Prof. Dr. M. Müller C. Aschermann, J. Hensel, J. Protze, P. Reble

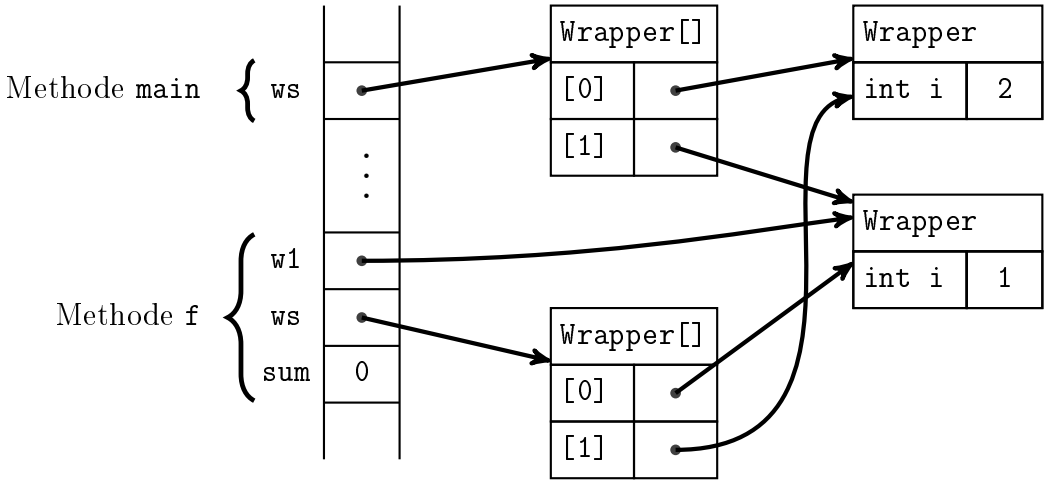
Tutoraufgabe 1 (Seiteneffekte):

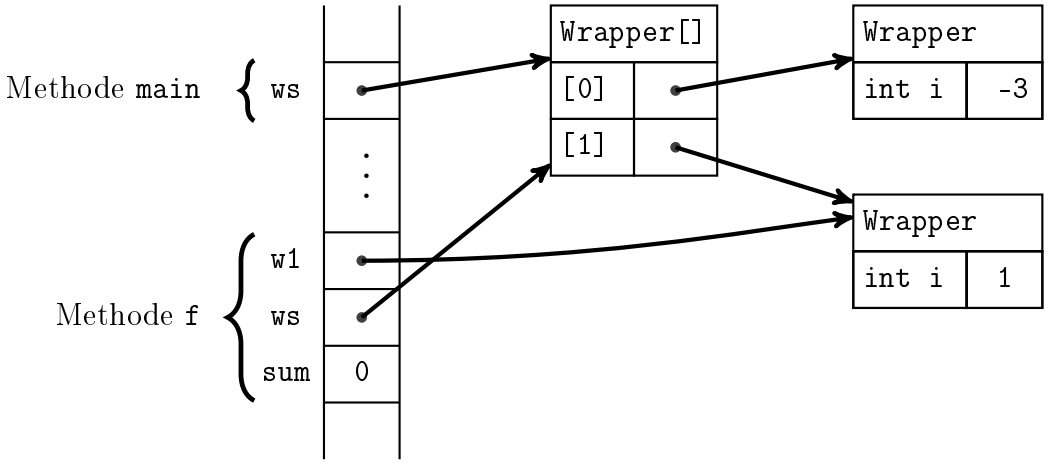
Betrachten Sie das folgende Programm:

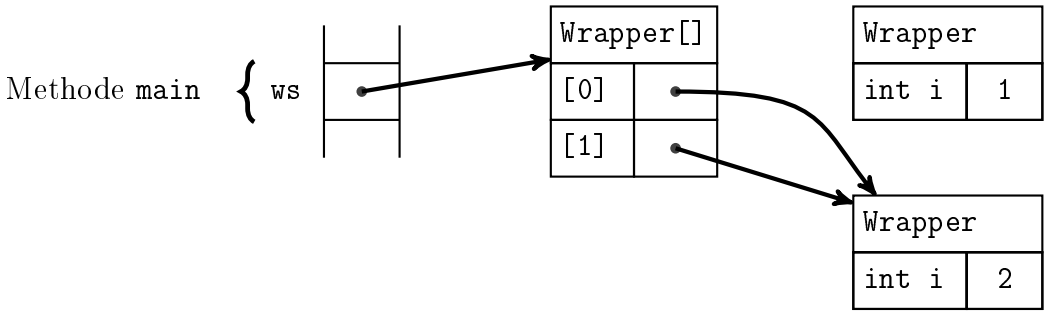
```
public class TSeiteneffekte {
    public static void main(String[] args) {
        Wrapper[] ws = new Wrapper[2];
        ws[0] = new Wrapper();
        ws[1] = new Wrapper();

        ws[0].setI(2);
        ws[1].setI(1);

        f(ws[1], ws[1], ws[0]);
        f(ws[1], ws);
        //Speicherzustand hier zeichnen
    }

    public static void f(Wrapper w1, Wrapper... ws) {
        int sum = 0;
        //Speicherzustand hier zeichnen
        for (int i = 0; i < ws.length; i++) {
            Wrapper w = ws[i];
            sum += w.getI();
            w.setI(i+1);
        }
        ws[0] = w1;
        w1 = ws[1];
        w1.setI(-sum);
    }
}

public class Wrapper {
    private int i;

    public void setI(int x) {
        i = x;
    }
    public int getI() {
        return i;
    }
}
```

