src/main/resources/h01.properties` anpassen.
- Zum legen und aufheben von Münzen sollen die Methoden `putCoin` und `pickCoin` verwendet werden.
- Ob eine Vorwärtsbewegung möglich ist soll mit der Methode `isFrontClear` überprüft werden.
- Ob eine Münze auf dem aktuellen Feld liegt soll mit der Methode `isOnACoin` überprüft werden.
- Sie dürfen davon ausgehen, dass die Werte in `h01.properties` immer einen sinnvollen Wert haben. Das heißt, bis auf den Random Seed sind alle Werte garantiert immer positiv.

## Einleitung

Das Konzept dieser Hausübung ist von einer Szene aus dem Film „Wall-E“ inspiriert, in der ein Putzroboter versucht, die dreckigen Reifenspuren, die Wall-E hinterlässt, zu reinigen und ihm hinterher zu putzen. Unser Ziel ist es, ein ähnliches Szenario unter der Verwendung des FOPBot-Frameworks zu simulieren.

Die Welt ist ein Labyrinth bestehend aus Wänden, wobei jedes Feld im Labyrinth erreichbar ist, aber die Bewegungsfreiheit durch die Platzierung der Wände eingeschränkt ist.

Es gibt zwei Arten von Robotern, die „Contaminants“ (also Verschmutzer), welche in den Klassen `Contaminant1` und `Contaminant2` implementiert werden, und den „Cleaner“ (also Reiniger) welcher in der Klasse `CleaningRobot` implementiert wird. Die Contaminants bewegen sich nach einem vordefinierten Muster im Labyrinth und „verschmutzen“ dabei die Welt, indem sie Münzen ablegen. Der Cleaner wird durch Tastatureingaben gesteuert. Er kann Münzen aufnehmen und ablegen, jedoch maximal `GameConstants.CLEANER_CAPACITY` Münzen gleichzeitig tragen. Die Aufgabe des Cleaners besteht darin, so viele Münzen wie möglich in ein vordefiniertes, gelb markiertes Feld zu bringen, das wir als Abladezone bezeichnen. Die Abladezone befindet sich immer bei  $(0, \text{World.getHeight()} - 1)$ .

Das Spiel hat klare Gewinnbedingungen: Die Contaminants gewinnen, wenn mindestens 50% der Spielfläche mit Münzen belegt ist. Der Cleaner hingegen gewinnt, wenn er entweder 200 Münzen in die Abladezone gelegt hat oder wenn den Contaminants die Münzen ausgehen. (diese schalten sich dann selbst aus)

