Ludwig-Maximilians-Universität München WS 2018/19
Institut für Informatik Übungsblatt 6
Prof. Dr. D. Sabel, Dr. P. Wendler

Übungen zu Einführung in die Informatik: Programmierung und Software-Entwicklung:

Lösungsvorschlag

Aufgabe 6-1 Überprüfen und Auswerten von Ausdrücken *Präsenz*

In dieser Aufgabe sollen Sie Ausdrücke auf Korrektheit untersuchen und gegebenenfalls auswerten. Als Grundlage für Ausdrücke im Kontext von Klassendeklarationen verwenden wir folgende EBNF-Grammatik mit der entsprechenden Erweiterung von `Value` um das Terminalsymbol `"null"`:

```
Expression = Variable |
             Value |
             Expression BinOp Expression |
             UnOp Expression |
             "(" Expression ")" |
             MethodInvocation |
             InstanceCreation

Variable = NamedVariable | FieldAccess
NamedVariable = Identifier
FieldAccess = Expression "." Identifier

MethodInvocation = Expression "." Identifier "(" [ActualParameters] ")"
ActualParameters = Expression {"," Expression}

InstanceCreation = ClassInstanceCreation
ClassInstanceCreation = "new" ClassType "(" [ActualParameters] ")"
```

Wir verwenden die Klassendeklarationen von `Point` und `Line` aus der Vorlesung. Seien `Point p;`, `Line l;` und `int i;` lokale Variablendeklarationen. Gegeben seien folgende Ausdrücke:

1. `l.x`
2. `p.move(1,2) == null`
3. `l.length()`
4. `p == null`
5. `i == null`
6. `new point(3,4)`
7. `(new Point(1,1)).x`

a) Geben Sie an, welche dieser Ausdrücke syntaktisch nicht korrekt und welche Ausdrücke syntaktisch korrekt, aber nicht typkorrekt sind (jeweils mit Begründung). Für typkorrekte Ausdrücke ist deren Typ anzugeben.

Lösung:
Überprüfen von Ausdrücken auf Korrektheit:

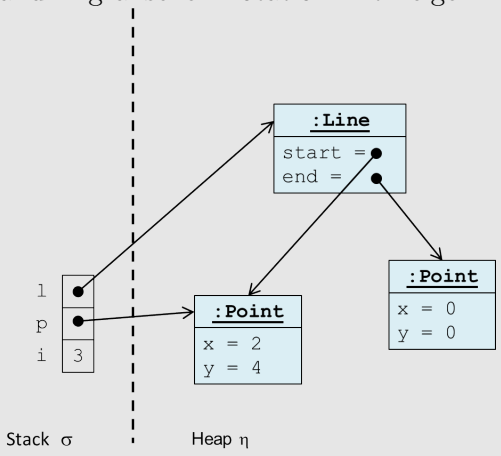
1. `l.x` ist nicht typkorrekt, da zwar `l` den Klassentyp `Line` hat, `x` jedoch kein Attribut der Klasse `Line` ist.
2. `p.move(1,2) == null` ist nicht typkorrekt, da die Methode `move` eine Methode ohne Ergebnis (zu erkennen am Rückgabetyp `void`) ist. Solche Methoden können nicht innerhalb eines geschachtelten Ausdrucks verwendet werden.
3. `l.length()` ist syntaktisch korrekt und typkorrekt und hat den Typ `double`.
4. `p == null` ist syntaktisch korrekt und typkorrekt und hat den Typ `boolean`.
5. `i == null` ist nicht typkorrekt, da `==` eine zweistellige Operation ist, die in beiden Argumenten den gleichen Typ erwartet, `i` aber vom Typ `int` ist und `null` einen namenlosen Typen passend zu jedem Klassentypen, aber nicht zu Grunddatentypen hat.
6. `new point(3,4)` ist nicht typkorrekt, da `point` kein Klassentyp ist (Achtung bei Groß- und Kleinschreibung!).
7. `(new Point(1,1)).x` ist syntaktisch korrekt und typkorrekt und hat den Typ `int`.

b) Gegegeben sei der Zustand (σ, η) mit

$$\begin{aligned}\sigma &= [(\texttt{i}, 3), (\texttt{p}, @123), (\texttt{l}, @789)] \\ \eta &= \{< (@123, \texttt{Point}), [(\texttt{x}, 2), (\texttt{y}, 4)] >, < (@456, \texttt{Point}), [(\texttt{x}, 0), (\texttt{y}, 0)] >, \\ &\quad < (@789, \texttt{Line}), [(\texttt{start}, @123), (\texttt{end}, @456)] >\}\end{aligned}$$

i) Geben Sie die grafische Repräsentation dieses Zustands mit Zeigern an.

