

It is the final Count—Zaunuuuun

Das Ende. Die 13 (Natürlich...)

Florian Sihler ◦ KW 7



Übungsblatt 13 - Aufgabe 1

Übungsblatt 13 - Aufgabe 1

Gegeben sei folgende abstrakte Klasse `RegularPolygon`, welche ein Polygon mit gleichen Seitenlängen bzw. Innenwinkeln repräsentieren soll:

Übungsblatt 13 - Aufgabe 1

Gegeben sei folgende abstrakte Klasse `RegularPolygon`, welche ein Polygon mit gleichen Seitenlängen bzw. Innenwinkeln repräsentieren soll:

```
public abstract class RegularPolygon {  
    protected final int numSides;  
    protected final double sideLength;  
  
    public RegularPolygon(int numSides, double sideLength) {  
        this.numSides = numSides;  
        this.sideLength = sideLength;  
    }  
  
    public int getNumSides() { return this.numSides; }  
    public double getCircumference() { return this.numSides * this.sideLength; }  
    public abstract double getArea();  
}
```

Übungsblatt 13 - Aufgabe 1

Übungsblatt 13 - Aufgabe 1

1. Implementieren Sie nun die zwei Klassen `EquilateralTriangle` und `Square`, welche ein gleichseitiges Dreieck bzw. ein Quadrat repräsentieren sollen. Beide Klassen sollen jeweils von `RegularPolygon` erben, und müssen deshalb die Methode `public double getArea()` definieren. Zusätzlich sollen beide Klassen einen Konstruktor anbieten, welcher die Seitenlänge als Parameter übernimmt ($A_{\triangle} = \frac{1}{4} \cdot \sqrt{3} \cdot a^2$).

Übungsblatt 13 - Aufgabe 1

1. Implementieren Sie nun die zwei Klassen `EquilateralTriangle` und `Square`, welche ein gleichseitiges Dreieck bzw. ein Quadrat repräsentieren sollen. Beide Klassen sollen jeweils von `RegularPolygon` erben, und müssen deshalb die Methode `public double getArea()` definieren. Zusätzlich sollen beide Klassen einen Konstruktor anbieten, welcher die Seitenlänge als Parameter übernimmt ($A_{\triangle} = \frac{1}{4} \cdot \sqrt{3} \cdot a^2$).
2. Wenn Sie die abstrakte Methode `public double getArea()` in den Subklassen implementieren, wird diese Methode dann überladen oder überschrieben? Begründen Sie ihre Antwort!

Übungsblatt 13 - Aufgabe 1

1. Implementieren Sie nun die zwei Klassen `EquilateralTriangle` und `Square`, welche ein gleichseitiges Dreieck bzw. ein Quadrat repräsentieren sollen. Beide Klassen sollen jeweils von `RegularPolygon` erben, und müssen deshalb die Methode `public double getArea()` definieren. Zusätzlich sollen beide Klassen einen Konstruktor anbieten, welcher die Seitenlänge als Parameter übernimmt ($A_{\triangle} = \frac{1}{4} \cdot \sqrt{3} \cdot a^2$).
2. Wenn Sie die abstrakte Methode `public double getArea()` in den Subklassen implementieren, wird diese Methode dann überladen oder überschrieben? Begründen Sie ihre Antwort!
3. Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden? Falls ja, geben Sie ein entsprechendes Interface an, falls nein, begründen Sie warum dies nicht möglich ist.

Übungsblatt 13 - Aufgabe 1

1. Implementieren Sie nun die zwei Klassen `EquilateralTriangle` und `Square`, welche ein gleichseitiges Dreieck bzw. ein Quadrat repräsentieren sollen. Beide Klassen sollen jeweils von `RegularPolygon` erben, und müssen deshalb die Methode `public double getArea()` definieren. Zusätzlich sollen beide Klassen einen Konstruktor anbieten, welcher die Seitenlänge als Parameter übernimmt ($A_{\triangle} = \frac{1}{4} \cdot \sqrt{3} \cdot a^2$).
2. Wenn Sie die abstrakte Methode `public double getArea()` in den Subklassen implementieren, wird diese Methode dann überladen oder überschrieben? Begründen Sie ihre Antwort!
3. Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden? Falls ja, geben Sie ein entsprechendes Interface an, falls nein, begründen Sie warum dies nicht möglich ist.
4. In der Vorlage wird der Modifier `protected` für die Instanzvariablen verwendet. Könnte hier auch der Modifier `private` verwendet werden? Begründen Sie ihre Antwort!

Übungsblatt 13 - Aufgabe 1

Übungsblatt 13 - Aufgabe 1

5. Welche der Aufrufe in folgendem Testprogramm würden zu einem Compiler-Fehler führen und welche sind korrekt? Begründen Sie ihre Antwort! (Nehmen Sie für diese Teilaufgabe an, dass die beiden Klassen `Square` und `EquilateralTriangle` korrekt implementiert wurden.)

