Funktionale und objektorientierte Programmierkonzepte

TECHNISCHE UNIVERSITÄT DARMSTADT

Alexander Mainka, Kim Berninger,
Dustin Glaser, Jan Bormet und
Julius Hardt
(Gesamtleitung: Prof. Dr. Karsten Weihe)

Wintersemester 20/21
v1.0

Übungsblatt 1

Themen: FopBot

Relevante Foliensätze: 01a bis 01c

Abgabe der Hausübung: 20.11.2020 bis 23:50 Uhr

V Vorbereitende Übungen

V1 FopBot ☆ ☆ ☆

Beschreiben Sie kurz in ihren eigenen Worten worum es sich bei FopBot handelt, wie die Welt aufgebaut ist und welche Grundfunktionen jeder Roboter beherrscht.

Für alle Aufgaben auf diesem und allen weiteren Übungsblättern: In der Abschlussklausur werden Sie keine Hilfsmittel zur Verfügung haben. Üben Sie also schon zu Beginn auch ohne Entwicklungsumgebung und nur mit Stift auf einem Blatt Papier zu programmieren. Abschließend können Sie dann ihre vollständige Lösung in die Entwicklungsumgebung übertragen und überprüfen.

V2 Liegen geblieben ☆ ☆ ☆

Betrachten Sie den folgenden Codeausschnitt/führen Sie ihn selbst einmal aus:

```
1 Robot bot = new Robot(0, 2, UP, 0);
2 bot.move();
3 bot.turnLeft();
4 bot.putCoin();
```

In welcher Zeile kommt es zu einem Problem und wieso?

V3 Rechteck ★ ☆ ☆

Schreiben Sie ein Programm, welches zwei Roboter `putbot` und `pickbot` erstellt. Dabei soll `putbot` mit Coins ein Rechteck der Höhe 5 und der Breite 3 zeichnen. Es sollen nur die Seiten des Rechtecks gezeichnet werden, die restlichen innenliegenden Felder des Rechtecks bleiben unberührt. Nachdem das Rechteck gezeichnet wurde, soll `pickbot` alle Coins wieder einsammeln. Überlegen Sie sich, wie Sie das Programm mit nur einer Schleife pro Roboter gestalten können.

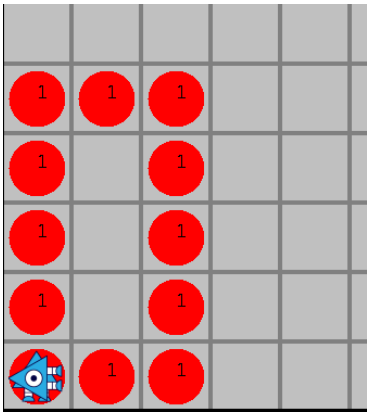

Abbildung 1: Fertiggestelltes Rechteck durch putbot

V4 Bedingungen I ★ ☆ ☆

Betrachten Sie folgenden Codeausschnitt:

```
1 Robot bot1 = new Robot(3, 1, UP, 1);
2 bot1.move();
3 if (bot1.isNextToACoin()) {
4     bot1.pickCoin();
5 } else {
6     bot1.putCoin();
7 }
```

Beschreiben Sie in eigenen Worten, welchem Zweck dieser Codeausschnitt dient. Erweitern Sie außerdem den Code so, dass `bot1` nur einen Coin ablegt, wenn er auch mindestens einen besitzt.

V5 Variablen ★ ☆ ☆

Legen Sie eine Variable `int a` an und setzen Sie ihren Wert auf 127. Jetzt legen Sie eine weitere Variable `int b` an und setzen ihren Wert auf 42. Was gibt nun der Ausdruck `int c = a % b` wieder? Beschreiben Sie in Ihren eigenen Worten, welche Berechnung mit dem `%` Operator durchgeführt wird.

V6 Bedingungen II ★ ☆ ☆

Ihr Kommilitone ist etwas tippfaul und lässt deswegen gerne einmal Klammern weg, um sich Arbeit zu sparen. Er hat in seinem Code eine Variable `int number` angelegt, in der er eine Zahl speichert. Ist diese Zahl kleiner als 0, so möchte er das Vorzeichen der Zahl umdrehen und sie anschließend um 1 erhöhen. Ist die Zahl hingegen größer als 0, so möchte er die Zahl verdoppeln. Dazu schreibt er folgenden Code:

```
if (number < 0) number = -number;
number = number + 1;
else number = number * 2;
```

Kann der Code so ausgeführt werden? Beschreiben Sie den Fehler, den ihr Kommilitone begangen hat.