Lösung:
Zustand in grafischer Notation mit Zeigern:



ii) Werten Sie alle syntaktisch korrekten und typkorrekten Ausdrücke bezüglich dieses Zustands aus.

Lösung:
Auswerten der korrekten Ausdrücke:

`l.length()`
$=_{(\sigma,\eta)}$ `@789.length()`
$=_{(\sigma,\eta)} \sqrt{2 * 2 + 4 * 4}$
$=_{(\sigma,\eta)} \sqrt{20}$

$=_{(\sigma,\eta)}$ 4.47

`p == null`
$=_{(\sigma,\eta)}$ `@123 == null`
$=_{(\sigma,\eta)}$ `false`

`(new Point(1,1)).x`
$=_{(\sigma,\eta)}$ `@000.x` mit Zustandsänderung zu (σ, η')
$=_{(\sigma,\eta')}$ `1`
mit $\eta' = \eta \cup \{< (@000, \texttt{Point}), [(\texttt{x}, 1), (\texttt{y}, 1)] >\}$

Aufgabe 6-2 Überprüfen und Auswerten von Ausdrücken *Hausaufgabe*

In dieser Aufgabe sollen Sie Ausdrücke auf Korrektheit untersuchen und gegebenenfalls auswerten. Als Grundlage für Ausdrücke im Kontext von Klassendeklarationen verwenden wir die EBNF-Grammatik aus Aufgabe 6-1.

Wir verwenden die Klassendeklarationen von `Point` und `Line` aus der Vorlesung. Seien `Point p;`, `Line l;` und `int i;` lokale Variablendeklarationen. Gegeben seien folgende Ausdrücke:

1. `i.y`
2. `p.i`
3. `l.start.y`
4. `p.GetX()`
5. `l.move(p)`
6. `i == p`
7. `new Line(3,4)`
8. `Point(3,4)`
9. `(new Line(new Point(i,i), p)).end.getY()`

a) Geben Sie an, welche dieser Ausdrücke syntaktisch nicht korrekt und welche Ausdrücke syntaktisch korrekt, aber nicht typkorrekt sind (jeweils mit Begründung). Für typkorrekte Ausdrücke ist deren Typ anzugeben.

Lösung:
Überprüfen der Ausdrücke:

1. `i.y` ist nicht typkorrekt, da `i` keinen Klassentyp hat.
2. `p.i` ist nicht typkorrekt, da zwar `p` den Klassentyp `Point` hat, `i` jedoch kein Attribut der Klasse `Point` ist.
3. `l.start.y` ergibt nach vollständiger Klammerung `(l.start).y`. Dieser Ausdruck ist syntaktisch korrekt und typkorrekt und hat den Typ `int`.
4. `p.GetX()` ist nicht typkorrekt, da keine Methode `GetX` in der Klasse `Point` deklariert ist (Achtung bei Groß- und Kleinschreibung!).
5. `l.move(p)` ist nicht typkorrekt, da die Anzahl der aktuellen Parameter nicht mit der Anzahl der formalen Parameter in der Deklaration der Methode `move` übereinstimmt.

6. `i == p` ist nicht typkorrekt, da `==` eine zweistellige Operation ist, die in beiden Argumenten den gleichen Typ erwartet, `i` aber vom Typ `int` und `p` vom Typ `Point` ist.
7. `new Line(3,4)` ist nicht typkorrekt, da zwar die Anzahl der aktuellen Parameter mit der Anzahl der formalen Parameter in der Konstruktordeklaration übereinstimmt, aber `3` und `4` den Typ `int` haben, der Konstruktor Paramter vom Typ `Point` erwartet.
8. `Point(3,4)` ist syntaktisch nicht korrekt, da zur Objekterzeugung ein `new` vorangestellt werden muss.
9. `(new Line(new Point(i,i), p)).end.getY()` ergibt nach vollständiger Klammerung `((new Line(new Point(i,i), p)).end).getY()` ist syntaktisch korrekt und typkorrekt und hat den Typ `int`.

b) Gegegeben sei der Zustand (σ, η) mit

$$\begin{aligned}\sigma &= [(\texttt{i}, 3), (\texttt{p}, @123), (\texttt{l}, @789)] \\ \eta &= \{< (@123, \texttt{Point}), [(\texttt{x}, 2), (\texttt{y}, 4)] >, < (@456, \texttt{Point}), [(\texttt{x}, 0), (\texttt{y}, 0)] >, \\ &\quad < (@789, \texttt{Line}), [(\texttt{start}, @123), (\texttt{end}, @456)] >\}\end{aligned}$$

Werten Sie alle syntaktisch korrekten und typkorrekten Ausdrücke bezüglich dieses Zustands aus.