Übungsblatt 13 - Aufgabe 1

5. Welche der Aufrufe in folgendem Testprogramm würden zu einem Compiler-Fehler führen und welche sind korrekt? Begründen Sie ihre Antwort! (Nehmen Sie für diese Teilaufgabe an, dass die beiden Klassen `Square` und `EquilateralTriangle` korrekt implementiert wurden.)

```
public class PolygonTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3); // A 1  
        EquilateralTriangle triangle = (EquilateralTriangle)polygon; // A 2  
        Square square = new RegularPolygon(4, 4); // A 3  
    }  
}
```

Übungsblatt 13 - Aufgabe 1

Übungsblatt 13 - Aufgabe 1

6. Würden die folgenden Aufrufe der Methode `getArea` funktionieren? Falls ja, welche Werte würden entsprechend ausgegeben werden? Begründen Sie in jedem Fall ihre Antwort! (Nehmen Sie an, dass die beiden Klassen `Square` und `EquilateralTriangle` korrekt implementiert wurden.)

Übungsblatt 13 - Aufgabe 1

6. Würden die folgenden Aufrufe der Methode `getArea` funktionieren? Falls ja, welche Werte würden entsprechend ausgegeben werden? Begründen Sie in jedem Fall ihre Antwort! (Nehmen Sie an, dass die beiden Klassen `Square` und `EquilateralTriangle` korrekt implementiert wurden.)

```
public class PolygonAreaTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3);  
        System.out.println(polygon.getArea());  
  
        polygon = new Square(2);  
        System.out.println(polygon.getArea());  
    }  
}
```


- Gimme those classes ([EquilateralTriangle.java](#) und [Square.java](#)):

- Gimme those classes (`EquilateralTriangle.java` und `Square.java`):

```
public class EquilateralTriangle extends RegularPolygon {
```

$$\}$$

- Gimme those classes (EquilateralTriangle.java und Square.java):

```
public class EquilateralTriangle extends RegularPolygon {  
    public EquilateralTriangle(int sideLength) {  
  
    }  
  
}
```

- Gimme those classes (EquilateralTriangle.java und Square.java):

```
public class EquilateralTriangle extends RegularPolygon {  
    public EquilateralTriangle(int sideLength) {  
        super(3, sideLength);  
    }  
  
}
```

- Gimme those classes (EquilateralTriangle.java und Square.java):

```
public class EquilateralTriangle extends RegularPolygon {  
    public EquilateralTriangle(int sideLength) {  
        super(3, sideLength);  
    }  
  
    @Override  
    public double getArea() {  
  
    }  
}
```

- Gimme those classes (EquilateralTriangle.java und Square.java):

```
public class EquilateralTriangle extends RegularPolygon {  
    public EquilateralTriangle(int sideLength) {  
        super(3, sideLength);  
    }  
  
    @Override  
    public double getArea() {  
        return 0.25 * Math.sqrt(3) * sideLength * sideLength;  
    }  
}
```



```
public class Square extends RegularPolygon {
```

$$\}$$


```
public class Square extends RegularPolygon {  
    public Square(int sideLength) {  
  
    }
```

```
}
```

```
public class Square extends RegularPolygon {  
    public Square(int sideLength) {  
        super(4, sideLength);  
    }  
  
}
```

```
public class Square extends RegularPolygon {  
    public Square(int sideLength) {  
        super(4, sideLength);  
    }  
  
    @Override  
    public double getArea() {  
  
    }  
}
```

```
public class Square extends RegularPolygon {  
    public Square(int sideLength) {  
        super(4, sideLength);  
    }  
  
    @Override  
    public double getArea() {  
        return sideLength * sideLength;  
    }  
}
```

Übungsblatt 13 - Aufgabe 1 b-d)

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)
(siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)
(siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?
Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)
(siehe [JLS17 8.4.8.1](#) für die genauen Regeln).
- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?
Das ist aus zwei Gründen nicht möglich:

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)
(siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

Das ist aus zwei Gründen nicht möglich:

1. Die (abstrakte) Klasse hat einen Konstruktor.

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)
(siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

Das ist aus zwei Gründen nicht möglich:

1. Die (abstrakte) Klasse hat einen Konstruktor.
2. Die (abstrakte) Klasse enthält Instanzvariablen.

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)
(siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

Das ist aus zwei Gründen nicht möglich:

1. Die (abstrakte) Klasse hat einen Konstruktor.
2. Die (abstrakte) Klasse enthält Instanzvariablen.

Schnittstellen definieren das Verhalten. Keinen Zustand!

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?

Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar)
(siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

Das ist aus zwei Gründen nicht möglich:

1. Die (abstrakte) Klasse hat einen Konstruktor.
2. Die (abstrakte) Klasse enthält Instanzvariablen.

Schnittstellen definieren das Verhalten. Keinen Zustand!

- Es wird der Modifier **protected** für die Instanzvariablen verwendet. Ginge auch **private**?

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?
Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar) (siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

Das ist aus zwei Gründen nicht möglich:

1. Die (abstrakte) Klasse hat einen Konstruktor.
2. Die (abstrakte) Klasse enthält Instanzvariablen.

Schnittstellen definieren das Verhalten. Keinen Zustand!

- Es wird der Modifier **protected** für die Instanzvariablen verwendet. Ginge auch **private**?
Für `numSides` ist dies möglich, diese Variable wird nur in `RegularPolygon` gebraucht.

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?
Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar) (siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?

Das ist aus zwei Gründen nicht möglich:

1. Die (abstrakte) Klasse hat einen Konstruktor.
2. Die (abstrakte) Klasse enthält Instanzvariablen.

Schnittstellen definieren das Verhalten. Keinen Zustand!

- Es wird der Modifier `protected` für die Instanzvariablen verwendet. Ginge auch `private`?
Für `numSides` ist dies möglich, diese Variable wird nur in `RegularPolygon` gebraucht. Für `sideLength` allerdings nicht, wir benötigen diese für `getArea()`.

Übungsblatt 13 - Aufgabe 1 b-d)

- Ist die Implementation von `getArea()` Überladung oder Überschreiben?
Überschreiben: die „implementierende“ Methode hat die selbe Signatur (und ist sichtbar) (siehe [JLS17 8.4.8.1](#) für die genauen Regeln).

- Könnte die abstrakte Klasse auch durch ein Interface ersetzt werden?
Das ist aus zwei Gründen nicht möglich:

1. Die (abstrakte) Klasse hat einen Konstruktor.
2. Die (abstrakte) Klasse enthält Instanzvariablen.

Schnittstellen definieren das Verhalten. Keinen Zustand!

- Es wird der Modifier **protected** für die Instanzvariablen verwendet. Ginge auch **private**?
Für `numSides` ist dies möglich, diese Variable wird nur in `RegularPolygon` gebraucht. Für `sideLength` allerdings nicht, wir benötigen diese für `getArea()`.
(Man kööönnte aber in diesem speziellen Fall auch eine neue Variable machen...)

Übungsblatt 13 - Aufgabe 1 e)

Übungsblatt 13 - Aufgabe 1 e)

```
public class PolygonTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3); // A 1  
        EquilateralTriangle triangle = (EquilateralTriangle)polygon; // A 2  
        Square square = new RegularPolygon(4, 4); // A 3  
    }  
}
```

Übungsblatt 13 - Aufgabe 1 e)

```
public class PolygonTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3); // A 1  
        EquilateralTriangle triangle = (EquilateralTriangle)polygon; // A 2  
        Square square = new RegularPolygon(4, 4); // A 3  
    }  
}
```

1. Das ist fein, da `EquilateralTriangle` von `RegularPolygon` erbt und damit eine Spezialisierung darstellt! (Polymorphie)

Übungsblatt 13 - Aufgabe 1 e)

```
public class PolygonTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3); // A 1  
        EquilateralTriangle triangle = (EquilateralTriangle)polygon; // A 2  
        Square square = new RegularPolygon(4, 4); // A 3  
    }  
}
```

1. Das ist fein, da `EquilateralTriangle` von `RegularPolygon` erbt und damit eine Spezialisierung darstellt! (Polymorphie)
2. Durch die explizite Typkonvertierung ist dies fein. Implizit würde dies nicht funktionieren! So geht das aber, da `polygon` tatsächlich auf eine Instanz vom Typ `EquilateralTriangle` zeigt.

Übungsblatt 13 - Aufgabe 1 e)

```
public class PolygonTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3); // A 1  
        EquilateralTriangle triangle = (EquilateralTriangle)polygon; // A 2  
        Square square = new RegularPolygon(4, 4); // A 3  
    }  
}
```

1. Das ist fein, da `EquilateralTriangle` von `RegularPolygon` erbt und damit eine Spezialisierung darstellt! (Polymorphie)
2. Durch die explizite Typkonvertierung ist dies fein. Implizit würde dies nicht funktionieren! So geht das aber, da `polygon` tatsächlich auf eine Instanz vom Typ `EquilateralTriangle` zeigt.
3. Böse, wir können keine Instanz einer abstrakten Klasse erzeugen!

