Karlsruher Institut für Technologie

Programmieren – Wintersemester 2016/17

Architecture-driven Requirements Engineering (ARE)
https://sdqweb.ipd.kit.edu/wiki/Programmieren
Jun.-Prof. Dr.-Ing. Anne Koziolek

Übungsblatt 2

Ausgabe: 15.11.2016 – 17:00
Abgabe: 30.11.2016 – 13:00

- Sie müssen bei Ihren Lösungen eine maximale Zeilenbreite von 120 Zeichen einhalten.
- Programmcode muss in englischer Sprache verfasst sein.
- Kommentieren Sie Ihren Code angemessen: So viel wie nötig, so wenig wie möglich.
- Kommentare können sowohl in englischer Sprache, als auch in deutscher Sprache geschrieben werden. Die Sprache innerhalb eines Übungsblattes muss jedoch einheitlich bleiben.
- Verwenden Sie keine Klassen der Java-Bibliotheken ausgenommen Klassen der Pakete `java.lang` und `java.io`, es sei denn die Aufgabenstellung erlaubt ausdrücklich weitere Pakete.
- Achten Sie auf fehlerfrei kompilierenden Programmcode[1]

Abgabemodalitäten

Die Praktomat-Abgabe wird am **Mittwoch, den 23. November 2016, um 13:00 Uhr**, freigeschaltet.

- Geben Sie Ihre Antworten zu Aufgabe A als `.java`-Dateien ab.

Bitte beachten Sie, dass das erfolgreiche Bestehen der öffentlichen Tests für eine erfolgreiche Abgabe nötig ist. Planen Sie entsprechend Zeit für Ihren ersten Abgabeversuch ein.

A Kalender (20 Punkte)

In dieser Aufgabe sollen die grundlegenden Funktionalitäten für einen Kalender implementiert werden. Das primäre Ziel ist es sich mit dem Definieren, Implementieren und Aufrufen von Methoden in Java vertraut zu machen. Darüber hinaus sollen Sie sich mit der Datenkapselung, als ein wichtiges Konzept der objektorientierten Programmierung, vertraut machen.

Sie können während dieser Aufgabe davon ausgehen, dass alle übergebenen Methodenparameter immer korrekt sind. Das heißt, es werden nie negative oder *null*-Werte, sowie keine logisch falschen Werte übergeben.
Beispielsweise wird nie versucht werden eine Uhrzeit mit -30 Sekunden, ein Datum am 31. Februar oder einen Termin, bei welchem das Endzeitpunkt vor dem Anfangszeitpunkt liegt, zu instanziieren. Folglich können Sie während dieser Aufgabe davon ausgehen, dass auf alle übergebenen Methodenparameter immer problemlos eine lokale Reaktion möglich ist.

Die Grundlage für die Uhrzeit- und Datum-Klassen bildet der heute gebräuchliche gregorianische Kalender und die übliche astronomische Stundenzählung. Zeitzonen und Schaltsekunden werden zur Vereinfachung nicht betrachtet. Die wichtigsten Punkte zur Zeitdarstellung sind nochmals zusammengefasst:

[1] Der Praktomat wird die Abgabe zurückweisen, falls diese Regel verletzt ist.

Jahr Besteht aus *12* geordneten *Monaten* und dauert entweder *365 Tage* oder im Schaltjahr *366 Tage* und beginnt am *1. Januar (01.01)* und endet am *31. Dezember (31.12)*.

Monat Besteht aus 28, 29, 30 oder 31 aufeinanderfolgenden Tagen. Die geordnete Reihenfolge der 12 Monate des Jahres ist: *Januar* mit 31 Tagen, *Februar* mit 28 Tagen oder im Schaltjahr mit 29 Tagen, *März* mit 31 Tagen, *April* mit 30 Tagen, *Mai* mit 31 Tagen, *Juni* mit 30 Tagen, *Juli* mit 31 Tagen, *August* mit 31 Tagen, *September* mit 30 Tagen, *Oktober* mit 31 Tagen, *November* mit 30 Tagen und *Dezember* mit 31 Tagen.