Nachdem Sie ihren Kommilitonen auf den obigen Fehler hingewiesen haben, überarbeitet er seinen Versuch. Wie sieht es mit folgender Variante aus?

```
if(counter > 0) counter = counter * 2;
if(counter < 0) counter = -counter;
counter = counter + 1;
```

Da müssen Sie wohl selbst ran. Erstellen Sie ein Codestück, um den Sachverhalt korrekt zu implementieren.

V7 Schleifen I ★ ☆ ☆

Schreiben Sie den folgenden Ausdruck mithilfe einer `for`-Schleife:

```
Robot bot = new Robot(0, 0, UP, 1);
int i = 3;
while (i < 9) {
    bot.move();
    i = i + 1;
}
```

V8 Schleifen II ★ ☆ ☆

Ihr klammerfauler Kommilitone hat auch diesmal wieder zugeschlagen und versucht den Code aus vorheriger Aufgabe kürzer zu schreiben. Was sagen Sie dazu?

```
Robot bot = new Robot(0, 0, UP, 1);
int i = 3;
while(counter < 9) bot.move(); i = i + 1;
```

V9 Anzahl an Umdrehungen ★ ★ ☆

Legen Sie eine Variable `int numberOfTurns` an und setzen Sie ihren Wert zu Beginn auf 0. Erstellen Sie dann einen neuen Roboter und platzieren Sie ihn an der Stelle $(8, 2)$. Er schaut dabei nach links und besitzt keine Coins. Lassen Sie den Roboter nun geradewegs auf die Stelle $(0, 2)$ zusteuern und alle Coins auf seinem Weg aufsammeln. Liegen mehrere Coins auf einer Stelle, so soll er alle Coins aufsammeln. Bei jedem Aufsammeln, erhöhen

Sie den Wert von `numberOfTurns` um 1. Hat er am Ende die Stelle $(0, 2)$ erreicht, soll er sich `numberOfTurns`-mal nach links drehen.

V10 Vorsicht Wand! ★ ★ ☆

Gehen Sie in dieser Aufgabe davon aus, dass Sie einen Roboter `wally` an der Position $(0, 0)$ erstellt haben und er nach rechts schaut. An der Position $(x, 0)$ befindet sich eine vertikale Wand die den Weg nach $(x + 1, 0)$ versperrt, die x-Koordinate ist allerdings unbekannt. Schreiben Sie ein kleines Programm, mit dem Sie den Roboter bis vor die Wand laufen lassen, direkt vor der Wand einen Coin ablegen, um dann wieder an die Ausgangsposition $(0, 0)$ zurückzukehren.

Hinweis: Es gibt die Funktion `isFrontClear()`, mit der getestet werden kann, ob sich in der Blickrichtung des Roboters direkt eine Wand befindet.

V11 Codeverständnis ★ ★ ☆

Beschreiben Sie ausführlich, welches Verhalten der nachfolgende Code umsetzt. Bei Fragen zur Funktionalität einzelner Methoden, werfen Sie einen Blick in die entsprechenden Vorlesungsfolien.

```
Robot robot = new Robot(0, 0, RIGHT, 99999);

int counter = 0;
while (robot.getX() < World.getWidth() - 1) {
    robot.move();
    counter = counter + 1;
}

robot.turnLeft();

for(int i = 0; i < counter; i++) {
    if(i % 2 == 0 && robot.hasAnyCoins()) {
        robot.putCoin();
    }
    robot.move();
}

robot.turnLeft();
while (robot.isFrontClear()) {
    robot.move();
}

robot.turnLeft();
while (robot.getX() != 0 || robot.getY() != 0) {
    robot.move();
}

robot.turnOff();
```

V12 Navigator ★ ★ ★

Gegeben seien vier Variablen:

```
int startX        int startY

int destinationX  int destinationY
```

Ihr Roboter befindet sich zu Beginn an der Position (`startX`, `startY`) und schaut in eine beliebige Richtung. Schreiben Sie ein Programm, das ihn von dieser Position auf die Position (`destinationX`, `destinationY`) laufen lässt.

