Universität Leipzig **WS 14/15**
Institut für Informatik **Dr. Monika Meiler**

Modellierung und Programmierung 1
Übungsserie 4

Abgabetermin: 21.12.2014, 23:55 Uhr

Grundsätzlich sind Nebenrechnungen anzugeben und Antworten zu begründen.
Einzureichen sind, bei mehreren Dateien als .zip-Archiv:
Lösungen und UML-Klassendiagramme als .pdf-Datei, Programme als Quellcode,
Ergebnisdateien und Online-Dokumentationen .

---

1. **Rekursion**

   a) Entwickeln Sie in einer Klasse **Quer** eine iterative und eine rekursive Klassenmethode zur Berechnung der Quersumme einer `long`-Zahl.

   ```
   public static int itQuer( long n){...}
   public static int reQuer( long n){...}
   ```

   b) Eine Verallgemeinerung sind gewichtete Quersummen, bei denen die Ziffern erst mit den Werten einer periodischen Zahlenfolge (a_n) multipliziert und die Ergebnisse dann addiert werden. Dabei wird mit dem Einer begonnen.
   Sei die periodisch fortgesetzte Folge

   $$1,\ 3,\ 2,\ -1,\ -3,\ -2,\ 1,\ 3,\ 2,\ -1,\ -3,\ -2,\ ...$$

   und die zu prüfende Zahl 422 625. Dann ist die gewichtete Quersumme

   $$5 \cdot \mathbf{1} + 2 \cdot \mathbf{3} + 6 \cdot \mathbf{2} - 2 \cdot \mathbf{1} - 2 \cdot \mathbf{3} - 4 \cdot \mathbf{2} = 5 + 6 + 12 - 2 - 6 - 8 = 7.$$

   Erweitern Sie Ihre Klasse **Quer** um die Klassenmethode

   ```
   public static int wQuer( long n){...},
   ```

   welche die gewichtete Quersumme mit der angegebenen Folge (a_n) bestimmt.
   Schreiben Sie als Hauptmethode **main** ein Testprogramm, welches mit **itQuer**, **reQuer** und **wQuer** die Quersumme der Zahlen

   $$77\ 131\ 834,\ 8\ 876\ 826,\ 99\ 506\ 204$$

   berechnet. Leiten Sie die Ergebnisse in **Quer.out** um.
   Hinweis: Um das n-te Element a_n der Folge (a_n) zu erhalten, können Sie zum Beispiel ein Feld der Länge 6 anlegen, auf welches Sie mittels n modulo 6 zugreifen.

   c) Die Klassenmethode **wQuer** liefert eine Teilbarkeitsregel für die Zahl 7. Formulieren und beweisen Sie diese.
   Hinweis: Verwenden Sie für den Beweis die Summendarstellung für Zahlen im Positionssystem zur Basis 10.

2. **Fehlerfortpflanzung**

$$f(x) = e^x = \sum_{i=0}^{\infty} \frac{x^i}{i!} = 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \dots$$

Das Implementieren der Reihenentwicklung für die Exponentialfunktion soll die Fehlerfortpflanzung, bedingt durch die Zahlendarstellung im Rechner, demonstrieren.
Hinweis zum Programmieren einer Reihenentwicklung:
Um die Anzahl der Operationen zu minimieren, berechnet man einen neuen Summanden aus dem alten und addiert ihn anschließend zu der bis dahin berechneten Summe. Das wird solange wiederholt, bis ein neuer Summand so klein ist, dass er keinen Beitrag zur Summenbildung liefert, d.h. die Summe sich nicht mehr verändert (Rechnergenauigkeit):

```
double alteSumme, neueSumme = 1.0, summand = 1.0; int i = 1;    // Startwerte
do                                                              // Reihenentwicklung
{
   summand *= x / i++;                          // Berechnen des neuen Summanden
   alteSumme = neueSumme;                              // Merken der alten Summe
   neueSumme += summand;                          // Berechnen der neuen Summe
} while( neueSumme != alteSumme);                              // Abbruchbedingung
```

Stellen Sie in einer Klasse **Exp** zwei Klassenmethoden für die Exponentialfunktion bereit:

a) Implementieren Sie die Reihenentwicklung der Exponentialfunktion in einer Klassenmethode

```
public static double myExp(double x){...}
```

Tabellieren Sie in einem Testprogramm **Exp.java** die Funktionswerte $f(x)$ für $x \in [-20, 0]$ mit der Schrittweite 0.5 und vergleichen Sie die Ergebnisse Ihrer Methode mit denen der Methode **double Math.exp(double)** der Klasse **java.lang.Math**.

b) Fehlerbehandlung: Für einige $x < 0$ wird das Ergebnis negativ. Man berechnet für den Fall $x < 0$ deshalb $e^{|x|}$ und setzt $e^x = \frac{1}{e^{|x|}}$. Geben Sie eine verbesserte Klassenmethode

```
public static double myExpBesser(double x){...}
```

an, die diesen Fall unter Verwendung der Klassenmethode **myExp** gesondert behandelt. Wiederholen Sie den Test.
Erzeugen Sie durch Ausgabeumleitung eine Datei **Exp.out** mit den Testergebnissen.