Lösung: Auswerten der korrekten Ausdrücke:

`(l.start).y`
$=_{(\sigma,\eta)}$ `(@789.start).y`
$=_{(\sigma,\eta)}$ `@123.y`
$=_{(\sigma,\eta)}$ `4`

`((new Line(new Point(i,i), p)).end).getY()`
$=_{(\sigma,\eta)}$ `((new Line(new Point(3,3), p)).end).getY()`
$=_{(\sigma,\eta)}$ `((new Line(@111, p)).end).getY()` mit Zustandsänderung zu (σ, η')
$=_{(\sigma,\eta')}$ `((new Line(@111, @123)).end).getY()`
$=_{(\sigma,\eta')}$ `(@222.end).getY()` mit Zustandsänderung zu (σ, η'')
$=_{(\sigma,\eta'')}$ `@123.getY()`
$=_{(\sigma,\eta'')}$ 4
mit $\eta' = \eta \cup \{< (@111, Point), [(x, 3), (y, 3)] >\}$
und $\eta'' = \eta' \cup \{< (@222, Line), [(start, @111), (end, @123)] >\}$

Aufgabe 6-3 **Deklaration von Klassen und Methoden** *Präsenz*

In dieser Aufgabe soll eine Klasse `Figur` deklariert werden, die einfache geometrische Figuren repräsentiert. Beispielsweise können ein Rechteck oder ein Kreis eine geometrische Figur sein. Jede geometrische Figur hat einen Mittelpunkt vom Typ `Point` und eine Farbe vom Typ `String`. Außerdem kann man angeben, ob die Figur mit dieser Farbe ausgefüllt ist.

a) Deklarieren Sie eine Klasse `Figur`, die oben genannte Eigenschaften realisiert. Die Klasse soll die folgenden Konstruktoren und zwei Methoden anbieten:

- einen Konstruktor mit formalen Parametern zur Initialisierung der Attribute,
- einen weiteren Konstruktor mit formalen Parametern zur Initialisierung der Attribute, so dass für den Mittelpunkt als Parameter dessen x- und y-Koordinaten übergeben werden,

- eine Methode `bewegen` ohne Rückgabetyp, die den Mittelpunkt einer Figur soweit verschiebt, wie es zwei formale Parameter `dx` und `dy` vom Typ `int` fordern,
- eine Methode `getMittelpunkt()` mit dem Rückgabetyp `Point`, die den Mittelpunkt einer Figur zurückgibt.

Hinweis: Um die Klasse ***`Point`*** *verwenden zu können, speichern Sie die Datei* ***`Point.java`*** *aus dem ZIP-Archiv des Übungsblatts im gleichen Ordner wie Ihre Klasse* ***`Figur`****. Diese Datei enthält die Klassendeklaration der Klasse* ***`Point`****, wie sie in der Vorlesung gezeigt wurde.*

