Objektorientiertes Software Engineering

Einführung in die Objektorientierung

Was passiert, bis das Produkt fertig ist?

Ziele professioneller Software-Entwicklung

- Korrektheit
- Robustheit
- Erweiterbarkeit
- Wiederverwendbarkeit
- Verträglichkeit (*compatibility*)

Nachteile Imperative Programmierung

- Basiert auf:
  - Zustand, Variablen, Zuweisungen, Schleifen
- Konzeptionelle Trennung von Code und Daten
  - Code ist ohne Daten unbrauchbar,
  - und umgekehrt

Objektorientierte Programmierung

- ... hat sich bisher bewährt
- ... ist für kleine bis große Problemstellungen anwendbar
- ... Modell ähnelt den Modellen anderer Problembereiche
  - ... heisst “Objektorientierung” weil die Objekte der realen Welt abgebildet werden
  - Modellierung im Problembereich: ermöglicht Kommunikation mit Kunden, Anwendern, Entwicklern, etc.

Grundidee der OOP: “Little Computing Agents”

- Alles ist ein Objekt
- Objekte arbeiten zusammen (*collaboration*)
  - durch Senden von Nachrichten (*message passing)*
- Objekte haben ein eigenes Gedächtnis (*state*)
- Objekte sind Exemplare (*instance*) von Klassen
- Die Klasse beschreibt das Verhalten ihrer Exemplare
- Klassen sind in einer Vererbungshierarchie organisiert

Metapher

- Ein laufendes Programm ist ein Universum miteinander interagierender Objekte
- Objekte senden einander Nachrichten
- Es ist wie in der wirklichen Welt:
  - Probleme werden durch Delegation gelöst
    - ... indem jemand anderes die Arbeit tut

