Funktionale und objektorientierte Programmierkonzepte Übungsblatt 01

TECHNISCHE UNIVERSITÄT DARMSTADT

Rückfragen zu diesem Übungsblatt vorzugsweise im moodle-Forum zu diesem Blatt!

Wintersemester 21/22	v1.0
Themen:	Programmieren in Java mit Hilfe von FopBot
Relevante Foliensätze:	01a bis 01c
Abgabe der Hausübung:	05.11.2021 bis 23:50 Uhr

> Screenshots der `World` mit Ihren Robotern darin können Sie unbedenklich mit anderen teilen und in Foren posten, um zu klären, ob Ihr Programm das tut, was es soll. Quelltext und übersetzten Quelltext dürfen Sie selbstverständlich nicht teilen, posten oder sonst wie weitergeben außer an die Ausrichter und Tutoren der FOP 21/22!

Auf diesem und auch weiteren Übungsblättern wird häufig davon die Rede sein, dass `rook` bzw. `bishop` dieses oder jenes „tut“. Solche vermenschlichenden Formulierungen sind natürlich nicht wirklich korrekt, aber einfacher und intuitiver und daher allgemein sehr beliebt. Zum Beispiel besagt die Formulierung „`rook` macht einen Vorwärtsschritt ohne zu prüfen, ob die `World` dadurch verlassen wird“ ganz genau formuliert so etwas wie:

> „Mit dem Verweis namens `rook` von Klasse `Robot` wird im Quelltext die Methode `move` aufgerufen, so dass das Zeilen- oder Spaltenattribut (`getRow()` bzw. `getColumn()`) in dem Objekt, auf das `rook` momentan verweist, je nach momentaner Blickrichtung (`getDirection()`) um 1 erhöht oder verringert wird (bzw. der Prozess wird mit Fehlermeldung abgebrochen, falls das Zeilen- bzw. Spaltenattribut dadurch kleiner als 0 oder größer/gleich der Zeilen- bzw. Spaltenzahl der `World` würde).“

Wir bleiben vielleicht erst einmal lieber bei der inkorrekten, vermenschlichenden Sprache, oder?

Schauen Sie in die Datei `RookAndBishop.java`, die Sie in der Vorlage auf moodle finden. In diese Datei, und zwar in den mit „`// Hier programmieren`“ bezeichneten Bereich, schreiben Sie ihren gesamten Java-Code für dieses Hausübungsblatt. Sie dürfen außerdem weitere Methoden erstellen, um ihren Code zu strukturieren und diese von der main-Methode aus ausrufen.

In dieser Hausübung soll ein Roboter `rook` eine Spur mit Münzen hinter sich herziehen, und ein zweiter Roboter `bishop` soll `rook` verfolgen. Kern Ihrer Lösung ist eine `while`-Schleife, die ab jetzt die *Hauptschleife* genannt wird. In der Hauptschleife legt `rook` in jedem Durchlauf eine Münze ab, und beide Roboter bewegen sich in einer Weise, die unten detailliert erläutert wird. Die Hauptschleife ist zu Ende, sobald eines der beiden folgenden Ereignisse eingetreten ist: (1) `Roboter rook` hat alle Münzen, die er bei seiner Einrichtung bei sich hatte, abgelegt; (2) `rook` und `bishop` sind auf demselben Feld der `World`. Im Fall (1) hat `rook` gewonnen, im Fall (2) `bishop`.

Neu ist bei Hausübung 01 gegenüber 00, dass die Zeilen- und Spaltenzahl der `World` nicht als bestimmte Zahlen festgelegt sind, sondern aus der Datei `fopbot.properties` (im Ordner src/main/resources) eingelesen werden. Die Anweisungen zum Einlesen und Speichern dieser beiden Werte finden Sie aber schon in `Task1.java`, dazu müssen Sie nichts selbst schreiben – Sie verwenden einfach nur die Konstanten `NUMBER_OF_ROWS` und `NUMBER_OF_COLUMNS`. Und wenn Sie eine andere Zeilen- und Spaltenzahl für die `World` haben wollen, müssen Sie gar nichts am Quelltext ändern, sondern einfach nur die beiden Zahlen in der Datei `fopbot.properties` ändern (abspeichern nicht vergessen!) und Ihr Programm einfach laufen lassen. Dann wird Ihr Programm wieder dasselbe tun wie vorher, aber nun mit einer anderen Größe der `World`.

