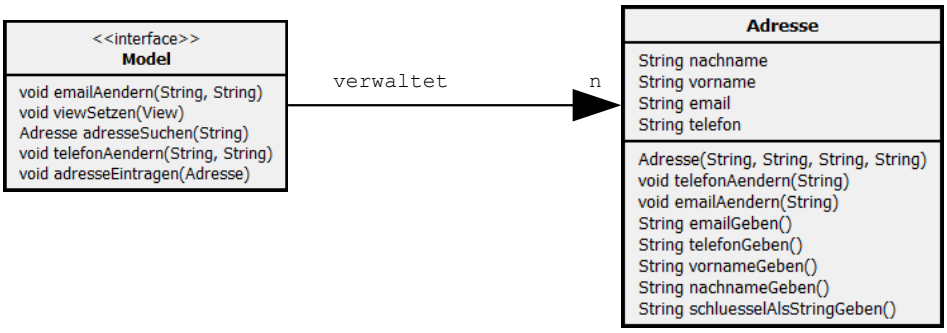
Das Ziel: Ein Programm zur Adressverwaltung

Sie sollen gemeinsam ein Programm entwickeln, mit dem man Adressen verwalten kann. Der Einfachheit halber gehört zu einer Adresse nur: Name (verpflichtend), Vorname (verpflichtend), eine E-Mail-Adresse (optional) und eine Telefonnummer (optional).
Der Benutzer soll eine neue Adresse hinzufügen können, und bei einem gegebenen Eintrag nachträglich die E-Mail-Adresse und die Telefonnummer ändern können. Ein Löschen ist nicht vorgesehen. Doppelte Einträge darf es nicht geben, als eidneutiger Schlüssel dient die Kombination aus Vor- und Nachnamen.

Der Entwurf

Eine Adressverwaltung verwaltet n Adressen. Das Interface dazu heißt aus Gründen, die später klar werden, „Model“:



Im Model ist bei „emailAendern“ und „telefonAendern“ jeweils das erste Argument der Suchschlüssel (z.B. „Rau, Thomas“) und das zweite Argument der neue Wert von Eemail bzw. telefon.

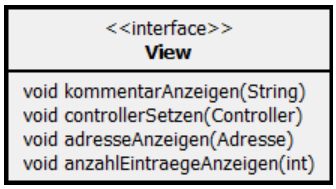
Implementierung

Die 1:n-Beziehung soll mit einem geordneten Binärbaum umgesetzt werden, da der beim Suchen bei vielen Elementen schneller ist als eine Liste. Als Schlüssel dienen dabei Name und Vorname in der Form „Rau, Thomas“, also Nachname + Komma + Leerzeichen + Vorname.
Zu diesem Zweck muss die Adresse die Schnittstelle „Datenelement“ benutzen, und Ihre Adressbuch-Klasse (die das Model-Interface benutzen muss) verwendet einen Binbaum zur Verwaltung der Datenelemente.

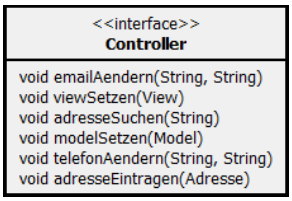
Verwendung des Model-View-Controller-Musters

Viele größere Projekte arbeiten nach dem arbeitsteiligen MVC-Muster. Dabei werden die Aufgabenbereiche auf drei möglichst unabhängig arbeitende Gruppen verteilt.

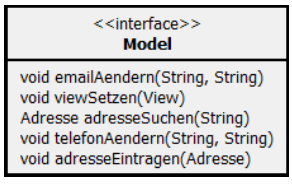
Die/der **View:** Das ist die grafische Benutzeroberfläche. Sie besteht oft aus Knöpfen (die man drücken kann), aus Eingabefeldern (in die man etwas schreiben kann und die man auslesen kann), und aus Beschriftungen (die beliebigen Text anzeigen können).
Der View stellt anderen Objekten Methoden zur Verfügung, um Informationen anzuzeigen. Im Projekt sind diese Methoden im Interface „View“ festgehalten:



Wenn die entsprechende Methode des View aufgerufen wird, reagiert der sinngemäß und zeigt zum Beispiel die Adresse an, die als Argument mitgeschickt wurde. Außerdem sendet die View Informationen an den zweiten Bestandteil, den Controller.