Exp.java (Grobstruktur)

```
public class Exp
{
   public static double myExp(double x){ }
   public static double myExpBesser(double x){ }
   public static void main(String[] args)
   {
      for(double x = 0.0; x > -20.0; x-=.5)
      {
         // Berechnung und Ausgabe der Ergebnisse
         // myExp(x), myExpBesser(x), Math.exp(x)
      }
   }
}
```

3. **Modellierung - Militärverwaltung**

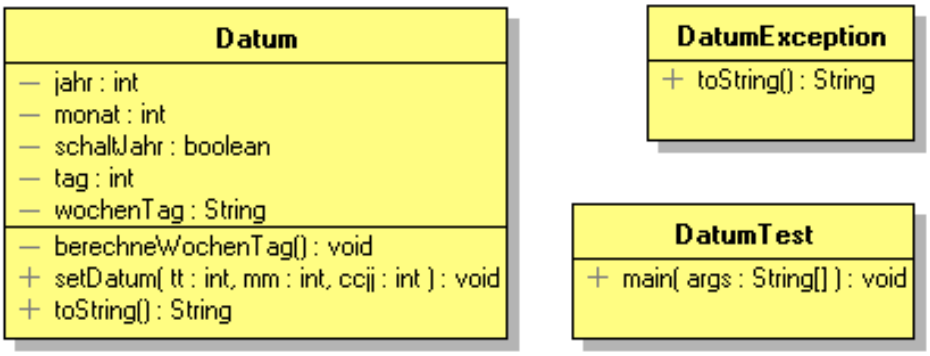
Ein nichtausbildendes *Regiment* beinhaltet *Kompanien* bestehend aus *Gefreiten*, *Unteroffizieren* und *Offizieren* sowie *zivilen Personen*. Jede *Kompanie* besitzt einen *Kompaniechef*, welcher stets ein *Offizier* ist. Der *Regimentskommandeur*, ebenfalls ein *Offizier*, möchte zur Verwaltung der Regimentskaserne ein Rechnerprogramm einsetzen, mit dem folgende Daten erfasst werden:
Vor- und *Nachname, Adresse (Strasse, Hausnummer, Postleitzahl, Ort), Geburtsdatum (Tag, Monat, Jahr)* aller Personen des Regiments, bei allen *Soldaten* die *abgeleisteten Dienstjahre* und bei allen *Zivilpersonen* deren *Tätigkeit* innerhalb des Regiments. Modellieren Sie die Regimentsstruktur als UML-Klassendiagramm (ohne Funktionalität). Achten Sie auf kleine, wiederverwendbare Klassen.

a) Geben Sie grafisch die Klassen einschließlich ihrer Instanzvariablen an und

b) stellen Sie die Beziehungen der Klassen untereinander dar.

4. **Objektorientierte Programmierung - Klasse Datum**

Gottfried Wilhelm Leibniz ist am 1.7.1646 geboren. Was für ein Wochentag war das? Um diese oder ähnliche Fragen zu beantworten, muss man den Wochentag aus dem Datum heraus berechnen können. Teilnehmer der Kopfrechenweltmeisterschaften erledigen diese Frage in wenigen Sekunden. Wir wollen eine Klasse **Datum** implementieren, welche diese Aufgabe für die Jahre zwischen 1600 und 3000 erfüllt und folgenden Klassenaufbau haben soll: Implementieren Sie die



Klasse mit ihren Methoden (einschließlich Online-Dokumentation):

a) Die Methode **setDatum** übernimmt ein korrektes Datum in die dafür vorgesehenen Instanzvariablen **tag**, **monat** und **jahr**, setzt **schaltJahr** und berechnet **wochenTag**. Ist das übergebene Datum nicht korrekt, setzt die Methode die Instanzvariablen **tag**, **monat**, und **jahr** auf 0 zurück und wirft eine Ausnahme (**DatumException**).

b) Die Methode **berechneWochentag** ist eine Hilfsmethode für die Methode **setDatum**. Sie berechnet aus dem Datum den Wochentag nach dem Gregorianischen Kalender, der seit 1582 gilt.

c) Die Methode **toString** erzeugt zu einem Datum einen String der Form:

„Mittwoch, am 15.12.2010“

Testen Sie Ihre Klasse in einem Programm **DatumTest.java**, welches wiederholt ein Datum einliest, es analysiert und bei Korrektheit als String ausgibt. Überprüfen Sie mit Ihrem Programm die folgenden Daten und ermitteln Sie den Wochentag:

- Die Gründung der Universität Leipzig fand am 2.12.1409 statt.
- Gottfried Wilhelm Leibniz wurde am 1.7.1646 geboren.
- Neil Armstrong betrat am 21.7.1969 als erster Mensch den Mond.
- Siegmund Jähn startete als erster deutscher Kosmonaut am 26.8.1978 in das Weltall.

Leiten Sie die Ergebnisse in eine Datei **DatumTest.out** um und erstellen Sie eine Online-Dokumentation für alle verwendeten Klassen mit

```
javadoc -private -d DatumDoc *.java
```

und beachten Sie, dass auch alle *privaten* Attribute und Methoden zu dokumentieren sind.

Hinweis:
Beachten Sie, dass ein Datum 29.2. nur in einem Schaltjahr korrekt ist. Schaltjahre sind durch 4 teilbare Jahre mit der Ausnahme der Jahrhunderwenden. Diese sind nur dann Schaltjahre, wenn auch das Jahrhundert durch 4 teilbar ist. Beispiel: 2000 war ein Schaltjahr (20 modulo 4 = 0), 1900 war kein Schaltjahr (19 modulo 4 = 3)!