Dieser Grad an Flexibilität wird für uns in Zukunft bei allen möglichen Eingabedaten, die prinzipiell von Programmlauf zu Programmlauf variieren könnten, aber in jedem einzelnen Programmlauf konstant sind, selbstverständlich sein: nicht den Quelltext ändern, nur die Eingabedaten.

H1 Initialisierungen vor der Hauptschleife — 4 Punkte

Vor der Hauptschleife richten Sie einen Verweis `rook` und einen Verweis `bishop` von Klasse `Robot` jeweils mit einem eigenen Roboter-Objekt ein. Initiale Zeile und Spalte werden bei beiden Roboter-Objekten zufällig gewählt. Insbesondere werden beide Roboter nicht außerhalb der `World` platziert.

***Hinweis*:** In Hausübung 00 haben Sie bereits gesehen, wie ein pseudozufälliger `boolean` generiert wird (mit `ThreadLocalRandom.current()`) und analog mit `nextInt()` statt `nextBoolean()` können sie auf diese Weise auch Ganzzahlen generieren. Dabei nimmt die Methode `nextInt()` auch `int`-Parameter die den Ereignisraum begrenzen. Jedes mögliche Ereignis hat dann annähernd die gleiche Chance, sprich es modelliert eine uniforme Verteilung. Das Statement `ThreadLocalRandom.current().nextInt(1, 7)` modelliert somit zum Beispiel den Wurf eines handelsüblichen sechs-seitigen Würfels. [a].

[a] Weitere Information finden Sie hier

***Tests zur eigenen Kontrolle* (0 Punkte):** Initialisieren Sie erst einmal beide Roboter mit den oben bestimmten Koordinaten, Ausrichtung `UP` und 0 als Anzahl der Münzen. Fügen Sie wie in Blatt 00 mit `System.out.println` ein paar Konsolenausgaben ein, mit denen Sie die die Zeilen- und Spaltenzahl der `World`, die Zufallswerte unmittelbar nach ihrer Generierung sowie nach Einrichtung der Roboter mit `getX` und `getY` die Koordinaten der Roboter ausgeben. Kompilieren Sie das Programm so schon einmal und lassen Sie es mehrfach laufen. Prüfen Sie, ob die Konsolenausgaben Ihrer Erwartung entsprechen, und falls nicht, nutzen Sie die Konsolenausgaben zur Fehlersuche. Sobald Sie bei mehreren Programmläufen hintereinander gesehen haben, dass alles stimmt, nehmen Sie die Konsolenausgaben wieder heraus und machen mit dem nächsten Absatz von H1 weiter:

Die Anzahl der Münzen von `rook` wird bei Einrichtung des Roboter-Objektes mit einem Zufallswert zwischen 12 und 20 initial festgelegt. Dazu richten Sie eine `int`-Variable `numberOfCoins` ein (Initialisierung nicht nötig), der ihr einen Zufallswert im Intervall $12 \ldots 20$ zuweist. Bei der Einrichtung des Objektes mit `new`, auf das `rook` verweisen soll, setzen Sie dann den Namen „`numberOfCoins`“ anstelle einer festen Anzahl von Münzen als vierten Parameter ein.

***Tests zur eigenen Kontrolle* (0 Punkte):** Fügen Sie eine Konsolenausgabe in die `while`-Schleife ein, mit der Sie sich die generierten Zufallszahlen ansehen können, kompilieren Sie das Programm und lassen Sie es mehrfach laufen. Nehmen Sie die eingefügte Konsolenausgabe wieder heraus, sobald Sie bei mehreren Programmläufen hintereinander gesehen haben, dass alles stimmt.

Jede der vier Himmelsrichtungen ist mit derselben Wahrscheinlichkeit die initiale Richtung von `rook` bzw. `bishop`. Dazu erzeugen Sie für `rook` bzw. `bishop` jeweils eine Zufallszahl $0 \ldots 3$. Dann richten Sie eine Variable `direction` von Typ `Direction` ein und setzen ihren Wert auf `UP` im Fall 0, `RIGHT` im Fall 1, `DOWN` im Fall 2 und `LEFT` im Fall 3. Bei der Einrichtung des Objektes mit `new`, auf das `rook` verweisen soll, setzen Sie dann den Namen „`direction`“ anstelle einer festen Richtung als dritten Parameter ein.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie auch hier analog zu den Tests oben mit Konsolenausgaben bei mehreren Programmdurchläufen, ob alles stimmt, bevor Sie die Konsolenausgaben wieder herausnehmen und mit H2 weitermachen.

