Hochschule Karlsruhe
University of Applied Sciences

Fakultät für **Informatik**
und Wirtschaftsinformatik

Prof. Dr. Carsten Sinz
Lehrgebiet Software-Entwicklung

Übungsblatt 4 zur Vorlesung Programmieren (fällig bis 30.11.25)

Aufgabe 1 (Komplexe Zahlen):

Die *komplexen Zahlen* sind eine Erweiterung der reellen Zahlen, so dass auch die Gleichung $x^2 = -1$ eine Lösung besitzt. Die Lösung dieser Gleichung wird mit dem Symbol i bezeichnet, d.h. es gilt $\mathrm{i}^2 = -1$ für die komplexe Zahl i, die auch *imaginäre Einheit* genannt wird.

Komplexe Zahlen z haben die Form $z = a + b\mathrm{i}$, wobei $a, b \in \mathbb{R}$. D.h. jede komplexe Zahl lässt sich durch ein Paar (a, b) von reellen Zahlen repräsentieren, wobei a auch der Realteil und b der Imaginärteil der Zahl z genannt wird.

Auf den reellen Zahlen lassen sich die üblichen arithmetischen Operationen wie Addition, Subtraktion, Multiplikation und Division definieren. Dies geschieht wie folgt, wobei wir annehmen, dass $z_1 = a + b\mathrm{i}$ und $z_2 = c + d\mathrm{i}$.

- $z_1 + z_2 = (a + b\mathrm{i}) + (c + d\mathrm{i}) = (a + c) + (b + d)\mathrm{i}$, d.h. die Summe von z_1 und z_2 ist die komplexe Zahl mit Realteil $a + c$ und Imaginärteil $b + d$.
- $z_1 - z_2 = (a - c) + (b - d)\mathrm{i}$
- $z_1 \cdot z_2 = (ac - bd) + (ad + bc)\mathrm{i}$
- $\frac{z_1}{z_2} = \frac{ac+bd}{c^2+d^2} + \frac{bc-ad}{c^2+d^2}\mathrm{i}$

a) Erstellen Sie eine Klasse `Complex` zur Repräsentation von komplexen Zahlen in einer Datei `Complex.java` mit folgender Funktionalität:

   - Realteil und Imaginärteil sollen als `double`-Werte abgespeichert werden.
   - Die Klasse soll zwei Konstruktoren besitzen, einen mit zwei `double`-Argumenten für Real- und Imaginärteil, und einen mit einer leeren Argumentliste (dieser soll die Attribute mit Standardwerten initialisieren).
   - Die Klasse soll Methoden für die vier Grundrechenarten auf komplexen Zahlen enthalten:

     ```
     Complex add(Complex other) { ... }
     Complex sub(Complex other) { ... }
     Complex mul(Complex other) { ... }
     Complex div(Complex other) { ... }
     ```

     Jede Methode soll die entsprechende Verknüpfung von 'this' und 'other' durchführen und das Ergebnis als neue komplexe Zahl zurückgeben. 'this' bleibt dabei unverändert.

(b.w.)

- Die Klasse soll eine Methode `public String toString()` enthalten, die die textuelle Darstellung einer komplexen Zahl berechnet. Wenn der Imaginärteil 0 ist, soll nur der Realteil ausgegeben werden.

  Beispiele für einige konkrete Paare (a, b), also Repräsentationen von $a + bi$:

(a, b)	Ausgabe
$(2, 3)$	$2 + 3i$
$(3, -4)$	$3 - 4i$
$(3, 0)$	3
$(0, 7)$	$0 + 7i$ (oder gerne auch: $7i$)
$(-2, -5)$	$-2 - 5i$
$(0, 0)$	0

- Die Klasse soll eine Methode `public boolean equals(Complex other)` enthalten, um zwei komplexe Zahlen auf Gleichheit zu prüfen.

  Hinweis: Denken Sie an die endliche Genauigkeit von `double`. Nutzen Sie die Methode `Math.abs(...)` und einen Größenvergleich mit einer sehr kleinen positiven Zahl, z.B. `1e-12`, anstatt auf exakte Gleichheit zu prüfen.

  (Zwei komplexe Zahlen sind genau dann gleich, wenn jeweils die beiden Realteile und die beiden Imaginärteile zueinander gleich sind.)

b) Erstellen Sie des Weiteren eine Klasse `ComplexTest` (u.a. mit einer `main`-Methode), mit der Sie Ihre Implementierung testen:

- Erstellen Sie vier Testfälle, je einen für `add`, `sub`, `mul` und `div`. Überlegen Sie sich jeweils passende Operanden, die zu nicht-trivialen Berechnungen führen (z.B. wäre $0+0$ trivial).

