Universität der Bundeswehr München
Fakultät für Informatik – Institut für Softwaretechnologie
Prof. Dr.-Ing Mark Minas

Klausur
Objektorientierte Programmierung

13.07.2020

Folgende Anweisungen sind zu beachten:

- Diese Klausur umfasst insgesamt **9** Seiten. Prüfen Sie, ob Ihre Unterlagen vollständig und alle Seiten einwandfrei zu lesen sind.
- Schreiben Sie in alle dafür vorgesehenen, umrahmten Felder Ihren Namen und Ihre Matrikelnummer. Aufgabenblätter ohne diese Angaben werden in der Korrektur nicht berücksichtigt.
- Verwenden Sie zum Lösen der Aufgaben entweder einen Füllfederhalter oder einen Kugelschreiber in blauer oder schwarzer Farbe. Verwenden Sie keine anderen Farben. Verwenden Sie insbesondere weder Bleistift noch Rotstift (auch nicht für Streichungen). Die Verwendung von Mitteln, welche die Dokumentenechtheit verletzen, z.B. Tipp-Ex oder Tintenkiller, ist ebenfalls untersagt. Bei Nichtbeachtung werden die entsprechenden Lösungen nicht gewertet.
- Pro Aufgabe kann nur eine Lösung gewertet werden. Ungültige Lösungen sind erkennbar **durchzustreichen**.

Kenntnis genommen

_____

(Name in **Blockschrift**, Matrikelnummer, Unterschrift)

Viel Erfolg!

	Testate	Aufg. 1	Aufg. 2	Aufg. 3	Aufg. 4	Summe
max. Punkte:	24	15	11	12	7	69
Erzielte Punkte:						

Name:	Matrikelnummer:

Aufgabe 1 (Turtle-Grafik-Programme, 15 Punkte)

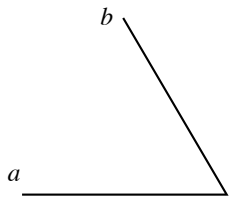
Bei der Turtle-Grafik lässt sich programmgesteuert eine virtuelle Schildkröte über das zweidimensionale Koordinatensystem bewegen. Um bei Bedarf eine Linie hinter sich herzuziehen, führt die Schildkröte einen Stift mit, den man absenken oder anheben kann und womit sich eine Zeichnung erstellen lässt. In unserem Fall bestehen solche Turtle-Grafik-Programme aus den folgenden Befehlen:

- `Go(`d`)` lässt die Schildkröte eine gerade Strecke der Länge d ($d \geq 0$) in ihrer Blickrichtung laufen.
- `Turn(`α`)` dreht die Schildkröte um die durch α angegebene Gradzahl nach links, wenn $\alpha > 0$, sonst um den Betrag von α nach rechts.
- `PenDown` senkt den Stift der Schildkröte auf die Zeichenfläche ab.
- `PenUp` hebt den Stift der Schildkröte von der Zeichenfläche an.
- `Sequence(`$s_1, s_2, \ldots$`)` ist ein Befehl, der nacheinander die Befehle $s_1, s_2, \ldots$ ausführt.

Beispiel: Die folgende Zeichnung wird von einer mit

```
Sequence(PenDown, Go(100), Turn(120), Go(100))
```

gesteuerten Schildkröte erstellt, die anfangs im Punkt a steht und nach Osten blickt. Am Ende befindet sich die Schildkröte dann im Punkt b.



Die Befehle sollen nun in Java jeweils als eigene Klasse implementiert werden, so dass sich das oben angegebene Programm als

```
new Sequence(new PenDown(), new Go(100), new Turn(120), new Go(100))
```

formulieren lässt. Alle diese Klassen sollen das Interface `Stmt` implementieren und das Visitor-Muster unterstützen, wofür das Interface `Visitor` benötigt wird. Ergänzen Sie dementsprechend die folgenden Interfaces und Klassen. Stellen Sie in der Klasse `Go` sicher, dass die Streckenlänge d im Befehl `Go(`d`)` nichtnegativ ist. Andernfalls soll eine geeignete Exception geworfen werden.

```
public interface Stmt
```

Name:	Matrikelnummer:

```
public interface Visitor
```

```
public class PenUp
```

```
public class PenDown
```

```
public class Turn
    public final double angle;
    public Turn(double angle) {
```