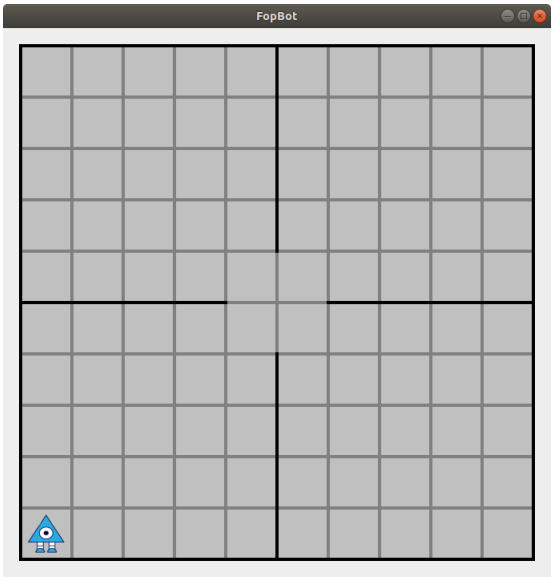

Abbildung 2: Diese Welt ist die Grundlage für das Hausübungsblatt 01. Auf diese Welt werden beide Aufgaben implementiert. Dabei beschreibt der blaue Roboter den Roboter *botProtagonist*, der von Ihnen zu kontrollieren ist.

H Erste Hausübung Gesamt 16 Punkte
Eintauchen in FopBot

Sie haben im nullten Übungsblatt Ihre ersten Schritte mit unserem FopBot erfolgreich gemeistert. Nun steht es Ihnen bevor, sich ausgiebiger mit dem Fopbot zu beschäftigen, und hierfür komplexere Aufgabenstellungen zu lösen.
Für die beiden folgenden Aufgaben sind die Welten, auf denen Sie agieren werden, identisch. Es handelt sich um eine Welt der Größe 10x10, mit zwei horizontalen und vertikalen Wänden, die jeweils zum Mittelpunkt der Welt gerichtet sind, und ein 2x2 großes Feld zur Mitte frei lassen. Der sogenannte Roboter *botProtagonist* ist die Hauptfigur, die Sie kontrollieren werden und startet in beiden Aufgaben bei Koordinaten (0,0). Eine Illustration ist unter Abbildung 2 zu sehen. Die Koordinaten folgen der in der Vorlesung eingeführten Terminologie von Spalten x, Zeilen y und Feldern (x,y).
Sie können, wie Ihnen aus der Vorlesung bekannt ist, den Roboter *botProtagonist* via *botProtagonist.move()* und *botProtagonist.turnLeft()* in der Methode *main()* der Klasse *task1* bewegen. Außerdem stehen Ihnen alle bekannten Aufrufe auf Methoden zur Verfügung, die Sie in den Foliensätzen 01a-01c kennengelernt haben. Weitere Anmerkungen zu den verbindlichen Anforderungen entnehmen Sie dieser Box auf Seite 7. Bei Verletzungen von verbindlichen Anforderungen müssen Sie mit Punktabzug rechnen.
Achten Sie darauf, dass wenn Sie Aufgaben 1 oder 2 bearbeiten möchten, Sie auch die in *Task1.java* dafür vorgesehene Variable *tasknumber* auf 1 oder 2 setzen. Außerhalb des annotierten Blocks, in dem Sie Ihre Lösung implementieren sollen, ist dies die einzige Variable, die Sie verändern dürfen.
Lassen Sie sich nicht beirren, dass die Datentypen des *botProtagonist* oder anderer vorkommender Roboter nicht dem gewohnten Datentyp *Robot* der Vorlesung gleichen. Die Funktionalitäten der Roboter sind identisch, und bei Erweiterungen der Funktionalitäten werden Ihnen diese in den Aufgabenstellungen erläutert.

Verbindliche Anforderungen:

- Sie dürfen lediglich den Roboter *botProtagonist* durch die Welt bewegen. Bewegungen anderer Roboter, die durch die Vorlage gedacht sind und ausgelöst werden, siehe Aufgabe H2, sind natürlich auch zugelassen.
- Stellen Sie sicher, dass Ihre Lösung nur in den davor vorgesehenen Teilen des Codes implementiert wurde. Hierfür sind in der Vorlage Abschnitte mit Kommentaren annotiert, wo Sie Ihren Code einzufügen haben.
- Sie dürfen **NUR** die in der Vorlesung vorkommenden Methoden, Aufrufe auf Methoden und Konzepte nutzen. Ein Beispiel wäre der Aufruf der Methode isFrontClear() auf einem Roboter. Ausnahmen sind den folgenden Punkten zu entnehmen.
- Sie dürfen alle in dieser Hausübung vorgestellten und eingeführten Aufrufe verwenden.
- Außerdem steht es Ihnen frei, folgende Aufrufe zu nutzen, allerdings ist dies freiwillig:

  getX() und *getY()*, wobei diese Aufrufe auf einem Roboter getätigt werden müssen, und die X-Koordinate bzw. Y-Koordinate des Roboters liefern