Es wird nun die Methode `main` ausgeführt. Stellen Sie den Speicher (d.h. alle (implizit) im Programm vorkommenden Arrays (außer `args`) und Objekte) bei jedem Aufruf der Methode `f` nach der Deklaration von `sum` und vor Ende des Programms graphisch dar.

Lösung:


Methode main
ws
Methode f
w1
ws
sum
0
Wrapper[]
[0]
[1]
Wrapper
int i
2
Wrapper
int i
1
Wrapper[]
[0]
[1]


Methode main
ws
Methode f
w1
ws
sum
0
Wrapper[]
[0]
[1]
Wrapper
int i
-3
Wrapper
int i
1


Methode main
ws
Wrapper[]
[0]
[1]
Wrapper
int i
1
Wrapper
int i
2

Aufgabe 2 (Seiteneffekte): (6 Punkte)

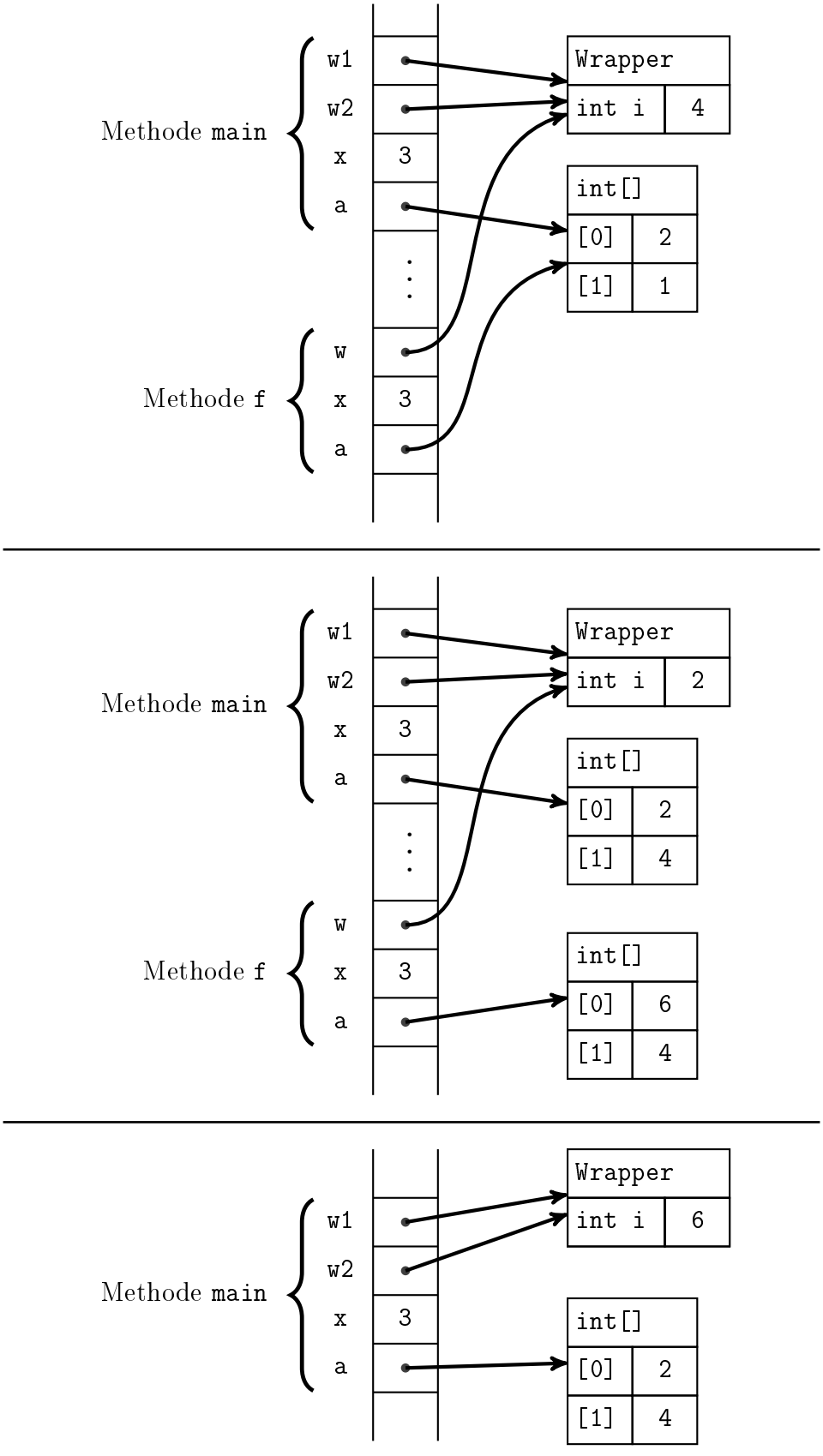
Betrachten Sie das folgende Programm:

```
public class HSeiteneffekte {
    public static void main(String[] args) {
        Wrapper w1 = new Wrapper();
        Wrapper w2 = w1;

        w1.setI(5);
        w2.setI(4);

        int x = 3;
        int[] a = { 2, 1 };

        f(w2, x, a);
        f(w1, x, 6, a[1]);
        //Speicherzustand hier zeichnen
    }

    public static void f(Wrapper w, int x, int... a) {
        //Speicherzustand hier zeichnen
        x = a[0];
        a[1] = w.getI();
        w.setI(x);
    }
}

public class Wrapper {
    private int i;

    public void setI(int x) {
        i = x;
    }
    public int getI() {
        return i;
    }
}
```

