Funktionale und objektorientierte Programmierkonzepte

Alexander Mainka und Lukas Röhrig
(Gesamtleitung: Prof. Karsten Weihe)

Wintersemester 19/20
v1.0

Übungsblatt 2

Themen:	FopBot, Referenzsemantik, Klassen, Arrays
Relevante Folien:	01a bis 01e
Abgabe der Hausübung:	08.11.2019 bis 23:55 Uhr

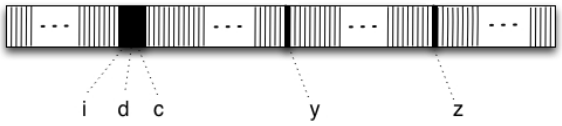
V Vorbereitende Übungen

V1 Referenzen

☆ ☆ ☆

Geben Sie in eigenen Worten wieder, was man unter einer Referenz versteht.

Betrachten Sie außerdem folgendes Schaubild und den Codeausschnitt aus der Vorlesung:




```java
public class X{
 int i;
 double d;
 char c;
 ...
}
```

Zeichnen Sie die Referenzpfeile nach den folgenden Aufrufen ein (ergänzen Sie auch die neuen Reservierungen des Speicherplatzes, wenn nötig):

```java
X y = new X();
X z = y;
y = new X();
```

V2 Zuweisen und Kopieren

☆ ☆ ☆

Erläutern Sie in Ihren eigenen Worten den Unterschied zwischen Zuweisen und Kopieren. In welchen Fällen sind beide Aktionen synonym zu betrachten?

Wie können Sie eine Zuweisung beziehungsweise eine Kopie in Java umsetzen? Nennen Sie jeweils ein Beispiel.

V3 Arrays ★ ☆ ☆

Welche Aussagen zu einem gegebenen Array `a` sind wahr?

(1) Alle Einträge des Arrays müssen vom selben Typ sein.

(2) Ein Array hat keine feste Größe und es können beliebig viele neue Einträge einem Array hinzugefügt werden.

(3) Um die Anfangsadresse einer Komponente an Index i zu bekommen, wird i-mal die Größe einer Komponente auf die Anfangsadresse von `a` addiert.

(4) Außer den eigentlichen Komponenten des Arrays enthält das Arrayobjekt nichts weiteres.

(5) Ein Array kann nur primitive Datentypen wie zum Beispiel `int`, `char` oder `double` speichern. Somit ist es insbesondere nicht möglich, Roboterobjekte in einem Array zu speichern.

Schreiben Sie die nötigen Codezeilen (auf Papier), um ein Array `a` der Größe 42 vom Typ `int` anzulegen. Füllen Sie danach das Array mithilfe einer Schleife, sodass an der Stelle `a[i]` der Wert `2i+1` steht. Nutzen Sie dabei zuerst eine while- und danach eine for-Schleife.

V4 Wettrennen ★ ☆ ☆

Sie haben einen schnellen Roboter `rabbit` erstellt und wollen ihm nun noch ein langsames Gegenstück `turtle` bauen. Beide starten an einem gemeinsamen Punkt, schauen in die gleiche Richtung und besitzen die gleiche Anzahl an Coins. Sie wollen nun schauen, wer in 10 Runden mehr Strecke zurücklegen kann. Jeder der beiden Kontrahenten kommt pro Runde genau einen Schritt voran, der schnelle `rabbit` erhält jedoch in jeder zweiten Runde einen Extraschritt. Betrachten Sie den folgenden Codeausschnitt, der die Situation implementieren möchte:

```
1  Robot rabbit = new Robot(0,0,RIGHT,0);
2  Robot turtle = rabbit;
3
4  for(int i = 0; i < 10; i++){
5
6   if(i / 2 == 0){
7    rabbit.move();
8   }
9
10  rabbit.move();
11  turtle.move();
12 }
```

Führen Sie den Code einmal selbst in Eclipse aus und schauen Sie was passiert! Beheben Sie danach **alle** vorhandenen Fehler in der Implementierung, um die oben beschriebene Situation exakt umzusetzen.