H2 Fehlermeldungen besser verstehen 0 Punkte

Bauen Sie folgende Fehler jeweils nacheinander ein und versuchen Sie, die Fehlermeldungen zu verstehen. Vergessen Sie nicht, vor Einbau des nächsten Fehlers bzw. vor Ihrer Weiterarbeit an den Hausübungen jeden Fehler wieder rückgängig zu machen!

Als erstes fügen Sie eine Zeile nach der Einrichtung der beiden Roboter-Objekte ein, in der Sie `NUMBER_OF_ROWS` einen Wert zuweisen. Lassen Sie den so geänderten Quelltext übersetzen. Der Compiler sollte eine Fehlermeldung ausgeben und daher Ihren Quelltext ***nicht*** in ausführbaren Java Byte Code übersetzen. Sie sollten Ihr Programm also nicht laufen lassen können. Macht die Fehlermeldung für Sie Sinn? Die Fehlermeldung sollte auch auftreten, wenn Sie `NUMBER_OF_ROWS` genau den Wert zuweisen, den `NUMBER_OF_ROWS` durch die Initialisierung schon hat. Macht das für Sie Sinn? Was passiert hingegen, wenn Sie `final` in der Definition von `NUMBER_OF_ROWS` entfernen und dann übersetzen lassen?

Nach Rückgängigmachung des Fehlers (also Zuweisung an `NUMBER_OF_ROWS` wieder löschen und `final` wieder einfügen) benennen Sie als nächsten Fehler die Datei `fopbot.properties` um. Jetzt sollte der Compiler Ihren Quelltext ohne Fehlermeldungen übersetzen, aber beim Laufenlassen sollte es einem vorzeitigen Abbruch mit einer Fehlermeldung geben, die Sie in `Task1.java` finden können. Zum Aktualisieren der Datei `fopbot.properties` müssen sie mithilfe von `gradle` erneut die Applikation bauen (`gradle run` bzw. `grade build`).

Die Fehlermeldungen enthalten wertvolle Hinweise auf die Ursache des Problems, auf der einen Seite im Bezug auf die Art des Fehlers und auf der anderen Seite an welcher Stelle das Problem aufgetreten ist. In diesem Fall weist sie (unter Anderem) auf Zeile `RookAndBishop.java:21` hin, in der eigentlich `NUMBER_OF_ROWS` eingelesen werden sollte. Damit ist klar, dass das Problem beim Einlesen von `NUMBER_OF_ROWS` passiert ist. Hier gibt die Fehlermeldung außerdem einen Hinweis über die Art des Fehlers, nämlich dass die Property-Datei (also `fopbot.properties`) nicht gefunden werden konnte.

Eine solche Struktur, die präzise detaillierte Hinweise über den Fehler liefert, wurde in dieser Hausübung bewusst integriert und kommt leider nicht von selbst. Wie Sie solche Fehlerbehandlungen mit `try` und `catch` selbst schreiben können, werden Sie allerdings erst in Kapitel 05 und in einer späteren Hausübung sehen. Hier reicht es, wenn Sie das schon einmal gesehen haben und mit dem Fehlerabbruch des Prozesses prinzipiell in Verbindung bringen.

Geben Sie der Datei wieder ihren eigentlichen Namen `fopbot.properties`, sodass das Programm wieder korrekt laufen kann.

Nun nehmen Sie eine der beiden Zeilen aus `fopbot.properties` heraus (abspeichern nicht vergessen!). Lassen Sie Ihr Programm nochmals laufen (eine neuerliche Übersetzung ist dafür nicht notwendig). Jetzt sollte das Programm wieder mit einer Fehlermeldung abbrechen, aber mit einer anderen. Suchen Sie die Fehlermeldung wieder im Quelltext von `Task1.java`; erkennen Sie ein Muster?

Wenn Sie möchten, probieren Sie, auch die anderen Fehlermeldungen in der Vorlage bewusst auzulösen, indem Sie die `fopbot.properties` Datei anpassen. Machen Sie diese Änderungen im Anschluss wieder rückgängig.

Nun fügen Sie die Zahl wieder in `fopbot.properties` ein, aber negativ, also mit einem Minuszeichen unmittelbar davor. Lassen Sie Ihr Programm wieder laufen. Nun sollte der Prozess vorzeitig abgebrochen werden mit einer Fehlermeldung, die in Klasse `KarelWorld` (also außerhalb der Vorlage) entsteht, sonst aber den anderen Fehlermeldungen ähnelt.