Es wird nun die Methode `main` ausgeführt. Stellen Sie den Speicher (d.h. alle (implizit) im Programm vorkommenden Arrays (außer `args`) und alle Objekte) an folgenden Programmstellen graphisch dar:

- bei jedem Aufruf der Methode `f`
- vor Ende der `main` Methode

Lösung:


Methode main
w1
w2
x
3
a
Methode f
w
x
3
a
Wrapper
int i
4
int[]
[0]
2
[1]
1
Methode main
w1
w2
x
3
a
Methode f
w
x
3
a
Wrapper
int i
2
int[]
[0]
2
[1]
4
int[]
[0]
6
[1]
4
Methode main
w1
w2
x
3
a
Wrapper
int i
6
int[]
[0]
2
[1]
4

Tutoraufgabe 3 (Kreise):

In dieser Aufgabe soll eine Klasse erstellt werden, mit der sich Kreise repräsentieren lassen. Ein solcher Kreis lässt sich mit den Koordinaten x und y für den Mittelpunkt und der Länge *radius* beschreiben, wobei x, y und *radius* Gleitkommazahlen sind. Der Radius eines Kreises darf nicht negativ sein.
Verwenden Sie die gegebene Klasse Punkt, um den Mittelpunkt darzustellen.
Ihre Implementierung sollte mindestens die folgenden Methoden beinhalten. Sie sollten dabei die Konzepte der Datenkapselung berücksichtigen. Hilfsmethoden müssen als `private` deklariert werden. In dieser Aufgabe dürfen Sie die in der Klasse `Utils` zur Verfügung gestellten Hilfsfunktionen, aber keine Bibliotheksfunktionen verwenden.

a) Erstellen Sie eine Klasse `Kreis` mit den Attributen `mittelpunkt` und `radius`.

b) Erstellen Sie die folgenden öffentlichen Methoden, um Objekte des Typs `Kreis` erzeugen zu können. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren:

```
Kreis newKreis(double x, double y, double radius)
Kreis clone()
```

Beachten Sie dabei folgende Punkte:

- Die Methode `newKreis` soll einen Kreis erzeugen, dessen Attribute jeweils die von den entsprechenden Parametern angegebenen Werte haben. Falls eines der Argumente einen unzulässigen Wert hat, muss eine geeignete Fehlermeldung ausgegeben und `null` zurückgeliefert werden. Zur Ausgabe einer Fehlermeldung kann die Methode `Utils.error(String msg)` genutzt werden.
- Die Methode `clone` soll einen Kreis kopieren.

c) Erstellen Sie die folgenden öffentlichen Selektoren, um die Koordinaten und den Radius eines Kreises auslesen zu können. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren:

```
double getX()
double getY()
double getRadius()
```

d) Erstellen Sie die folgenden öffentlichen Methoden, mit denen man Eigenschaften von Kreisen überprüfen kann. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren und begründen Sie Ihre Entscheidung kurz:

```
boolean contains(Kreis... others)
double size(Kreis... kreiss)
```

Bei den in diesem Aufgabenteil geforderten Methoden muss der Aufrufer (und nicht Sie als Implementierer der Klasse `Kreis`) sicherstellen, dass die Parameter, mit denen die Methoden aufgerufen werden, nicht den Wert `null` haben bzw. enthalten.

