Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

Programmierung 1
WiSe 25/26
Moritz Balg
15.12.2025

Übungsblatt 11

- Vererbung und Substitutionsprinzip -

Aufgabe 1: Weihnachtsbaum

Schreiben Sie eine Klassenmethode `public static void weihnachtsbaum(int n)`. Diese Methode soll einen Weihnachtsbaum mit n Reihen aus Sternen (*) auf der Konsole ausgeben (siehe Beispiel).

1. Der Baum besteht aus n Ebenen, wobei jede Ebene aus einer ungeraden Anzahl von Sternen besteht.
   - In der ersten Ebene steht ein Stern (*-Symbol).
   - In jeder folgenden Ebene erhöht sich die Anzahl der Sterne um 2.
2. Der Baum soll mittig ausgerichtet sein.
   - Dazu muss vor jeder Ebene eine passende Anzahl von Leerzeichen ausgegeben werden, sodass alle Zeilen die gleiche Gesamtbreite besitzen.
3. Nach den Ebenen soll ein Baumstamm ausgegeben werden, bestehend aus einem einzelnen senkrechten Strich (|-Symbol).
   - Auch dieser soll zentriert unter dem Baum stehen.

Info

Entwickeln Sie zuerst eine Hilfsmethode `public static String wiederhole(char c, int n)`, die einen übergebenen Charakter n-mal hintereinander in einem String zurückgibt.

Beispiel

Der Weihnachtbaum für $n = 5$ sieht sieht wie folgt aus:

```
    *
   ***
  *****
 *******
*********
    |
```

Aufgabe 2: Abstrakte Klassen

a) Gegeben sind folgende Klassen. Ändern Sie die Klassenstruktur und Klassen so, dass eine gemeinsame, nicht instanziierbare Oberklasse mit sinnvollem Inhalt existiert und die beiden instanziierbaren Klassen davon abgeleitet werden. Die Gesamtfunktionalität der beiden Klassen soll erhalten bleiben. Geben Sie in ihrer Lösung alle Klassen komplett an.

```
class Birne {
    static int anzahl;        // verkaufte Birnen
    int gewicht;              // Gewicht in Gramm

    Birne() {
        this(95);             // StandardBirne
    }
```

```
    Birne(int gewicht) {
        anzahl++;
        this.gewicht = gewicht;
    }
}

class Apfel {
    static int anzahl;        // verkaufte Äpfel
    int gewicht;              // Gewicht in Gramm

    Apfel() {
        this(65);
    }

    Apfel(int gewicht) {
        anzahl++;
        this.gewicht = gewicht;
    }
}
```

b) Machen Sie Äpfel untereinander vergleichbar, indem Sie die `equals`-Methode aus der Klasse `Object` überschreiben. Machen Sie zusätzlich auch Birnen untereinander vergleichbar. Äpfel und Birnen gelten als gleich, wenn Sie das gleiche Gewicht haben.

Achtung

Achten Sie darauf, dass Sie nicht Äpfel mit Birnen vergleichen!

Beispiel

```
public static void main(String[] args){
   Birne b1 = new Birne(101);
   Birne b2 = new Birne();
   Birne b3 = new Birne();
   Apfel a1 = new Apfel(101);

   System.out.println(b1.equals(b2)); //false;
   System.out.println(b2.equals(b3)); //true;
   System.out.println(a1.equals(a1)); //true;
   System.out.println(b1.equals(a1)); //false;
   System.out.println(b1.equals(null)); //false;
 }
```

Aufgabe 3: Zahlen

Definieren Sie eine abstrakte Klasse Zahl, die also nicht instanziierbar sein soll. In der Klasse sollen abstrakte Instanzmethoden für die vier Grundrechenarten definiert werden: *addieren,subtrahieren, multiplizieren, dividieren*. Ein Argument einer solchen Operation ist die Instanz selber, das zweite Argument der Operation ist ein Parameter der Methode vom Typ Zahl. Das Resultat einer Methode soll eine (neue) Zahl sein (also ein neues Zahl-Objekt). Beispiel: `public abstract Zahl addieren(Zahl z)`.Weiterhin soll es eine abstrakte Methode `toString` geben mit der gleichen Signatur wie in Object.

