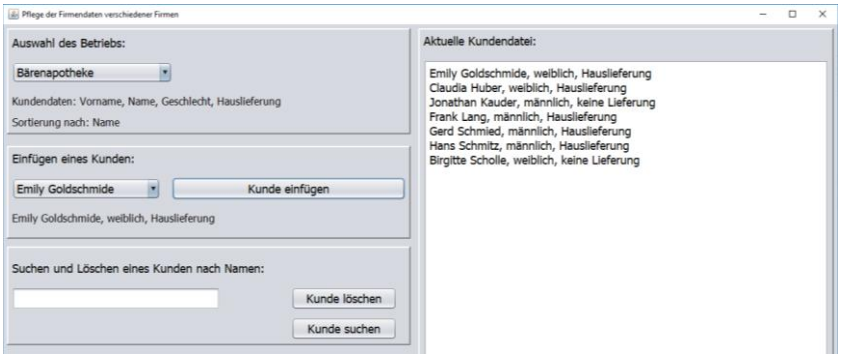
Projekt 13 Firmendaten – Arbeiten mit ComparableContent

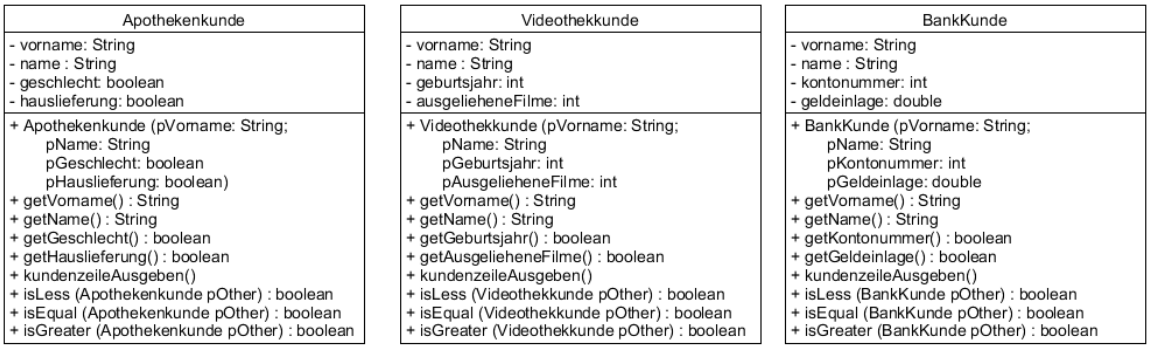


Eine Informatikfirma hat mehrer Kunden, für die sie eine App zur Verwaltung der Kunden erstellen soll. Jeder Kunde möchte andere Merkmale ihrer Kunden speichern und die Reihenfolge soll auch nach verschiedenen Merkmalen sortiert werden. In unserem Beispiel hat die Compterfirma drei Kunden:

Kunde	**Kundenmerkmale, die gespeichert werden sollen**	**Kundenmerkmal, nachdem die Kunden sortiert werden sollen**
Bärenapotheke	Vorname (String) Name (string) Geschlecht (boolean, true entspricht weiblich) Hauslieferung (boolean, true wenn Kunden Ware nach Hause geliefert bekommen)	Name
Videothek Sunview	Vorname (String) Name (string) Geburtsjahr (int) Zahl ausgeliehener Filme (int)	Zahl ausgeliehener Filme
Citybank	Vorname (String) Name (string) Kontonummer (int) Geldeinlage (double)	Geldeinlage

Es soll eine Listenverwaltung implementiert werden, die die Daten der Firmen verwaltet: Kunden müssen eingefügt und gelöscht werden können, außerdem soll nach einem Kunden gesucht werden können. Um den Aufwand zu minimieren, muss eine einzige Listenverwaltung jedoch mit allen drei Kundenlisten funktionieren. Dazu benutzt man ein sogenanntes **Interface** mit dem Namen **ComparabelContent** (zu Deutsch vergleichbare Information). Dieses Interface macht nur eine Vorgabe, dass welcher Datentyp in der Liste später auch immer benutzt wird, er die Möglichkeit haben muss, dass man zwei der Listenelemente miteinander vergleichen können muss. Im Interface werden dabei keine vollständigen Methoden, sondern nur die Methodendeklarationen angegeben:

Methode aus ComparableContent	**Beispiel eines Methodenaufrufs aus der Listenverwaltung**
public boolean isGreater(ContentType pContent); prüft, ob das Listenelement elementEins größer ist als das Listenelement elementZwei ist, verglichen werden die Merkmale, nach denen sortiert werden soll	if (elementEins.isGreater(elementZwei)){ …}
public boolean isEqual(ContentType pContent); prüft, ob das Listenelement elementEins gleich dem Listenelement elementZwei ist, verglichen werden die Merkmale, nach denen sortiert werden soll	if (aktuelleselement.isEqual(dieListe.getContent())){ …}
public boolean isLess(ContentType pContent); prüft, ob das Listenelement elementEins kleiner als das Listenelement elementZwei ist, verglichen werden die Merkmale, nach denen sortiert werden soll	if (aktuelleselement.isLess(dieListe.getContent())){ …}



Für jeden Kunden gibt es nun eine Klasse, die – neben den üblichen get-Methoden - die drei geforderten Methoden des Interfaces implementiert: Die Methode **isLess** für den VideothekKunden sieht z. B. folgendermaßen aus, es wird nach dem Merkmal ausgeliehene Filme verglichen:

```
public boolean isLess(VideothekKunde pOther) {
    if ausgelieheneFilme < pOther.ausgelieheneFilme){
      return true;
    }
    else return false;
}
```

Wenn man später bei der Methode kundenEinfügen zwei Kunden kunde1 und kunde2 vom Typ Content vergleichen will, kann man folgen Befehl benutzen:

```
if (kunde1.isLess (kunde2)) {
  area1.setText („Kunde 1 steht vor Kunde 2“);
}
```

Wenn man die Methode isLess für alle drei Firmen implementiert hat, wird der obige Befehl in der Listenverwaltung auch für alle drei Kunden funktionieren, er muss nur einmal implementiert werden.

Aufgabe 1: Kopiere nun das Projekt „Firmendaten“ in deinen persönlichen Ordner. Öffne die Klasse **VideothekKunde** und implementiere die drei Methoden **isLess**, **isEqual** und **isGreater**, alle anderen Methoden sind bereits fertig. Wiederhole den Vorgang für die Klassen BankKunde und ApothekenKunde. Da die Klasse ApothekenKunde mit dem Attribut Name vergleicht, müssen hier zum Vergleich die String-Methoden **compareTo** und **isEqual** benutzt werden.

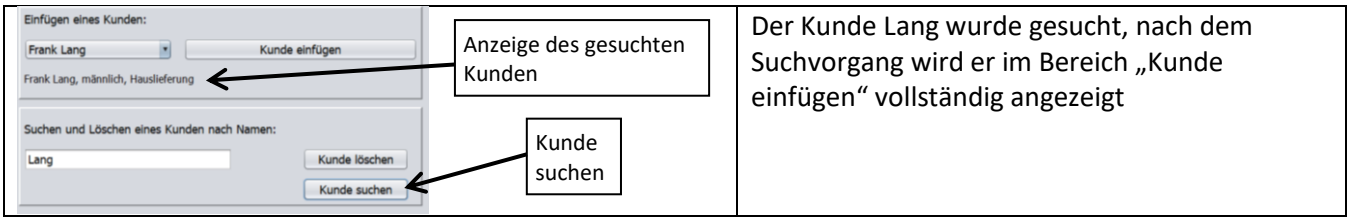
Aufgabe 2: Es soll in der Klasse **OrderedList** eine Listenverwaltung für den beliebigen Datentyp ContentType geschrieben werden, die Kunden sortiert nachdem jeweiligen Sortier-Merkmal einfügen und löschen kann. Außerdem soll man nach einem bestimmten Kunden suchen können. Rufe dazu die Klasse OrderedList auf, wir beginnen mit der Implementation der Methode insert (ContentType pContent), die das neue Kunden-Element pContent in die Liste einfügen soll:

```
Gehe an den Anfang der Liste dieliste
Während (die Liste hat ein aktuelles Element und
          das aktuelleElement ist kleiner als das einzufügende
Element pContent) dann
  gehe zum nächsten Element
Ende Während
Wenn (die Liste hat ein aktuelles Element)
     füge pContent an der aktuellen Stelle ein
Sonst
    hänge pContent an die Liste an
Ende Wenn
```

Die Klasse **OrderedList** greift auf die Klasse **List** zurück (s. im oberen Teil der Klasse **OrderedList**: dieListe ist als **List<ContentType>** deklariert), d. h. die bekannten Methoden **getContent()**, **next()**, **toFirst()**, **hasAccess()** dürfen benutzt werden.

Wenn die Methode **insert (ContentType pContent)** fertig implementiert ist, kannst du das Programm zum ersten Mal testen: Wenn du einen Kunden aus der **Combobox kundenCombo** auswählst und anschließend auf „Kunden einfügen“ klickst, muss der Kunde in die Liste eingefügt werden und im rechten Fenster erscheinen. Alle Methoden zur Ausgabe der Kunden sind bereits fertig und in der Klasse Gui enthalten. Die Klasse **Verwaltung** steurert unmittelbar die Methoden aus deiner Klasse **OrderedList** an. Wenn du weitere Kunden einfügst, erscheinen diese ebenfalls im Ausgabefenster – sortiert nach ihrem Namen, denn zum Programmstart ist die Kundendatei der Bärenapotheke eingestellt. Wenn du in der **Combobox FirmenCombo** auf die Videothek wechselst, kannst du Kunden der Sunrise Videothek einfügen, diese werden jetzt nach der Zahl der ausgeliehenen Filme sortiert, außerdem siehst du, wie die geforderten Kundenmerkmale angezeigt werden. Probiere dasselbe auch aus für die Kunden der Citybank.

Aufgabe 3: Bevor wir zum Löschen von Kunden kommen, implementieren wir die Methode **public ContentType search(ContentType pContent)** in der Klasse **OrderedList**. Beginnen wir mit der Kundendatei Bärenapotheke: Wenn der Benutzer im **JTextField namenEingabeBox** einen Namen eingibt, soll die Methode einen Kunden mit dem entsprechenden Namen suchen. Gibt es diesen Kunden, soll die Methode den entsprechenden Kunden als **ContentType** zurückgeben, sodass er mit vollständigen Daten (also auch Vorname, Geschlecht, Auslieferung) ausgegeben werden kann. Gibt es diesen Kunden nicht, liefert die Methode als Rückgabewert **null**. Du kannst die Methode direkt testen: Die Kundenausgabe ist bereits fertig implemenmtiert: wurde ein Kunde durch deine Methode gefunden, wird dieser im bereich „Kunde einfügen“ vollständig angezeigt. Gibt man einen falschen Kundennamen ein, erfolgt ein entsprechender Hinweis im **nichtgefundenLab**. Man kann natütlich nur einen Kunden finden, wenn dieser vorher in die Kundenliste eingefügt wurde. Teste deine Methode für alle drei Kunden aus.



Aufgabe 4: Wenn die Methode search implementiert ist, ist die Programmierung der Methode remove(ContentType pContent) sehr einfach, benutze zum Löschen deine fertige Methode search, außerdem können die Methoden der Liste verwendet werden. Teste auch diese Methode für die verschienden Kunden aus.

Zusatzaufgabe 1: In einem Label soll die Kundenzahl sowohl der erste bzw. letzte Kunde der Liste mit einem sinnvollen Kommentar angezeigt werden.

Zusatzaufgabe 2: Schau dir an, mit welchen Schlüsselwörtern das Interface mit den Klassen Apothekenkunde, VideothekKunde und BnakKunde miteinander verknüpft wird. Woher weiß die Klasse OrderedList und die Klasse Verwaltung, dass die mit dem Interface zusammenarbeiten muss? Bereite einen kleinen Vortrag dazu vor.