Hochschule
Albstadt-Sigmaringen
Albstadt-Sigmaringen University

Fakultät Informatik
Nemirovski

Programmieren 2 Prof. Dr. German
SoSe 18

Aufgabenblatt 2

Grundlagen

Anzahl der Aufgaben: **5**
Abgabefrist: **26.04.18**

Aufgabe 2.1

a) Gegeben ist ein Python-Programm, welches entscheidet, seit wann Sie volljährig sind / in wieviel Jahren Sie volljährig sind und wählen dürfen:

```
# Programm, welches berechnet,
# - in wievielen Jahren Sie waehlen duerfen (falls Sie noch nicht volljaehrig sind)
# -- seit wievielen Jahren Sie waehlen duerfen (falls Sie schon volljaehrig sind)

def frageNutzer():
    # Tipp: Nehmen Sie fuer diese 3 Zeilen  in Java die Scanner-Klasse!
    namestr = input("Wie heissen Sie? ")
    alterstr = input("Wie alt sind Sie? ")
    alter = int(alterstr) # Konvertierung int -- str
    ########################################################################
    print("Hallo",namestr)
    if (alter < 18):
        wahlalter = 18 - alter
        print("Sie duerfen in",wahlalter,"Jahren waehlen!")
    else:
        wahlalter = alter - 18
        print("Sie duerfen seit",wahlalter,"Jahren waehlen!")

frageNutzer()
```

Schreiben Sie eine Java-Klasse, die mit äquivalenten Kontrollstrukturen die gleiche Berechnung durchführt.

b) Gegeben ist ein Python-Programm mit einem kleinen Eingabemenü. Sie können mit einer der Zahlen 1 bis 4 auswählen, welche Berechnung das Programm ausführt.

```
import math

def menu():
    auswahl = 1
    while auswahl != 4:

        print ('Sie haben die Auswahl zwischen folgenden Berechnungen: ')
        print ('[1] |x|')
        print ('[2] sin(x)')
        print ('[3] cos(x)')
        print ('[4] Ende der Berechnung')
        auswahlstr=input('Treffen Sie eine Auswahl 1 bis 4: ')
        auswahl = int(auswahlstr)
        if auswahl == 1:
            xstr=input('Bitte geben Sie x ein: ')
            x=float(xstr)
            ergebnis = math.fabs(x)
            print('|',x,'|=',ergebnis)
        elif  auswahl == 2:
            xstr=input('Bitte geben Sie x ein: ')
            x=float(xstr)
            ergebnis = math.sin(x)
            print('sin(',x,')=',ergebnis)
        elif  auswahl == 3:
            xstr=input('Bitte geben Sie x ein: ')
            x=float(xstr)
            ergebnis = math.cos(x)
            print('cos(',x,')=',ergebnis)
    print ('Auswahl beendet!')

menu()
```

Schreiben Sie eine Java-Klasse, welche ein äquivalentes Menü mit einer do-while-Schleife und einer switch-case-Anweisung aufbaut und mit der Math-Klasse von Java die gleichen Berechnungen ausführt!

Das folgende Ein/Ausgabeszenario zeigt das Verhalten des Python-Programms mit einigen Beispielwerten:

```
Sie haben die Auswahl zwischen folgenden Berechnungen:
[1] |x|
[2] sin(x)
[3] cos(x)
[4] Ende der Berechnung
Treffen Sie eine Auswahl 1 bis 4: 2
Bitte geben Sie x ein: 33.7
sin( 33.7 )= 0.756221658786063
Sie haben die Auswahl zwischen folgenden Berechnungen:
[1] |x|
[2] sin(x)
```

```
[3] cos(x)
[4] Ende der Berechnung
Treffen Sie eine Auswahl 1 bis 4: 4
Auswahl beendet!
```

Tipp: Das Scanner-Objekt in Java sollte hier etwas anders erzeugt werden:

```
Scanner tastatur = new Scanner(System.in).useLocale(Locale.US);
```

Ansonsten wird bei der Eingabe der Gleitkomma-Zahlen 2,5 statt 2.5 erwartet.