Geben Sie aufbauend auf dieser abstrakten Klasse Zahl zwei konkrete Klassen an: `ReelleZahl` und `KomplexeZahl`, die jeweils die vier Operationen konkret umsetzen mit der entsprechenden Bedeutung sowie die toString-Methode. Die toString Methode der rellen Klasse soll einfach die entsprechende Zahl als String liefern (Beispiel: 4.1). Die toString Methode der komplexen Klasse soll eine Ausgabe erzeugen der Form $(a +$

$b * i$), wobei a und b jeweils reelle Zahldarstellungen sind (Beispiel: $14.0 + 8.0 * i$). Reelle Zahlen sind in ihrer Bedeutung bekannt. Eine Instanz der Klasse ReelleZahl besitzt also eine Instanzvariable mit einer reellen Zahl vom Typ double. Über einen Konstruktor ReelleZahl(double r) kann man ein Objekt dieses Typs erzeugen. Komplexe Zahlen bestehen aus zwei Teilen, dem Realteil a und dem Imaginärteil b, a und b sind jeweils reelle Zahlen. Nutzen Sie zur Darstellung von a und b nicht ihre Klasse ReelleZahl, sondern ganz normal double. Sehen Sie auch hier einen Konstruktor vor, der zwei Argumente für a und b nimmt. Die übliche Notation für eine komplexe Zahl ist $a + b * i$ mit Realteil a und Imaginärteil b. Die Rechenregeln zu zwei komplexen Zahlen sind wie folgt, wobei $a + b * i$ und $c + d * i$ zwei komplexe Zahlen sind:

- $(a + b * i) + (c + d * i) = (a + c) + (b + d) * i$
- $(a + b * i) - (c + d * i) = (a - c) + (b - d) * i$
- $(a + b * i) * (c + d * i) = (\mathrm{ac} - \mathrm{bd}) + (\mathrm{ad} + \mathrm{bc}) * i$
- $(a + b * i)/(c + d * i) = \frac{\mathrm{ac}+\mathrm{bd}}{c^2+d^2} + \frac{\mathrm{bc}-\mathrm{ad}}{c^2+d^2} * i$

Beispiel

$(3 + 1 * i) * (5 + 1 * i) = (3 * 5 - 1 * 1) + (3 * 1 + 1 * 5) * i = 14 + 8 * i$

Sie können davon ausgehen, dass eine Instanz des Typs **`ReelleZahl`** bzw. **`KomplexeZahl`** nur mit einer Instanz des gleichen Typs kombiniert wird und keine Mischung auftritt.

Machen Sie sich sorgfältig Gedanken, wie Sie die abstrakten Methoden, die ja auf dem Typ Zahl arbeiten, in den konkreten Klassen umsetzen. Hinweis: Cast-Operator.

Testen Sie anschließend ihre Implementierung mit dem folgenden Programm. Dieses Testprogramm sollen Sie nicht in den Praktomat hochladen, es dient nur für Sie als Test:

```
public static void main(String[] args) {
    Zahl z1 = new ReelleZahl(3.0);
    Zahl z2 = new ReelleZahl(5.0);

    System.out.println(z1.multiplizieren(z2));

    z1 = new KomplexeZahl(3.0, 1.0);
    z2 = new KomplexeZahl(5.0, 1.0);

    System.out.println(z1.multiplizieren(z2));
}
```

Freiwillige Erweiterung für zuhause: Schreiben Sie eine Variante der Klasse KomplexeZahl, in der Sie zur Speicherung des Real- und Imaginärteils ihre eigene Klasse ReelleZahl nutzen. Verwenden Sie auch für die Berechnungen in der komplexen Klasse die in der Klasse ReelleZahl definierten Methoden.