Name:	Matrikelnummer:

```
public class Go
    public final double dist;
    public Go(double d) {
```

```
public class Sequence
    public final Stmt[] seq;
    public Sequence(Stmt... seq) {
```

Name:	Matrikelnummer:

Aufgabe 2 (Visitor-Muster, 11 Punkte)

Im Folgenden geht es aber nicht um die Standardinterpretation der Turtle-Grafik. Vielmehr soll die Schildkröte die Märchenfiguren *Hänsel und Gretel* modellieren, die man mit einem Turtle-Grafik-Programm durch einen dunklen, zweidimensionalen Wald steuert. Dabei sollen sie in regelmäßigen Abständen (vorgegeben durch eine Strecke s) Brotkrümel (*bread crumbs*) fallenlassen, um so eine Spur zu hinterlassen, jedoch nur bei abgesenktem Stift. Wie Hänsel und Gretel dabei genau vorgehen, ist durch das Zustandsdiagramm in der nächsten Aufgabe beschrieben und hier nicht weiter von Belang.

Um Turtle-Grafik-Programme auf diese Weise zu interpretieren, soll die Klasse `HanselGretelVisitor` (siehe Seite 9) als Visitor realisiert werden. Das Interface `DarkForest` repräsentiert dabei den dunklen, zweidimensionalen Wald, dessen `breadCrumb`-Methode aufgerufen wird, um einen Brotkrümel an der angegebenen Position fallenzulassen. Die Klasse `State` im Klassendiagramm wird erst in der nächsten Aufgabe benötigt.

a) Das Klassendiagramm auf Seite 9 enthält das Interface `Visitor` aus der vorigen Aufgabe. Ergänzen Sie im Klassendiagramm alle Angaben (z.B. Klassen, Attribute, Methoden, Generalisierungsbeziehungen etc.), um mit `HanselGretelVisitor` einen geeigneten Visitor zu realisieren. Die Methoden von `Visitor` müssen im Klassendiagramm nicht ergänzt werden!

b) Vervollständigen Sie den folgenden Java-Code für den Konstruktor der Klasse `HanselGretelVisitor`. Der Parameter `s` sei der Abstand zwischen aufeinanderfolgenden Krümeln der Brotkrümelspur. Ergänzen Sie auch eventuell fehlende Attribute der Klasse.

c) Vervollständigen Sie nun den fehlenden Code, um den Visitor zu realisieren. Rufen Sie dabei die Methoden `up()`, `down()` und `go(`d`)` auf, um die Turtle-Grafik-Befehle `PenUp`, `PenDown` bzw. `Go(`d`)` auszuführen. Diese Methoden implementieren Sie aber erst in Aufgabe 3.

Der Code der Klasse `HanselGretelVisitor` besteht aus dem Java-Code auf dieser und der folgenden Seite sowie dem auf Seite 7.

```
public class HanselGretelVisitor
    private double gap;









    public HanselGretelVisitor(double s, Turtle turtle, DarkForest forest) {
```

Name:	Matrikelnummer:

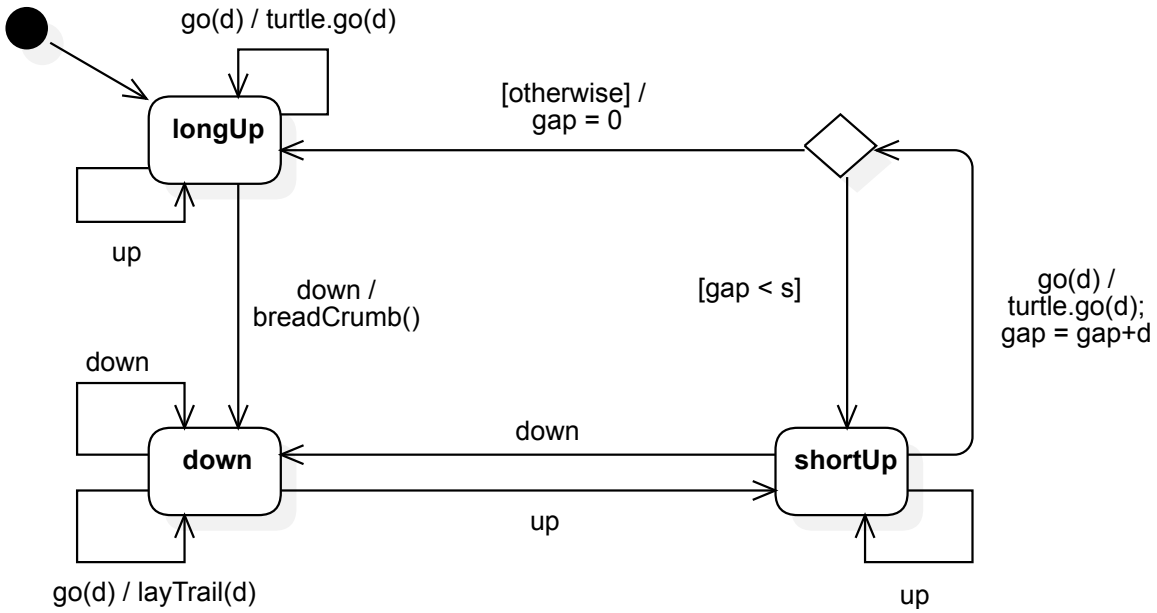
```
private void breadCrumb() {
    forest.breadCrumb(turtle.getX(), turtle.getY());
}

// private void layTrail(double len)  muss nicht geschrieben werden
// private void go(double d)          wird in Aufgabe 3 geschrieben
// private void down()                wird in Aufgabe 3 geschrieben
// private void up()                  wird in Aufgabe 3 geschrieben
```