  World.getWidth() und *World.getHeight()*, wobei diese Aufrufe die Breite und Länge der vorliegenden Welt liefern

  isFacingUp(), *isFacingDown()*, *isFacingLeft()* und *isFacingRight()*, wobei diese Aufrufe auf einem Roboter getätigt werden müssen, und liefern, ob ein Roboter in die bestimmte Richtung schaut
- Die Endposition des Roboters *botProtagonist* nach Bearbeitung einer Aufgabe darf nicht (0,0) betragen.

H1 Navigation und Arithmetische Operationen 6 Punkte

In dieser Aufgabe erhält unser Roboter *botProtagonist* 10 Münzen und unsere Welt wird um weitere 3 Roboter erweitert. Diese 3 Roboter befinden sich in den 3 freien Ecken der Welt, illustriert in Abbildung 3. Ihre Aufgabe ist es nun, diese 10 Münzen auf die neuen Roboter *botPrime*,*botEven* und *botOdd* zu verteilen, wobei Sie die Positionen der Roboter der Abbildung 3 und dessen Beschreibung entnehmen können. Hierbei soll der Roboter *botPrime* 6 Münzen erhalten, der Roboter *botEven* 3 Münzen, und der Roboter *botOdd* 1 Münze. Die Reihenfolge des Ablegens, Aufnehmens und Besuchen der Roboter ist irrelevant. Ihre Aufgabe lautet, den Roboter *botProtagonist* zu den restlichen Robotern zu navigieren, und die Münzen nach den hier definierten Kriterien zu verteilen, bis der Roboter *botProtagonist* keine Münzen mehr besitzt. Mit Verteilen ist gemeint, dass Sie auch dafür sorgen müssen, dass die Roboter die Münzen aufheben. Beachten Sie, es existiert keine “nullte Münze”.

H2 Navigation unter erschwerenden Umständen 10 Punkte

In dieser Aufgabe finden Sie wieder eine ähnliche Situation wie in H1 vor, allerdings dürfte Ihnen die Navigation um einiges schwieriger fallen. Dieses Mal befinden sich Münzen in den drei frei gebliebenen Ecken, jedoch versucht ein Roboter, Ihnen den Weg zu den Münzen zu versperren. Die Ausgangssituation können Sie der Abbildung 4 entnehmen.

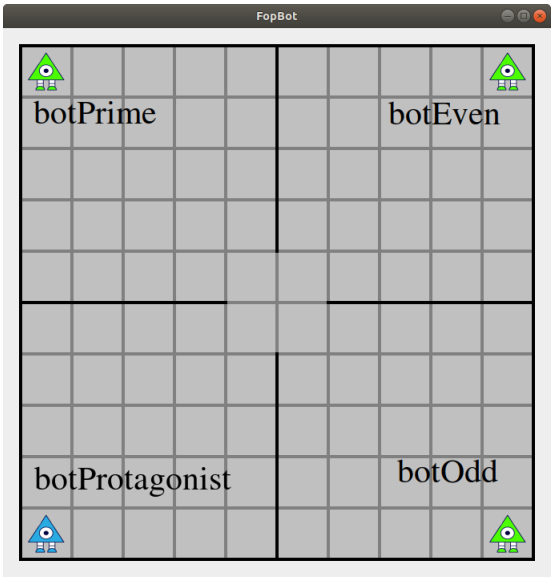

Abbildung 3: Diese Welt ist der Ausgangspunkt der Aufgabe H1. Der Roboter *botProtagonist* befindet sich an Koordinaten (0,0). Die Roboter *botPrime*,*botEven* und *botOdd* befinden sich an den Koordinaten (0,9), (9,9) und (9,0).

