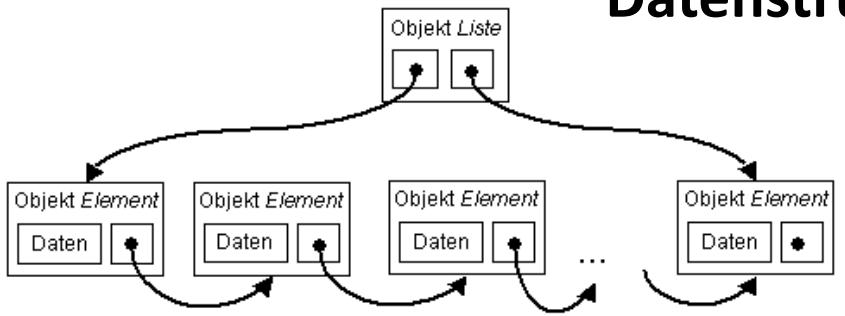
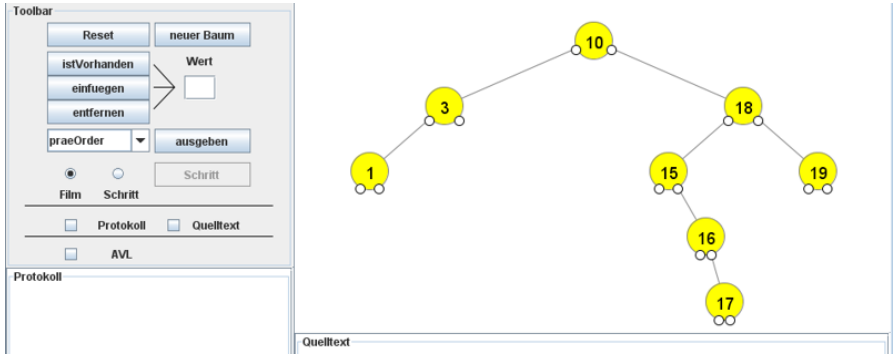
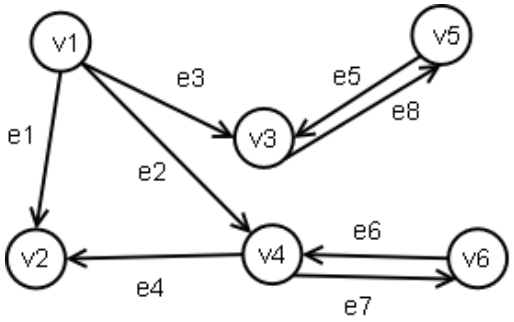
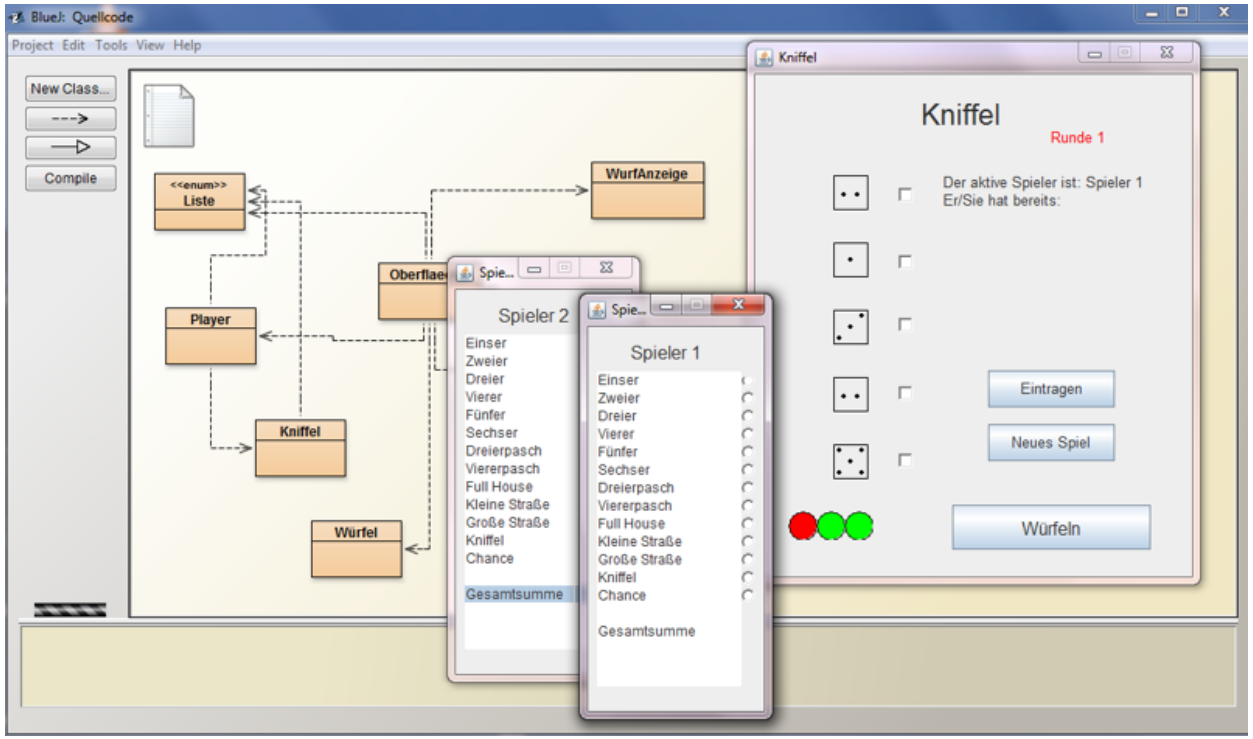
Informatik 11