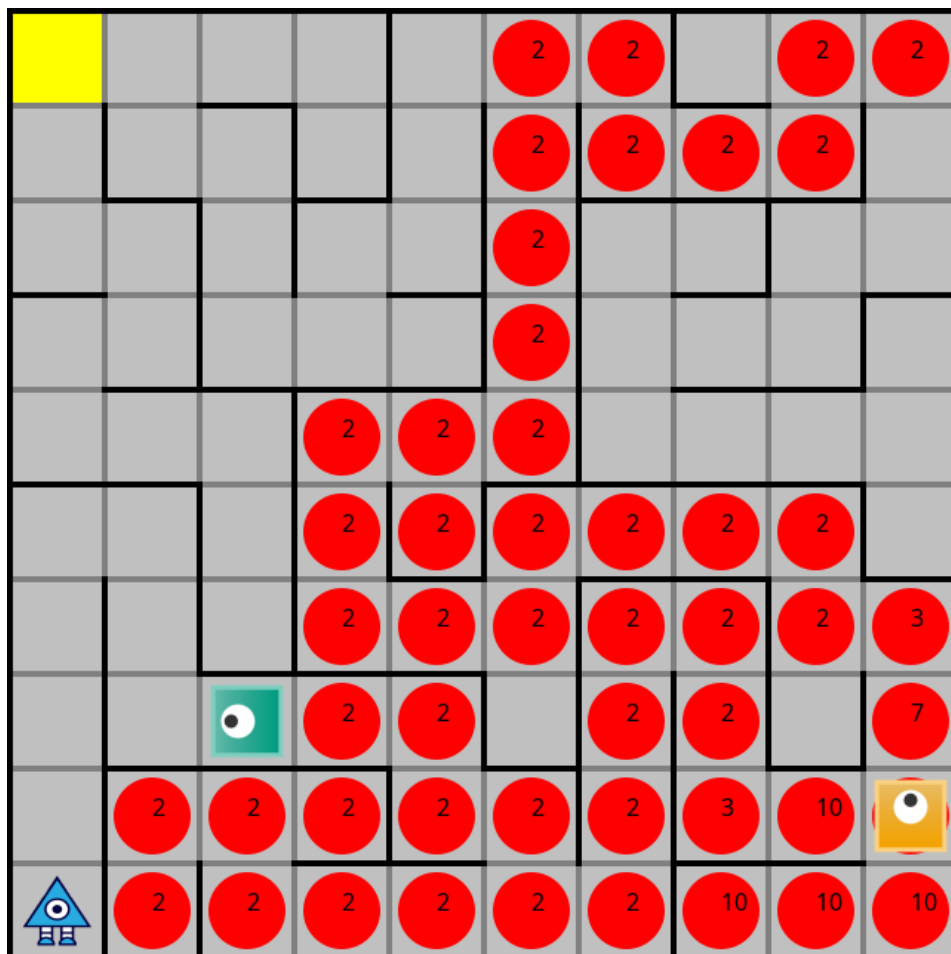


Abbildung 1: Beispiel für eine Spielsituation

**H1: Steuerung des „Cleaner“-Roboters****4 Punkte**

Zunächst wollen wir die Steuerung des „Cleaner“-Roboters implementieren. Der Roboter kann über Tasteneingaben gesteuert werden. Die Tastenbelegung ist wie folgt:

- **Pfeiltasten:** Roboter bewegt sich in die entsprechende Richtung
- **Keine Taste:** Roboter bleibt stehen
- **Leertaste:** Roboter legt eine Münze ab
- **R:** Roboter nimmt eine Münze auf

Die Tastatureingaben werden schon automatisch von der vorgegebenen Implementierung abgefangen, Sie müssen diese also nicht selbst implementieren. Ihre Aufgabe ist es nun, eine Methode zu vervollständigen, die dann die Steuerung des Roboters möglich macht.

Vervollständigen Sie die Methode `handleKeyInput` der Klasse `CleaningRobot` (Datei `CleaningRobot.java`). Diese bekommt drei Werte übergeben:

- Parameter `direction`: eine Ganzzahl, die für eine vom Spieler gedrückte Taste auf der Tastatur steht. Die Pfeiltaste nach oben hat den Wert 0, die Pfeiltaste nach rechts den Wert 1, die Pfeiltaste nach unten den Wert 2 und die Pfeiltaste nach links den Wert 3.
- Parameter `shouldPutCoins`: ein boolescher Wert, der angibt, ob der Roboter eine Münze ablegen soll.
- Parameter `shouldPickCoins`: ein boolescher Wert, der angibt, ob der Roboter eine Münze aufheben soll.

Die Methode führt drei Schritte in genau der gegebenen Reihenfolge aus:

1. Münzen ablegen, bzw. aufheben, sofern die entsprechenden Parameter gesetzt sind.
2. Drehung des Roboters in die entsprechende Richtung, sofern `direction` zwischen 0 und 3 liegt (beide inklusive).
3. Bewegung des Roboters in die entsprechende Richtung (1 Schritt), sofern der Weg nicht durch eine Wand versperrt ist.

Falls `shouldPutCoins` den Wert `true` hat, soll der Roboter eine Münze auf dem aktuellen Feld ablegen, wenn er noch Münzen hat. Falls `shouldPickCoins` den Wert `true` hat, soll der Roboter eine Münze aufheben, sofern sich eine auf dem aktuellen Feld befindet. Der Cleaner kann maximal `GameConstants.CLEANER_CAPACITY` Münzen tragen. Wenn `shouldPickCoins` den Wert `true` hat und sich eine Münze auf dem Feld befindet, der Roboter aber bereits `GameConstants.CLEANER_CAPACITY` Münzen trägt, so wird die Münze einfach liegen gelassen. Sie dürfen davon ausgehen, dass `shouldPutCoins` und `shouldPickCoins` nie gleichzeitig den Wert `true` haben.

Sollte der Methode ein `direction`-Wert übergeben werden, der nicht zwischen 0 und 3 liegt (beide inklusive), so soll der Cleaner sich nicht bewegen. Liegt der Wert im vorher genannten Bereich, so soll der Roboter zunächst in die Richtung blicken, die der gedrückten Pfeiltaste entspricht (auch wenn er dann in Richtung einer Wand blicken würde). Also UP bei gedrückter Pfeiltaste nach oben usw. Nachdem der Roboter in die entsprechende Richtung blickt, soll er einen Schritt in eben diese Richtung gehen, sofern der Weg nicht durch eine Wand versperrt ist. Ist der Weg versperrt, so verbleibt der Roboter an seiner momentanen Position.

