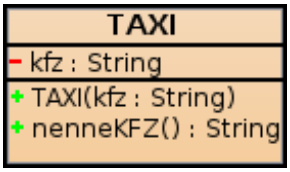
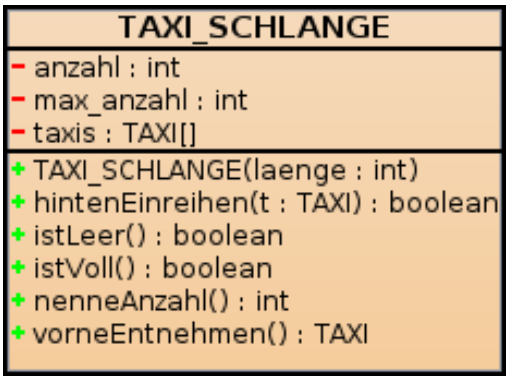
Datenstruktur S C H L A N G E

Eine ***Schlange*** ist in der Informatik eine Datenstruktur, bei der man an einem Ende etwas einreihen und am anderen Ende wieder entnehmen kann. Die Elemente innerhalb der Schlange werden in genau der Reihenfolge entnommen, wie man eingefügt hat. Dieses Prinzip nennt man in der Informatik ***FIFO*** (**F**irst **I**n **F**irst **O**ut). Die Kapazität der Schlange kann begrenzt sein, muss aber nicht.

Realisierung durch ein Array

Unser erster Versuch wird die Schlange mit Hilfe eines Arrays abbilden. Hier haben wir es also mit einer Schlange von begrenzter Kapazität zu tun. Hierzu erstellen wir zuerst eine Klasse TAXI und danach eine Klasse TAXI_SCHLANGE nach folgenden Vorlagen:





Aufgabe 1:

*Lege ein **neues, leeres BlueJ-Projekt** **SchlangeArray** an und erstelle die beiden Klassen TAXI und TAXI_SCHLANGE. **Benenne alle Attribute und Methoden genau wie in den Klassenkarten**. Du wirst später eine TESTER-Klasse bekommen, die*

automatisch alle Funktionen deiner Schlange überprüfen kann. Dies funktioniert aber nur, wenn die darin aufgerufenen Methoden auch bei dir genau so heißen.

Tipp: *Schreibe als erstes in jeder Klasse für jede Methode die entsprechende Signatur (nur Kopf der Methode mit noch leeren geschweiften Klammern).*
Im Falle von Rückgaben returnierst du 0, null, false, ...

Softwareentwicklung – Teil 1

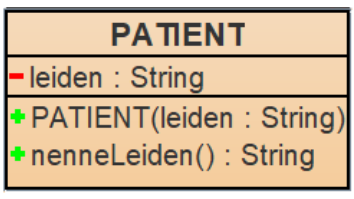
Du sollst im Rahmen des Informatikunterrichts der 11. Jahrgangsstufe eines über das Schuljahr verteilten **Softwareentwicklungspraktikums** sollst du nach und nach verschiedene Aspekte kennenlernen und guten von schlechtem Code unterscheiden lernen. Das vor der Implementierungsphase ***entworfene Modell*** und dessen Struktur ist dabei von größter Bedeutung.

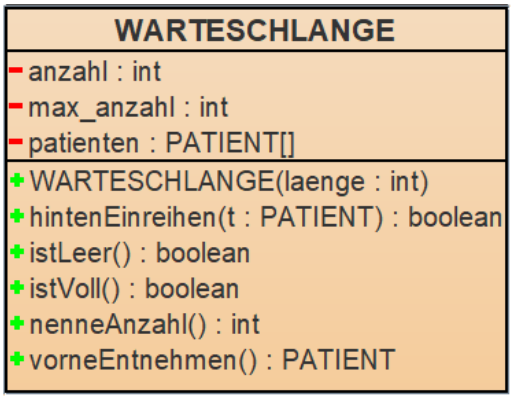
Ein Aspekt hierbei ist die ***Wiederverwendbarkeit des Codes***. Schafft man es, so zu modellieren und später zu implementieren, dass man bei einem ähnlichen Problem möglichst viel des vorherigen Codes unverändert wiederverwenden kann, so kann die Entwicklung weiterer (ähnlicher) Software-Produkte erheblich schneller ablaufen.