Übungsblatt 13 - Aufgabe 1 f)

Übungsblatt 13 - Aufgabe 1 f)

```
public class PolygonAreaTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3);  
        System.out.println(polygon.getArea());  
  
        polygon = new Square(2);  
        System.out.println(polygon.getArea());  
    }  
}
```

Übungsblatt 13 - Aufgabe 1 f)

```
public class PolygonAreaTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3);  
        System.out.println(polygon.getArea());  
  
        polygon = new Square(2);  
        System.out.println(polygon.getArea());  
    }  
}
```

- Beide fein! Durch *dynamische Bindung* wird `EquilateralTriangle::getArea` und `Square::getArea` für `polygon.getArea()` aufgerufen.

Übungsblatt 13 - Aufgabe 1 f)

```
public class PolygonAreaTest {  
    public static void main(String[] args) {  
        RegularPolygon polygon = new EquilateralTriangle(3);  
        System.out.println(polygon.getArea());  
  
        polygon = new Square(2);  
        System.out.println(polygon.getArea());  
    }  
}
```

- Beide fein! Durch *dynamische Bindung* wird `EquilateralTriangle::getArea` und `Square::getArea` für `polygon.getArea()` aufgerufen. So wird zunächst der Flächeninhalt des Dreiecks (3.87911...) und dann der Flächeninhalt des Quadrates (4) ausgegeben.

This is it.

This is it.

Aus.

This is it.

Aus.

Vorbei.

This is it.

Aus.

Vorbei.

Noch Fragen?

This is it.

Aus.

Vorbei.

Noch Fragen? Viel Erfolg!



Org



Org



0



Org



0



1



Org



0



1



2



Org



0



1



2



3



Org



0



1



2



3



4



Org



0



1



2



3



4



5

Motivation!



Org



0



1



2



3



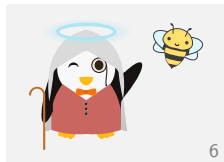
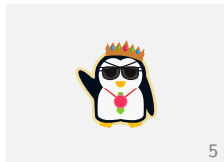
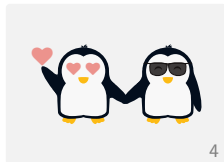
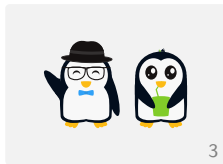
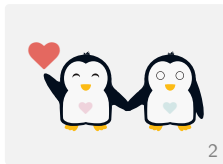
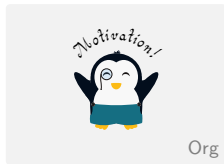
4



5



6



Motivation!



Org



0



1



2



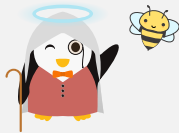
3



4



5



6



7



8

Motivation!



Org



0



1



2



3



4



5



6



7



8



9

Motivation!



Org



0



1



2



3



4



5



6



7



8



9



10

Motivation!



Org



0



1



2



3



4



5



6



7



8



9



10



11

Motivation!



Org



0



1



2



3



4



5



6



7



8



9



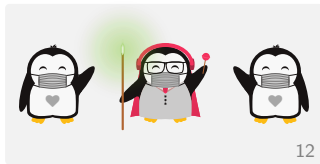
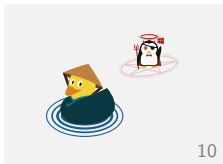
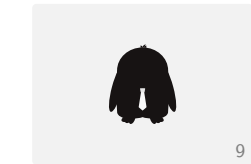
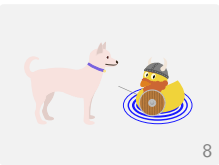
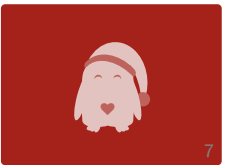
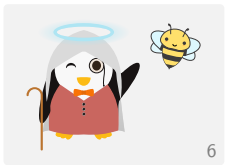
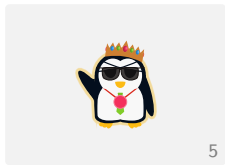
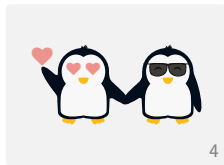
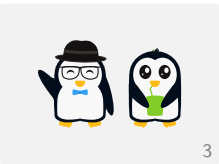
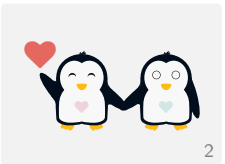
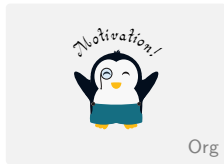
10



11



12





Hat Spaß gemacht.



Hat Spaß gemacht. Viel Erfolg an der Klausur!



Hat Spaß gemacht. Viel Erfolg an der Klausur! Bis dann und... *waddle on!*



Hat Spaß gemacht. Viel Erfolg an der Klausur! Bis dann und... waddle on!