**Hinweis:**

in den Verbindliche Anforderungen für die gesamte Hausübung finden Sie neben den verbotenen Methoden auch einige nützliche Methoden, die Sie verwenden können.

---

**H2: Steuerung der „Contaminant“-Roboter****8 Punkte**

---

Nun wollen wir die Contaminants implementieren. Es gibt zwei verschiedene Arten von Contaminants. Die erste Art wählt ihre Richtung zufällig aus und ist etwas langsamer, kann dafür aber mehr Münzen ablegen. Die zweite Art hingegen wählt ihre Richtung nicht zufällig aus, sondern versucht, sich immer an der linken Wand zu orientieren. Sie ist etwas schneller, kann dafür aber immer nur zwei Münzen ablegen.

**Hinweise** (Für die gesamte H2):

- Für das generieren einer Zufallszahl wird die Methode `Utils.getRandomInteger(int min, int max)` verwendet. Diese Methode gibt eine Zufallszahl zwischen `min` und `max` (beide inklusive) zurück.
- Um Abzufragen, wie viele Münzen auf einem Feld liegen, kann die Methode `Utils.getCoinAmount(int x, int y)` verwendet werden. Diese Methode gibt die Anzahl der Münzen auf dem Feld mit den Koordinaten `x` und `y` zurück.

---

**H2.1: Contaminant1****4 Punkte**

---

Vervollständigen Sie die parameterlose Methode `doMove` der Klasse `Contaminant1` (Datei `Contaminant1.java`). Die Methode führt die folgenden Schritte in genau der gegebenen Reihenfolge aus:

1. Falls der Roboter keine Münzen mehr besitzt, so wird er ausgeschaltet.
2. Falls der Roboter ausgeschaltet ist oder keine Münzen mehr hat, so wird die Methode abgebrochen. (also die restlichen Schritte werden nicht ausgeführt)
3. Der Roboter bestimmt eine Zufallszahl mittels `Utils.getRandomInteger`, zwischen `GameConstants.CONTAMINANT_ONE_MIN_PUT_COINS` und `GameConstants.CONTAMINANT_ONE_MAX_PUT_COINS` (beide inklusive), und legt entsprechend viele Münzen ab. Dabei dürfen maximal 20 Münzen auf einem Feld liegen. Wenn also 20 Münzen oder mehr auf dem Feld liegen werden keine weiteren Münzen gelegt. Falls der Roboter weniger Münzen hat, als er ablegen möchte, so legt er alle Münzen ab, die er hat. (Hier wird die Methode nicht abgebrochen, falls der Roboter keine Münzen mehr hat)
4. Der Roboter dreht sich einmal gegen den Uhrzeigersinn um die eigene Achse und prüft dabei in jeder Richtung, ob er sich bewegen kann. Dannach hat er die gleiche Blickrichtung wie zuvor.
5. Der Roboter wählt aus den Richtungen, in die er sich bewegen kann, eine zufällige Richtung aus und dreht sich in diese Richtung.
6. Der Roboter bewegt sich in die entsprechende Richtung, sofern der Weg nicht durch eine Wand versperrt ist (1 Schritt). Ist der Weg versperrt, so verbleibt der Roboter an seiner momentanen Position.

**H2.2: Contaminant2****4 Punkte**

Vervollständigen Sie die parameterlose Methode `doMove` der Klasse `Contaminant2` (Datei `Contaminant2.java`). Die Methode führt die folgenden Schritte in genau der gegebenen Reihenfolge aus:

1. Falls der Roboter keine Münzen mehr besitzt, so wird er ausgeschaltet.
2. Falls der Roboter ausgeschaltet ist oder keine Münzen mehr hat, so wird die Methode abgebrochen.
3. Wenn auf dem Feld, auf dem sich der Roboter befindet, weniger als zwei Münzen liegen, dann erhöht der Roboter die Anzahl der Münzen auf zwei. Ansonsten wird die Anzahl der Münzen auf dem Feld nicht verändert. Falls der Roboter weniger Münzen hat, als er ablegen möchte, so legt er alle Münzen ab, die er hat. (Hier wird die Methode nicht abgebrochen, falls der Roboter keine Münzen mehr hat)
4. Der Roboter dreht sich einmal gegen den Uhrzeigersinn um die eigene Achse und prüft dabei in jeder Richtung, ob er sich bewegen kann. Dannach hat er die gleiche Blickrichtung wie zuvor.
5. Jetzt orientiert sich der Roboter an der linken Wand. Dazu dreht er sich in die erste Richtung, in die er sich bewegen kann (beginnend mit der Richtung, die sich links von seiner aktuellen Blickrichtung befindet, weiter im Uhrzeigersinn). Falls er sich in keine Richtung bewegen kann, so verbleibt der Roboter an seiner momentanen Position und behält seine Blickrichtung bei.
6. Der Roboter bewegt sich in die entsprechende Richtung, sofern der Weg nicht durch eine Wand versperrt ist (1 Schritt). Ist der Weg versperrt, so verbleibt der Roboter an seiner momentanen Position.