V5 Coins im Laufen ablegen ★ ☆ ☆

In dieser Aufgabe sollen Sie eine erste eigene Methode implementieren. Als Orientierung betrachten Sie die Methode `public void move(int numberOfSteps)` der Klasse `FastRobot` aus der Vorlesung. Schreiben Sie nun eine neue Methode

```
public void coinMove(int numberOfSteps)
```

für den `SymmTurner`-Roboter, den Sie in der Vorlesung kennengelernt haben. Diese soll `numberOfSteps` Schritte nach vorne gehen und dabei jedes Mal einen Coin ablegen. Sollte die geforderte Anzahl an Schritten größer sein als die Anzahl an Coins soll der Roboter einfach stehen bleiben und sich ausschalten (dafür gibt es bereits die Methode `public void turnOff()`). Sollten mehr Coins vorhanden sein als geforderte Schritte, soll er an seiner finalen Position alle verbleibenden Coins ablegen.

V6 Richtungsdreher ★ ☆ ☆

In vielen Aufgaben reichen uns die eingeschränkten Methoden eines Roboters der FopBot-Werke nicht. Daher definieren wir uns neue Roboter, welche die technischen Anforderungen erfüllen. In den Foliensätzen zu FopBot haben Sie bereits Beispiele wie den `SymmTurner` Roboter dazu gesehen. In dieser Aufgabe sollen Sie eine neue Roboterklasse definieren, welche sich in alle beliebigen Richtungen drehen kann. Dafür ist folgendes Grundgerüst gegeben:

```
1  public class DirectionTurner extends Robot{
2
3   public DirectionTurner (int x, int y,
4                           Direction direction, int
                               numberOfCoins){
5          super(x,y,direction,numberOfCoins);
6          }
7
8      ..........
9
10 }
```

An der Stelle ..... schreiben wir die neuen Funktionalitäten des Roboters mithilfe von Methoden (vergleiche vorherige Aufgabe). Ergänzen Sie vier neue `public void`-Methoden namens `turnUp()`, `turnDown()`, `turnLeft()` und `turnRight()`, damit sich der Roboter gezielt in alle vier Richtungen drehen kann.

V7 CoinMover ★ ☆ ☆

Legen Sie eine neue Klasse `CoinMover` an, die von `Robot` erbt. Die `move()`-Methode überladen Sie mit den Anweisungen, die in V5 die Methode `coinMove` definiert haben. Legen Sie zusätzlich eine neue Funktion `putAllCoins()` an, welche alle Coins des Roboters ablegt. Nutzen Sie diese Methode, um die `move()`-Methode zu implementieren.

V8 Roboter miteinander vergleichen ★ ☆ ☆

Schreiben Sie eine Methode `int robotsEqual(Robot a, Robot b)`. Diese bekommt zwei Roboter übergeben und soll 2 zurückgeben, wenn die Attribute `x`, `y`, `direction` und `numberOfCoins` bei beiden Robotern die gleichen Werte haben. 1 soll zurückgegeben werden wenn sich beide Roboter nur auf demselben Feld befinden, andernfalls gibt die Methode 0 zurück.

V9 Primitive Datentypen ★ ★ ☆

Schreiben Sie eine Methode `char smallestPDT(long n)`. Diese bekommt eine ganze Zahl übergeben und soll den primitiven Datentyp zurückgeben, der den wenigsten Speicherplatz verbraucht, aber immer noch die übergebene Zahl speichern kann. Geben sie `'l'` für den Datentyp `long` zurück, `'i'` für `integer` usw.

Schreiben Sie nun eine Methode `char[] smallestPDTs(long[] a)`. Diese bekommt ein Array von ganzen Zahlen übergeben und soll ein Array zurückgeben, dass die Methode `smallestPDT(long n)`, in der gleichen Reihenfolge wie in `a`, auf jede Zahl in `a` anwendet.