> Beachten Sie generell, dass es einfacher ist, Kompilierfehler zu verfolgen und zu beheben als Laufzeitfehler. Wie wir durch das Herumspielen mit Namen und Inhalt der Datei `fopbot.properties` gesehen haben, müssen Laufzeitfehler nicht einmal bei jedem Lauf auftreten, was die Rückverfolgung noch schwieriger macht (siehe Kapitel 12, Abschnitt zu Fehlern auf der logischen und der spezifikatorischen Ebene, also Folien 81-90, für eindrucksvolle Beispiele aus der Praxis).

Wie gesagt: nicht vergessen, alle Fehler wieder rückgängig zu machen!

H3 Die Hauptschleife — 10 Punkte

Die Fortsetzungsbedingung der Hauptschleife soll der boolesche Wert `true` sein, das heißt, die Hauptschleife wird durch eine `break`-Anweisung im Schleifenrumpf beendet (vgl. Kapitel 01b, Folien 48-62).

Verbindliche Anforderung: Die Methode `isFrontClear` der Klasse `Robot` darf in dieser Hausübung nicht verwendet werden, sprich z.B. `rook.isFrontClear();` ist nicht erlaubt. Analog ist die Methode `isNextToARobot` ebenfalls von der Klasse `Robot` verboten.

H3.1 Bewegung von `rook` — 4 Punkte

In jedem Durchlauf durch die Hauptschleife legt `rook` erst einmal wie schon erwähnt eine Münze ab. Als nächstes prüft `rook`, ob sein nächster Vorwärtsschritt aus der `World` hinausführen würde. Das ist natürlich in vier Fällen der Fall: Einer der vier Fälle ist, dass die momentane Richtung Osten ist und die momentane Spaltennummer die höchste in der `World` ist. Die anderen drei Fälle für die anderen drei Himmelsrichtungen sind analog.

Falls der nächste Vorwärtsschritt von `rook` nicht aus der `World` hinausführt, geht `rook` einen Schritt vorwärts und dreht sich dann mit jeweils 25% Wahrscheinlichkeit nach links oder nach rechts (also mit 50% Wahrscheinlichkeit keine Drehung). Andernfalls dreht `rook` sich mit einer Wahrscheinlichkeit von 50% um 180 Grad, und jede der beiden Drehungen um 90 Grad hat eine Wahrscheinlichkeit von 25% (also auf jeden Fall eine Drehung).

***Hinweis*:** Um 50% bzw. 25% Wahrscheinlichkeit für ein Ereignis hervorzurufen können Sie analog zu der Erstellung der Roboter einen Pseudozufallszahlengeneratoren verwenden. Stellen Sie sich die Operation wie den Wurf eines Würfels vor. Bei einem handelsüblichen 6-seitigen Würfel gibt es zum Beispiel in 50% der Fälle eine Zahl kleiner als 4. Wie viele Seiten sollte Ihr Würfel hier haben? Achtung: In der Informatik wird häufig ab 0 gezählt.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie auch hier mit Konsolenausgaben bei mehreren Programmdurchläufen, ob die Bewegungen von `rook` stimmen, bevor Sie mit H3.2 weitermachen. Da die bisher erstellte `while`-Schleife noch eine Endlosschleife ist, fügen Sie schon einmal die zu Beginn von H3 erwähnte `break`-Anweisung ein, aber erst einmal mit einer vorläufigen Abbruchbedingung: dass `rook` keine Münzen mehr hat. Diese Anweisung lassen Sie erst einmal so, bis Sie sie in H3.3 in ihre endgültige Form bringen.

H3.2 Bewegung von `bishop` — 3 Punkte

Der Roboter `bishop` hingegen läuft in jedem Durchlauf durch die Hauptschleife diagonal bis zur nächsten Münze oder – falls er über kein Feld mit Münze kommt – bis zum Rand der `World`. Dabei bedeutet diagonale Bewegung im FopBot zwei Schritte, z.B. für Ausrichtung `LEFT` ein Schritt nach links und ein weiterer nach oben. Dafür implementieren Sie nach dem Quelltext für H3.1, aber noch innerhalb des Rumpfes der Hauptschleife eine weitere `while`-Schleife, die ab jetzt dfie *`bishop`-Schleife* genannt wird.

Wie in 01c (Folien 37 und 54), richten Sie eine boolesche Variable `notFinished` ein, die Sie vor der `bishop`-Schleife (aber im Rumpf der Hauptschleife) mit `true` initialisieren und in der `bishop`-Schleife auf `false` setzen, sobald `bishop` auf einem Feld mit mindestens einer Münze oder am Rand der `World` angekommen ist.

