Aufgabe 3 Terminkalender

Aufgabenstellung Überblick

Sie sollen ein Modell für einen Terminkalender implementieren und dabei eine Reihe an vorgegebenen Interfaces implementieren.

Die Interfaces sind im mitgelieferten Projekt bereits vorhanden.

Teilweise sind auch Implementierungen (für die komplizierteren Aufgaben) vorhanden.

Aufgabenstellung

Kopieren Sie das mitgelieferte Projekt in Ihre Eclipse-Umgebung und implementieren Sie die Interfaces / Klassen, wie beschrieben.

Implementieren / ergänzen Sie alle Methoden in den Klassen, die mit TODO gekennzeichnet sind.

Implementieren Sie die Methoden ***toString*** und die Methoden ***equals*** und ***hashCode*** geeignet.

Achten Sie bei ***TerminMitWiederholungImpl*** darauf, dass ***equals*** und ***hashCode*** nicht alle Termine expandieren.

Versuchen Sie wann immer möglich und sinnvoll mit Streams zu arbeiten.

Versuchen Sie sich zuerst an den **einfachen** Klassen.

Überblick über das Modell

Terminkalender

Das Interface ***TerminKalender*** spezifiziert die Funktionalität für die Implementierung eines Terminkalenders.

```
public interface TerminKalender {
	/*
	 * Trägt eine Termin in den Kalender ein
	 */
	public boolean eintragen(Termin termin);

	/*
	 * Verschiebt das Datum des Termins auf das übergebene Datum
	 */
	public void verschiebenAuf(Termin termin, Datum datum);

	/*
	 * Löscht den Termin termin aus dem Kalender gibt true zurück, wenn termin
	 * enthalten ist, sonst false
	 */
	public boolean terminLoeschen(Termin termin);

	/*
	 * Prüft, ob termin im Kalender enthalten ist
	 */
	public boolean enthaeltTermin(Termin termin);

	/*
	 * Liefert alle Termine eines Tages als Tabelle zurück. Unter einem Datum
	 * sind alle Termine mit gleichem Datum eingetragen Der Schlüssel der
	 * Tabelle ist ein Datum. Der Wert die Liste der Termine mit gleichem Datum.
	 *
	 * Die Termine eines Tages, sind die Termine, die zwischen 00:00 Uhr und
	 * 23:59 liegen
	 */
	public Map<Datum, List<Termin>> termineFuerTag(Tag tag);

	/*
	 * Liefert alle Termine einer Woche als Tabelle zurück. Unter einem Datum
	 * sind alle Termine mit gleichem Datum eingetragen Der Schlüssel der
	 * Tabelle ist ein Datum. Der Wert die Liste der Termine mit gleichem Datum.
	 *
	 * Die Termine einer Woche, sind die Termine, die zwischen 00:00 Uhr des
	 * ersten Tages der Woche und 23:59 des letzen Tages der Woche liegen.
	 */
	public Map<Datum, List<Termin>> termineFuerWoche(Woche woche);

	/*
	 * Liefert alle Termine eines Monats als Tabelle zurück. Unter einem Datum
	 * sind alle Termine mit gleichem Datum eingetragen Der Schlüssel der
	 * Tabelle ist ein Datum. Der Wert die Liste der Termine mit gleichem Datum.
	 *
	 * Die Termine eines Monats, sind die Termine, die zwischen 00:00 Uhr des
	 * ersten Tages des Monats und 23:59 des letzen Tages des Monats liegen.
	 */
	public Map<Datum, List<Termin>> termineFuerMonat(Monat monat);
}
```

