Aufgabe 1: "Appointments"

Gesucht ist die Realisierung eines **Terminplaners**, welche es erlaubt **beliebig viel Termine** zu verwalten. Realisieren Sie dazu die folgenden Klassen:

Aufgabe 1a: "Appointment" [12 Punkte]

Jeder **Termin** (Klasse **`Appointment`**) findet an einem bestimmten **Tag** (`Date`), zu einem **Startzeitpunkt** und eine **Endzeitpunkt** (jeweils `TimeOfDay`) statt, wird durch eine global eindeutige **Id** (`int`) identifiziert und hat eine beliebige **Bezeichnung** (`String`). Darüber hinaus, kann zu jedem Termin eine **beliebige Anzahl von Teilnehmern** hinzugefügt werden, welche an dem Termin teilnehmen sollen, von denen (nur) deren **eMail** gespeichert werden soll.

Der **Tag** an dem der Termin stattfindet wird durch ein **`Date`-Objekt** repräsentiert (dh. Termine können sich nicht über mehrer Tage erstrecken). Verwenden Sie dazu die zur Verfügung gestellte Klasse `Date`.

Der **Startzeitpunkt** und der **Endzeitpunkt** wird jeweils durch ein **`TimeOfDay`-Objekt** repäsentiert. Verwenden Sie dazu die zur Verfügung gestellte Klasse `TimeOfDay`.

Die **Id** des Termins soll *global eindeutig* sein und beim Erzeugen eines neuen Termins *automatisch vergeben* werden - diese kann nicht verändert werden (vgl. Beispiel `Account` aus Vorlesung).

Zu jedem Termin soll eine beliebige Anzahl von **Teilnehmern** (genauer, nur deren eMail-Addresse) gespeichert werden können. **Realisieren Sie dies als einfach-verkette Liste**. Die Liste muss dazu keiner bestimmten Ordnung folgen.

Ein gültige **eMail** muss zumindest das Zeichen `'@'` aufweisen. Andere Überprüfungen müssen nicht erfolgen.

Stellen Sie beim **Erzeugen eines Termins** sicher, dass nur gültige Termine erstellt werden können. Ist kein Datum gegeben, soll der `1.1.1900` verwendet werden. Ist kein Beginnzeitpunkt gegeben, soll dieser zum Zeitpunkt `0:00` angelegt werden. Ist kein Endzeitpunkt gegeben, dann soll der Beginnzeitpunkt angenommen werden. Liegt der Endzeitpunkt vor dem Beginnzeitpunkt, soll der führe Zeitpunkt als Beginn verwendet werden.

Stellen Sie folgende Methoden zur Verfügung:

`getId()`	Gibt die Id des Termins zurück.
`getDate()`	Gibt das Datum des Termins zurück.
`getBegin()`	Gibt den Beginnzeitpunkt des Termins zurück.
`getEnd()`	Gibt den Endzeitpunkt des Termins zurück.
`getDescription()`	Gibt die Beschreibung des Termins zurück.
`duration()`	Gibt die Dauer des Termin in Minuten zurück.
`alterDescription(String description)`	Ändert die Beschreibung eines Termins.
`addParticipant(String eMail)`	Fügt eine Teilnehmer (deren `eMail`) zu dem Termin hinzu. Ist keine gültige eMail gegeben (dh. es muss genau ein `'@'` vorkommen) oder befindet sich der Teilnehmer bereits in der Liste, soll die Liste der Teilnehmer nicht verändert werden und die Methode `false` zurück liefern.
`removeParticipant(String eMail)`	Entfernt den Teilnehmer (dh. die `eMail`) aus der List der Teilnehmer eines Termins. Der Rückgabewert soll anzeigen, ob die eMail erfolgreich gelöscht werden konnte oder nicht.
`compareTo(Appointment a)`	Vergleicht den Termin mit dem Termin gegeben in a und liefert einen negativen Wert, falls der Termin zeitlich vor dem Termin a liegt, einen positiven Wert, falls der Termin nach dem Termin a

	liegt und `0` falls die beiden Termine am gleichen Tag zur gleichen Zeit beginnen.
`toString()`	Gibt eine textuelle Repräsentation des Termins zurück. Diese soll beispielsweise wie folgt aussehen: `2020/JUN/01 10:00-11:30 "Vorlesung ESOFT" (90 min): hugo@geeemail.com bertha@heissmail.com sepp@gmax.com`

Hinweis:

- Realisieren Sie die **einfach-verkettete Liste** von Teilnehmer **durch eigene Klassen** selbst (e.g. **`ParticipantList`**, **`ParticipantNode`**)
- Sie können die Methode `indexOf()` und `lastIndexOf()` der Klasse `String` verwenden um auf das (einmalige) Vorkommen des Zeichens `'@'` in einer eMail zu überprüfen.
- Die Klasse `TimeOfDay` stellt mit `minBetween(TimeOfDay t)` eine Methode zur Verfügung, um die Zeitspanne in Minuten zwischen zwei Zeitpunkten zu erhalten. Diese Methode gibt einen positiven Wert zurück gibt, falls der Zeitpunkt vor(!) dem Zeitpunkt `t` liegt, die einen negativen Wert zurück gibt, falls der Zeitpunkt nach(!) dem Zeitpunkt `t` liegt und `0` falls beide den gleichen Zeitpunkt repräsentieren.
- Die Klasse `Date` stellt eine Methode `compareTo(Date d)` zur Verfügung, die einen negativen Wert zurück gibt, falls das Datum vor dem Datum `d` liegt, die einen positiven Wert zurück gibt, falls das Datum nach dem Datum `d` liegt und `0` falls beide das gleiche Datum repräsentieren.

Aufgabe 1b: "Schedule" [12 Punkte]

Ein **Terminplaner** (Klasse **`Schedule`**) verwaltet eine **beliebige Anzahl von Terminen** (Klasse **`Appointment`**) in Form einer **einfach-verketteten Liste**. Termine sollen dabei nach ihrem Beginn (also Datum und Startzeitpunkt) ***sortiert gehalten werden***.

Jeder Terminplaner hat eine textuelle **Bezeichnung**. Ist beim Erzeugen keine textuelle Bezeichnung gegeben, soll der Standard-Text "`Calendar`" verwendet werden.

Die Klasse **`Schedule`** soll folgende Methoden sowie Konstruktoren zur Verfügung stellen:

`Schedule(String name)`	Initialisiert den Schedule mit dem Name. Verwendet ggf. den Standard-Text.
`noAppointments()`	Gibt die Anzahl der Termine in dem Schedule zurück.
`schedule(Date d, TimeOfDay b, TimeOfDay e, String desc)`	Fügt einen neuen Termin mit den entsprechenden Informationen in den Schedule ein; dabei soll die *Ordnungsreihenfolge* beachtet werden. Die Methode liefert die Id des eingefügten Termins retour. Ist kein Termin gegeben oder sind ungültige Informationen gegeben, soll ein negativer Wert zurück gegeben werden.
`remove(int id)`	Entfernt einen durch `id` identifizierten Termin aus dem Schedule. Die Methode gibt `false` zurück, falls der Termin nicht im Schedule gefunden wurde.
`addParticipant(int id, String eMail)`	Fügt einen Teilnehmer (dh. die eMail) zu dem durch `id` identifizierten Termin hinzu. Kann der Teilnehmer nicht hinzugefügt werden, soll `false` zurück gegeben werden.
`removeParticipant(int id, String eMail)`	Entfernt einen Teilnehmer von dem durch `id` identifizierten Termin. Kann der Teilnehmer nicht entfernt werden, soll `false` zurück gegeben werden.

`toString()`	Gibt eine textuelle Repräsentation des Schedules zurück geben. Diese soll den Namen des Schedules enthalten, sowie eine nach dem Beginn sortierte Liste der Termine.
	Beispielsweise:
	`"Lehre-Termine"`
	`----------`
	`2020/JUN/01 08:00-10:00 "Einfuehrung in die Philosophie" (120 min): sepp@gmax.com`
	`2020/JUN/01 10:00-11:30 "Vorlesung ESOFT" (90 min): hugo@geeemail.com bertha@heissmail.com sepp@gmax.com`
	`2020/JUN/03 14:00-15:00 "Tutorium ESOFT" (60 min): bertha@heissmail.com sepp@gmax.com`

Hinweis:

- Realisieren Sie die **einfach-verkettete Liste** von Teilnehmer **durch eigene Klassen** selbst (e.g. **`AppointmentList`**, **`AppointmentNode`**)
- Definieren Sie ggf. **eigene (private) Hilfs-Methoden**.

Im MOODLE finden Sie `ScheduleDemo.java`, das zeigt, wie Ihre Klassen verwendet werden können sollen.

Darüber können Sie die Klassen `Date` und `TimeOfDay` zur Verwendung im MOODLE herunterladen und diese in Ihrer Realisierung direkt übernehmen.

Abgabe:

➔ **`Schedule.java` mit der Klasse `Schedule`.**

➔ **`Appointment.java` mit der Klasse `Appointment`.**

➔ **`Weitere Klassen` Ihrer Realisierung (e.g. `ParticipantList`, `ParticipantNode`, `AppointmentList`, `AppointmentNode`)**