V10 TeamRobot ★ ★ ★

In dieser Aufgabe sollen Sie ihre erste eigene Roboterklasse von Grund auf implementieren. Erstellen Sie dazu eine neue Klasse `TeamRobot`, die die Klasse `Robot` erweitert, also von ihr erbt. Der Konstruktor der Klasse `TeamRobot` übernimmt die Parameter des Konstruktors der Oberklasse `Robot` und besitzt zusätzlich die Parameter `int left` und `int right`. Der Parameter `int left` gibt an, wie viele zusätzliche Roboter beim Aufruf des Konstruktors links neben des `TeamRobot`s platziert werden. Der Parameter `int right` ist analog, für die Roboter rechts. Der `TeamRobot`, sowie die Roboter links und rechts von ihm bilden ein Team. Die zusätzlichen Roboter werden vom `TeamRobot` im Konstruktor erzeugt. Bekommt der `TeamRobot` einen Befehl, so soll dieser von allen Robotern im Team ausgeführt werden. Die zusätzlichen Roboter selbst sind dabei nicht ansprechbar, dass heißt auf ihnen können keine Methoden aufgerufen werden. Überlegen Sie sich, wie Sie die Roboter des Teams in der `TeamRobot`-Klasse speichern können und wie Sie die Befehle die ein `TeamRobot` erhält, an alle Roboter im Team weiterreichen können. Die Befehle meinen hier die Methoden: `move()`, `turnLeft()`, `pickCoin()` und `putCoin()`.

Beispiel: Beim Erstellen eines `TeamRobot`s mit den Parametern `right = 1` und `left = 2` an der Position $(4, 4)$, werden zusätzlich 3 Roboter erstellt, die dem Team angehören, nämlich an den Position $(2, 4)$, $(3, 4)$ (links) und $(5, 4)$ (rechts).

H Zweite Hausübung — Gesamt 25 Punkte
Schach

Schach ist ein strategisches Brettspiel, bei dem zwei Spieler abwechselnd Spielsteine (die Schachfiguren) auf einem Spielbrett (dem Schachbrett) bewegen. In dieser Hausübung wollen wir dieses Brettspiel mit dem FopBot-Framework implementieren. Eine GUI, die es Ihnen erlaubt, das eigentliche Spiel zu spielen, ist bereits in der Vorlage enthalten. Ihre primäre Aufgabe ist es deshalb, die Zugregeln der unterschiedlichen Schachfiguren zu implementieren.

Aufbau der Vorlage und Zugregeln

Die Schachfiguren dürfen, abhängig von ihrem Typ, nur nach bestimmten Regeln gezogen werden. Für jede Schachfigur existiert eine eigene Klasse im package `chesspieces`. Alle Schachfiguren sind von der Klasse `ChessPiece` abgeleitet, diese ist wiederum von der ihnen bekannten `Robot`-Klasse des FopBot-Frameworks abgeleitet. Die `ChessPiece`-Klasse hat ein zusätzliches Attribut color vom vorgegebenen Enum-Typ `ChessColor`. Ein Attribut oder eine Variable vom Typ `ChessColor` kann nur die Werte `ChessColor.WHITE` oder `ChessColor.BLACK` annehmen. Mit dem `color`-Attribut der Klasse `ChessPiece` bestimmen wir also die Farbe der jeweiligen Spielfigur. Über den zusätzlichen Konstruktor `public ChessPiece(int x, int y, ChessColor color)` wird die Farbe einer jeden Spielfigur festgelegt. Über die Methode `public ChessColor getColor()` können Sie die Farbe einer jeden Spielfigur abfragen. Jede Spielfigur hat zusätzlich zwei von Ihnen zu implementierende Methoden `public Point[] getMoveFields()` sowie `public Point[] getAttackFields()`. Die beiden Methoden sind für die Realisierung der Zugregeln verantwortlich. Konkret gibt die Methode `getMoveFields()` ein `Point`-Array zurück, das alle Felder enthält, auf die sich die jeweilige Spielfigur, von ihrer momentanten Position aus, bewegen kann, ohne eine gegnerische Spielfigur zu schlagen. Die Methode `getAttackFields()` hingegen gibt ein `Point`-Array zurück, dass alle Felder enthält, auf die sich die jeweilige Spielfigur, von ihrer momentanten Position aus, bewegen kann, in dessen Folge eine gegnerische Spielfigur geschlagen wird. Kann sich die Spielfigur auf ihrer momentanten Position nicht fortbewegen oder kann sie keine gegnerische Figur schlagen, so geben die jeweiligen Methoden `null` zurück. Die Klasse `Point` der Java-Standardbibliothek hat zwei `public` Attribute `x` und `y` sowie einen Konstruktor `public Point(x, y)` mit dem Sie einen neuen Punkt erstellen können.