Tag Besteht immer aus *24* gleich langen Stunden. Die Zeitangabe für die erste Stunde reicht von 00:00:00 Uhr bis 00:59:59 Uhr, für die 24. Stunde von 23:00:00 Uhr bis 23:59:59 Uhr. Gezählt wird ab Mitternacht, in der Zählung wechselt aber die Sekunde um 23:59:59 Uhr direkt auf die Sekunde um 00:00:00 Uhr des Folgetages. Jedem Tag im Jahr, lässt sich einer der sieben Wochentage zuordnen: *Montag*, *Dienstag*, *Mittwoch*, *Donnerstag*, *Freitag*, *Samstag* oder *Sonntag*.

Stunde Besteht immer aus *60* gleich langen Minuten.

Minute Besteht immer aus *60* gleich langen Sekunden.

Sekunde Wird nicht weiter in kleinere Einheiten zerteilt.

A.1 Musterlösung

Laden Sie sich zuerst die Musterlösung für Aufgabe E des letzten Übungsblattes herunter [2]. Implementieren Sie die Lösung dieses Übungsblattes auf Basis der Musterlösung für Aufgabe E des ersten Übungsblattes. Ändern Sie hierzu auf gar keinen Fall die Datentypen und Bezeichner der Attribute, Methoden, sowie der Klassen. Auch die Vorgaben der Signaturen von Konstruktoren und Methoden auf diesem Übungsblatt müssen exakt übernommen werden. Außer den explizit geforderten Attributen, Methoden und Konstruktoren dürfen Sie weitere Attribute, Methoden und Konstruktoren den Klassen hinzufügen, um Code-Duplikate zu vermeiden.

Folglich müssen die Klassen `TimeZone`, `TimeZoneOffset` und `ZonedDateTime` weder implementiert, noch hochgeladen werden.

A.2 Konstruktoren

Implementieren Sie jeweils die vorgegebenen Konstruktoren für die vier Klassen, um einen konsistenten Anfangszustand zu garantieren. Hierbei müssen im Rumpf des Konstruktors, die Werte der Parameter des Konstruktors den entsprechenden Attributen der zugehörigen Klasse zugewiesen werden. Eine Ausnahme bildet der zweite Konstruktor der `Appointment`-Klasse: Hier müssen Sie den Endzeitpunkt anhand des Anfangszeitpunktes und der Dauer errechnen.

A.2.1 Time

Implementieren Sie für die `Time`-Klasse den folgenden Konstruktor:

```
public Time(int hour, int minute, int second)
```

A.2.2 Date

Implementieren Sie für die `Date`-Klasse den folgenden Konstruktor:

```
public Date(int year, int month, int dayOfMonth)
```

[2] `https://sdqweb.ipd.kit.edu/wiki/Programmieren`

A.2.3 DateTime

Implementieren Sie für die `DateTime`-Klasse den folgenden Konstruktor:

```
public DateTime(Date date, Time time)
```

A.2.4 Appointment

Implementieren Sie für die `Appointment`-Klasse die folgenden Konstruktoren:

```
public Appointment(String name, DateTime from, DateTime to)
```

```
public Appointment(String name, DateTime from, Time duration)
```

A.3 Fabrikmethode

Fügen Sie den zwei Klassen `Time` und `Date` jeweils eine Methode hinzu, welche ein neues `DateTime`-Objekt instanziiert und dieses zurückgibt:

A.3.1 Date

Implementieren Sie hierzu für die `Date`-Klasse die folgende Methode:

```
public DateTime atTime(Time time)
```

A.3.2 Time

Implementieren Sie hierzu für die `Time`-Klasse die folgende Methode:

```
public DateTime atDate(Date date)
```

A.4 Immutable

Ein `Immutable`-Objekt ist in der objektorientierten Programmierung ein unveränderbares Objekt, dessen Zustand nach dessen Initialisierung nicht mehr geändert werden kann. Dies bedeutet, dass bei einem `Immutable`-Objekt alle relevanten Attribute mit dem Modifizierer `final` versehen sind. Einer Variable, welche mit dem Modifizierer `final` versehen ist, kann nur einmal ein Wert zugewiesen werden. Daher müssen finale Attribute immer bereits im Konstruktor initialisiert werden. Zum Beispiel ist die Klasse String[3] nach ihrer Initialisierung unveränderbar.

Versehen Sie jedes Attribut der Klassen `Time`, `Date` und `DateTime` mit dem Modifizierer `final` um diese drei Klassen nach der Initialisierung unveränderbar (immutable) zu machen.

A.5 Datenkapselung

Versehen sie jede Klasse, jedes Attribut, jede Methode und jeden Konstruktor aus dieser Aufgabe mit dem jeweils sinnvollsten Zugriffsmodifikatoren (`private`, `protected` oder `public` oder keiner). Erweitern Sie anschließend Ihre Klassen um die vorgegebenen Zugriffsfunktion (`getter-` / `setter`-Methoden), um die jeweilige Attribute der Objekte abzufragen oder ändern zu können.

[3] `https://docs.oracle.com/javase/8/docs/api/java/lang/String.html`

A.5.1 Time

Erweitern Sie die `Time`-Klasse um die folgenden `getter-` / `setter`-Methoden:

```
public int getHour()
public int getMinute()
public int getSecond()
```

A.5.2 Date

Erweitern Sie die `Date`-Klasse um die folgenden `getter-` / `setter`-Methoden:

```
public int getYear()
public int getMonthValue()
public int getDayOfMonth()
```

A.5.3 DateTime

Erweitern Sie die `DateTime`-Klasse um die folgenden Methoden:

```
public Date getDate()
public Time getTime()
```

A.5.4 Appointment

Erweitern Sie die `Appointment`-Klasse um die folgenden `getter-` / `setter`-Methoden:

```
public String getName()
public DateTime getFrom()
public DateTime getTo()
public void setName(String name)
public void setTo(DateTime to)
```

A.6 Textuelle Repräsentation

Fügen Sie folgenden fünf Klassen jeweils eine Methode (`public String toString()`) hinzu, um einen String für die textuelle Repräsentation zu erzeugen und zurückzugeben. Das Datumsformat entspricht der Form TT-MM-JJJJ, das Zeitformat entspricht der Form hh:mm:ss. Werden die beiden Formate kombiniert, entspricht die Zeitform TT-MM-JJJJThh:mm:ss.

Einstellige Werte bei den Monat-, Tag-, Stunden, Minuten- und Sekunden-Angaben werden stets mit einer `Null` auf zwei Stellen aufgefüllt. Zwischen Datumseinheiten wird das Zeichen Bindestrich, d.h. `-`, und zwischen Zeiteinheiten der Doppelpunkt, d.h. `:`, als Trennzeichen verwendet. Der Großbuchstabe `T` ist das Trennzeichen von Datum und Uhrzeit. Ein Beispiel für das Datum ist 14-06-2014 (14. Juni 2014), für die Uhrzeit 23:34:30 (23 Uhr, 34 Minuten und 30 Sekunden) und für beides zusammen 14-06-2014T23:34:30.

Für Termine wird eine Zeichenkette zurückgeben, der mit dem Namen des Termins beginnt, gefolgt von einem Leerzeichen und der textuellen Repräsentation des Anfangszeitpunktes, gefolgt von einem Leerzeichen und der textuellen Repräsentation des Endzeitpunktes.

In folgenden Beispielen wird das Zeitdarstellen für die jeweiligen Klassen illustriert:

A.6.1 Time

```
20:42:04
```

A.6.2 Date

```
30-01-2030
```

A.6.3 DateTime

```
13-12-2014T01:11:00
```

A.6.4 Appointment

```
Blatt 15-11-2016T13:00:00 30-11-2016T13:00:00
```

A.7 Kalenderrechnen

Erweitern Sie die `Date`-Klasse um vier neue Methoden, um zusätzliche Funktionalitäten zum Kalenderrechnen bereitzustellen.