  Tipp: Für die Division ist es hilfreich, Realteil und Imaginärteil des Nenners so zu wählen, dass die Summe deren Quadrate 25 oder 100 ergibt; dann sind Ihre im-Voraus-Rechnungen auf Papier einfacher als bei "krummen" Werten. Beispielsweise ist

  $$3^2 + (-4)^2 = 9 + 16 = 25 \qquad \text{oder} \qquad (-6)^2 + (-8)^2 = 36 + 64 = 100$$

- Erstellen Sie einen weiteren Testfall für die `mul`-Operation, der nachweist, dass $i^2 = -1$ gilt.

Dazu Hinweise:

- Ein Testfall beginnt hier mit einer Beispielrechnung, die Sie selbst schriftlich ausführen (bei der Abgabe vorzuzeigen!). Dabei sollten Sie die Rechenoperationen genauso ausführen wie oben im Aufgabentext beschrieben. Ihr Rechenergebnis (benannt z.B. als `expectedValue` und als fester Wert in Ihrem Programm eingetragen) benötigen Sie dann für den eigentlichen Test:
- Dabei lassen Sie jeweils die gleiche Rechnung von der Klasse `Complex` ausführen – dazu müssen Sie die beiden Operanden der Rechnung an die jeweilige Rechenmethode übergeben (und natürlich zuvor erzeugen!); das Rechenergebnis (benannt z.B. als `calculatedValue`) vergleichen Sie dann mit dem zuvor selbst errechneten Wert (nutzen Sie dazu die `equals`-Methode).
- Wichtig: Der Test soll automatisch ablaufen – keine Benutzerinteraktion über die Konsole vorgesehen! (b.w.)

- Die Ausgabe eines jeden Testfalls soll dabei einerseits angeben, welche Operation mit welchen Operanden durchgeführt wurde, und andererseits, ob das Ergebnis mit dem erwarteten Wert übereinstimmt:
  - Ist dies der Fall, so soll "`Ok`" ausgegeben werden
  - Ansonsten: "`FAILED: expected` X`, got` Y", wobei X und Y das erwartete und das berechnete Ergebnis sind.

  Mögliche Beispiel-Ausgaben wären:
  - `Addition of (3 + 4i) and (-2 -i): result is (1 + 3i). Ok`
  - `Multiplication of (1 + i) and (1 - i): FAILED: expected (2), got (1 + 2i)`

Aufgabe 2 (Tic-Tac-Toe):

Tic-Tac-Toe ist ein einfaches Strategiespiel für zwei Personen (siehe z.B. `https://de.wikipedia.org/wiki/Tic-Tac-Toe`).

In dieser Aufgabe soll das Spiel Tic-Tac-Toe implementiert werden, so dass zwei menschliche Gegner am (selben) Computer gegeneinander spielen können. Die Eingabe der Züge sowie die Ausgabe des aktuellen Spielstandes soll auf der Konsole erfolgen. Gehen Sie wie folgt vor:

a) Erstellen Sie (schriftlich) Storyboards für drei Spielabläufe, jeweils von Anfang bis Ende:
   - Spieler `X` gewinnt
   - Spieler `O` gewinnt
   - Spiel endet mit einem Patt

   Dazu notieren Sie sich die Abfolge des (ganzen) Spielfeld-Inhalts Schritt für Schritt.

b) Überlegen Sie sich Datenstrukturen zur Darstellung des Spielfeldes und ggf. der Spieler.

c) Entwickeln Sie eine Methode zur Ausgabe des aktuellen Stands des Spiels auf der Konsole.

   Denken Sie daran, dass der Benutzer erkennen können sollte, welche Eingabe von Koordinaten für den gewünschten Spielzug nötig ist. Das Feld sollte also entsprechend beschriftet sein.

d) Schreiben Sie eine Methode, die es erlaubt, einen Spielstein für einen Spieler auf ein leeres Feld zu setzen. Die Methode soll den Rückgabetyp `boolean` besitzen und `true` genau dann zurückgeben, wenn das Feld frei war und der Stein gesetzt werden konnte.

   Schließen Sie außerdem im Voraus aus, dass Koordinaten eingegeben werden, die gar nicht auf dem Spielfeld liegen.

   Falls der Zug nicht möglich ist, soll die Aufforderung zum Ziehen erneut gestellt werden, bis ein gültiger Spielzug eingegeben wurde – also hier kein Programmabbruch. Geben Sie in der erneuten Aufforderung deren Grund mit an, damit die BenutzerInnen geeignet reagieren können.

e) Erstellen Sie eine Methode, die feststellt, ob das Spiel für einen Spieler gewonnen ist.

f) Fügen Sie eine Spiellogik hinzu, die Spielzüge der beiden Spieler abwechselnd entgegennimmt so lange, bis das Spiel beendet ist. Das Ergebnis des Spiels soll anschließend ausgegeben werden.

g) Spielen Sie Ihre in a) erstellten Szenarien testweise durch.