Auf diese Art ***modular aufgebauter Code*** hat auch den Vorteil, dass das gesamte Softwareprodukt **kompatibel** ist und man im Falle einer späteren Änderung des Modells ***einzelne Klassen einfach austauschen kann.*** nicht jedes bisher erstellte Softwareprodukt daraufhin untersuchen muss, ob es von dieser Änderung betroffen ist. Man muss auch nicht an unzähligen Stellen diese Änderung einpflegen. Ebenso wenig läuft man Gefahr, die Änderung doch an einer Stelle zu vergessen. Man ändert einfach einmalig etwas in einer Klasse und diese Änderung kann dann von allen bisher produzierten Softwareprodukten genutzt werden.

Aufgabe 2:

Du sollst herausfinden, wie geeignet unser bisheriges Konzept einer Warteschlange bzgl. einer Wiederverwendung in einem ähnlichen Sachzusammenhang ist. ***Kopiere dazu dein BlueJ-Projekt SchlangeArray*** *und erstelle die Klasse Patient. Ändere die Klasse TAXI_SCHLANGE so ab, dass sie der WARTESCHLANGE entspricht:*





Erfahrungen:

Was hast du einfach unverändert übernehmen können?

Wo waren kleinere Änderungen nötig?

Was musstest du quasi vollständig neu implementieren?

Würde ein Arzt einen Patienten abweisen, nur weil er im Wartezimmer keinen Stuhl mehr frei hat?

Verbesserungsmöglichkeiten:

Welche Nachteile hast du bisher an unserem Modell entdeckt?

Welche Ansprüche stellst du an eine mögliche Verbesserung?

Softwareentwicklung – Teil 2

Du hast gesehen, dass dein Code insbesondere deshalb nicht vernünftig wieder verwendet werden konnte, weil die Struktur der SCHLANGE zu fest an die konkrete Klasse TAXI gebunden war.

Insgesamt ist es sehr schlecht, mehrere verschiedene Anliegen in einer Klasse umzusetzen. Es gibt die ***Prämisse***: **Jede Klasse hat ihre eigene Aufgabe**. So soll die ***Trennung zwischen Struktur*** (SCHLANGE) ***und Inhalt*** (TAXI, PATIENT, ...) erlangt werden, die wieder einen modularen Aufbau und die Wiederverwendbarkeit gewährleistet (siehe oben).

So sollte sich die SCHLANGE nur um das FIFO-Prinzip kümmern, also um das hinten Einreihen, das vorne Entnehmen und das Aufrücken. Was in der Datenstruktur SCHLANGE gepuffert wird, muss für die Datenstruktur SCHLANGE unerheblich sein.

Aus Sicht eines TAXIs ist es allerdings unabdingbar, über eine Fähigkeit zu verfügen, sich in einer SCHLANGE zurecht zu finden, denn kein Taxi wird um eine Einreihung in eine SCHLANGE herum kommen.

Da aber auch jeder Patient oder jeder Druckauftrag in einer SCHLANGE verwaltet werden können sollte, müssen auch diese über die Fähigkeit des Einreihens in eine SCHLANGE verfügen. All diese Klassen, deren Objekte in eine SCHLANGE eingereiht werden können sollen, haben also ***eine gemeinsame Fähigkeit***. Gemeinsamkeiten werden durch Vererbung oder durch Interfaces abgebildet. Nun haben aber PATIENT, TAXI und DRUCKAUFTRAG so absolut gar nichts gemeinsam, so dass sich keine sinnvolle Vererbungshierarchie ergibt. Daher bietet sich hier das Interface an, da sie eh nur eine Fähigkeit gemeinsam haben: sich in eine SCHLANGE einreihen zu lassen.

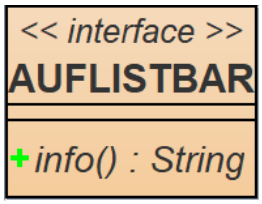
Wiederholung:

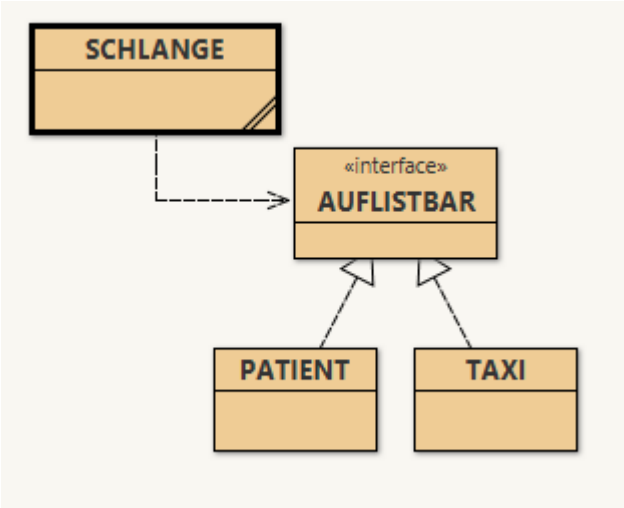
Während eine klassische Vererbung (extends KLASSE) ein ***„ich bin ein“*** zum Ausdruck bringt (ein DACKEL ist ein HUND), so verwendet man ein Interface (implements INTERFACE) anstatt eine Superklasse immer dann, wenn man ein ***„ich kann ... tun“*** zum Ausdruck bringen möchte.

Interfaces	beschreiben also ein oder mehrere Methoden, die nötig sind, um eine Fähigkeit „nachzurüsten“.
Superklassen	hingegen bringen zum Ausdruck, dass man eine Spezialisierung einer allgemeineren Spezies darstellt.

Aufgabe 3:

*Erweitere dein Projekt **SchlangeArray**, um das Interface AUFLISTBAR und ändere die Datentypen in der Klasse SCHLANGE auf AUFLISTBAR, sodass die Schlange Patienten und Taxen verwalten kann.*





Was fällt dir beim Testen der Schlange mit verschiedenen Objekten (TAXT, PATIENT) auf? Besitzt die Schlange Möglichkeiten, die so nicht sein sollten?

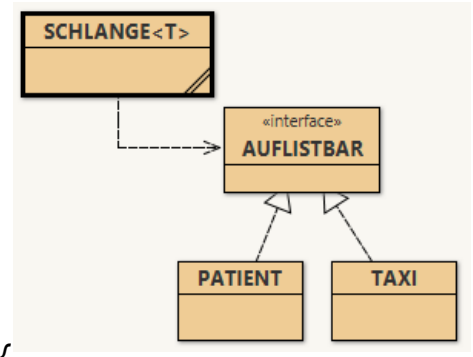
Generische Datentypen

Jetzt muss man nur noch der SCHLANGE beibringen, dass sie beliebige Objekte aufnehmen darf (bereits erledigt), aber ***sobald man ein Objekt eines Typ aufgenommen wurde, darf die SCHLANGE nur noch Objekte diesen einen Typs zulassen***. Dies realisiert man in Java mit Hilfe von ***Generischen Datentypen***.

- Ein generischer Datentyp ist ein ***Platzhalter für einen noch unbekannten Datentyp***.
- Der Platzhalter wird in ***dreieckige Klammern*** geschrieben.
  z.B. ***<T>***

- Der Platzhalter muss direkt nach dem Klassennamen erwähnt werden.
  z.B. ***public class SCHLANGE<T>***
- Innerhalb der dreieckigen Klammern kann man **optional** auch noch angeben, dass die unbekannte Klasse von einer anderen Klasse erben oder ein bestimmtes Interface implementieren soll.
  z.B. ***<T extends AUFLISTBAR>***
- Beim Aufruf des Konstruktors einer solchen Klasse muss man dann den konkreten Typ in dreieckigen Klammern mit angeben. Bei der Instanziierung der Klasse zur Laufzeit (wenn das Programm ausgeführt wird) wird sie an den angegebenen Datentyp (hier: TAXI) gebunden.
  z.B. ***new SCHLANGE<TAXI>()***

Aufgabe 4:



*Erweitere dein Projekt **SchlangeArray**, um den generischen Datentyp.*

Einige Code-Ausschnitte, bei denen generische Typen oder Interfaces Verwendung finden:

```
public class SCHLANGE<T extends AUFLISTBAR> {
  …
  private AUFLISTBAR[ ] elemente;
  …

  public boolean hintenEinreihen(T t) {
     …
  }

  public T vorneEntnehmen() {
     if ( !istLeer() ) {
        AUFLISTBAR a = this.elemente[0];
        for ( int i=0 ; i<this.akt_zahl-1 ; i++ ) {
           this.elemente[i] = this.elemente[i+1];
        }
        this.akt_zahl--;
        this.elemente[this.akt_zahl] = null;
        return (T)a;
     }
     else {
        return null;
  } } }
```