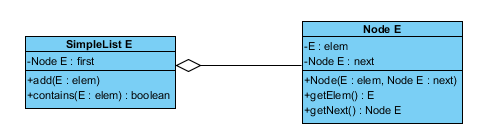
Übung 1:

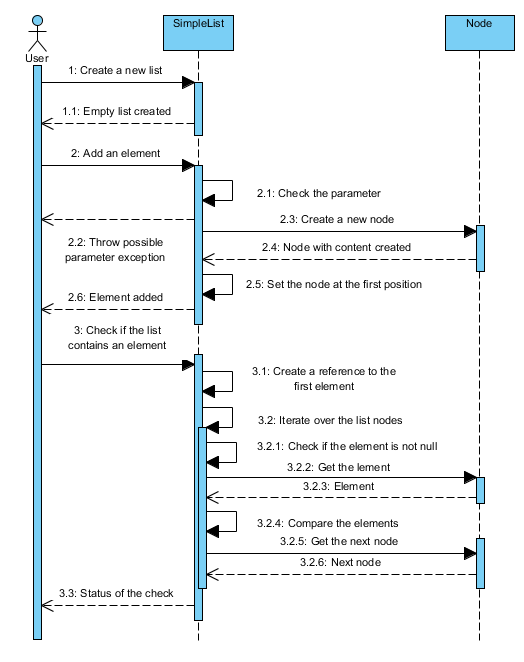
# Logical/Development View der List

## Logical View

Generell lässt sich die Liste als Klassendiagram modellieren. Die Klasse SimpleList kann dabei Nodes enthalten, muss sie aber nicht (leere Liste):

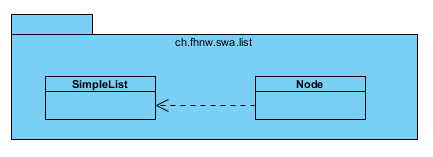


Das Verhalten lässt sich mittels Sequenzdiagram modellieren:



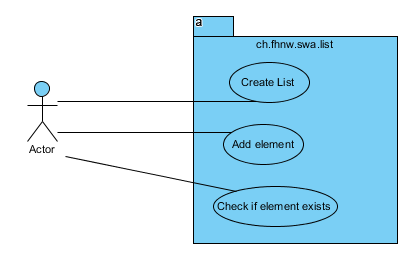
## Development View

Für den Development View lässt sich analog zum Klassendiagram ein Packagediagram realisieren:



## Use Case View

Der Use Case View lässt sich als typisches Use Case Diagram modellieren:



# Abstraction/Components and Interactions of Observers

Welche „Eigenschaften“ des gesamten Systems gewährleistet die „Observer-Architektur“:

* Es gewährleistet die Benachrichtigung zwischen einem oder mehreren Komponenten, sodass diese auf die Änderung einer Komponente reagieren können, zum Beispiel eine grafische Komponente, welche ein Datenmodell beobachtet

Welche nicht Architektur-relevanten Entwurfsentscheidungen sind den einzelnen Komponenten (bzw. deren Programmierern) überlassen? (Geben Sie jeweils auch die entsprechende Komponente an.):

* Die Architektur beschreibt nicht, wie das Pattern konkret implementiert ist, das wäre also nonarchitectural, respektive designbedingt. Das Design gibt vor, ob die Synchronisierung im gleichen Prozess, über Sockets oder sogar Interprocesscommunication erfolgen soll 🡪 Die Architektur besagt einfach, dass die Komponenten sich beobachten können müssen
* Die Architektur beschreibt auch nicht die Kardinalität zwischen den Subjects und den Observern (Sie kann, sollte es gefordert werden). So kann in Java Swing nur ein Listener, in JavaFX aber mehre Observer installiert werden (Diese Information spiegelt sich im Design wieder)

Welcher Architectural View (gemäss dem 4+1-Modell von Kruchten) gehört das oben darstellte Strukturdiagramm? Begründen Sie Ihre Aussage:

* Durch das Klassendiagram kann man das Das Diagram ganz klar dem Logic View einordnen. Das Diagram zeigt dabei die Funktionalität nach Aussen sowie die Abhängigkeiten der einzelnen Klassen untereinander

# Qualitätseigenschaften von Software-Systemen

Bewertung der App zur Fahrplanauskunft für Kunden (Lebensdauer 2 – 3 Jahre, danach Redesign):

* Änderbarkeit/Wartbarkeit: Dieser Kritikpunkt ist für den Kunden komplett unwichtig, da er nicht mit der Wartung und dem Unterhalt der Applikation beauftragt ist. Für ihn muss die Applikation einfach funktionieren
* Benutzbarkeit: Die Benutzbarkeit stellt eines der wichtigsten Kriterien dar. Können Kunden die Applikation nicht oder schlecht bedienen, können sie nicht oder nur schlecht Auskünfte einholen und verpassen möglicherweise einen Zug oder steigen falsch um -ein Imageverlust droht
* Effizienz: Nach der Benutzbarkeit stellt die Effizienz das zweitwichtigste Kriterium dar, da Kunden schnell eine Auskunft einholen möchten (Ich stehe am Bahnhof, habe ich in 5 Minuten einen Zug nach Hause?)
* Funktionalität: Die Funktionalität ist bis auf das Heraussuchen der Fahrplanauskunft meiner Meinung nach zweitranging und verhältnismässig unwichtig. Zwar gibt es in gewissen Applikationen Features wie «Bring me home», doch stellen diese nicht die Kernfunktionalität der Applikation dar, nämlich das heraussuchen der Fahrpläne
* Übertragbarkeit: Die Übertragbarkeit ist für den Kunden schwarz oder weiss: Entweder es gibt die Applikation für seine Platform oder es gibt sie nicht. Mit den heutigen Frameworks scheint diese Eigenschaft aber in den Hintergrund zu rücken (Mobileoptimierte Webseite oder Hybridapplikation), respektive ist leicht zu bewältigen
* Zuverlässigkeit: Nach der Effizienz der drittwichtigste Kritikpunkt. Ein Fehler oder eine falsche Aussage verärgert den Kunden oder lässt ihn einen Zug verpassen, doch geht davon die Welt nicht unter

Bewertung der Software eines Eisenbahn-Stellwerks (Lebensdauer in der Dimension von 30 – 50 Jahren):

* Änderbarkeit/Wartbarkeit: Die Änderbarkeit ist verhältnismässig unwichtig, da die Systeme für eine lange Zeitspanne konzipiert werden. Sie verfügen über eine gewisse Grundkomplexität, welche kurzfristige Änderungen verunmöglicht – auch um die Zuverlässigkeit und Integrität der Systeme zu garantieren. Hingegen müssen die Systeme gut wartbar sein, so dass sie ihren Dienst mit minimsten bis keinen Ausfällen tätigen können
* Benutzbarkeit: Eine gute Benutzbarkeit ist zwar wünschenswert und unterstützt das Personal und verhindert möglicherweise Falschkommunikation oder Unfälle, doch ist die Zuverlässigkeit und Integrität des Systems essentiell.
* Effizienz: Wie bei der Benutzbarkeit ist die Effizienz zwar wünschenswert, aber auch untergeordnet. Kann eine Aufgabe in akzeptabler Zeit bewältigt werden, muss die Effizienz nicht noch verbessert werden – ausser sie fördert das Verhindern von Unfällen (z.B. wenn ein System einen übermüdeten Fahrdienstleiter erkennt und einen Austausch veranlagt oder einen Alarm auslöst)
* Funktionalität: Die geforderten Anforderungen betreffend Funktionalität müssen erfüllt sein. Mehrfunktionalität ist nur bedingt interessant
* Übertragbarkeit: Die Übertragbarkeit ist faktisch inexistent, da die Software für einen sehr spezifischen Zweck konzipiert und realisiert worden ist. Ein Wechsel auf ein anderes System muss sehr sorgfältig geplant und umgesetzt werden (z.B. kann ein Stellwerksystem der Deutschen Bahn nicht ohne Planung und tiefgreifende Änderungen bei der SBB eingesetzt werden)
* Zuverlässigkeit: Die Zuverlässigkeit der Stellwerksoftware ist das wichtigste Qualitätskriterium und stellt alles andere in den Schatten. Da die Software in einem systemkritischen Bereich eingesetzt wird, muss sie robust und redundant geplant und realisiert werden. Es darf nicht vorkommen, dass durch einen Fehler zwei Züge zusammenstossen und es 200 Tote gibt – da nimmt man lieber ein benutzerunfreundliches User Interface in Kauf, welches aber solche Situation verhindert.

Abschliessend lässt sich sagen, dass die Fahrplanapplikation kundenorientiert und benutzerfreundlich gestaltet werden sollte. Es bringt nur bedingt Vorteile, viel Geld und Ressourcen zu investieren, da die Applikation innert fünf Jahren sicher wieder umgebaut oder neugeschrieben wird. Für die Software des Stellwerks steht die Zuverlässigkeit aber absolut an erster Stelle. Es ist besser, wenn Änderungsprozesse mehr Vorlaufzeit benötigen, dafür aber die Integrität und Sicherheit der Systeme aber nicht leidet.