Termin

Das Interface ***Termin*** spezifiziert die Funktionalität von Terminen. In dem Terminkalender sollen **einfache Termine**, Termine ohne Wiederholungen, und komplexere Termine, **Termine mit Wiederholungen** verwaltet werden. Das Interface ***Termin*** spezifiziert die Funktionalität für einfache Termine vollständig. Termine sind vergleichbar. Die Ordnung ergibt sich aus der Ordnung ihres Datums.

```
public interface Termin extends Comparable<Termin> {
	/*
	 * Liefert die Beschreibung eines Termins, die bei der Erzeugung übergeben
	 * wird.
	 */
	public String getBeschreibung();
	/*
	 * Liefert das Datum eines Termins, das bei der Erzeugung übergeben wird.
	 */
	public Datum getDatum();
	/*
	 * Liefert die Dauer eines Termins, die bei der Erzeugung übergeben wird.
	 */
	public Dauer getDauer();
	/*
	 * Verschiebt den Termin auf das übergebene Datum.
	 */
	public Termin verschiebeAuf(Datum datum);
	/*
	 * Liefert Termine in einem Monat. Um einfache Termine und Termine mit
	 * Wiederholungen konsistent zu behandeln, liefert auch ein einfacher Termin
	 * eine Map mit Datum und Termin.
	 *
	 * Für einfache Termine enthält die Map maximal ein Element, den Termin
	 * selbst, wenn dieser im geforderten Monat liegt.
	 *
	 * Für Termine mit Wiederholungen kann die Map mehrere Einträge enthalten-
	 * alle Wiederholungen, die in dem Monat liegen. Dabei soll für verschiedene
	 * Datumsangaben nur die Referenz auf den Termin mit Wiederholungen
	 * eingetragen werden.
	 *
	 * Zur weiteren Erläuterung siehe auch termineFuerMonat in TerminKalender.
	 */
	public Map<Datum, Termin> termineIn(Monat monat);
	/*
	 * Liefert die Termine einer Woche.
	 *
	 * Zur weiteren Erläuterung siehe termineIn(Monat) sowie termineFuerWoche in
	 * TerminKalender.
	 */
	public Map<Datum, Termin> termineIn(Woche woche);
	/*
	 * Liefert die Termine eines Tages.
	 *
	 * Zur weiteren Erläuterung siehe termineIn(Monat) sowie termineFuerTag in
	 * TerminKalender.
	 */
	public Map<Datum, Termin> termineAn(Tag tag);
}
```

TerminMitWiederholung

Ein Termin mit Wiederholung ist ein Termin, der sich täglich oder wöchentlich in einem gewissen Zyklus (z.B. alle 2 Wochen) wiederholt und eine obere Grenze für die Wiederholungen hat. Diese obere Grenze kann sehr groß werden, wenn es sich um einen Routine-Termin handelt. Termine ohne Begrenzung, d.h. unendliche Wiederholungen, soll es zunächst nicht geben, wären aber denkbar.

Daher dürfen wir einen Termin mit Wiederholungen nicht „expandieren“, d.h., alle Wiederholungen explizit aufzählen. Das könnte bei sehr vielen Wiederholungen die **Performance** beeinträchtigen und wäre bei unendlichen Wiederholungen **unmöglich**.

Um die Wiederholungen dennoch zu erhalten, betrachten wir nur **Ausschnitte =** Intervalle für ***Datum***-Objekte. Die Ausschnitte sind definiert über die Datumsgrößen ***Tag, Woche und Monat***. Für diese Größen werden Ausschnitte, also die Indizes des Intervalls bestimmt, in dem zugesichert Wiederholungen des Termins liegen. Die Methoden, die zu einer Datumsgröße die Indizes für das Intervall liefern, sind bereits implementiert. (Zur Erläuterung siehe die innere Klasse ***WiederholungImpl*** in der Klasse ***TerminMitWiederholungImpl***).

```
public interface TerminMitWiederholung extends Termin, Iterable<Termin> {
	/*
	 * Liefert ein Objekt von Typ Wiederholung zurück. Die Wiederholung kennt
	 * die Anzahl der Wiederholungen, den Typ der Wiederholung und den Zyklus
	 * der Wiederholung. Die Wiederholung wird beim Erzeugen des Termins
	 * übergeben.
	 */
	public Wiederholung getWdh();
	/*
	 * Ein Iterator, der in einem vorgebenen Intervall das Datum der
	 * Wiederholungen des Termins, die in dem Intervall liegen, zurückliefert.
	 *
	 * von ist der erste Intervallindex (inkl.), bis letzte Intervallindex
	 * (inkl.).
	 *
	 * Zu Details siehe Erläuterung zur Implementierung in der Klasse
	 * TerminMitWiederholungImpl.
	 */
	public IntervallIterator<Datum> intervallIterator(int von, int bis);
	/*
	 * Liefert alle Wiederholung, die innerhalb einer DatumsGroesse (Tag, Woche,
	 * Monat) liegen, also z.B. alle Wiederholungen, die in einer Woche liegen.
	 *
	 * Diese Methode lässt sich generisch implementieren, zu Details siehe die
	 * Erläuterungen zur Implementierung in der Klasse
	 * TerminMitWiederholungImpl.
	 */
	public Map<Datum, Termin> termineFuer(DatumsGroesse groesse);

}
```

Datum

```
/*
 * Datum kapselt die Darstellung von Datums und Zeitangaben als Calendar-Objekt
 * in Java.
 * Ein Datum besteht aus Datumsangabe (eindeutig durch Tag, Woche, Monat und Jahr)
 * und einer Zeitangabe dargestellt als Uhrzeit.
 *
 * Intern verwendet Datum eine Calendar-Objekt für die Berechnungen und
 * Umrechnungen.
 */
public interface Datum extends Comparable<Datum> {
	/*
	 * Liefern den Tag, Woche, Monat, Uhrzeit des Datums
	 */
	public Tag getTag();
	public Woche getWoche();
	public Monat getMonat();
	public Uhrzeit getUhrzeit();
	/*
	 * Zugriff auf z.T. nicht eindeutige Datumsanteile. Wrapper um die get
	 * Methoden von Calendar.
	 */
	public int getJahr();
	public int getTagImMonat();
	public int getTagImJahr();
	public int getWocheImMonat();
	public int getWocheImJahr();
	public int getMonatImJahr();
	/*
	 * Addieren und Subtrahieren einer Dauer. Addiert und Subtrahiert werden die
	 * Minuten einer Dauer. Addiert/subtrahiert wird unter Verwendung der
	 * add-Methode von Calendar. (--> inBasis())
	 */
	public Datum add(Dauer dauer);
	public Datum sub(Dauer dauer);
	/*
	 * Berechnet den Abstand zwischen zwei Datums-objekten als Dauer durch
	 * Umrechnung beider Datumsobjekte in Minuten. Darf negativ werden.
	 */
	public Dauer abstand(Datum d);
	/*
	 * Berechnet die zeitliche Differenz zwischen zwei Datums-objekten in Anzahl
	 * Tagen. Nutzt dazu die Methode differenzInTagen von Tag.
	 */
	public long differenzInTagen(Datum d);
	/*
	 * Rechnet das Datums-objekt in Minuten seit dem 1.1.1970 00:00:00 um.
	 * Verwendet dazu die Methode getTimeMillis() von Calendar.
	 */
	public int inMinuten();
	/*
	 * Transformiert ein Datums-objekt und erzeugt ein neues Calendar Objekt,
	 * bzw. cloned das interne Calendar Objekt.
	 */
	public Calendar inBasis();
}
```

Dauer

Dauer repräsentiert eine Zeitspanne, die auch negativ werden darf. Die Basiseinheit für die Darstellung von Dauer sind Minuten, d.h. eine verlustfreie Umwandlung einer Dauer ist nur bei der Umrechnung in Minuten garantiert. Die Ordnung für Dauerobjekte wird über die Minutendarstellung definiert.

```
/*
 * Dauer repräsentiert eine Zeitspanne (auch negative).
 * Dauern sind vergleichbar ueber ihre Minutendarstellung
 */
public interface Dauer extends Comparable<Dauer> {
	/*
	 * Gibt den Wert der Dauer in Minuten, Tagen, Stunden, Wochen wieder. Die
	 * Anteile in Tagen, Stunden und Wochen sind verlustbehaftet, wenn die
	 * Minuten nicht ein Vielfaches der jeweiligen Groesse sind.
	 */
	public int inMinuten();
	public int inStunden();
	public int inTagen();
	public int inWochen();
	/*
	 * Berechnet die Anteile für die Zeitgroessen an der Gesamtdauer.
	 * Die Addition der Anteile muss den Wert der Dauer ergeben.
	 */
	public int anteilMinuten();
	public int anteilStunden();
	public int anteilTage();
	public int anteilWochen();
}
```

DatumsGroesse

Das Interface ***DatumsGroesse*** fasst Funktionalitäten der Typen ***Tag***, ***Woche***, ***Monat*** zusammen. ***Datum*** ist keine Datumsgröße, da Daten neben Datumsangaben auch die Uhrzeit als Eigenschaft haben.

```
public interface DatumsGroesse {
	/*
	 * Gibt das ersten Datum einer DatumsGroesse zurück.
	 * Das erste Datum hat immer die Uhrzeit 00:00.
	 */
	public Datum getStart();
	/*
	 * Gibt das letzte Datum einer DatumsGroesse zurück.
	 * Das letzte Datum hat immmer die  Uhrzeit 23:59.
	 */
	public Datum getEnde();
}
```

Tag, Woche, Monat

```
/**
 *          Tag repraesentiert einen eindeutigen Tag im Kalender. Eindeutig wird
 *          ein Tag im Kalender durch Kombination von Jahr, Monat und Tag im Monat
 *          oder durch Kombination von Jahr und Tag im Jahr. Eine dieser
 *          Kombinationen muss bei der Erzeugung von Tagen übergeben werden.
 *
 *          Eine Implementierung sollte intern ein Calendar-Objekt für die
 *          Transformationen und Umrechnungen verwenden.
 *
 *          Tage lassen sich über die Transformation in ein Calendar-Objekt
 *          vergleichen.
 */
public interface Tag extends DatumsGroesse, Comparable<Tag> {
	/*
	 * berechnet Jahr, Monat, TagImJahr, TagImMonat als int. verwendet geeignete
	 * get Methoden von Calendar
	 */
	public int getJahr();
	public int getMonat();
	public int getTagImJahr();
	public int getTagImMonat();
	/*
	 * Berechnet die Differenz von Tagen (this-other). Dazu werden die Tage in
	 * die Calendar-Basis umgerechnet und die Differenz der
	 * Millisekundendarstellung in Anzahl Tage umgerechnet.
	 */
	public long differenzInTagen(Tag other);
	/*
	 * Transformiert Tag in ein Calendar-Objekt. Lässt sich durch Clonen der
	 * internen Darstellung erreichen.
	 */
	public Calendar inBasis();
}
```

