Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences
Fachbereich Informatik
Prof. Dr. Rudolf Berrendorf

Einführung in die Programmierung
WS 19/20
Ausgabe : 09.12.2019
Bearbeiten bis : 15.12.2019

Übung 11

Aufgabe 1 :
Gegeben ist folgendes Programm:

```
class K1 {
    public static int s;
    public int i;

    K1() {
        s++;
    }
    K1(int i) {
        i = i;
    }
    public int f(int x) {
        s += x;
        return i++;
    }
    public static void g() {
        ++s;
    }
    public void h() {
        f(s);
        g();
    }
    public String toString() {
        return "s=" + s + ", i=" + i;
    }
}
```

```
class K2 extends K1 {
    private static int s;
    private int i;

    K2(int i1) {
        i = i1;
    }
    K2(int i1, int i2) {
        super(i1);
        i = i1;
    }
    public int f() {
        i++;
        super.g();
        g();
        return i;
    }
    public static void g() {
        s++;
    }
    public void h() {
        s++;
    }
    public String toString() {
        return "s=" + s + ", i=" + i;
    }
}
```

```
public class TestK1K2 {
    public static void main(String[] args) {
        K1 k1 = new K1(1);
        K2 k2 = new K2(2);
        int i;
        k1.f(i = 1);
        k2.f(i = 1);
        System.out.println("k1: " + k1);
        System.out.println("k2: " + k2);
        K2 k3 = new K2(3,4);
        k3.f();
        k3.h();
        System.out.println("k1: " + k1);
        System.out.println("k2: " + k2);
        System.out.println("k3: " + k3);
        K1 k4 = new K2(5,6);
        System.out.println("k4: " + k4);
        k4.f(2);
        k4.h();
        System.out.println("k4: " + k4);
    }
}
```

Bei Ausführung des Programms kommt es zu folgender Ausgabe:

```
> java TestK1K2
k1: s=3, i=1
k2: s=0, i=2
k1: s=4, i=1
```

```
k2: s=2, i=2
k3: s=2, i=4
k4: s=2, i=5
k4: s=3, i=5
```

Erläutern Sie *im Detail*, wieso es zu genau dieser Ausgabe kommt. Oder anders ausgedrückt: was passiert genau während des Programmablaufs?

Hinweis (siehe Vorlesung): Bei Instanziierung eines Objekts werden zu Beginn alle Instanzvariablen mit dem entsprechenden Null-Wert initialisiert.

Aufgabe 2 (1 Punkt) :
Dies ist eine Praktomataufgabe!

In dieser Aufgabe soll ein Web Shop programmiert werden.

Ein Kunde (Klasse **`Kunde`**) wird beschrieben durch einen Vornamen und Nachnamen. Gute und treue Kunden (Klasse **`GuterKunde`**) sind Kunden, die zusätzlich einen Rabatt auf Bestellungen bekommen (z.B. 0.05 für 5% Rabatt).

Ein Artikel (Klasse **`Artikel`**) im Web Shop wird beschrieben mit einem Namen, dem zugehörigen Preis in Euro (**`double`**) und dem aktuellen Bestand (wie viele Exemplare derzeit vorhanden sind; ganzzahlig).

In einem Web Shop (Klasse **`WebShop`**) können maximal 10 Kunden und 10 Artikel registriert sein. Ein neuer Artikel kann dem Web Shop hinzugefügt werden über eine Methode **`void neuerArtikel(String name, double preis, int anzahl)`**, wobei **`anzahl`** die Anzahl an Exemplaren ist, die zu diesem Artikel im Web Shop anschließend vorhanden sein soll. Ein neuer Kunde kann in einem Web Shop registriert werden über **`Kunde neuerKunde(String vorname, String nachname)`**, gute Kunden über die (dann überladene) Methode **`Kunde neuerKunde(String vorname, String nachname, double nachlass)`**. Beachten Sie hierbei, dass der Ergebnistyp dieser Methode **`Kunde`** ist, aber in der Methode ein **`GuterKunde`** erzeugt werden soll (siehe auch Hinweise unten).