Datenstrukturen und Softwareentwicklung







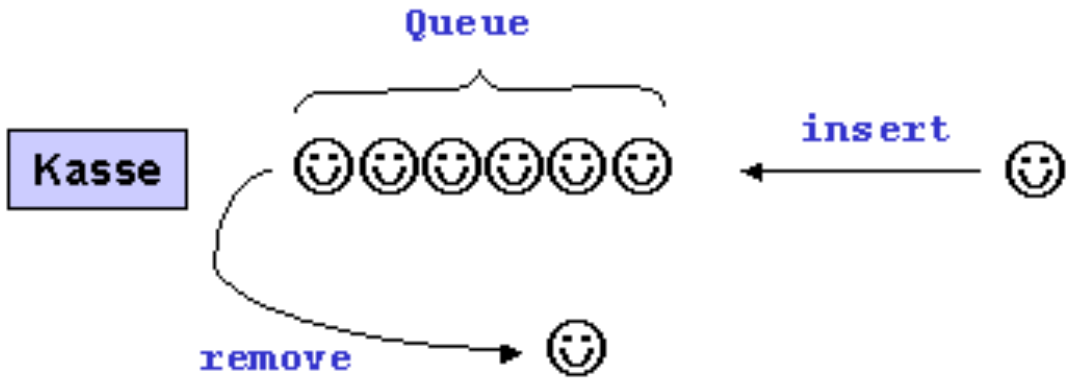


1. Die rekursive Datenstruktur Liste

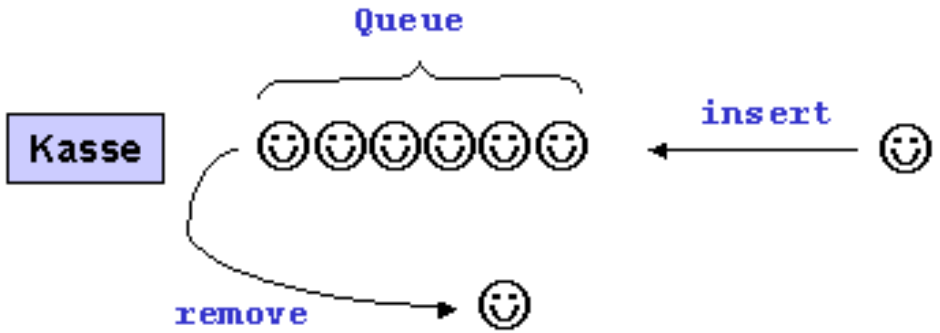
1.1 Die Datenstruktur Warteschlange

In einer Warteschlange (französisch: Queue) an einer Kasse im Supermarkt geschieht das Verlassen und Einreihen nach dem Prinzip **“First In, First Out“ (FIFO):**

Wer sich zuerst in die Schlange einreiht, verlässt sie auch als erster wieder.



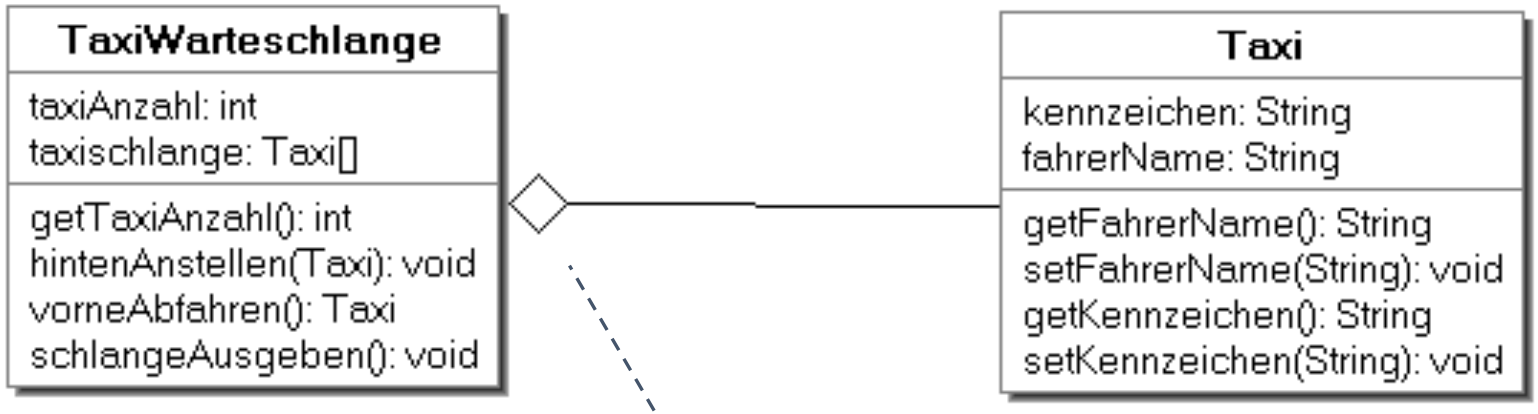
Andere Beispiele, bei denen das Prinzip FIFO möglich (aber nicht zwingend) ist:



- Autos an einer Zapfsäule
- Taxis an einem Taxistand
- Patienten im Wartezimmer
- Prozesse in einem Computerbetriebssystem

Die Situation „Warteschlange mit Taxis“ soll mithilfe eines **Feldes (Array)** modelliert und implementiert werden.

Erweitertes Klassendiagramm:

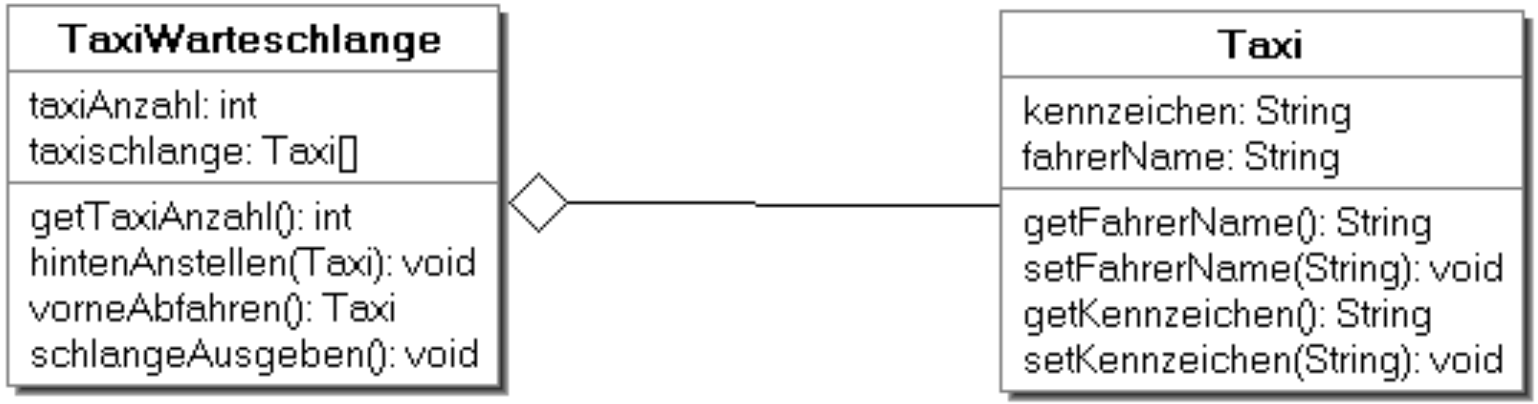


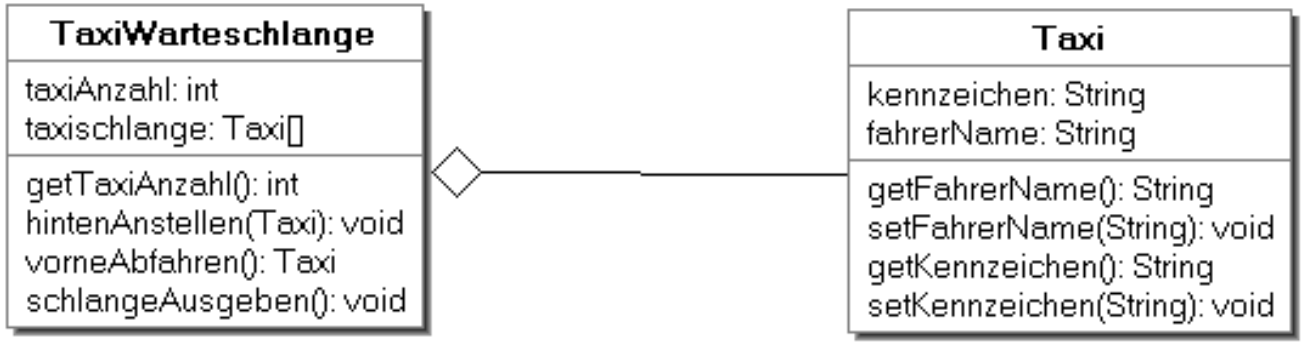
Hinweis:
Im Klassendiagramm kann ähnlich wie in JAVA der Datentyp,die Referenzklasse und der Rückgabebezeichner zuerst stehen
int taxiAnzahl
Taxi[] taxischlange
void hintenAnstellen(Taxi)
...

Aggregation,
d.h. ein Objekt von TaxiWarteschlange referenziert mehrere Objekte von Taxi.

Übung 1 Implementieren einer Warteschlange als Array

Implementiere nach dem Klassendiagramm die beiden Klassen *Taxi* und *Taxischlange*.
Alle Attribute haben den Sichtbarkeitsmodifikator *private*.
Im Konstruktor von *TaxiWarteschlange* wird ein leeres Feld mit 6 Plätzen erzeugt, für *Taxi* genügt der Standardkonstruktor.
Die Methode *schlangeAusgeben()* listet in einer Zeile die Taxikennzeichen auf.
Erzeuge einige Objekte und teste dein Programm ausführlich.

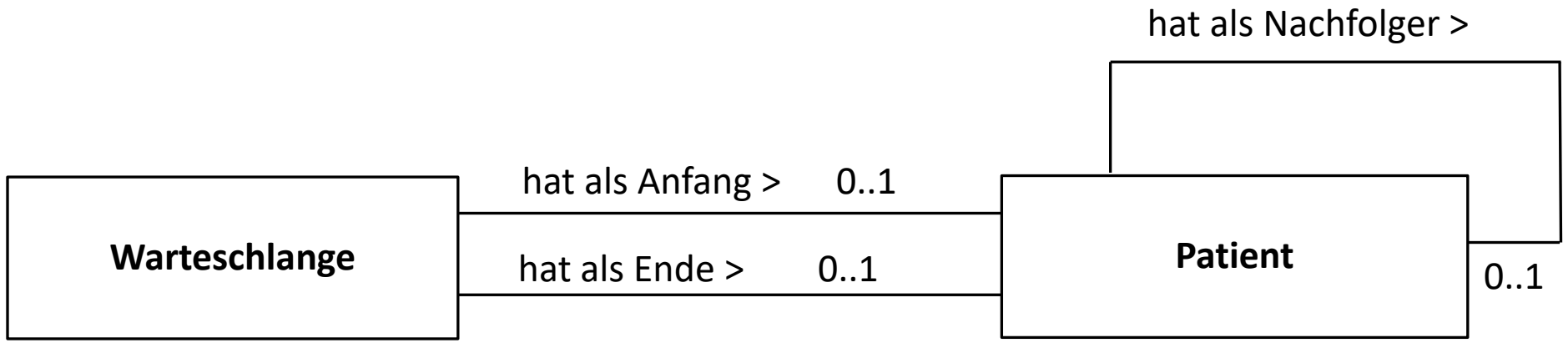




Probleme bei der Modellierung als Feld (Array):

