Praktikum Programmieren
Studiengang Angewandte Informatik
Prof. Dr. Bernd Kahlbrandt
Dipl.-Ing. Gerhard Oelker
Dipl.-Wirtschaftsinformatiker (FH) Sven Berding
Hochschule für Angewandte Wissenschaften Hamburg
Department für Informatik
18. März 2020

Aufgabenblatt A02: Java Grundlagen, Fortsetzung

1 Java Denksportaufgaben

1. Dieser Code besteht aus einer einfachen immutablen Klasse, die aus einem Namen, bestehend aus Vor- und Nachname besteht und einer main-Methode, die ein Name-Objekt in ein Set einfügt und prüft, ob das Objekt in dem Set enthalten ist. Was gibt das Program aus?

```
public class Name {
    private String first, last;

    public Name(String first, String last) {
        this.first = first;
        this.last = last;
    }

    public boolean equals(Object o) {
        if (!(o instanceof Name))
            return false;
        Name n = (Name)o;
        return n.first.equals(first) && n.last.equals(last);
    }

    public static void main(String[] args) {
        Set<Name> s = new HashSet<>();
        s.add(new Name("Mickey", "Mouse"));
        System.out.println(s.contains(new Name("Mickey", "Mouse")));
    }
}
```

Überlegen Sie sich die Antwort bitte zunächst theoretisch ohne den Code auszuführen und begründen Sie, warum es zu der Ausgabe kommt! Geben Sie ggfs. bitte an, ob man das anders programmieren muss und falls ja, wie!

2. Der Code besteht aus einer Klasse Name und einer main-Methode, die ein Name-Objekt in ein Hash Set einfügt und prüft, ob das Set das Name-Objekt enthält. Was gibt dieses Programm aus?

```
public class Name {
    private String first, last;

    public Name(String first, String last) {
        this.first = first; this.last = last;
    }

    public boolean equals(Name n) {
     return n.first.equals(first) && n.last.equals(last);
    }

    public int hashCode() {
        return 31 * first.hashCode() + last.hashCode();
    }

    public static void main(String[] args) {
        Set<Name> s = new HashSet<Name>();
        s.add(new Name("Donald", "Duck"));
        System.out.println(s.contains(new Name("Donald", "Duck")));
    }
}
```

Überlegen Sie sich die Antwort bitte zunächst theoretisch ohne den Code auszuführen und begründen Sie, warum es zu der Ausgabe kommt! Geben Sie ggfs. bitte an, ob man das anders programmieren muss und falls ja, wie!

2 Klasse Complex

Schreiben Sie bitte eine Klasse *Complex*, deren Objekte komplexe Zahlen repräsentieren! Meine Motivation für diese Aufgabe ist durch m. E. interessante Aufgaben (GUI am Beispiel Fraktale, Apfelmännchen etc.) im letzten Drittel des Semesters begründet. Außerdem haben Sie in der Klasse Complex aus Ruby ein Muster, an dem Sie sich bis zu einem gewissen Grad orientieren können.

Ich mache Ihnen in dieser Aufgabe einige Vorgaben. Sie haben aber immer noch sehr viele Freiheiten für die genaue Form Ihrer Lösung.

Insbesondere sollen Sie hieran dies lernen bzw. nun in Java weiter üben:

- Konstruktoren, auch Default-Konstruktor. Das ist in Java etwas anders als in Ruby, insbesondere über Sie überladen.
- Fabrikmethoden.
- Überschreiben der Methoden *equals*, *hashCode* und *toString*.
- Utility-Klassen.

Nun eine kurze Erinnerung an komplexe Zahlen:

1. Komplexe Zahlen können Sie als Punkte in einer Ebene mit re und im Achse betrachten. In dieser Sichtweise ist eine komplexe Zahl c ein Punkt in dieser Ebene und kann so dargestellt werden:
$$c = re + i \cdot im$$
re steht dabei für Realteil, im für Imaginärteil. i ist die „imaginäre Einheit“, $i^2 = -1$.
Die ist die Darstellung durch *kartesische Koordinaten*.
2. Die Punkte in einer solchen Ebene können Sie aber auch durch den Abstand vom Nullpunkt (0,0) und den Winkel φ mit der positiven Richtung der re-Achse beschreiben:
$$c = abs \cdot e^{i\varphi}$$
Dies ist die Darstellung durch *Polarkoordinaten*. Siehe Abb. 1[1].
3. Komplexe Zahlen haben die üblichen elementaren Rechenoperationen Addition, Multiplikation, Subtraktion, Division.
4. Viele bekannte Funktionen, wie Exponentialfunktion oder trogonometrische Methoden können auch für komplexe Zahlen definiert werden.

Ihre Klasse soll bitte diesen Anforderungen genügen:

1. Die Klasse Complex soll mutable sein. Das ist vor allem eine Entscheidung für bessere Performance der geplanten Anwendungen.
2. Komplexe Zahlen (also Objekte der Klasse Complex) wollen sowohl aus *kartesischen Koordinaten* als auch aus *Polarkoordinaten* erzeugt werden können.
3. Die Rechenmethoden verändern die komplexe Zahl, für die sie aufgerufen werden.
4. In Ruby sind die Rechenoperationen (+,-,*,/) Methoden, die Sie auch in einer Klasse Complex in Ruby schreiben können. In Java sind diese Rechenoperationen Operatoren, die können Sie in Java nicht überschreiben, überladen etc. Sie müss also Methode add, subtract, mult, gerne mit sinnvoll abgekürzten Methodennamen schreiben.

[1] Vielen Dank an Marcel Tuleweit, der mir die Bilder im Juli 2014 geschickt hat.

Abb. 1: Kartesischer und Polar-Bär

5. Die Methoden *equals* und *hashCode* werden konsistent überschrieben. Diese Methoden sind analog zu == und *hash* in Ruby.

6. Die Methode *toString* wird sinnvoll überschrieben. Diese Methode entspricht der Methode *to_s*, die Sie aus Ruby kennen.

7. Es gibt Methoden, um die Bestandteile einer komplexen Zahl sowohl als kartesischen Koordinaten als auch als Polarkoordinaten zu bekommen.

8. Funktionen, die Sie in der Klasse *Complex* z. B. für Umrechnungen zwischen den Koordinatendarstellungen benötigen implementieren Sie bitte als Klassenmethoden in einer Utility-Klasse (Arbeitstitel: *ComplexMath*!

Der Abgabetermin für alle ist:

Montag, 30.03.2020, 08:00