In einem Web Shop wird die Bestellung eines Kunde bearbeitet über die Methode
`String bestellen(Kunde kunde, String[] artikel)`
Der Parameter **`kunde`** steht für den bestellenden Kunden (was auch ein guter Kunde sein kann) und das String-Feld enthält die Namen (als String) der Artikel, die bestellt werden sollen (jeweils ein Exemplar davon pro Eintrag). Wenn ein Artikel ausgegeben wird, so reduziert sich der Bestand dieses Artikels um 1. Das Ergebnis der Methode ist ein String mit der Rechnung zu dieser Bestellung (siehe Beispiel unten).

Das nachfolgende Beispiel (in Klasse **`WebShopTest`**) erläutert den Aufbau der Rechnung, inklusive aller möglichen Fälle. Falls ein gewünschter Artikel nicht im Bestand vorhanden ist, soll eine entsprechende Ausgabe auf der Rechnung erfolgen (**`nicht gefunden`**). Und falls ein Artikel ausverkauft ist (der Bestand ist also 0), dann soll auch eine entsprechende Meldung auf der Rechnung ausgegeben werden (**`nicht mehr vorhanden`**). Gute Kunden bekommen ihren Rabatt bei jedem bestellten Artikel angerechnet. Alle diese Sonderfälle sind im Beispiel unten beim zweiten (guten) Kunden zu sehen und auch die vorgegebe Ausgabe für alle Fälle.

Für das folgende Beispielprogramm

```
public static void main(String[] args) {
    // erzeuge Webshop
    WebShop w = new WebShop();

    // neuer Kunde
    Kunde kunde1 = w.neuerKunde("Helga", "Herrlich");
    // neuer guter Kunde mit 5% Preisnachlass
    // Beachte: Ein GuterKunde-Objekt wurde erzeugt, aber der Typ der Variablen ist Kunde!
    Kunde kunde2 = w.neuerKunde("Werner", "Winzig", 0.05);

    // neue Artikel (2x Leberwurst, 1x Nutella)
    w.neuerArtikel("Leberwurst", 1.95, 2);
    w.neuerArtikel("Nutella", 3.95, 1);

    // Kunde 1 bestellt eine Leberwurst und ein Nutella
    String[] bestellung1 = {"Leberwurst", "Nutella"};
    String rechnung1 = w.bestellen(kunde1, bestellung1);
    // Rechnung Kunde 1 ausgeben
    System.out.println(rechnung1);
```

```
        // Kunde 2 bestellt eine Leberwurst, ein Nutella und eine Butter
        String[] bestellung2 = {"Leberwurst", "Nutella", "Butter"};
        String rechnung2 = w.bestellen(kunde2, bestellung2);
        // Rechnung Kunde 2 ausgeben
        System.out.println(rechnung2);
    }
```

soll die Ausgabe (also die beiden erstellten Rechnungen hintereinander) folgendermaßen aussehen:

```
Rechnung fuer Helga Herrlich:
Leberwurst : 1.95
Nutella : 3.95
Gesamtpreis : 5.9

Rechnung fuer unseren guten Kunden Werner Winzig, Preisnachlass 5.0%:
Leberwurst : 1.8524999999999998
Nutella : nicht mehr vorhanden
Butter : nicht gefunden
Gesamtpreis : 1.8524999999999998
```

Hinweise:

- Polymorphismus
- **`instanceof`**
- Explizite Downcasts und implizite Upcasts
- Überladen und Überschreiben von Methoden
- Vergeben Sie in allen Klassen sinnvoll Zugriffsrechte für Instanz- und Klassenvariablen sowie Methoden.

Aufgabe 3 (1 Punkt) :
Dies ist eine Praktomataufgabe!

In der Vorlesung zu Abstrakte Datentypen haben wir einen Stack kennengelernt. Entwickeln Sie einen allgemeinen Stack, der ein Behälter für Objekte eines beliebigen Referenztyps darstellt. Nutzen Sie dabei den Typ **`Object`** in geeigneter Weise.