- Die Anzahl der Feldlänge ist nicht variabel.
- Nicht benötigte Feldplätze beanspruchen Speicherplatz.
- Die Umsortierung (z.B. bei remove) der Referenzen auf die Feldplätze kann unnötig komplex werden.
- Modellierung nicht ganz korrekt, da es Referenzen auf alle Elemente gibt. (eigentlich benötigt man nur den Anfang und evt. das Ende der Schlange)

Modellierung der Situation „Patienten im Wartezimmer“ als **Warteschlange:**

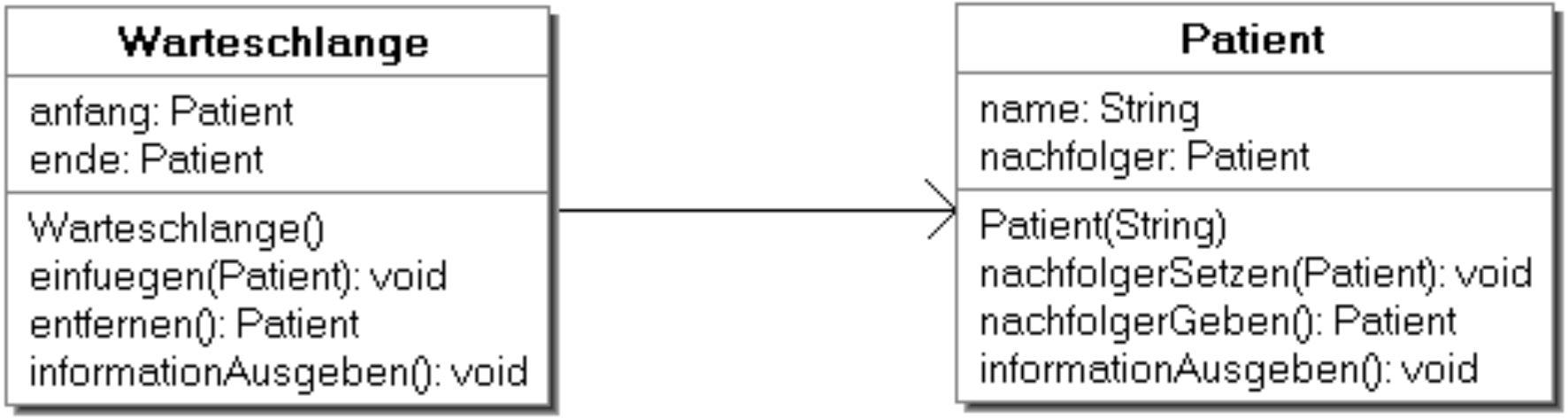


„hat als Nachfolger“ ist eine **rekursive Beziehung**.

Übung 2 Implementieren einer Warteschlange

a)

Implementiere nach dem Klassendiagramm die beiden Klassen *Warteschlange* und *Patient*.
Die Methode *informationAusgeben()* von Warteschlange soll die Methode *informationAusgeben()* des ersten Patienten in der Schlange aufrufen.
Erzeuge einige Objekte und teste dein Programm ausführlich.
Ergänze die Klasse *Warteschlange* um eine Methode *alleInformationenAusgeben()*, die die Namen aller Patienten in der Schlange in ihrer Reihenfolge ausgibt.



Übung 2 Implementieren einer Warteschlange – Beschreibung der Klassen

Klasse Patient
String name (eindeutiger) Nachname des Patienten **Patient nachfolger** Referenz auf den Nachfolger in der Warteschlange
Patient(String name) Konstruktor mit dem Übergabeparameter name. Der Nachfolger ist noch unbestimmt und wird auf null gesetzt. **void nachfolgerSetzen(Patient nf)** Setzt den Nachfolger des Patienten in der Warteschlange fest. **Patient nachfolgerGeben()** Sondierende Methode, die den Nachfolger des Patienten zurückgibt. **void informationAusgeben()** Schreibt den Namen des Patienten in ein Textfenster.

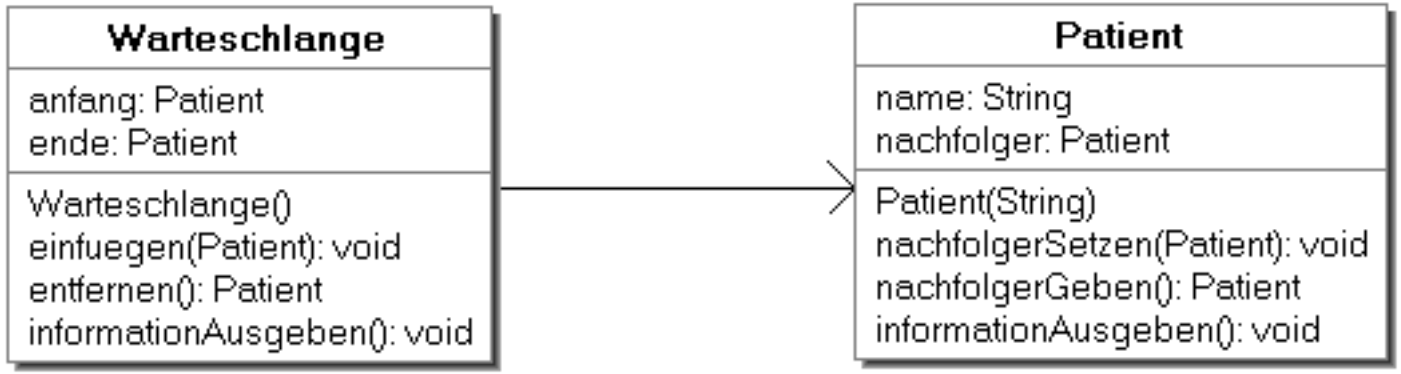
Übung 2 Implementieren einer Warteschlange – Beschreibung der Klassen

Klasse Warteschlange
Patient anfang, ende Referenzen auf den Anfang und auf das Ende der Warteschlange. Die Referenz auf das Ende ist dabei nicht zwingend notwendig. (Übung 2c)
Warteschlange() Konstruktor; anfang und ende werden auf null gesetzt. **void einfuegen(Patient patientNeu)** Fügt die Referenz auf einen neuen Patienten am Ende der Wartschlange ein. Ist die Warteschlange nicht leer, erhält ende patientNeu als neuen Nachfolger. Ist die Warteschlange leer, werden anfang und ende auf patientNeu gesetzt. **Patient entfernen()** Entfernt den ersten Patienten in der Warteschlange und gibt ihn zurück. Ist die Warteschlange nicht leer, so wird anfang neu gesetzt und zwar auf seinen Nachfolger. Für den Fall, dass nur ein Patient vor dem Entfernen in der Schlange war, muss ende auf null gesetzt werden. **void informationAusgeben()** Ruft die Methode informationAusgeben() des ersten Patienten in der Schlange auf. **void alleInformationenAusgeben()** Ruft der Reihe nach jeweils die Methode informationAusgeben() des Patienten in der Schlange auf.

Übung 2 Testen der Implementierung

b)
Erzeuge einige Objekte von Patient, ein Objekt von Warteschlange und teste deine Methoden.
Überprüfe auch die Sonderfälle leere Schlange und Schlange mit einem Patienten.

Der folgende Sonderfall kann zu einem Fehler in der Schlange führen:
Der aus der Schlange entfernte Patient reiht sich gleich wieder ein. Was passiert dann beim Aufruf der Methode alleInformationenAusgeben() ? Wie lässt sich dieser Fehler leicht beheben?



Übung 2 Ändern der Modellierung, Implementierung

c)

Das Verwalten des Endes der Warteschlange ist nicht zwingend notwendig.

Das Klassendiagramm zeigt die geänderte Modellierung.

Die Methode endeGeben() durchläuft die Schlange solange bis der Nachfolger des aktuellen Elements null ist.

Passe die Implementierung an die neue Modellierung an und teste ausführlich.