Aufgabe 2.2

a) Einige Bewohner eines kleinen gallischen Dorfes starten einen Wettbewerb im Hinkelstein-Weitwurf. Das folgende Python-Programm gibt aus, welcher Wettbewerbs-Teilnehmer wie weit geworfen hat. Außerdem berechnet es, wer Sieger war:

```
# Hinkelstein-Weitwurf-Teilnehmer
teilnehmer=['asterix','obelix','gutemine','majestix','falbala','tragicomix']

# Geworfene Entfernungen
hinkelsteinentfernung = [20,50,30,40,18,25]

# Loesung 1: Ausgabe der Weitwurfergebnisse
i=0
siegerentfernung=hinkelsteinentfernung[0]
sieger=teilnehmer[0]
# while-Schleife gibt alle geworfenen Entfernungen aus
# Ausserdem wird der Sieger des Wettbewerbs ermittelt
while (i<len(teilnehmer)):
    print('Dorfmitglied',teilnehmer[i],'wirft
Hinkelstein',hinkelsteinentfernung[i],'Meter');
    if (siegerentfernung < hinkelsteinentfernung[i]):
        siegerentfernung = hinkelsteinentfernung[i]
        sieger = teilnehmer[i]
    i+=1
print('Sieger ist',sieger,'mit',siegerentfernung,'Metern')
```

Schreiben Sie eine Java-Klasse, die den gleichen Algorithmus mit zwei festen Arrays umsetzt. Verwenden Sie für die Schleifenkonstruktion eine for-Schleife! (Zähl- oder Iterationsschleife – was Sie lieber mögen).

Hilfestellung: Arrays mit fester Initialisierung werden in Java folgendermaßen vereinbart:

```
String [] teilnehmer = {"Asterix","Obelix","Gutemine","Majestix","Falbala","Tragicomix"};
int [] hinkelsteinEntfernung= {20,50,30,40,18,25};
```

b) Gegeben ist folgendes Java-Programm:

```
public class Datentypen {


	public static void main(String[] args) {
		short x = 25000;
		short y = 555;


		// Frage 1: Was fuer ein Ergebnis wird hier produziert?
		// Frage 2: Was geschieht mit dem "richtigen" Ergebnis dabei
		//          auf Bitebene?
		short ergebnis = (short) (x * y);
		System.out.println("Ergebnis: " + ergebnis);
	} // end main()
} // end class
```

Geben Sie es ein, bringen Sie es zum Laufen und beantworten Sie die beiden Fragen im Quelltext!

Aufgabe 2.3

Implementieren Sie eine Klasse *Complex*, welche komplexe Zahlen repräsentieren soll mit passenden Attributen und Methoden.

(a) Implementieren Sie eine sinnvolle Konstruktor-Methode, so dass die Klasse wie in den Beispielen unten gezeigt "funktioniert“.

(b) Implementieren Sie eine *toString()*-Methode, die eine *String*-Repräsentation eines Complex-Objekts wie folgt zurückliefert:

```
Complex z = new Complex(1.0, 2.0);
z.toString(); // liefert den String "1.0 + 2.0i"
```

(c) Implementieren Sie die Methode *add(c)* der Klasse *Complex*, die die komplexe Zahl *c* zum Objekt hinzuaddiert. Die Methode soll ein neues *Complex*-Objekt zurückliefern.
(c) Implementieren Sie die Methode *sub(c)* der Klasse *Complex*, die die komplexe Zahl *c* vom Objekt abzieht. Die Methode soll ein neues *Complex*-Objekt zurückliefern.
(d) Implementieren Sie die Methode *mult(c)* der Klasse *Complex*, die die komplexe Zahl *c* mit dem Objekt multipliziert. Die Methode soll ein neues *Complex*-Objekt zurückliefern.
(e) Implementieren Sie die Methode *div(c)* der Klasse Complex, die das Objekt durch die Komplexe Zahl c teilt. Die Methode soll ein neues Complex-Objekt zurückliefern.

Beispielfunktionsweise:

```
Complex z1 = new Complex(1,2);
Complex z2 = new Complex(2,3);
Complex z3;
z3 = z1.add(z2);
System.out.Println(z1.toString()); // gibt aus: "1.0 + 2.0i"
System.out.Println(z3); // gibt aus: "3.0 + 5.0i"
```

Aufgabe 2.4

Der julianische Kalender wurde vom Julius Cesar eingeführt. Seit dem 16. Jahrhundert wurde er in Europa durch den gregorianischen Kalender in mehreren Schritten ersetzt.