Aufgabe 4: Schnittstellen und dynamische Bindung

Erweitern Sie Ihre Lösung zu der Zahlen-Aufgabe, indem Sie folgendes Interface vorsehen:

```
public interface Vergleichbar {

    /**
     * Toleranzschwelle für Vergleiche von Zahlen.
     */
    final double epsilon = 1e-12;

    /**
     * Vergleicht dieses Objekt mit der angegebenen Zahl.
     * Die Implementierung soll folgendes liefern:
     *
```

```
     *     0: wenn die Werte innerhalb der Toleranz epsilon sind</li>
     *     1: wenn dieses Objekt größer ist als z</li>
     *    -1: wenn dieses Objekt kleiner ist als z</li>
     *
     * @param z die Zahl, mit der verglichen werden soll
     * @return ein Integer entsprechend der Vergleichsrelation
     */
    public int vergleicheMit(Zahl z);
}
```

Erweitern Sie ihre Zahlklassen so, dass diese Schnittstelle in den konkreten Klassen jeweils anwendbar ist, also Zahlen vergleichbar sind. Realisieren Sie diese Funktion nur in der abstrakten Oberklasse, wobei Sie auf Methoden in den abgeleiteten Klassen zurückgreifen können (siehe nachfolgende Beschreibung). Zwei Zahlen x und y können allgemein wie folgt verglichen werden:

$$\text{vergleicheMit(x,y)} := \begin{cases} 0, & \text{wenn } |x| - |y| < \varepsilon \wedge |x| - |y| > -\varepsilon \\ 1, & \text{wenn } |x| - |y| > \varepsilon \\ -1, & \text{sonst} \end{cases}$$

Für eine komplexe Zahl $a + b * i$ ist der Betrag definiert durch: $|a + b * i| = \sqrt{a^2 + b^2}$.

Beispiel

```
Zahl z1 = new KomplexeZahl(3.0, 1.0);
Zahl z2 = new KomplexeZahl(5.0, 1.0);

z1.vergleicheMit(new KomplexeZahl(3.0, 1.0 + 1e-14)); //Ergebnis: 0
z1.vergleicheMit(new KomplexeZahl(3.0, 1.0 + 1e-5)); //Ergebnis: -1
z2.vergleicheMit(z1); // Ergebnis: 1
```

Aufgabe 5: Anonyme Klasse 👾

Nehmen Sie folgende Klassen und Schnittstelle als Basis:

```
public class TestKlasse2 {

    public static void main(String[] args) {
        System.out.println(gibZahl(4711));
    }

    // in dieser Methode dürfen Sie Ergänzungen vornehmen
    public static String gibZahl(int wert) {
        ZahlKlasse z = new ZahlKlasse(wert);
        return z.toString();
    }
}

// ab hier keine Änderungen erlaubt

public interface ZahlFunktionalitaet {
    public int getZahl();
    public String toString();
}

public abstract class ZahlKlasse implements ZahlFunktionalitaet {

    private int zahl;
```

```
    public ZahlKlasse(int zahl) {
        this.zahl = zahl;
    }

    public int getZahl() {
        return zahl;
    }

    public String toString() {
        return " " + zahl;
    }
}
```

Dieser Code ist so nicht übersetzbar, weil in main versucht wird, ein Objekt der Klasse ZahlKlasse zu instanziieren, was aber aufgrund der Eigenschaft, dass diese Klasse abstrakt ist, nicht möglich ist. An der Schnittstelle sowie an der Klasse Zahlklasse darf jetzt aber nichts geändert werden (zum Beispiel, weil dies zwischen Ihnen und einem zweiten Programmierer so vereinbart wurde). Wir werden auch genau diesen gegebenen Source-Code im Praktomat anwenden. Realisieren Sie durch entsprechende Ergänzungen in der Methode gibZahl in TestKlasse2, dass ein Objekt mit dem im Parameter wert übergebenen Wert instanziiert werden kann (ein Aufruf getZahl liefert dann auch den Wert, mit dem dieses Objekt erzeugt wurde) und dass ein Aufruf der toString-Methode dieses Objekts als Resultat aber einen String liefert, in dem der Zahlwert des Objekts (also der in zahl gespeicherte Wert) verdoppelt ist. Hinweis: anonyme Klasse. Beispiel: Mit dem Beispielparameterwert 4711 bei der Erzeugung soll als Ergebnis auf dem Bildschirm der Wert 9422 (=2*4711) erscheinen. Geben Sie im Praktomat nur ihren Code der Klasse TestKlasse2 in einer Datei TestKlasse2.java ab. Geben Sie keinen Code für die Schnittstelle ZahlFunktionalitaet und die Klasse Zahlklasse ab. Achten Sie darauf, dass die Klasse TestKlasse2 wie angegeben public ist.