**Hinweis:**

Wenn sich der Roboter „im Urzeigersinn“ drehen soll, so muss er sich dreimal gegen den Uhrzeigersinn drehen, da es keine `turnRight` Methode für `Robots` gibt.

**H3: Gewinnbedingungen****4 Punkte**

Als letztes wollen wir nun die Gewinnbedingungen überprüfen, um das Spiel zu vervollständigen. Vervollständigen Sie dazu die Methode `checkWinCondition` der Klasse `GameController` (Datei `GameController.java`). Die Methode wird automatisch nach jedem Spielschritt (tick) aufgerufen und überprüft, ob eine Partei gewonnen hat. Zur Erinnerung, hier nochmal die zwei Gewinnbedingungen:

- Der Cleaner hat gewonnen, wenn alle Contaminants ausgeschaltet sind oder sich in der Abladezone mindestens 200 Münzen befinden.
- Die Contaminants hingegen haben gewonnen, wenn mindestens 50% der Felder mit Münzen belegt sind, oder mathematisch ausgedrückt:  $\frac{\text{Anzahl der Felder mit Münzen}}{\text{Anzahl der Felder insgesamt}} \geq 0.5$ .

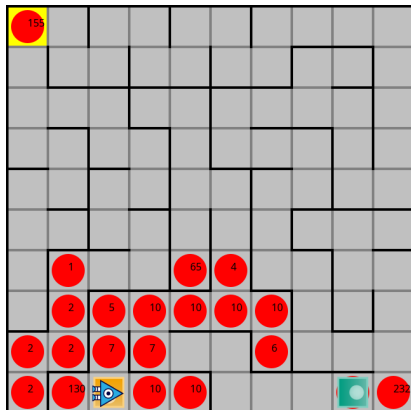
In beiden Gewinnfällen sollen die Roboter der unterlegenen Partei ausgeschaltet werden, die Gewinnnachricht in die Konsole ausgegeben werden und die Methode `stopGame()` aufgerufen werden. Die Gewinnnachricht soll dabei folgendermaßen aussehen:

- Cleaner gewinnt: "Cleaning robot won!"
- Contaminants gewinnen: "Contaminants won!"

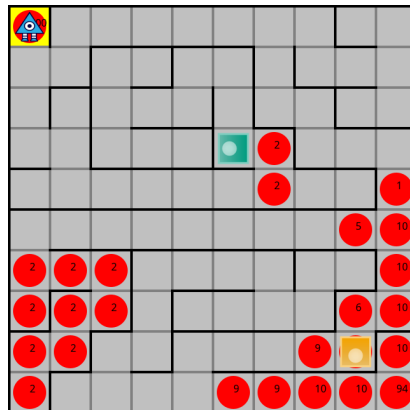
Falls beide Parteien gleichzeitig gewinnen, so soll der Cleaner gewinnen.

**Hinweise:**

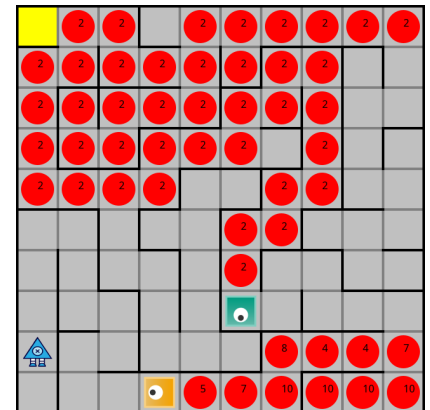
- Die Instanzen der drei Roboter erhalten Sie über die Methoden `getCleaningRobot()`, `getContaminant1()` und `getContaminant2()`.
- Nutzen Sie wieder die Methode `Utils.getCoinAmount(int x, int y)`, um die Anzahl der Münzen auf einem Feld zu erhalten.



(a) Cleaner gewinnt, da alle Contaminants ausgeschaltet sind und ihre Münzen verbraucht haben.



(b) Cleaner gewinnt, da er mehr als 200 Münzen in die Abladezone gelegt hat.



(c) Contaminants gewinnen, da mehr als 50% der Felder mit Münzen belegt sind.

Abbildung 2: Beispiele für Gewinnbedingungen