Lösung:

```
1  /**
2   * Diese Klasse repraesentiert eine geometrische Figur.
3   */
4  public class Figur {
5     private Point mittelpunkt;
6     private String farbe;
7     private boolean ausgefuellt;
8
9     /**
10     * Konstruktor einer geometrischen Figur, wobei deren Mittelpunkt und die
11     * Farbe gegeben sein muessen sowie ob die Figur ausgefuellt ist oder nicht.
12     *
13     * @param p
14     *            der Mittelpunkt der Figur
15     * @param farbe
16     *            die Farbe der Figur
17     * @param ausgefuellt
18     *            ob die Figur ausgefuellt ist oder nicht
19     */
20    public Figur(Point p, String farbe, boolean ausgefuellt) {
21       this.mittelpunkt = p;
22       this.farbe = farbe;
23       this.ausgefuellt = ausgefuellt;
24    }
25
26    /**
27     * Konstruktor einer geometrischen Figur, wobei deren Mittelpunkt und die
28     * Farbe gegeben sein muessen sowie ob die Figur ausgefuellt ist oder nicht.
29     *
30     * @param x
31     *            x-Koordinate des Mittelpunkts der Figur
32     * @param y
33     *            y-Koordinate des Mittelpunkts der Figur
34     * @param farbe
35     *            die Farbe der Figur
36     * @param ausgefuellt
37     *            ob die Figur ausgefuellt ist oder nicht
38     */
39    public Figur(int x, int y, String farbe, boolean ausgefuellt) {
40       this.mittelpunkt = new Point(x, y);
41       this.farbe = farbe;
42       this.ausgefuellt = ausgefuellt;
43    }
44
45    /**
46     * Diese Methode versetzt den Mittelpunkt der Figur
47     *
48     * @param dx
49     *            Versatz in x-Richtung
50     * @param dy
51     *            Versatz in y-Richtung
52     */
53    public void bewegen(int dx, int dy) {
```

```
54        this.mittelpunkt.move(dx, dy);
55     }
56
57     /**
58      * Diese Methode gibt den Mittelpunkt der Figur zurueck
59      *
60      * @return der Mittelpunkt
61      */
62     public Point getMittelpunkt() {
63        return this.mittelpunkt;
64     }
65 }
```

b) Mit folgendem Programmcode können Sie Ihre Klasse `Figur` testen (die Datei `FigurTest.java` muss im gleichen Ordner wie Ihre Klasse `Figur` gespeichert werden):

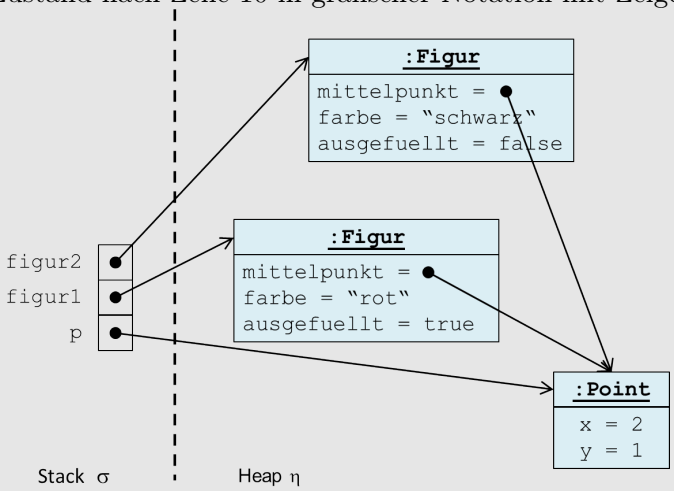
```
1  /**
2   * Diese Klasse testet die Implementierung der Klasse Figur.
3   * Es wird eine Figur figur1 erstellt sowie eine Figur figur2.
4   * Anschliessend wird die Figur figur1 bewegt.
5   */
6  public class FigurTest {
7     public static void main(String[] args) {
8        Point p = new Point(2, 1);
9        Figur figur1 = new Figur(p, "rot", true);
10       Figur figur2 = new Figur(p, "schwarz", false);
11
12       System.out.println("figur1: Mittelpunkt=("
13             + figur1.getMittelpunkt().getX() + ","
14             + figur1.getMittelpunkt().getY() + ")");
15       System.out.println("figur2: Mittelpunkt=("
16             + figur2.getMittelpunkt().getX() + ","
17             + figur2.getMittelpunkt().getY() + ")");
18
19       figur1.bewegen(2, 2);
20
21       System.out.println("figur1: Mittelpunkt=("
22             + figur1.getMittelpunkt().getX() + ","
23             + figur1.getMittelpunkt().getY() + ")");
24       System.out.println("figur2: Mittelpunkt=("
25             + figur2.getMittelpunkt().getX() + ","
26             + figur2.getMittelpunkt().getY() + ")");
27    }
28 }
```

Geben Sie für dieses Programm jeweils den Zustand des Speichers (Stack und Heap) nach Zeile 10 und nach Zeile 19 an. Sie können die Zustände entweder in der abstrakten Syntax mit Objektreferenzen Ihrer Wahl angeben (siehe Aufgabe 6-1 und 6-2) oder die grafische Repräsentation mit Zeigern verwenden.