Während den letzten zwei Jahrhunderte (von 1900 bis 2099) besteht zwischen dem julianischen und dem gregorianischen Kalender eine Differenz von 13 Tagen. D.h. am 1. Oktober nach dem julianischen Kalender haben wir den 13. Oktober nach dem gregorianischen Kalender.

Erstellen Sie eine Klasse, welche die gregorianischen Daten aus der Datei daten-greg.csv, einliest und in das julianische Format konvertiert. Speichern Sie dieses Ergebnis in der gleichen Struktur in der Datei daten-jul.csv.

Der Inhalt der Datei daten-greg.csv sieht wie folgt aus:

```
2001-03-12, 2010-12-20, 2003-11-01

2002-05-16, 2013-10-11, 1997-03-05

2007-06-22, 2017-01-01 2018-12-26
```

Aufgabe 2.5

Schreiben Sie eine Klasse *Stck*, die eine Stackdatenstruktur ***mithilfe eines Strings*** implementiert. Der Stack enthält einzelne Zeichen und implementiert die bekannten Stack-Operationen *push(c)* und *pop()*, die Elemente auf den Stack legen bzw. Elemente vom Stack entfernen.

Sie werden *String*-Methoden benötigen, die wir bisher in der Vorlesung nicht behandelt haben. Verwenden Sie zum Nachschlagen bitte ausschließlich die offiziellen Java-Docs.

(a) Erstellen Sie eine neue Klasse *Stck* mit einem geeigneten Attribut und den beiden Methoden *push(c)* und *pop()*. Folgendes Beispiel zeigt das Verhalten der Klasse:

```
Stck s = new Stck();
s.push('a'); // Jetzt liegt oben auf dem Stack "a"
s.push('e'); // Jetzt liegt oben auf dem Stack "e"
s.push('x'); // Jetzt liegt oben auf dem Stack "x"
s.pop(); // liefert den char 'x' zurueck. Jetzt liegt
         // oben auf dem Stack 'e'
s.push('y'); // Jetzt liegt oben auf dem Stack "y"
s.pop(); // liefert 'y' zurueck
s.pop(); // liefert 'e' zurueck
```

(b) Schreiben Sie einen Konstruktor, der einen String erwartet und den Stack dann mit dem *String* vorbelegt. Folgendes Beispiel zeigt das Verhalten:

```
Stck s = new Stck("bla");
s.pop(); // liefert 'a' zurueck
s.pop(); // liefert 'l' zurueck
```

(c) Schreiben Sie eine Methode *len()*, welche die Länge des Stacks zurückliefert.

(d) Schreiben Sie eine Methode *ops*(), die zurückliefert, wie viele *push()*- und *pop()*-Operationen insgesamt bisher auf dem Stack ausgeführt wurden.

(f) Schreiben Sie eine Methode *plus()*, die die beiden obersten Stack-Elemente addiert - vorausgesetzt es sind Ziffern - und das Ergebnis wieder auf dem Stack speichert. Sind die beiden obersten Stackelemente keine Ziffern, so soll eine Fehlermeldung ausgegeben werden und nichts passieren. Kommt bei der Summe ein Wert von 10 oder mehr heraus, so soll 'a' für 10, 'b' für 11, … , 'i' für 18 oben auf dem Stack abgelegt werden.

Folgendes Beispiel zeigt das Verhalten:

```
Stck s = new Stck("bla1234");
s.plus(); // Nichts wird zurueckgeliefert, aber der Stack
          // verändert sich, nämlich, 3 und 4 werden addiert und ersetzt mit 7
s.pop(); // Es wird der char-Wert '7' zurueckgeliefert.
```

```
s.push('9');
s.plus(); // hier werden 2 und 9 addiert, das Ergebnis 11
          // wird als 'b' anstelle der beiden Zahlen im Stack abgelegt
s.pop(); // Es wird 'b' zurueckgeliefert.
```