Im Detail sieht ein Durchlauf durch die `bishop`-Schleife so aus: Zuerst wird wie bei `rook` geprüft, ob ein Vorwärtsschritt aus der `World` hinausgehen würde. Falls ja, drehen Sie `bishop` um 90 Grad nach links und setzen `notFinished` auf `false` (denken Sie hier und im Folgenden an die geschweiften Klammern um die Anweisungen im `if`-Teil!).

Andernfalls soll `bishop` einen Schritt vorwärts machen, sich um 90 Grad nach links drehen und dann ein weiteres Mal prüfen, ob ein Vorwärtsschritt aus der `World` hinausgehen würde. Falls ja, dreht sich `bishop` um 180 Grad nach links und Sie setzen `notFinished` auf `false`. Andernfalls soll `bishop` einen Schritt vorwärtsgehen und sich dann um 90 Grad nach rechts drehen.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie auch hier mit Konsolenausgaben bei mehreren Programmdurchläufen, ob die Bewegungen von `bishop` stimmen, bevor Sie mit dem nächsten Absatz weitermachen.

Danach sollen in jedem Durchlauf durch die `bishop`-Schleife (das heißt nach je 2 Schritten, bzw. 1 Schritt wenn der zweite nicht möglich war) noch zwei Dinge passieren: (i) Falls `bishop` und `rook` auf demselben Feld stehen, setzen Sie `notFinished` auf `false`. (ii) Falls auf dem Feld, auf dem `bishop` steht, in diesem Moment mindestens eine Münze liegt, hebt `bishop` genau eine davon auf, und auch in diesem Fall setzen Sie `notFinished` auf `false`.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie nun, ob Ihr Programm neben allem bisher Getestetem das tut, was im letzten Absatz steht.

***Hinweis für Experimentierfreudige und Fortgeschrittene*:** Insgesamt dreimal wird in H3.1 und H3.2 also auf exakt identische Art geprüft, ob ein Roboter die `World` bei einem Vorwärtsschritt verlassen würde. Falls Sie es sich zutrauen, dürfen Sie diese Logik auch in eine separate boolesche Methode mit aussagekräftigem („sprechendem“) Namen auslagern und diese Methode dreimal aufrufen.

H3.3 Beendigung der Hauptschleife — 3 Punkte

***Erinnerung*:** Die Hauptschleife soll beendet werden, sobald eines der beiden folgenden Ereignisse eingetreten ist: (1) `Roboter rook` hat alle Münzen, die er bei seiner Einrichtung bei sich hatte, abgelegt; (2) `rook` und `bishop` sind auf demselben Feld der `World`. Im Fall (1) hat `rook` gewonnen, im Fall (2) `bishop`.

Am Ende der Hauptschleife ist also noch zu prüfen, ob eine dieser beiden Abbruchbedingungen erfüllt ist. In jedem der beiden Fälle wird eine entsprechende Botschaft auf die Konsole geschrieben: „Der Turm hat gewonnen!“ bzw. „Der Läufer hat gewonnen!“, und die Hauptschleife wird in jedem der beiden Fälle unmittelbar nach dieser Konsolenausgabe durch eine `break`-Anweisung beendet. Im seltenen Fall, dass beide Bedingungen erfüllt sind soll ebenfalls „Der Turm hat gewonnen!“ ausgegeben werden.

***Verbindliche Anforderung*:** Für den Test, ob `rook` und `bishop` auf demselben Feld sind, verwenden Sie eine einzige `if`-Anweisung (auch keine ineinander geschachtelten `if`-Anweisungen) und verknüpfen die Bedingungen darin mit Operator „`&&`“ (siehe Kapitel 01b, Folien 164-171).

***Hinweis*:** Je nach Tastatur-Layout und Encoding-Einstellungen kann es möglichweise schwierig sein den Umlaut in „Der Läufer hat gewonnen“ einzugeben. Bei Problemen suchen Sie im Internet nach der hexadezimalen Unicode-Nummer für „ä“ und orientieren sich dann an Kapitel 01b, Folien 184-195.

***Tests zur eigenen Kontrolle* (0 Punkte):** Testen Sie nun abschließend, ob Ihr Programm alle Anforderungen dieser Hausübung erfüllt (abgesehen von H3.3 haben Sie ja alles schon schrittweise getestet, also nicht mehr viel zu testen an dieser Stelle).