Lösung:
Zustand nach Zeile 10 in abstrakter Syntax:

$$
\begin{aligned}
\sigma &= [(\mathtt{p}, @111), (\mathtt{figur1}, @222), (\mathtt{figur2}, @333)] \\
\eta &= \{< (@111, \mathtt{Point}), [(\mathtt{x}, 2), (\mathtt{y}, 1)] >, \\
&\quad < (@222, \mathtt{Figur}), [(\mathtt{mittelpunkt}, @111), (\mathtt{farbe}, \text{“}rot\text{”}), (\mathtt{ausgefuellt}, true)] >, \\
&\quad < (@333, \mathtt{Figur}), [(\mathtt{mittelpunkt}, @111), (\mathtt{farbe}, \text{“}schwarz\text{”}), (\mathtt{ausgefuellt}, false)] >\}
\end{aligned}
$$

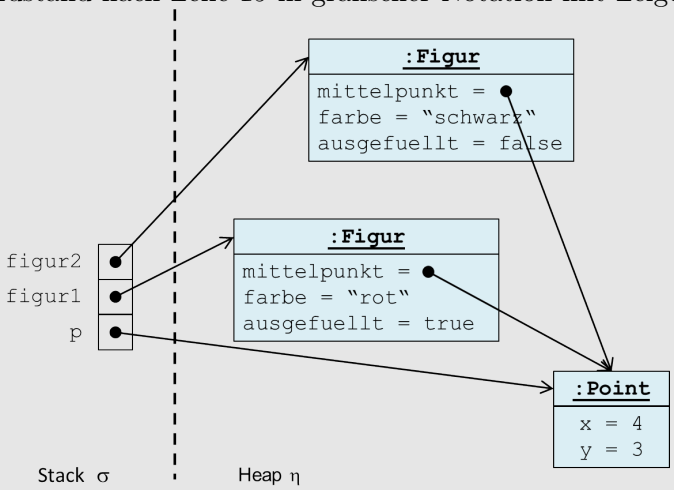
Zustand nach Zeile 10 in grafischer Notation mit Zeigern:



Zustand nach Zeile 19 in abstrakter Syntax:

$$\begin{aligned}\sigma &= [(\texttt{p}, @111), (\texttt{figur1}, @222), (\texttt{figur2}, @333)] \\ \eta &= \{< (@111, \texttt{Point}), [(\texttt{x}, 4), (\texttt{y}, 3)] >, \\ &\quad < (@222, \texttt{Figur}), [(\texttt{mittelpunkt}, @111), (\texttt{farbe}, \text{“}rot\text{”}), (\texttt{ausgefuellt}, true)] >, \\ &\quad < (@333, \texttt{Figur}), [(\texttt{mittelpunkt}, @111), (\texttt{farbe}, \text{“}schwarz\text{”}), (\texttt{ausgefuellt}, false)] >\}\end{aligned}$$

Zustand nach Zeile 19 in grafischer Notation mit Zeigern:



c) Welches Problem fällt Ihnen bei der Analyse der Zustandsänderung auf und wie können Sie es beheben?

Lösung:
Bei der Analyse der Zustandänderung fällt auf, dass nicht nur der Mittelpunkt von `figur1` bewegt wurde, sondern auch der Mittelpunkt von `figur2`. Dies liegt daran, dass beide Figuren im Attribut `mittelpunkt` auf den gleichen Punkt referenzieren. Wird dieser Punkt verändert, verändert er sich automatisch für beide Figuren.

Verbesserte Klasse `FigurTest`:

```
1 public class FigurTest {
2   public static void main(String[] args) {
3     Figur figur1 = new Figur(2, 1, "rot", true);
4     Figur figur2 = new Figur(2, 1, "schwarz", false);
5     figur1.bewegen(2, 2);
```

```
6   }
7 }
```

Aufgabe 6-4 **Deklaration von Klassen und Methoden** *Hausaufgabe*

In dieser Aufgabe soll eine Klasse `Fahrzeug` deklariert werden, die beliebige Fortbewegungsmittel repräsentiert. Beispielsweise können ein Fahrrad, ein Auto oder eine Eisenbahn ein Fahrzeug sein. Jedes Fahrzeug hat eine aktuelle Position vom Typ `Point`, eine bestimmte Anzahl an Rädern, ein Leergewicht und eine aktuelle Geschwindigkeit.

a) Deklarieren Sie eine Klasse `Fahrzeug`, die oben genannte Eigenschaften definiert. Die Klasse soll den folgenden Konstruktor und drei Methoden anbieten:

- einen Konstruktor mit formalen Parametern zur Initialisierung der Attribute, so dass für den Mittelpunkt als Parameter dessen x- und y-Koordinaten übergeben werden,
- eine Methode `beschleunigen` ohne Rückgabetyp, die die aktuelle Geschwindigkeit um einen bestimmten Wert erhöht bzw. erniedrigt (bei negativer Beschleunigung). Der Wert soll als Parameter übergeben werden können,
- eine Methode `fahren` ohne Rückgabetyp, die das Fahrzeug auf eine neue Position verschiebt, wie es zwei formale Parameter `dx` und `dy` vom Typ `int` fordern,
- eine Methode `getAktuelleGeschwindigkeit` mit dem Rückgabetyp `double`, die die aktuelle Geschwindigkeit des Fahrzeugs zurückgibt.

Hinweis: Um die Klasse `Point` verwenden zu können, speichern Sie die Datei `Point.java` aus dem ZIP-Archiv des Übungsblatts im gleichen Ordner wie Ihre Klasse `Fahrzeug`. Diese Datei enthält die Klassendeklaration der Klasse `Point`, wie sie in der Vorlesung gezeigt wurde.

Lösung:

```
1  /**
2   * Diese Klasse repraesentiert ein Fahrzeug.
3   */
4  public class Fahrzeug {
5     private Point position;
6     private int anzahlRaeder;
7     private double leergewicht;
8     private double aktuelleGeschwindigkeit;
9
10    /**
11     * Konstruktor eine Fahrzeugs, wobei die Position des Fahrzeugs, die Anzahl
12     * der Raeder, das Leergewicht und die aktuelle Geschwindigkeit des
13     * Fahrzeugs gegeben sein muessen.
14     *
15     * @param x
16     *             x-Koordinate der Position des Fahrzeugs
17     * @param y
18     *             y-Koordinate der Position des Fahrzeugs
19     * @param anzahlRaeder
20     *             die Anzahl der Raeder des Fahrzeugs
21     * @param leergewicht
22     *             das Leergewicht des Fahrzeugs
23     * @param aktuelleGeschwindigkeit
24     *             die aktuelle Anfangsgeschwindigkeit des Fahrzeugs
25     */
26    public Fahrzeug(int x, int y, int anzahlRaeder, double leergewicht,
27          double aktuelleGeschwindigkeit) {
28       this.position = new Point(x, y);
```

```
29        this.anzahlRaeder = anzahlRaeder;
30        this.leergewicht = leergewicht;
31        this.aktuelleGeschwindigkeit = aktuelleGeschwindigkeit;
32     }
33
34     /**
35      * Diese Methode beschleunigt das Fahrzeug um die angegebene
36      * Geschwindigkeit.
37      *
38      * @param beschleunigung
39      *             die Geschwindigkeit, um die das Fahrzeug beschleunigt werden
40      *             soll
41      */
42     public void beschleunigen(double beschleunigung) {
43        this.aktuelleGeschwindigkeit = this.aktuelleGeschwindigkeit
44              + beschleunigung;
45     }
46
47     /**
48      * Diese Methode bewegt den Mittelpunkt des Fahrzeugs
49      *
50      * @param dx
51      *             Versatz in x-Richtung
52      * @param dy
53      *             Versatz in y-Richtung
54      */
55     public void fahren(int dx, int dy) {
56        this.position.move(dx, dy);
57     }
58
59     /**
60      * Diese Methode gibt die aktuelle Geschwindigkeit des Fahrzeugs zurueck
61      *
62      * @return die aktuelle Geschwindigkeit
63      */
64     public double getAktuelleGeschwindigkeit() {
65        return this.aktuelleGeschwindigkeit;
66     }
67 }
```

b) Mit folgendem Programmcode können Sie Ihre Klasse `Fahrzeug` testen (die Datei `FahrzeugTest.java` muss im gleichen Ordner wie Ihre Klasse `Fahrzeug` gespeichert werden):

```
1  /**
2   * Diese Klasse testet die Implementierung der Klasse Fahrzeug.
3   * Es wird ein Fahrzeug auto erstellt sowie ein Fahrzeug fahrrad.
4   * Anschliessend wird das Fahrzeug auto beschleunigt.
5   */
6  public class FahrzeugTest {
7     public static void main(String[] args) {
8        Fahrzeug auto = new Fahrzeug(1, 2, 4, 1234.5, 100.0);
9        Fahrzeug fahrrad = new Fahrzeug(3, 4, 2, 33.3, 18.3);
10
11       System.out.println("Auto: aktuelle Geschwindigkeit="
12             + auto.getAktuelleGeschwindigkeit());
13       System.out.println("Fahrrad: aktuelle Geschwindigkeit="
14             + fahrrad.getAktuelleGeschwindigkeit());
15
16       auto.beschleunigen(11.1);
17
18       System.out.println("Auto: aktuelle Geschwindigkeit="
19             + auto.getAktuelleGeschwindigkeit());
20       System.out.println("Fahrrad: aktuelle Geschwindigkeit="
21             + fahrrad.getAktuelleGeschwindigkeit());
```

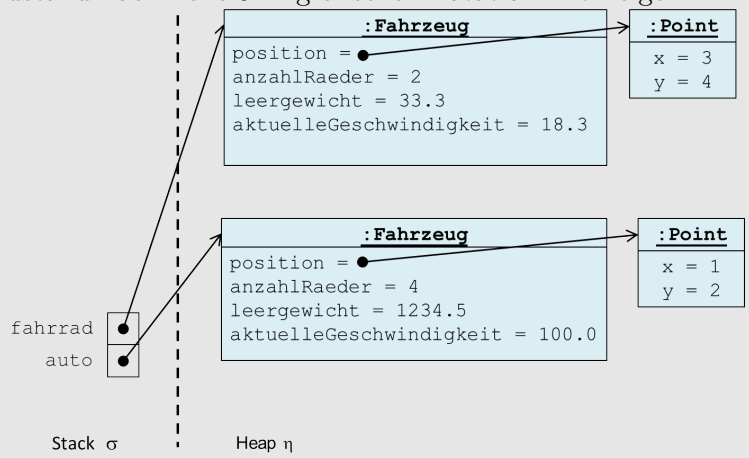
```
22
23          fahrrad.fahren(5, 3);
24      }
25  }
```

Geben Sie für dieses Programm jeweils den Zustand des Speichers (Stack und Heap) nach Zeile 9, nach Zeile 16 und nach Zeile 23 an. Sie können die Zustände entweder in der abstrakten Syntax angeben (siehe Aufgabe 6-1 und 6-2) oder die grafische Reräsentation mit Zeigern von Hand malen und abfotografieren oder ein beliebiges Zeichenprogramm (z.B. PowerPoint) verwenden. Bitte geben Sie nur `.txt`-, `.jpg`- oder `.pdf`-Dateien ab!

Lösung:
Zustand nach Zeile 9 in abstrakter Syntax:

$$
\begin{aligned}
\sigma &= [(\texttt{auto}, @111), (\texttt{fahrrad}, @222)] \\
\eta &= \{< (@111, \texttt{Fahrzeug}), [(\texttt{position}, @333), (\texttt{anzahlRaeder}, 4), \\
&\quad (\texttt{leergewicht}, 1234.5), (\texttt{aktuelleGeschwindigkeit}, 100.0)] >, \\
&\quad < (@222, \texttt{Fahrzeug}), [(\texttt{position}, @444), (\texttt{anzahlRaeder}, 2), \\
&\quad (\texttt{leergewicht}, 33.3), (\texttt{aktuelleGeschwindigkeit}, 18.3)] >, \\
&\quad < (@333, \texttt{Point}), [(\texttt{x}, 1), (\texttt{y}, 2)] >, \\
&\quad < (@444, \texttt{Point}), [(\texttt{x}, 3), (\texttt{y}, 4)] >\}
\end{aligned}
$$

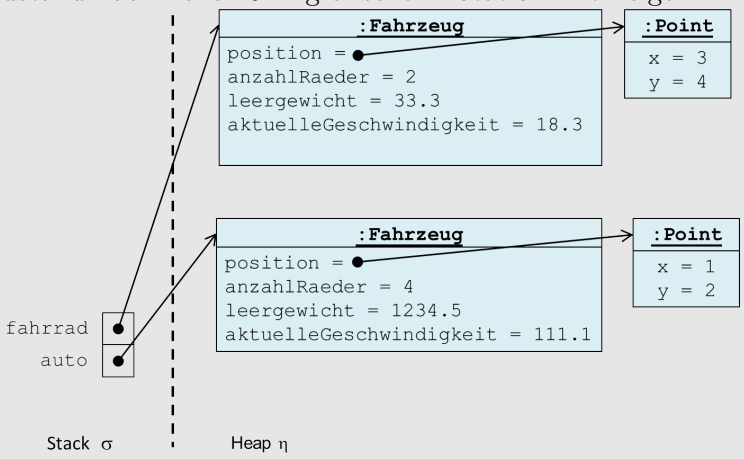
Zustand nach Zeile 9 in grafischer Notation mit Zeigern:



Zustand nach Zeile 16 in abstrakter Syntax:

$$
\begin{aligned}
\sigma &= [(\texttt{auto}, @111), (\texttt{fahrrad}, @222)] \\
\eta &= \{< (@111, \texttt{Fahrzeug}), [(\texttt{position}, @333), (\texttt{anzahlRaeder}, 4), \\
&\quad (\texttt{leergewicht}, 1234.5), (\texttt{aktuelleGeschwindigkeit}, 111.1)] >, \\
&\quad < (@222, \texttt{Fahrzeug}), [(\texttt{position}, @444), (\texttt{anzahlRaeder}, 2), \\
&\quad (\texttt{leergewicht}, 33.3), (\texttt{aktuelleGeschwindigkeit}, 18.3)] >, \\
&\quad < (@333, \texttt{Point}), [(\texttt{x}, 1), (\texttt{y}, 2)] >, \\
&\quad < (@444, \texttt{Point}), [(\texttt{x}, 3), (\texttt{y}, 4)] >\}
\end{aligned}
$$

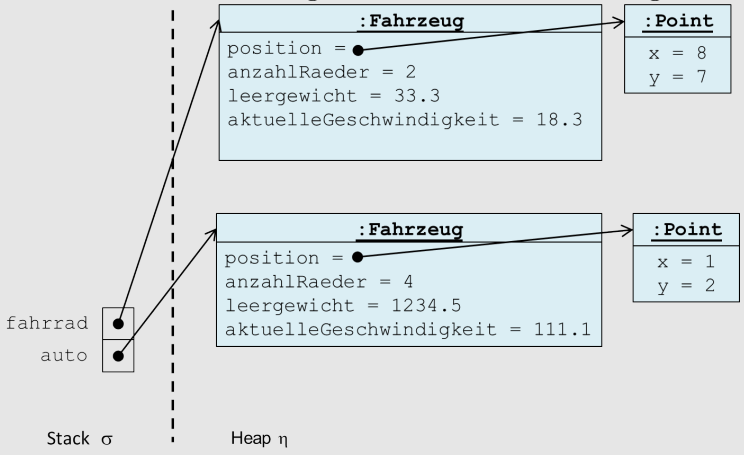
Zustand nach Zeile 16 in grafischer Notation mit Zeigern:



Zustand nach Zeile 23 in abstrakter Syntax:

$$
\begin{aligned}
\sigma &= [(\texttt{auto}, @111), (\texttt{fahrrad}, @222)] \\
\eta &= \{< (@111, \texttt{Fahrzeug}), [(\texttt{position}, @333), (\texttt{anzahlRaeder}, 4), \\
&\quad (\texttt{leergewicht}, 1234.5), (\texttt{aktuelleGeschwindigkeit}, 111.1)] >, \\
&\quad < (@222, \texttt{Fahrzeug}), [(\texttt{position}, @444), (\texttt{anzahlRaeder}, 2), \\
&\quad (\texttt{leergewicht}, 33.3), (\texttt{aktuelleGeschwindigkeit}, 18.3)] >, \\
&\quad < (@333, \texttt{Point}), [(\texttt{x}, 1), (\texttt{y}, 2)] >, \\
&\quad < (@444, \texttt{Point}), [(\texttt{x}, 8), (\texttt{y}, 7)] >\}
\end{aligned}
$$

Zustand nach Zeile 23 in grafischer Notation mit Zeigern:



Besprechung der Präsenzaufgaben in den Übungen ab 23.11.2018. Abgabe der Hausaufgaben bis Mittwoch, 05.12.2018, 14:00 Uhr über UniworX (siehe Folien der ersten Zentralübung).

- *Erstellen Sie zu jeder Aufgabe eine Klasse, die den Namen trägt, der in der Aufgabe gefordert ist.*
- *Der Java-Code in ihrer Abgabe muss als separate* `.java`*-Datei abgegeben werden. Wir benötigen* ***nicht*** *Ihre* `.class`*-Dateien.*