Name:	Matrikelnummer:

Aufgabe 3 (State-Muster, 12 Punkte)

Das nachstehende Zustandsdiagramm beschreibt das Verhalten eines `HanselGretelVisitor`-Objekts, wenn die Methoden `up`, `down` und `go` aufgerufen werden.



a) Ergänzen Sie im Klassendiagramm auf Seite 9 alle Angaben (z.B. Klassen, Attribute, Methoden, Generalisierungsbeziehungen etc.), um das Zustandsdiagramm mit Hilfe des State-Musters umzusetzen.

b) Realisieren Sie ausschließlich den Zustand *longUp* im Java-Code auf der nächsten Seite, indem Sie den vorgegebenen Code gemäß State-Muster vervollständigen. Den Code für die anderen Zustände müssen Sie nicht schreiben.

c) Vervollständigen Sie im Java-Code der Klasse `HanselGretelVisitor` den Code für `up`, `down` und `go` unter Nutzung des State-Musters. Ergänzen Sie dabei auch eventuell fehlende Attribute, die gegebenenfalls geeignet initialisiert werden müssen.

Name:	Matrikelnummer:

```
// der folgende Code ist eine Fortsetzung von Seite 4 sowie 5 und gehört auch
// zur Klasse HanselGretelVisitor

private abstract class State {
    abstract void go(double dist);
    void up() {}
    void down() {}
}

private final State shortUp = // muss nicht geschrieben werden
private final State down = // muss nicht geschrieben werden

private final State longUp =
```

```
private void go(double d) {
```

```
private void down() {
```

```
private void up() {
```

Name:	Matrikelnummer:

Aufgabe 4 (Softwaretest, 7 Punkte)

Die Implementierung soll mit JUnit getestet werden. Hier sollen Sie aber nur einen Testfall `testTriangle` schreiben, der das Turtle-Grafik-Programm

```
Sequence(PenDown, Go(50), Turn(120), Go(50), Turn(120), Go(50))
```

ausführt. Die Schildkröte soll sich anfangs an der Position $(2, 1)$ befinden und nach Osten blicken. Da sie damit ein gleichseitiges Dreieck abfährt, muss sich die Schildkröte danach wieder an ihrer ursprünglichen Position befinden.

Ergänzen Sie die unten angegebene Testklasse `HanselGretelTest` um diesen Testfall, wobei sie den `HanselGretelVisitor` mit $s = 10$ nutzen.

Hinweis: Gehen Sie davon aus, dass es eine Klasse `BlackForest` gibt, die das Interface `DarkForest` implementiert und einen Konstruktor mit leerer Parameterliste hat.

```
import org.junit.Test;
import static org.junit.Assert.assertEquals;

public class HanselGretelTest {
    public static final double EPS = 1e-10;
    public static final Stmt PROG = new Sequence(new PenDown(), new Go(50),
                                                 new Turn(120), new Go(50),
                                                 new Turn(120), new Go(50));
    private Turtle turtle;

    @Before
    public void setup() {
        turtle = new Turtle(2, 1, 0);
    }
```

Name:	Matrikelnummer: