Aufgabe 3 Modell eines Reisebuchungssystems (Teil A)

Lernziele

1. Ein etwas umfangreicheres Modell implementieren.
2. Abstrakte Klassen und Interfaces verwenden.
3. Wissen über Collection-Klassen vertiefen
4. Wissen über Streams für Collections vertiefen.
5. Wissen über Enums anwenden.
6. Das Rechnen mit Währungsgrößen realisieren.

Hintergrund

Reisebuchungssysteme finden Sie im Internet vielfältig. Sie sollen mit dieser Aufgabe ein vereinfachtes Modell eines Reisebuchungssystems für Reisen, die aus Flüge, Hotelbuchungen und Mietwagen bestehen, umsetzen. Das Modell ist in der Struktur vorgegeben. Ihre Aufgabe wird es sein, das Modell zu verstehen und die abstrakten Methoden und Interfaces zu implementieren.

Aufbau des Projektes

Das Projekt enthält 6 Packages:

1. ***de.hawhh.akteure***: Natürliche und juristische Personen, die bei einer Reisebuchung auftreten. (gegeben)
2. ***de.hawhh.kosten:*** Typen zur Repräsentation von Preisen und Rabatten.
3. ***de.hawhh.reisebuchung***: Allgemeine Typen für eine Reisebuchung
4. ***de.hawhh.reisebuchung.flug***: Typen für das Buchen von Flügen
5. ***de.hawhh.reisebuchung.hotel***: Typen für das Buchen von Hotels
6. ***de.hawhh.reisebuchung.mietwagen***: Typen für das Buchen von Mietwagen

Das Modell (noch ohne Hotel und Mietwagen)

Package: de.hawhh.reisebuchung

Eine Reisebuchung enthält neben den Referenzen auf Kunde, das Reisebüro und eine Referenz auf die gebuchte Reise.

Eine ***ReiseBuchung*** gibt Auskunft über Beginn, Ende, Dauer, StartOrt, EndOrt, und den Preis einer Reise. Diese Funktionalität wird durch das Interface ***Buchung*** gefordert.

Eine ***Reise*** ist eine ***Buchung***. Im Unterschied zu einer Reisebuchung enthält sie keine Referenzen auf Kunde, Reisebüro und sich selbst. Dieser Sachverhalt lässt sich durch Interface-Vererbung darstellen. ***Reise*** ist ein sogenanntes Tag-Interface.

Eine Reise setzt sich aus mehreren Reisebausteinen zusammen. Das gilt für alle konkreten Subtypen von Reise. Den Aufbau aus mehreren Reisebausteinen können wir daher in einer gemeinsame Klasse ***AbstractReise*** abstrahieren.

AbstractReise verwaltet eine Liste von Reisebausteinen und implementiert auf Basis dieser Liste alle Methoden des Interfaces Reise. Darüber hinaus enthält die Klasse Methoden zum Hinzufügen und Löschen von Reisebausteinen (***add***/***remove***). Diese Methoden dürfen nur für ableitende Klassen sichtbar sein.

Ein ***ReiseBaustein*** ist eine Reise. Da es unterschiedliche konkrete Bausteine gibt, kann von diesem Typ kein Objekt erzeugt werden. Allerdings soll ein ***ReiseBaustein*** aus Beginn und Ende einer Reise die Dauer berechnen.

Wir haben zwei konkrete Klassen für Reisen: ***EinfachReise*** und ***RundReise***.

Eine ***EinfachReise*** besteht aus einen Hin,- und Rueckflug (***ToFroFlug***), einem ***HotelBaustein*** und optional einem ***MietwagenBaustein***. Die Bausteine werden bei der Erzeugung dieser Reise übergeben und können danach nicht mehr modifiziert werden.

Eine ***RundReise*** besteht aus beliebig vielen Bausteinen in Anzahl und Typ. Einzelne Bausteine sollen der Rundreise hinzugefügt und aus ihr entfernt werden können.

Package: de.hawhh.reisebuchung.flug

Ein ***Flug*** ist ein ***ReiseBaustein***. Wir unterscheiden ***OneWayFlug*** und ***ToFroFlug*** (Hin und Rückflug). ***OneWayFlug*** wird nochmals unterschieden in ***DirektFlug*** und ***StopOverFlug***.

Ein ***DirektFlug*** ist gekennzeichnet durch die ***FlugNummer***, die ***IataAirline***, den Abflug- sowie den Ankunftsflughafen (***IataAirport***), die Abflugs- und Ankunftszeit (***LocalDateTime***), sowie einen Preis (***GeldBetrag***). Diese Eigenschaften werden bei der Erzeugung von DirektFlug übergeben.

DirektFlug muss die noch offenen abstrakten Methoden von ***ReiseBaustein*** / ***Reise*** geeignet implementieren.

Ein ***StopOverFlug*** ist eine ***OneWayFlug*** mit 1 oder mehreren Zwischen-Stops. Wir modellieren einen StopOverFlug als Sammlung von Direktflügen. Der StopOverFlug hat Methoden zum Hinzufügen und Löschen von Direktflügen (***add***/***remove***) und implementiert alle noch offenen abstrakten Methoden von ***ReiseBaustein*** / ***Reise*** durch geeignete Verarbeitung der Liste der Direktflüge. (Modellieren Sie die Sammlung als ***TreeSet*** und geben dem TreeSet einen ***Comparator*** mit, der die einzelnen Direktflüge in eine chronologische Reihenfolge bringt.) Beim Hinzufügen eines Direktfluges soll geprüft werden, ob es eine Lücke in der Liste der Direktflüge gibt, in die der neue Direktflug passt. Wenn nicht, soll die Methode ***add false*** zurückgeben.