A.7.1 getMonth

Abhängig vom Wert des `monthValue`-Attributes, gibt diese Methode die entsprechende Instanz des `Month`-Enum aus der Musterlösung zurück. Hierbei sind die zwölf Monate eines Jahrs auf natürliche Weise durchnummeriert, von Januar = 1 bis Dezember = 12.

A.7.2 getDayOfYear

Um angeben zu können, den wievielten Tag des Jahres die Instanz darstellt, gibt diese Methode eine Zahl von *1* bis entweder *365* oder im Schaltjahr *366* zurück.

A.7.3 getDayOfWeek

Diese Methode gibt die entsprechende Instanz des `DayOfWeek`-Enums aus der Musterlösung zurück. Hierbei sind die sieben Wochentage einer Woche auf natürliche Weise durchnummeriert, von Montag = 1 bis Sonntag = 7. Die Berechnung des Wochentages erfolgt nach folgender Formel:

$$d = c \bmod 7$$

$$w = \begin{cases} d + 6 & d = 0 \\ d - 1 & d > 0 \end{cases} \tag{1}$$

Hierbei wird mit mod die Modulo-Operation bezeichnet.

w ist der gesuchte Wochentag gemäß folgender Tabelle:

0	1	2	3	4	5	6
Sonntag	Montag	Dienstag	Mittwoch	Donnerstag	Freitag	Samstag

c ist die Anzahl der Tage des Datums, die seit dem 01.01.0000 vergangen sind.

A.7.4 isLeapYear

Der weltweit verbreitete Gregorianische Kalender nutzt Schalttage, um die durchschnittliche Länge eines Kalenderjahres an die Länge eines Sonnenjahres anzupassen. Denn während ein übliches Kalenderjahr aus 365 Tagen besteht, hat das Sonnenjahr eine Länge von 365,24219 Tagen. Ein Jahr, das durch Einschaltung eines zusätzlichen Schalttages verlängert wird, bezeichnet man als Schaltjahr. Ein Jahr ist genau dann ein Schaltjahr, wenn die Jahreszahl sich ohne Rest durch 400 teilen lässt oder wenn die Jahreszahl sich ohne Rest durch 4 teilen lässt, aber nicht durch 100. Ist die Jahreszahl durch 100 teilbar, aber nicht durch 400, ist es kein Schaltjahr.

Die Methode gibt nur dann `true` zurück, wenn das Jahr ein Schaltjahr ist, also aus 366 Tagen besteht.

A.8 Zusatzmethoden

Vereinfachen Sie die Aufrufe der Zugriffsfunktion auf den Attributen der `DateTime`-Klasse durch die Bereitstellung zusätzlicher Methoden in der `DateTime`-Klasse. Diese neuen Methoden rufen die ursprüngliche Methode in der `Time`- und/oder `Date`-Klasse auf:

A.8.1 DateTime

```
public int getYear ()
public int getMonthValue ()
public Month getMonth ()
public int getDayOfYear ()
public int getDayOfMonth ()
public DayOfWeek getDayOfWeek ()
public int getHour ()
public int getMinute ()
public int getSecond ()
```

A.9 Zeitrechnen

Erweitern Sie die drei Klassen `Time`, `Date` und `DateTime` um zusätzliche Methoden `plus` und `minus` zum Zeitrechnen bereitzustellen. Die Methoden verändern den Zustand des Objektes nicht, sondern liefern eine neue Instanz Plus / Minus den angegebenen Betrag zurück:

A.9.1 Time

Implementieren Sie für die `Time`-Klasse die folgenden Methoden:

```
public Time plus(Time time)
```

```
public Time minus(Time time)
```