Das Gerüst der Stack-Klasse ist wie folgt vorgegeben:

```
public class Stack {
    public Stack(int maximalGroesse) {...}
    public void push(Object o) {...}
    public Object pop() {...}
    public boolean isEmpty() {...}
```

Dem Konstruktur übergibt man die maximal Größe des Stacks. Die **`pop`**-Methode liefert das oberste Stack-Element und löscht es auch gleichzeitig vom Stack (im Gegensatz zur der pop-Funktion des abstrakten Datentyps damals). Füllen Sie sinnvoll diese Klasse aus. Sie können/müssen zusätzliche Instanzvariablen anlegen. Als kleines Testprogramm können Sie nehmen:

```
    public static void main(String[] args) {
        Stack st = new Stack(10);
        st.push("a");
        st.push("b");
        st.push("c");
        System.out.println(st.pop());
        System.out.println(st.pop());
        System.out.println(st.pop());
        System.out.println(st.isEmpty());
    }
```

Als Anwendung des Stacks wurde in der Vorlesung vorgestellt, wie man mit Hilfe eines Stacks eine Folge von Werten spiegeln kann. Passen Sie das Verfahren auf die neu programmierte Stack-Klasse an. Nehmen Sie statt einer Folge einen String. Nutzen Sie statt der damals definierten Folge-Operationen (**`isemptyfolge`**, **`first`**, **`rest`**, ...) in sinnvoller Weise String-Operationen als Basisoperationen:

- Über `str.length()` kann man ermitteln, ob ein String leer ist oder nicht.
- Über `str.charAt(0)` kann man das erste Element eines Strings bestimmen, das vom Typ ein `char` ist. Dies ist ein primitiver Typ und kein Referenztyp. Unser Stack arbeitet aber nur mit Referenztypen!
- Über `""+ c` kann man aus einem `char`-Wert (primitiver Typ) einen `String`-Wert machen (Referenztyp).
- Über `str.substring(...)` kann man einen Teil eines String bekommen, zum Beispiel den letzten Teil außer dem ersten Zeichen.

Die Methodensignatur in einer Klasse `Spiegeln` ist vorgegeben mit `public static String spiegeln(String s)`. Ein Testprogramm ist dann etwa:

```
public static void main(String[] args) {
    String s = "hallo";
    System.out.println(s + " gespiegelt ist " + spiegeln(s));
}
```

Aufgabe 4 (1 Punkt) :
Dies ist eine Praktomataufgabe!

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:

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

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)); // Ausgabe muss 15.0 sein

    z1 = new KomplexeZahl(3.0, 1.0);
    z2 = new KomplexeZahll(5.0, 1.0);
    System.out.println(z1.multiplizieren(z2)); // Ausgabe muss 14.0+8.0*i sein
}
```

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 5 :
Legen Sie ein neues eclipse-Projekt zu dieser Aufgabe an und kopieren ihre Klassen der letzten Aufgabe (Klasse **`Zahl`** und weitere) dort hinein.

Ändern Sie ihre Realisierung so ab, dass statt einer abstrakten Klasse **`Zahl`** eine Schnittstelle **`Zahl`** definiert wird mit der gleichen Funktionalität.

Aufgabe 6 :
Legen Sie ein neues eclipse-Projekt zu dieser Aufgabe an und kopieren ihre Klassen der vorletzten Aufgabe dort hinein (Klasse **`Zahl`** und weitere, mit abstrakter Klasse).

Gegeben ist die Schnittstelle **`Vergleichen`**:

```
public interface Vergleichen {
    final double epsilon = 1e-12;
    public boolean istGleich(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 gelten als gleich, wenn gilt: $||x|-|y|| < \varepsilon$ (ε aus der Schnittstelle). Die äußeren $||$ bedeuten den Absolutwert des Ausdrucks (**`Math.abs`**), die inneren den Betrag der Zahl. Für reelle Zahlen ist dies ebenfalls wieder **`Math.abs`**, für eine komplexe Zahl $a+b*i$ ist der Betrag definiert durch: $|a+b*i| = \sqrt{a^2+b^2}$.

Erweitern Sie ihr Testprogramm wie folgt:

```
...
// Ausgabe false
System.out.println(z1.istGleich(z2));
// Ausgabe true
System.out.println(z1.istGleich(new KomplexeZahl(3.0,1.0+1e-14)));
// Ausgabe false
System.out.println(z1.istGleich(new KomplexeZahl(3.0,1.0+1e-5)));
```