Beachten Sie dabei folgende Punkte:

- Die Methode `contains(Kreis... others)` prüft, ob alle in `others` enthaltenen Kreise *vollständig* innerhalb des Kreises liegen, auf dem die Methode aufgerufen wird.
  Hinweis: Es empfiehlt sich, eine Hilfsmethode zu implementieren, um zu überprüfen, ob *ein* Kreis in einem anderen enthalten ist.
- Die Methode `size(Kreis... kreise)` gibt die Summe der Flächen aller Kreise in `kreise` zurück.
  Hinweis: Es empfiehlt sich, eine Hilfsmethode zu implementieren, die die Größe *eines* Kreises berechnet.

e) Erstellen Sie ebenfalls eine Implementierung für die öffentliche Methode

```
String toString()
```

Entscheiden Sie dabei selbst, ob Sie die Methode als statisch deklarieren. Die Methode `toString()` erstellt eine textuelle Repräsentation eines Kreises, aus der die Koordinaten und der Radius hervorgehen. Zum Beispiel stellt der String `(10|11),12` den Kreis mit den Koordinaten 10 und 11 und dem Radius 12 dar. Kann die toString() Methode des Mittelpunktes verwendet werden?

f) Dokumentieren Sie alle Methoden, die als `public` markiert sind, indem Sie die Implementierung mit Javadoc-Kommentaren ergänzen. Diese Kommentare sollten eine allgemeine Erklärung der Methode sowie weitere Erklärungen jedes Parameters und des `return`-Wertes enthalten. Verwenden Sie innerhalb des Kommentars dafür die Javadoc-Anweisungen `@param` und `@return`.

Benutzen Sie das Programm `javadoc`, um Ihre Javadoc-Kommentare in das HTML-Format zu übersetzen. Überprüfen Sie mit einem Browser, ob das gewünschte Ergebnis generiert wurde.

g) Auf unserer Homepage finden Sie eine Klasse `XKreis`. Diese Klasse stellt Instanzen der Klasse `Kreis` in einer graphischen Benutzeroberfläche dar. Die graphische Benutzeroberfläche bietet die Möglichkeit, den dargestellten Kreis mit Hilfe der Pfeiltasten zu verschieben. Außerdem bietet sie die Möglichkeit, den dargestellten Kreis mit Hilfe von zwei Buttons zu vergrößern bzw. zu verkleinern. Damit die graphische Benutzeroberfläche benutzt werden kann, müssen Sie ein `enum` mit dem Namen `KreisAction` erstellen. Dieses `enum` muss die folgenden Elemente enthalten:

- `UP`, `DOWN`, `LEFT` und `RIGHT` repräsentieren das Verschieben des Kreises.
- `BIGGER` repräsentiert das Vergrößern des Kreises.
- `SMALLER` repräsentiert das Verkleinern des Kreises.

Außerdem müssen Sie in der Klasse `Kreis` die Methode

```
public void performAction(KreisAction action)
```

implementieren. Je nach Argument verringert bzw. vergrößert diese Methode den Radius des Kreises um den Wert `10` oder sie verschiebt den Kreis um den Wert `10` in die entsprechende Richtung. Benutzen Sie zur Implementierung der Methode `performAction` eine `switch`-Anweisung, um zu testen, welche Aktion das Argument repräsentiert. Ergänzen Sie auch für diese Methode einen geeigneten Javadoc-Kommentar. Achten Sie bei Ihrer Implementierung darauf, dass Kreise nicht beliebig klein werden können.

Nachdem Sie die Implementierung vervollständigt und alle Klassen mit `javac` kompiliert haben, können Sie die graphische Benutzeroberfläche mit `java XKreis` starten.

Lösung:

Listing 1: Kreis.java

```
/**
 * Ein Objekt der Klasse Kreis repraesentiert einen Kreis.
 */
public class Kreis {

    private Punkt mittelpunkt;
    private double radius;

    /**
     * Erzeugt einen neuen Kreis.
     * @param x der x-Anteil des Mittelpunktes
     * @param y der y-Anteil des Mittelpunktes
     * @param radius der nicht negative Radius
     * @return den neuen Kreis
     */
    public static Kreis newKreis(double x, double y, double radius) {
        if (radius < 0) {
            Utils.error("Trying to create kreis with negative radius " + radius + "!");
            return null;
        }
        Kreis res = new Kreis();
        res.mittelpunkt = new Punkt(x,y);
        res.radius = radius;
        return res;
    }

    /**
     * Erzeugt eine Kopie dieses Kreises.
     * @return eine Kopie dieses Kreis
     */
    public Kreis clone() {
        return newKreis(mittelpunkt.getX(), mittelpunkt.getY(), radius);
    }
```

```
    /**
     * Liefert den Radius dieses Kreises.
     * @return den Radius dieses Kreises
     */
    public double getRadius() {
        return radius;
    }

    /**
     * Setzt den Radius dieses Kreises.
     * @param radius der neue, nicht negative Radius dieses Kreises
     */
/*    public void setRadius(double radius) {
        if (radius < 0) {
            Utils.error("Trying to set radius to negative value " + radius + "!");
        } else {
            this.radius = radius;
        }
    }*/

    /**
     * Liefert den x-Anteil des Mittelpunktes dieses Kreises.
     * @return den x-Anteil des Mittelpunktes dieses Kreises
     */
    public double getX() {
        return mittelpunkt.getX();
    }

    /**
     * Setzt den x-Anteil des Mittelpunktes dieses Kreises.
     * @param x der neue x-Anteil des Mittelpunktes dieses Kreises
     */
/*    public void setX(double x) {
        this.x = x;
    }*/

    /**
     * Liefert den y-Anteil des Mittelpunktes dieses Kreises.
     * @return den y-Anteil des Mittelpunktes dieses Kreises
     */
    public double getY() {
        return mittelpunkt.getY();
    }

    /**
     * Setzt den y-Anteil des Mittelpunktes dieses Kreises.
     * @param y der neue y-Anteil des Mittelpunktes dieses Kreises
     */
/*    public void setY(double y) {
        this.y = y;
    }*/

    /*
     * Ueberprueft, ob der uebergebene Kreis in diesem Kreis enthalten ist.
     */
    private boolean containsOne(Kreis that) {
        double dist = mittelpunkt.distanz(that.mittelpunkt);
        return this.radius >= that.radius + dist;
    }

    // Aufgabenstellung suggeriert, dass die Methode nicht statisch sein kann.
    /**
     * Prueft, ob alle uebergebenen Kreise in diesem Kreis enthalten sind.
     * @param others jene Kreise, fuer die ueberprueft werden soll, ob sie in
     *               diesem Kreis enthalten sind
     * @return true falls alle uebergebenen Kreise in diesem Kreis enthalten sind,
     *         false sonst
     */
    public boolean contains(Kreis... others) {
        for (Kreis o : others) {
            if (!containsOne(o)) {
                return false;
            }
        }
        return true;
    }

    /*
     * Gibt die Flaeche dieses Kreises zurueck.
     */
    private double singleSize() {
        return radius * radius * Utils.PI;
    }
```

```
    /*
     * Begruendung fuer static: Alle Kreise spielen die gleiche Rolle, ohne static
     * suggeriert die Signatur eine "Sonderrolle" von this.
     */
    /**
     * Berechnet die Summe der Flaechen aller uebergebenen Kreise.
     * @param kreiss jene Kreise, deren Flaechen berechnet werden sollen
     * @return die Summe der Flaechen der uebergebenen Kreise
     */
    public static double size(Kreis... kreiss) {
        double res = 0;
        for (Kreis r : kreiss) {
            res += r.singleSize();
        }
        return res;
    }

    /**
     * Gibt eine String-Repraesentation dieses Kreises zurueck.
     * @return die String-Repraesentation dieses Kreises
     */
    public String toString() {
        return mittelpunkt + "," + this.radius;
    }

    /**
     * Fuehrt die als Argument uebergebene Aktion aus.
     * @param action die auszufuehrende Aktion
     */
    public void performAction(KreisAction action) {
        switch (action) {
            case UP:
                mittelpunkt.verschiebe(0,-10);
                break;
            case DOWN:
                mittelpunkt.verschiebe(0,10);
                break;
            case LEFT:
                mittelpunkt.verschiebe(-10,0);
                break;
            case RIGHT:
                mittelpunkt.verschiebe(10,0);
                break;
            case BIGGER:
                radius += 10;
                break;
            case SMALLER:
                if (radius >= 10)
                    radius -= 10;
                break;
            default:
                Utils.error("Unknown action " + action);
                break;
        }
    }
}
```

Listing 2: KreisAction.java

```
public enum KreisAction {

    UP, DOWN, LEFT, RIGHT, BIGGER, SMALLER;

}
```

Aufgabe 4 (Dreiecke): (1 + 2 + 3 + 6 + 2 + 3 + 3 + 3 = 23 Punkte)

In dieser Aufgabe soll eine Klasse erstellt werden, mit der sich Dreiecke repräsentieren lassen. Wir wählen das Koordinatensystem so, dass eine Ecke stets im Ursprung liegt. Ein solches Dreieck lässt sich also mit den Koordinaten x_1, y_1, x_2 and y_2 für die beiden anderen Ecken beschreiben wobei x_i, y_i Fliesskommazahlen (double) sind. Die drei Ecken eines Dreiecks dürfen nicht auf einer Geraden liegen. Behandeln Sie die Ecken als Punkte unter Verwendung der bereitgestellten Punkt-Klasse.
Ihre Implementierung sollte mindestens die folgenden Methoden beinhalten. Sie sollten dabei die Konzepte der Datenkapselung berücksichtigen. Hilfsmethoden müssen als `private` deklariert werden. In dieser Aufgabe dürfen Sie die in der Klasse `Utils` zur Verfügung gestellten Hilfsfunktionen, aber keine Bibliotheksfunktionen verwenden.

a) Erstellen Sie eine Klasse `Dreieck` mit den Attributen `ecke1`, `ecke2` und `ecke3`. Einer der Punkte ist implizit (0|0)

b) Erstellen Sie die folgenden öffentlichen Methoden, um Objekte des Typs `Dreieck` erzeugen zu können. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren:

```
Dreieck newDreieck(double x1, double y1, double x2, double y2)
Dreieck copy(Dreieck toCopy)
```

Beachten Sie dabei folgende Punkte:

- Die Methode `newDreieck` soll ein Dreieck erzeugen, dessen Ecken jeweils die von den entsprechenden Parametern angegebenen Werte haben. Falls eines der Argumente einen unzulässigen Wert hat, muss eine geeignete Fehlermeldung ausgegeben und `null` zurückgeliefert werden. Zur Ausgabe einer Fehlermeldung kann die Methode `Utils.error(String msg)` genutzt werden.
- Die Methode `copy` soll ein Dreieck kopieren.

c) Erstellen Sie die folgenden Selektoren, um Eckpunkte auslesen zu können. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren:

```
Punkt getEcke1()
Punkt getEcke2()
Punkt getEcke3()
```

d) Erstellen Sie die folgenden Methoden. Entscheiden Sie dabei selbst, welche Methoden Sie als statisch deklarieren und begründen Sie Ihre Entscheidung kurz:

```
Dreieck rotiere(double... Angles)
double perimeter(Dreieck... Dreiecks)
```

Bei den in diesem Aufgabenteil geforderten Methoden muss der Aufrufer (und nicht Sie als Implementierer der Klasse `Dreieck`) sicherstellen, dass die Parameter, mit denen die Methoden aufgerufen werden, nicht den Wert `null` haben bzw. enthalten.

Beachten Sie dabei folgende Punkte:

- Die Methode `rotiere(double... Angles)` soll ein Dreieck zurückgeben, das um die übergebenen Winkel um den Ursprung gedreht wurde.
  Hinweise: Es empfiehlt sich, eine Hilfsmethode zu implementieren, die *eine einzelne* Drehung eines Dreiecks berechnet. Sie können dazu die Methoden der Punkt-Klasse verwenden.

- Die Methode `perimeter(Dreieck... Dreiecks)` soll den Gesamtumfang aller Dreiecke zurückgeben.
  Hinweise: Es empfiehlt sich, eine Hilfsmethode zu implementieren, die den Umfang *eines* Dreiecks berechnet.

e) Erstellen Sie ebenfalls eine Implementierung für die öffentliche Methode

```
String toString()
```

Entscheiden Sie dabei selbst, ob Sie die Methode als statisch deklarieren. Die Methode `toString()` erstellt eine textuelle Repräsentation des aktuellen Dreiecks. Zum Beispiel stellt der String

```
#######__
_######__
__#####__
___####__
____####_
_____###_
______##_
_______#_
________#
```

das Dreieck mit den Koordinaten $(0|0), (6|0), (8|8)$ dar. Achten Sie darauf, dass Teile des Dreiecks aus dem sichtbaren Bereich verschwinden, wenn die x- oder die y-Koordinate einer Ecke negativ ist.

f) Erstellen Sie ebenfalls eine Implementierung für die öffentliche Methode

```
Kreis inkreis()
```

Entscheiden Sie dabei selbst, ob Sie die Methode als statisch deklarieren. Die Methode `inkreis()` erstellt eine Instanz eines `Kreis`, die dem Inkreis des Dreiecks entspricht.
Hinweis: Berechnung eines Inkreises finden Sie unter: `https://de.wikipedia.org/wiki/Inkreis#Inkreis_eines_Dreiecks`
Hinweis: Sie können die `Kreis` Klasse verwenden, die in der Tutoraufgabe erstellt wurde, oder Sie verwenden die bereitgestellte `Kreis` Klasse.

g) Dokumentieren Sie alle Methoden, die als `public` markiert sind, indem Sie die Implementierung mit Javadoc-Kommentaren ergänzen. Diese Kommentare sollten eine allgemeine Erklärung der Methode sowie weitere Erklärungen jedes Parameters und des `return`-Wertes enthalten. Verwenden Sie innerhalb des Kommentars dafür die Javadoc-Anweisungen `@param` und `@return`.

Benutzen Sie das Programm `javadoc`, um Ihre Javadoc-Kommentare in das HTML-Format zu übersetzen. Überprüfen Sie mit einem Browser, ob das gewünschte Ergebnis generiert wurde und senden Sie die HTML-Dateien an Ihren Tutor bzw. Tutorin. Bitte drucken Sie die generierten Dateien, der Umwelt zuliebe, *nicht* aus.

h) Auf unserer Homepage finden Sie eine Klasse `XDreieck`. Diese Klasse stellt Instanzen der Klasse `Dreieck` in einer graphischen Benutzeroberfläche dar. Die graphische Benutzeroberfläche bietet die Möglichkeit, das dargestellte Dreieck mit Hilfe der linken bzw. rechten Pfeiltasten zu rotieren. Außerdem bietet sie die Möglichkeit, das dargestellte Dreieck mit Hilfe von vier Buttons zu vergrößern bzw. zu verkleinern. Damit die graphische Benutzeroberfläche benutzt werden kann, müssen Sie ein `enum` mit dem Namen `DreieckAction` erstellen. Dieses `enum` muss folgende Elemente enthalten:

- `ROTATE_RIGHT` repräsentiert Rotieren nach rechts.
- `ROTATE_LEFT` repräsentiert Rotieren nach links.
- `NARROW` repräsentiert Verringern der Breite.
- `WIDEN` repräsentiert Vergrößern der Breite.
- `ENLARGE` repräsentiert Vergrößern des Dreiecks um 10%.
- `SHRINK` repräsentiert Verkleinern des Dreiecks um 10%.