Analog sind die Interfaces für ***Woche*** und ***Monat*** zu interpretieren.

```
public interface Monat extends DatumsGroesse {
	public int getMonat();
	public int getJahr();
}

public interface Woche extends DatumsGroesse {
	public int getJahr();
	public int getMonat();
	public int getWocheImMonat();
	public int getWocheImJahr();
}
```

Implementierung von ***Woche*** ist als Beispiel für das Berechnen von Start und Ende ***Datum*** enthalten.

Uhrzeit

Uhrzeit kapselt die Zeitangaben von Kalender-Objekten in Java. Für die interne Repräsentation gilt das Gleiche wie für ***Datum, Tag, Woche und Monat***. Implementierung ist als Beispiel für den Umgang mit Calendar-Objekten enthalten.

```
public interface Uhrzeit extends Comparable<Uhrzeit>{
	public int getStunde();
	public int getMinuten();

}
```

Wiederholung

```
/**
 * @author Birgit Wendholt Das Interface Wiederholung repräsentiert alle für
 *         eine Terminwiederholung relevanten Aspekte (Typ, Zyklus, Anzahl)
 *
 *         und spezifiziert Methoden, um
 *
 *         abhängige Größen wie Intervalllänge und maxIntervalIndex zu berechnen
 *         Wiederholungsintervalle zu berechnen
 *         das Datum der naechsten (n)Wiederholung(en) zu berechnen
 *         den Wiederholungszähler zu inkrementieren / dekrementieren
 *
 *         Eine Implementierung findet sich als innere Klasse der Klasse
 *         TerminMitWiederholungImpl
 */
public interface Wiederholung {
	/*
	 * Eigenschaften einer Wiederholung, die bei Erzeugen übergeben werden
	 * müssen
	 *
	 * Typ: tägliche / wöchentliche Wiederholung Zyklus: Wiederholungszyklus
	 * (z.B. alle 2 Wochen) Anzahl: Anzahl der Wiederholungen
	 */
	public WiederholungType getType();
	public int getZyklus();
	public int anzahl();
	/*
	 * Der Abstand zwischen zwei Wiederholungen in Anzahl von Tagen Produkt aus
	 * Anzahl der Tage des WiederholungsTyps und dem Zyklus
	 */
	public int intervallLaenge();
	/*
	 * Ist gleich der Anzahl der Wiederholungen. Bezeichnet das letzte
	 * Intervall, in dem noch eine Wiederholung des Termins liegt.
	 */
	public int maxIntervallIndex();
	/*
	 * Zu einem beliebigen Datum wird bestimmt, in welchem Intervall des Termins
	 * relativ zum Datum eine Wiederholung liegt. Das Ergebnis ist ein
	 * Intervall-Index für eine Wiederholung, die nach dem Datum liegt. Mit
	 * Hilfe dieses Index lassen die die Wiederholungen zu einem Termin mit
	 * naechstesDatum direkt berechnen.
	 */
	public int naechstesIntervall(Datum datum);
```

```
	/*
	 * Analog naechstesIntervall nur dass hier der Intervallindex vor dem Datum
	 * berechnet wird.
	 */
	public int vorherigesIntervall(Datum datum);
	/*
	 * Das Datum der nachfolgenden Wiederholung. siehe auch die "naive" Iterator
	 * Implementierung in TerminMitWiederholungImpl
	 */
	public Datum naechstesDatum();
	/*
	 * Das Datum der k-ten Wiederholung des Termin, k ist auch der
	 * Intervall-Index. Die Methoden naechstesIntervall/vorigesIntervall
	 * berechnen diesen Index.
	 */
	public Datum naechstesDatum(int k);
	/*
	 * Addition und Subtraktion eines wdhCount mit/von der Anzahl der
	 * Wiederholungen
	 */
	public Wiederholung sub(int wdhCount);
	public Wiederholung add(int wdhCount);
}
```

WiederholungType

Enum zur Verwaltung von Wiederholungstypen. Achten Sie darauf, dass die Enum-konstanten die Umrechnung in Tage vornehmen!!!!

```
public enum WiederholungType {
	TAEGLICH("täglich") {
		@Override
		public int inTagen() {
			return 1;
		}
	},
	WOECHENTLICH("wöchentlich") {
		@Override
		public int inTagen() {
			return 7;
		}
	};

	private String name;

	private WiederholungType(String name) {
		this.name = name;
	}

	@Override
	public String toString() {
		return name;
	}

	public abstract int inTagen();
};
```