Wir werden nicht alle Zugregeln und Eigenheiten implementieren, bspw. *en passant* beim Bauern, die *Rochade* mit Turm und König oder die Umwandlung eines Bauerns in eine andere Spielfigur beim Betreten der gegnerischen Grundreihe. Implementieren Sie also nur die von uns aufgeführten Regeln.

Um entsprechende Spielfelder des Schachbretts (auf das Vorhandensein von Spielfiguren) zu überprüfen, nutzen Sie die Klassenmethode `ChessBoard.getAllChessPieces()`. Diese Methode gibt ihnen alle auf dem Spielfeld befindlichen Schachfiguren in einem Array zurück. Das Array hat immer die Länge 32. Die Reihenfolge in der die Spielfiguren im Array auftreten wird bei jedem Methodenaufruf zufällig gewählt. Spielfiguren, die

geschlagen wurden, sind auch im Array vorhanden, allerdings wurden sie über die Methode `turnOff()` ausgeschalten, somit gibt die Methode `isTurnedOff()` dieser Spielfiguren `true` zurück.

Das Spiel wird gestartet, wenn Sie die Klasse `ChessBoard` im Package `main` ausführen. Über dem Schachbrett der GUI können Sie ablesen, welcher Spieler an der Reihe ist. Klicken Sie auf eine Spielfigur, um sie auszuwählen. Wird eine nicht gegnerische Spielfigur ausgewählt, so werden die möglichen Spielfelder, auf die sich die Spielfigur bewegen kann, angezeigt. In Grün werden diejenigen Felder angezeigt, auf die sich die Spielfigur bewegen kann, ohne eine gegnerische Figur zu schlagen (Methode: `getMoveFields()`); In Rot eben diejenigen Felder, auf der eine gegnerische Figur geschlagen werden kann (Methode: `getAttackFields()`). Gelb zeigt die momentan ausgewählte Figur an. Durch Klicken auf ein grünes oder rotes Feld wird der jeweilige Spielzug ausgeführt. Nachdem Sie die Methoden einer jeweiligen Klasse implementiert haben, können sie ihre Implementationen ganz leicht über einen Klick auf die jeweilige Figur testen. Abbildung 1 zeigt die Spielumgebung[1].