Außerdem müssen Sie in der Klasse `Dreieck` die Methode

```
public void performAction(DreieckAction action)
```

implementieren. Je nach Argument verringert bzw. vergrößert diese Methode die Größe bzw. Weite des Dreieckes bzw. dreht das Dreieck um den Wert `10 %` bzw. `10 Grad`. Beachten Sie, dass das Dreieck nicht beliebig klein werden kann. Öffnen des Dreiecks bedeutet, dass eine Ecke fest bleibt, während die andere gedreht wird. Verwenden Sie für die Drehungen die Methode `rotiere` von `Punkt`. Benutzen Sie zur Implementierung der Methode `performAction` eine `switch`-Anweisung, um zu testen, welche Aktion das Argument repräsentiert. Ergänzen Sie auch für diese Methode einen geeigneten Javadoc-Kommentar.

Nachdem Sie die Implementierung vervollständigt und alle Klassen mit `javac` kompiliert haben, können Sie die graphische Benutzeroberfläche mit `java XDreieck` starten.

Lösung:

Listing 3: Dreieck.java

```
public class Dreieck {
  Punkt ecke1;
  Punkt ecke2;
  Punkt ecke3;
```

```
private Dreieck () {}

private Dreieck ( double x1 , double y1 , double x2 , double y2 ) {
  ecke1 = new Punkt(0,0);
  ecke2 = new Punkt(x1,y1);
  ecke3 = new Punkt(x2,y2);
}


  /**
   * Erzeugt ein neues Dreieck.
   * @param x1 der x-Anteil der ersten Ecke
   * @param y1 der y-Anteil der ersten Ecke
   * @param x2 der x-Anteil der zweiten Ecke
   * @param y2 der y-Anteil der zweiten Ecke
   * @return das neue Dreieck
   */
public static Dreieck newDreieck ( double x1 , double y1 , double x2 , double y2 ) {
  if (x1*y2 == x2*y1)
    return null;
  return new Dreieck (x1,y1,x2,y2);
}

/**
  * Erzeugt eine Kopie dieses Dreiecks.
  * @return eine Kopie dieses Dreiecks
  */
public Dreieck copy() {
  return newDreieck(ecke2.getX(), ecke2.getY(), ecke3.getX(), ecke3.getY());
}

/**
  * Liefert die erste Ecke dieses Dreiecks.
  * @return die erste Ecke dieses Dreiecks
  */
public Punkt getEcke1() {
  return ecke1;
}

/**
  * Liefert die zweite Ecke dieses Dreiecks.
  * @return die zweite Ecke dieses Dreiecks
  */
public Punkt getEcke2() {
  return ecke2;
}

/**
  * Liefert die dritte Ecke dieses Dreiecks.
  * @return die dritte Ecke dieses Dreiecks
  */
public Punkt getEcke3() {
  return ecke3;
}

private double umfang() {
  return ecke1.distanz(ecke2) + ecke2.distanz(ecke3) + ecke3.distanz(ecke1);
}

/**
  * Berechnet die Summe der Umfaenge aller uebergebenen Dreiecke.
  * @param dreiecks jene Dreiecke derren Umfang berechent werden soll
  * @return Summe der Umfaenge aller uebergebenen Dreiecke
  */
public static double perimeter(Dreieck ... dreiecks) {
  double res = 0;
  for (Dreieck d : dreiecks) {
      res += d.umfang();
  }
  return res;
}

private void rotate(double angle) {
  ecke2.rotiere(angle);
  ecke3.rotiere(angle);
}
private void rotate(int angle) {
  ecke2.rotiere(angle);
  ecke3.rotiere(angle);
}

/**
  * Berechnet die Summe der Umfaenge aller uebergebenen Dreiecke.
  * @param dreiecks jene Dreiecke derren Umfang berechent werden soll
```

```
    * @return Summe der Umfaenge aller uebergebenen Dreiecke
    */
  public Dreieck rotiere(double ... angles) {
    Dreieck res = copy();
    for (double a : angles) {
        res.rotate(a);
    }
    return res;
  }

  public double flaeche() {
    return Utils.abs(0.5*(ecke1.getX()*(ecke2.getY()-ecke3.getY())+ecke2.getX()*(ecke3.getY()-ecke1.getY())+ecke3.getX()*
  }

    private double minimalX() {
    return Utils.min(Utils.min(ecke1.getX(),ecke2.getX()),ecke3.getX());
  }

  private double minimalY() {
    return Utils.min(Utils.min(ecke1.getY(),ecke2.getY()),ecke3.getY());
  }

  private double maximalX() {
    return Utils.max(Utils.max(ecke1.getX(),ecke2.getX()),ecke3.getX());
  }

  private double maximalY() {
    return Utils.max(Utils.max(ecke1.getY(),ecke2.getY()),ecke3.getY());
  }

  private boolean inDreieck(double x, double y) {
    double dx1 = ecke2.getX()-ecke1.getX();
    double dy1 = ecke2.getY()-ecke1.getY();