In folgenden Beispielen wird das Zeitrechnen für die `Time`-Klasse illustriert:

```
12:00:00 + 12:00:00 = 00:00:00
23:59:59 + 00:00:01 = 00:00:00
11:11:11 + 11:11:11 = 22:22:22
12:00:00 - 12:00:00 = 00:00:00
00:00:00 - 00:00:01 = 23:59:59
11:11:11 - 01:01:01 = 10:10:10
```

A.9.2 Date

Implementieren Sie für die `Date`-Klasse die folgenden Methoden:

`public Date plus(Date date)`

`public Date plusYears(int years)`

`public Date plusMonths(int months)`

`public Date plusDays(int days)`

`public Date minus(Date date)`

`public Date minusYears(int years)`

`public Date minusMonths(int months)`

`public Date minusDays(int days)`

In folgenden Beispielen wird das Zeitrechnen für die `Date`-Klasse illustriert:

```
01-01-1000 + 01-01-1000 = 02-02-2000
01-01-1000 + 28-01-1000 = 01-03-2000
01-01-1000 + 28-01-1001 = 01-03-2001
02-02-2000 - 01-01-1000 = 01-01-1000
29-02-2000 - 28-01-1000 = 01-01-1000
01-03-2001 - 28-01-1001 = 01-01-1000
```

Somit muss beim Addieren zunächst der Tag, dann der Monat und dann das Jahr betrachtet werden.

A.9.3 DateTime

Implementieren Sie für die `DateTime`-Klasse die folgenden Methoden [4]:

`public DateTime plus(DateTime datetime)`
`public DateTime minus(DateTime datetime)`
`public DateTime plus(Date date)`
`public DateTime minus(Date date)`
`public DateTime plus(Time time)`
`public DateTime minus(Time time)`
`public DateTime plusYears(int years)`

[4] Für die Berechnung sollen keine Leap-Sekunden beachtet werden

`public DateTime plusMonths(int months)`
`public DateTime plusDays(int days)`
`public DateTime minusYears(int years)`
`public DateTime minusMonths(int months)`
`public DateTime minusDays(int days)`

In folgenden Beispielen wird das Zeitrechnen für die `DateTime`-Klasse illustriert:

```
01-01-1000T12:00:00 + 01-01-1000T12:00:00 = 03-02-2000T00:00:00
01-01-1000T23:59:59 + 28-01-1000T00:00:01 = 02-03-2000T00:00:00
01-01-1000T11:11:11 + 28-01-1001T11:11:11 = 01-03-2001T22:22:22
27-01-2000T00:00:00 + 05-01-0000T00:00:00 = 01-03-2000T00:00:00
```

Somit muss beim Addieren zunächst die Uhrzeit und dann das Datum betrachtet werden.

A.10 Ordnungsrelation

Implementieren Sie für die drei Klassen `Time`, `Date` und `DateTime` jeweils drei Methoden `public boolean isBefore (Time other)`, `public boolean isEqual (Time other)` und `public boolean isAfter (Time other)` für die `Time`-Klasse, `public boolean isBefore (Date other)`, `public boolean isEqual (Date other)` und `public boolean isAfter (Date other)` für die `Date`-Klasse und `public boolean isBefore (DateTime other)`, `public boolean isEqual (DateTime other)` und `public boolean isAfter (DateTime other)` für die `DateTime`-Klasse mit welchen sich die Objekte miteinander vergleichen lassen:

- Die `isBefore`-Methode gibt nur dann `true` zurück, falls das Objekt, auf dem die Methode aufgerufen wird, zeitlich *vor* der Zeitangabe des Parameters vorkommt.
- Die `isEqual`-Methode gibt nur dann `true` zurück, falls das Objekt, auf dem die Methode aufgerufen wird und das Objekt, das als Parameter übergeben wird, *zeitlich gleich* sind.
- Die `isAfter`-Methode gibt nur dann `true` zurück, falls das Objekt auf dem die Methode aufgerufen wird, zeitlich *nach* der Zeitangabe des Parameters vorkommt.