Wie der Abbildung zu entnehmen ist, sehen Sie einen zusätzlichen Roboter, der sich in der Mitte der Welt befindet. Diese "Wache" *botGuard* patroulliert im 2x2 Feld des Zentrums der Welt. Genauer, mit jedem einzelnen Schritt des Roboters *botProtagonist* bewegt sich die Wache einen Stelle nach vorn, und dreht sich relativ zur eigenen Blickrichtung nach links. Dies führt dazu, dass sich die Wache im freigelassenen 2x2 Feld der Mitte im Kreis bewegt, und somit die Münzen der Ecken "bewacht".
Ihre Aufgabe besteht nun darin, der Roboter *botProtagonist* zu allen drei Ecken mit Münzen zu navigieren, **OHNE** dabei in Kollision mit der Wache zu geraten, und alle Münzen aufzusammeln. Eine "Kollision" vermeiden Sie, indem Sie garantieren, dass sich zu keinem Zeitpunkt der Roboter *botProtagonist* auf demselben Feld befindet, wie der Roboter *botGuard*. Für jede Art von "Kollisionen" erhalten Sie Punktabzug.
Um diese Aufgabe erfolgreich lösen zu können, stehen Ihnen weitere Funktionalitäten auf dem Roboter *botProtagonist* zur Verfügung. Mithilfe des Aufrufes *isFrontClearOfGuard()* auf einem Roboter können Sie überprüfen, ob sich die Wache direkt vor diesem Roboter oder am Rand der Welt befindet. Falls Sie nur auf die Existenz der Wache prüfen möchten, benutzen Sie die Methode *isFrontClearOfGuardOnly()*.
Des Weiteren können Sie durch Aufruf *waitOneRound()* auf einem Roboter eine Iteration der Wache erhöhen, i.e., die Wache schreitet in der Kreisbewegung voran.

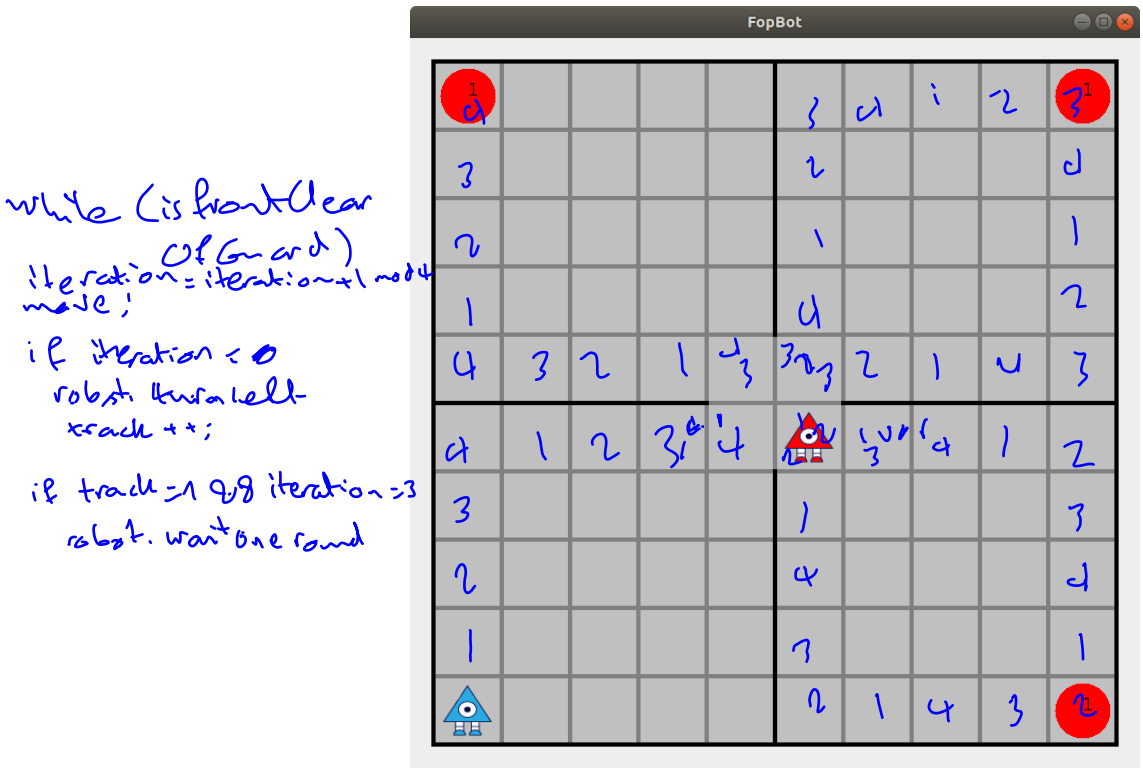

Abbildung 4: Diese Welt ist der Ausgangspunkt der Aufgabe H2. Der Roboter *botProtagonist* befindet sich an Koordinaten (0,0). In den Ecken (0,9), (9,9) und (9,0) befinden sich Münzen. Außerdem befindet sich in der Mitte der Welt ein Roboter, der eine patrouillierende Wache darstellt.