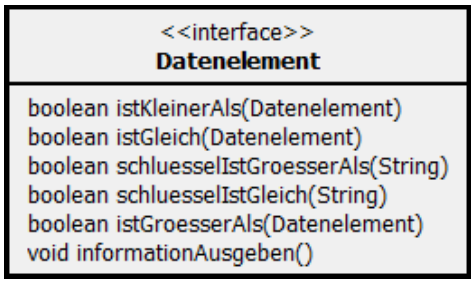
Der **Controller:** Er empfängt Informationen vom View – zum Beispiel welcher Knopf gedrückt worden ist und was gerade in den Textfeldern steht. Der Controller gibt diese Informationen bzw. den Auftrag an das Model weiter. (Vorher könnte er aber noch überprüfen, ob die Information auch sinnvoll ist, so dass z.B. doppelte Einträge oder inkorrekte E-Mail-Adressen gar nicht erst an das Modell übermittelt werden. Das ist für den Anfang wohl zu schwer und wird erst in einem zweiten Schritt umgesetzt.)

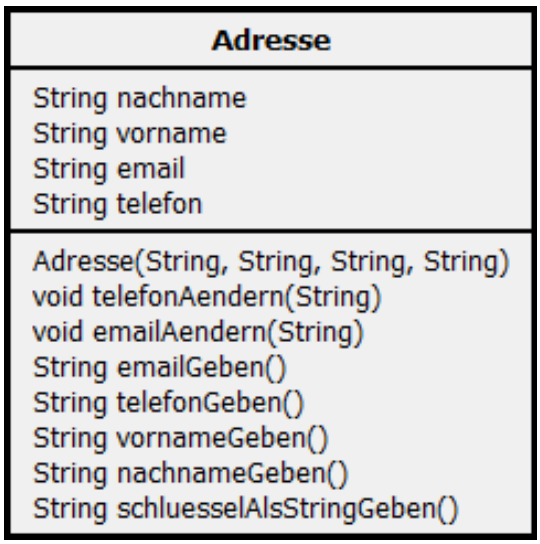


Das **Model:** Ist im Prinzip das, was Sie bisher kennen, das Herz des Programms. Man kann Adressbucheinträge hinzufügen, ändern und suchen. Hirn hat das Model keines – ob ein Eintrag doppelt ist oder eine Eingabe korrekt, das überprüft vorher der Controller.
Wenn sich etwas Wichtiges am aktuellen Zustand des Modells ändert, informiert es den View darüber, damit der das dem Benutzer anzeigen kann. Bei uns ist das nur die aktuelle Gesamtzahl an verwalteten Adressen. Wenn sich an dieser etwas ändert, wird der View informiert.

Team 1 (zwei oder drei Schülerinnen und Schüler)

Vervollständigen Sie die Klasse „Adresse“, so dass sie den Bedingungen im Klassendiagramm entspricht. Die Klasse erbt dazu von unserer alten Klasse „Datenelement“, die jemand – am besten Ihre Gruppe – in das Projekt kopieren muss. Ändern Sie nichts an der Datenelement-Klasse selber.





Ergänzen Sie außerdem die notwendigen Attribute und Methoden von „Adresse“ nach dem Klassendiagramm.

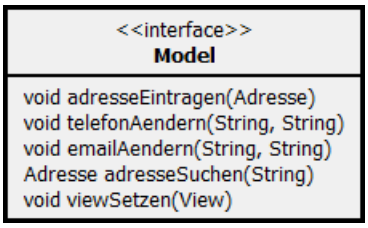
Der Suchschlüssel soll sein:

nachname+Komma+Leerzeichen+vorname,

also zum Beispiel „Rau, Thomas“

Team 2 (drei oder vier Schülerinnen und Schüler)

Kopieren Sie unseren Binbaum in das Projekt, mit allen beteiligten Klassen (Binbaum, Baumelement, Knoten, Abschluss, und Datenelement, falls das letztere nicht bereits durch ein anderes Team ergänzt worden ist.) Legen Sie eine Unterklasse zu „ModellUmsetzung“ an, die die Methoden, die diese Unterklasse noch erfüllen muss, implementiert. Das sind 4 der 5 Methoden, die im Interface „Model“ verlangt werden (die fünfte ist bereits in der abstrakten Oberklasse gegeben):



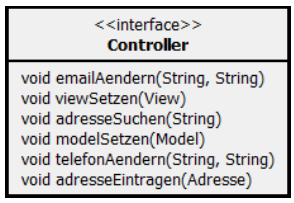
Ihre Unterklasse kann nicht von „Binbaum“ erben, da sie ja bereits von einer anderen Klasse erbt; sie muss also ein Objekt der Klasse „Binbaum“ benutzen, um die ihr übertragenen Aufgaben erfüllen zu können – so ähnlich, wie das bei all unseren Projekten bisher war.

Ändern Sie die Klassen Binbaum, Baumelement, Knoten und Abschluss dahingehend, dass Sie danach berechnen können, wie viele Element im Binbaum enthalten sind, falls das bei Ihrem Projekt nicht ohnehin schon gemacht ist.

Die Methode „emailAendern(String, String)“ funktioniert so: Das *erste* Argument ist der Suchschlüssel (z.B. „Rau, Thomas“). Dann sucht die Methode nach dem Datenelement, das dem Suchschlüssel entspricht, wandeln es in eine Adresse um, und ersetzt dann den Wert des Email-Attribut mit der Methode „emailAendern(String)“, wobei statt „String“ das *zweite* ursprüngliche Argument verwenden wird. Der Aufruf der Methode sieht dann so aus: emailAenden(„Rau, Thomas“, „neue@mail.de“).
Analog bei „telefonAendern“.

Optional:
Danach ändern Sie Ihren Code so, dass Ihre Klasse immer dann, wenn in ihr die Methode „adresseHinzufuegen“ ausgeführt wird, die ererbte Methode „anzahlEintraegeSenden(int i)“ aufgerufen wird, wobei Sie statt „int i“ die aktuelle Anzahl der verwalteten Datenelemente setzen.

Team 3 (drei oder vier Schülerinnen und Schüler – oder gleich Lehrer)



Legen Sie eine Unterklasse zur Klasse „ControllerUmsetzung“ an. Da diese Klasse das Interface „Controller“ implementiert, müssen Sie die fehlenden Methoden ergänzen. Es sind 4 Methoden, da „modelSetzen“ und „viewSetzen“ bereits durch die abstrakte Oberklasse gegeben sind.

adresseEintragen: Wie soll Ihre Klasse reagieren, wenn diese Methode (mit einer Adresse als Argument) aufgerufen wird? Sie soll die logisch passende Methode des Models aufrufen (das als Attribut in der Oberklasse „ControllerUmsetzung“ deklariert ist).
Danach ruft Sie die Methode „kommentarAnzeigen(String)“ des Views auf (der als Attribut in der Oberklasse „ControllerUmsetzung“ deklariert ist), wobei statt des „String“ eine kurze Textnachricht steht, z.B. „Eingetragen“, die der View danach anzeigen wird. (Wie das geschieht, ist Aufgabe des View-Teams.)

*emailAendernEmpfangen:*Wie soll Ihre Klasse reagieren, wenn diese Methode (mit zwei Strings als Argument) aufgerufen wird? So ähnlich wie oben: Den Auftrag an das Model weiterleiten und den View darüber informieren.

telefonAendernEmpfangen: wie oben.

adresseSuchenEmpfangen: Sie geben den Auftrag an das Model weiter, von dem Sie als Antwort die gewünschte Adresse erhalten (oder *null*). Rufen Sie dann die Methode *adresseAnzeigen* des View auf, mit der erhaltenen Antwort als Argument. Der Suchschlüssel ist name+Komma+Leerzeichen+vorname.

Optional: Lassen Sie den Controller vor dem Einfuegen eine Adresse oder dem Ändern von Email oder Telefon kontrollieren, ob die Email-Adresse und/oder Telefon ein gültiges Format haben. Andernfalls geben Sie den Auftrag nicht an das Model weiter, sondern schicken dem View eine Fehlermeldung über die Funktion *kommentarAnzeigen.*
Optional: Lassen Sie den Controller vor dem Einfuegen einer Adresse überprüfen, ob der Eintrag bereits vorhanden ist. Dazu müssen Sie aus der übergebenen Adresse den Suchschlüssel erzeugen, das Model danach suchen lassen, und nur wenn die Antwort *null* ist, führen Sie den Auftrag aus. (Sonst: Fehlermeldung an View.)

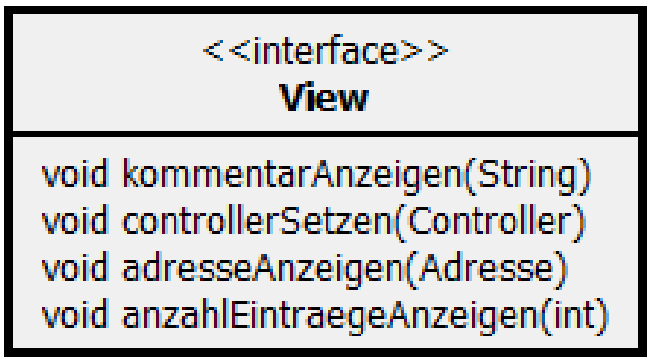
Team 4 (ein oder zwei Schüler oder Schülerinnen)

Zusammenbau und Test. Legen Sie eine Klasse „Main“ an. Diese Klasse soll drei Attribute haben, jeweils vom Typ View, Controller und Model. Außerdem soll es eine Methode „start()“ geben, in der

- die Variable vom Typ „View“ initialisiert wurd mit einem Objekt der von Ihren Mitschülern erzeugten Unterklasse dazu, also etwa *View view = new Mitschuelerklasse();*
- analog mit dem Controller-Attribut
- und dem Model-Attribut.
- Sorgen Sie mit den Methoden modelSetzen, viewSetzen, controllerSetzen dafür, dass die notwendigen Referenzattribute all dieser Objekte nicht *null* sind

Team 5 und Team 6 (jeweils drei oder vier Schülerinnen und Schüler)

Entwickeln Sie ein GUI für das Projekt. Anleitung dazu siehe separates Material. Ihr GUI muss das Interface View implementieren, also folgende Methoden bereitstsllen



Wenn die Methode adresseAnzeigen(Adresse a) aufgerufen wird, holt sich das GUI aus dem Argument die Information. (Ihre Mitschüler haben diese Methoden inzwischen hoffentlich implementiert.)
Dann soll Ihre GUI die Information auch anzeigen. Das geht alles etwa so:

```
eingabefeldName.setText(a.nachnameGeben());
eingabefeldVorname.setText(a.vornameGeben());
eingabefeldTelefon.setText(a.telefonGeben());
eingabefeldEmail.setText(a.emailGeben());
```

Entsprechend gehen Sie bei den beiden anderen Methoden vor.
Außerdem braucht Ihre Klasse ein Refernzattribut vom Typ Controller und eine passende controllerSetzen-Methode dazu.