Ein ***ToFroFlug*** besteht zwei ***OneWay*** Flügen, einem für den Hinflug und einem für den Rückflug. Diese werden bei der Erzeugung übergeben. Bei der Erzeugung muss geprüft werden, ob der Rückflug mindestens 2 Stunden nach dem Landen startet, sonst muss mit einer ***IllegalArgumentException*** das Erzeugen verhindert werden.

Klassen IataAirport, IataAirline, FlugNummer

IataAirport ist die Klasse aus Praktikumsaufgabe 2 erweitert um die Methode ***getLocation***(). Ergänzen Sie Aufgabe 2, um den Ort eines Flughafens zu extrahieren und in die IataAirport Objekte zu schreiben.

IataAirline ist die Klasse aus Praktikumsaufgabe 2.

FlugNummer setzt sich zusammen aus einem IataAirline Code gefolgt von 4 Ziffern.

Die Interfaces Buchung und Reise

```
public interface Buchung {
      public LocalDateTime getBeginn();
      public LocalDateTime getEnde();
      public GeldBetrag getPreis();
      public Duration getDauer();
      public Ort getEndOrt();
      public Ort getStartOrt();
}
```

Beginn und Ende einer Reise werden durch den Javatyp ***LocalDateTime*** repräsentiert. Dieser enthält Angaben zu Datum und Uhrzeit. Die Dauer Reise durch den Javatyp ***Duration***. Mit der statischen Methode ***Duration.between*** lässt sich die Dauer zwischen zwei ***LocalDateTime***-Objekten berechnen.

Die Klasse ***Ort*** ist gegeben.

Package: de.hawhh.kosten

```
public interface GeldBetrag {

	public double inBasis();
	public GeldBetrag ausBasis(double basis);

	public GeldBetrag add(GeldBetrag p2);
	public GeldBetrag sub(GeldBetrag p2);
	public GeldBetrag div(double d);
	public GeldBetrag mult(double d);

	public int getMajor();
	public int getMinor();
	public String getMinorSymbol();
	public String getMajorSymbol();
}
```

Das Interface dient der Modellierung von Geldbeträgen in unterschiedlichen Währungen. Um mit Geldbeträge unterschiedlicher Währungen rechnen zu können, legen wir eine Währung als Basis zugrunde, in die alle Geldbeträge transformiert werden (***inBasis***). Mit derart normalisierten Geldbeträgen können wir rechnen. Nach der Rechnung transformieren wir das Ergebnis mit ***ausBasis*** wieder in unsere Währungseinheit (Faktoren für die Umrechnung zwischen Währungen finden sich im Netz.)

getMajor gibt den Anteil der Vorkomma-stelle, ***getMinor*** den Anteil nach der Nachkommastelle zurück. (z.B. 3,5 €: 3-major, 5-minor) . MajorSymbol / MinorSymbol: (€ /ct) oder ($/ct) etc.

Die Klasse ***AbstractGeldBetrag*** soll die mathematischen Operation auf Basis der Methoden ***inBasis*** und ***vonBasis*** implementieren.

Die Klasse ***RabattModell*** Anzahl von Wochen einen Rabatt zwischen 0..1 zu. Rabattmodelle werden mit einem Array für Wochenanzahlen und einem gleichlangen Array für Rabatte initialisiert. Die Inhalte der beiden Arrays sollen in eine Tabelle (***Map***) übersetzt werden. RabattModell hat nur eine öffentliche Methode ***get(long wochenanzahl)***, die den zugehörigen Rabatt zurückliefert. Rabatte werden bei der Bepreisung von Mietwagen und Hotels benötigt.

TODO

Implementieren Sie die Klassen der Packages *de.hawhh.reisebuchung* und *de.hawhh.reisebuchung.flug* und stellen Sie geeignete Vererbungs- bzw. Implementierungs-Beziehungen her. Die benötigten Interfaces sind gegeben.

Generieren Sie für alle konkreten Klassen eine geeignete Methode ***toString***. Generieren Sie geeignete Methoden für Gleichheit und den Hash Code.

Achten Sie bei der Implementierung darauf, dass Sie die Funktionalität an geeigneter Stelle abstrahieren.

Informieren Sie sich über die Klasse ***TreeSet*** / ***Comparator*** und nutzen Sie die Besonderheiten der Klasse ***TreeSet*** für das chronologische Anordnen von Direktflügen.

Verwenden Sie, wenn immer möglich Streams bei der Verarbeitung von Collections!

Informieren Sie sich hier http://www.oracle.com/technetwork/articles/java/jf14-date-time-2125367.html und in der jeweiligen API Dokumentation über die Klassen ***LocalDateTime*** und ***Duration***.

Implementieren Sie die Klasse ***AbstractGeldBetrag*** und die mathematischen Operationen.

Schreiben Sie die konkreten Klassen ***Euro*** und ***Dollar*** für Geldbeträge. Die Objekte der Klassen werden mit ganzzahligen Wert/Werten initialisiert.

Implementieren Sie die Klasse ***RabattModell***.