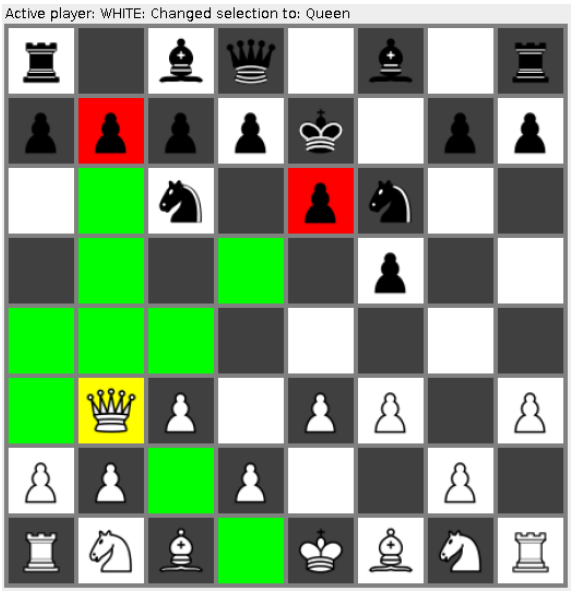


Abbildung 1: Spielumgebung Schach

[1] Alle Spielfiguren stammen von Colin M.L. Burnett (User:Cburnett auf Wikimedia). Spielfiguren entnommen aus folgendem englischsprachigen Wikipedia-Eintrag: Chess piece (Version: 09:25, 11 May 2019)

H1 Bauer — 5 Punkte

Implementieren Sie nun die beiden Methoden `getMoveFields()` und `getAttackFields()` der Klasse `Pawn` (zu dt. Bauer). Beachten Sie dabei die folgenden Regeln (entnommen aus[2]):

- Der Bauer kann einen Schritt nach vorne ziehen, wenn das Zielfeld leer ist.
- Wurde der Bauer noch nicht gezogen und befindet sich somit noch in der Ausgangsstellung, kann er wahlweise auch zwei Schritte vorrücken, sofern das Feld vor ihm und das Zielfeld leer sind.
- Der Bauer schlägt vorwärts diagonal ein Feld weit. Ist ein diagonal vor ihm liegendes Feld jedoch leer, kann er nicht darauf ziehen.

H2 Turm — 4 Punkte

Implementieren Sie die beiden Methoden `getMoveFields()` und `getAttackFields()` der Klasse `Rook` (zu dt. Turm). Beachten Sie dabei die folgenden Regeln (entnommen aus [2].):

- Ein Turm darf auf Linien und Reihen, also horizontal und vertikal, beliebig weit ziehen, ohne jedoch über andere Figuren zu springen.

H3 Läufer — 4 Punkte

Implementieren Sie die beiden Methoden `getMoveFields()` und `getAttackFields()` der Klasse `Bishop` (zu dt. Läufer). Beachten Sie dabei die folgenden Regeln (entnommen aus [2]):

- Läufer ziehen in diagonaler Richtung beliebig weit über das Brett. Über andere Figuren hinweg dürfen auch sie nicht ziehen.

H4 Dame — 4 Punkte

Implementieren Sie die beiden Methoden `getMoveFields()` und `getAttackFields()` der Klasse `Queen` (zu dt. Dame). Beachten Sie dabei die folgenden Regeln (entnommen aus [2]):

- Die Dame darf in horizontaler, vertikaler und diagonaler Richtung beliebig weit ziehen, ohne jedoch über andere Figuren zu springen. Sie vereint somit die Zugmöglichkeiten eines Turms und eines Läufers in sich.

[2] Regeln der einzelnen Spielfiguren entnommen aus folgendem Wikipedia-Beitrag: Schach (Version: 00:27, 14. Jun. 2019)

H5 König 4 Punkte

Implementieren Sie die beiden Methoden `getMoveFields()` und `getAttackFields()` der Klasse `King` (zu dt. König). Beachten Sie dabei die folgenden Regeln (entnommen aus [2]):

- Der König kann horizontal, vertikal oder diagonal auf das unmittelbar angrenzende Feld ziehen (auch wenn das Feld bedroht ist).

H6 Springer 4 Punkte

Implementieren Sie die beiden Methoden `getMoveFields()` und `getAttackFields()` der Klasse `Knight` (zu dt. Springer). Beachten Sie dabei die folgenden Regeln (entnommen aus [2]):

- Der Springer darf auf eines der Felder ziehen, die seinem Standfeld am nächsten, aber nicht auf gleicher Reihe, Linie oder Diagonale mit diesem liegen. Das bedeutet: Ein Springerzug kann formal auch als Hintereinanderausführung eines einschrittigen Turmzuges und eines einschrittigen Läuferzuges in dieselbe Richtung angesehen werden, wobei das Zwischenfeld nicht unbesetzt zu sein braucht.