    double dx2 = ecke3.getX()-ecke1.getX();
    double dy2 = ecke3.getY()-ecke1.getY();
    double a;
    double b;
    if (dx1 == 0) {
      b = (x - ecke1.getX()) / dx2;
      a = (dx2 * (y - ecke1.getY())  + dy2 * (ecke1.getX() - x)) / (dx2 * dy1);
    } else {
      b = (dx1 * (y - ecke1.getY()) + dy1 * (ecke1.getX() - x)) / (dx1 * dy2 - dx2 * dy1);
      a = (x - ecke1.getX() - b * dx2) / dx1;
    }
    return a >= 0 && b >= 0 && a + b <= 1;
  }

  /**
    * Berechnet den Inkreis zum aktuellen Dreieck.
    * @return Kreis, der dem Inkreis des Dreiecks entspricht
    */
  public Kreis inkreis() {
/* Laenge der gegenueberliegenden Kanten */
    double a1 = ecke2.distanz(ecke3);
    double a2 = ecke3.distanz(ecke1);
    double a3 = ecke1.distanz(ecke2);
    double P = (a1+a2+a3);
/* Radius: */
    double r = flaeche() * 2 / P;
/* Koordinaten: */
    double kx = (a1 * ecke1.getX() + a2 * ecke2.getX() + a3 * ecke3.getX()) / P ;
    double ky = (a1 * ecke1.getY() + a2 * ecke2.getY() + a3 * ecke3.getY()) / P ;
    return Kreis.newKreis(kx, ky, r);
  }

  private static Dreieck eingabe() {
    System.out.print("Geben Sie die X-Koordinate des 1. Punktes ein: ");
    double x1 = Double.parseDouble(System.console().readLine());
    System.out.print("Geben Sie die Y-Koordinate des 1. Punktes ein: ");
    double y1 = Double.parseDouble(System.console().readLine());
    System.out.print("Geben Sie die X-Koordinate des 2. Punktes ein: ");
    double x2 = Double.parseDouble(System.console().readLine());
    System.out.print("Geben Sie die Y-Koordinate des 2. Punktes ein: ");
    double y2 = Double.parseDouble(System.console().readLine());
    return newDreieck(x1,y1,x2,y2);
  }

  private void ausgabe() {
    System.out.println("Dreieck "+ecke1+", "+ecke2+", "+ecke3);
  }

  /**
   * Fuehrt die als Argument uebergebene Aktion aus.
   * @param action die auszufuehrende Aktion
```

```
     */
    public void performAction(DreieckAction action) {
        switch (action) {
            case ROTATE_LEFT:
                rotate(-10);
                break;
            case ROTATE_RIGHT:
                rotate(10);
                break;
            case ENLARGE:
                ecke1 = new Punkt(ecke1.getX()*1.1, ecke1.getY()*1.1);
                ecke2 = new Punkt(ecke2.getX()*1.1, ecke2.getY()*1.1);
                ecke3 = new Punkt(ecke3.getX()*1.1, ecke3.getY()*1.1);
                break;
            case SHRINK:
                ecke1 = new Punkt(ecke1.getX()*.9, ecke1.getY()*.9);
                ecke2 = new Punkt(ecke2.getX()*.9, ecke2.getY()*.9);
                ecke3 = new Punkt(ecke3.getX()*.9, ecke3.getY()*.9);
                break;
            case NARROW:
                ecke2.rotiere(-10);
                break;
            case WIDEN:
                ecke2.rotiere(10);
                break;
            default:
                Utils.error("Unknown action " + action);
                break;
        }
    }

    /**
      * Gibt eine String-Repraesentation dieses Dreiecks zurueck.
      * @return die String-Repraesentation dieses Dreiecks
      */
    public String toString() {
      String ret = "";
      for (int y = 0; y <= maximalY(); y++) {
        for (int x = 0; x <= maximalX(); x++) {
          if (inDreieck(x,y)) {
            ret += "#";
          } else {
            ret += "_";
          }
        }
        ret += "\n";
      }
      return ret;
    }

    public static void main(String[] args) {
      Dreieck d = eingabe();
      if (d==null) {
        Utils.error("Ungueltiges Dreieck ");
        return;
      }
      System.out.println(d.toString());
      d.ausgabe();
      System.out.println("Umfang: " + d.umfang());
      System.out.println("Inkreis: " + d.inkreis());
    }

}
```

Listing 4: DreieckAction.java

```
public enum DreieckAction {

    ROTATE_LEFT, ROTATE_RIGHT, ENLARGE, SHRINK, NARROW, WIDEN;

}
```