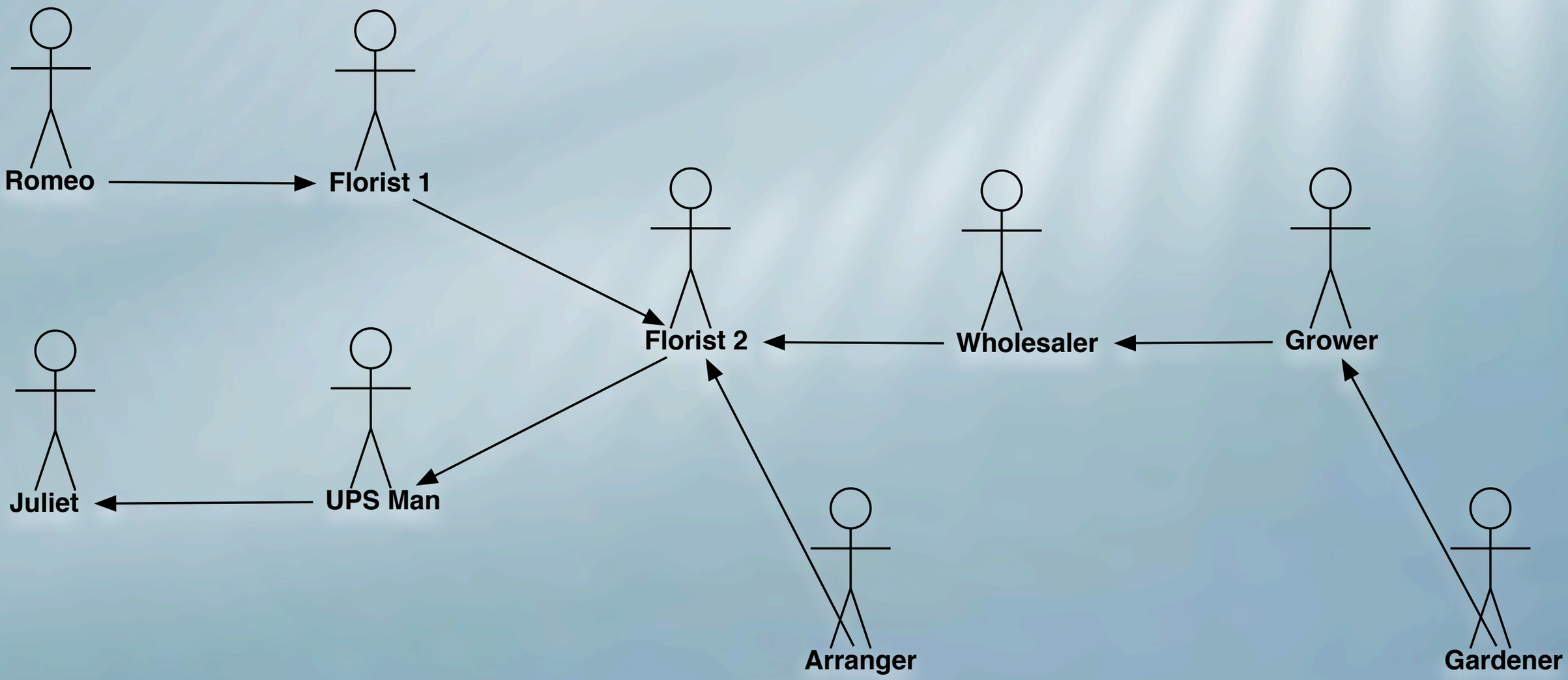
Blumen für Julia

- Romeo möchte seiner Freundin Blumen schicken, aber sie lebt in einer anderen Stadt

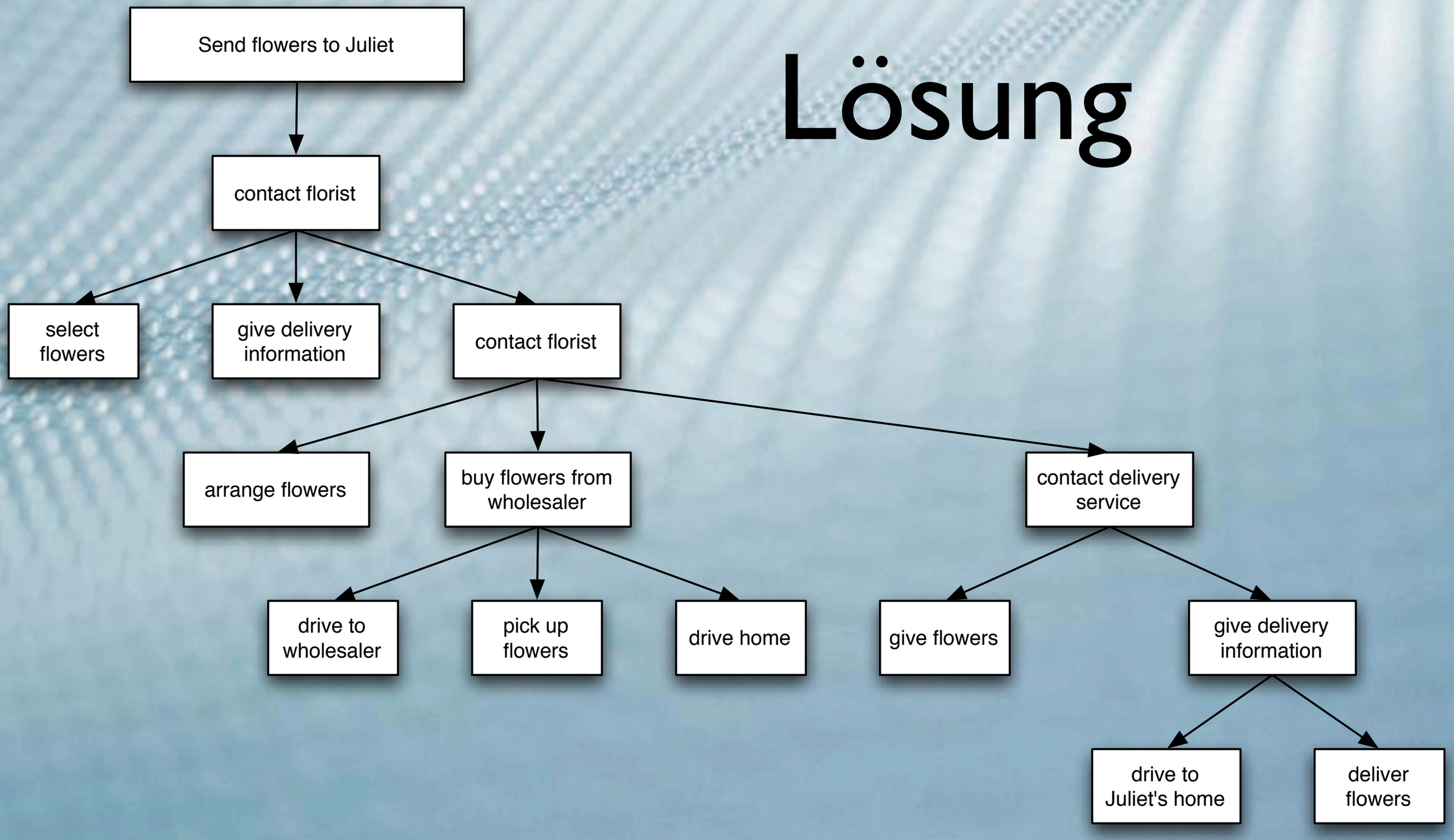
- Service eines Floristen; notwendige Informationen:

- Julias Adresse
- welche Blumen
- Kosten

- Romeos Florist kontaktiert seinen Kollegen
- Er leitet die Informationen weiter
- Dieser
- arrangiert den Strauss
- beauftragt einen Lieferservice
- dieser liefert die Blumen


Romeo
Florist 1
Florist 2
Wholesaler
Grower
Juliet
UPS Man
Arranger
Gardener

Strukturierte Lösung



OO Prinzipien

- Abstraktion (*abstraction*)
- Kapselung (*encapsulation*)
- Vererbung (*inheritance*)
- Polymorphismus (*polymorphism*)

OO-Prinzip Abstraktion (*abstraction*)

- ist notwendig, um komplexe Sachverhalte zu verstehen
- ausblenden *gegenwärtig* irrelevanter Details oder Aspekte
- die Essenz des Problems wird sichtbar
- Abstraktion ist abhängig vom Problembereich

OO Prinzip: Abstraktion

- Hat-Beziehung (*has-a*, *aggregation*)
- Ganzes-Teil-Beziehung (*composition*)
- Ist-ein-Beziehung (*is-a*) : Verallgemeinerung bzw. Spezialisierung
- Kapselung und Austauschbarkeit
- *service view*
- Entwurfsmuster

OO-Prinzip: Kapselung (*encapsulation*)

- Geheimnisprinzip (*information hiding*)
  - Details der Implementierung bleiben verborgen
  - Zugriff ist nur über die Schnittstelle möglich
- Trennung von Schnittstelle (*interface*) und Implementierung
  - Schnittstelle stellt *service view* dar
  - Implementierung kann ausgetauscht werden

OO-Prinzip: Kapselung (*encapsulation*)

- Klassen
  - sind Bauplan ihrer Instanzen
  - Vereinigen Verhalten (Methoden, *methods*) und Eigenschaften (*members*, *attributes*)
  - haben eine öffentliche Schnittstelle
  - haben eine private Implementierung
  - Modellieren Konzepte aus dem Problem- und Lösungsbereich

OO-Prinzip: Kapselung (*encapsulation*)

- Objekte
  - sind *Exemplare* (*instances*) ihrer Klassen
  - werden zur Laufzeit erzeugt (*constructor*) und wieder zerstört (*destructor*)
  - haben eine *Zustand* (*state*), nämlich die Ausprägung Ihrer Eigenschaften
  - haben ein *Verhalten* (gegeben durch ihre Klasse)
  - haben eine *Identität*

OO-Prinzip: Kapselung (*encapsulation*)

- Assoziation
  - Beschreibt eine Beziehung zwischen Objekten
    - Kardinalität: Anzahl der beteiligten Objekte
    - Rolle: die Rolle der beteiligten Objekte
- Aggregation (Hat-Beziehung)
- Komposition (Ganzes-Teil-Beziehung)

OO-Prinzip: Vererbung (*inheritance*)

- Beschreibt die Beziehung zwischen Klassen (Oberklasse und Unterklasse, *super*- and *subclass*)
- Spezialisierung oder Generalisierung
- The Litmus Test:
  - Whenever you can say that class *A* "is-a kind of" *B*, then class B is a *subclass* of class *A*, and class *A* is a superclass of *B*.

OO-Prinzip: Vererbung (*inheritance*)

- Formen der Vererbung:
  - ... for specialization
  - ... for specification
  - ... for construction
  - ... for extension
  - ... for limitation
  - ... for variance
  - ... for combination/categorization

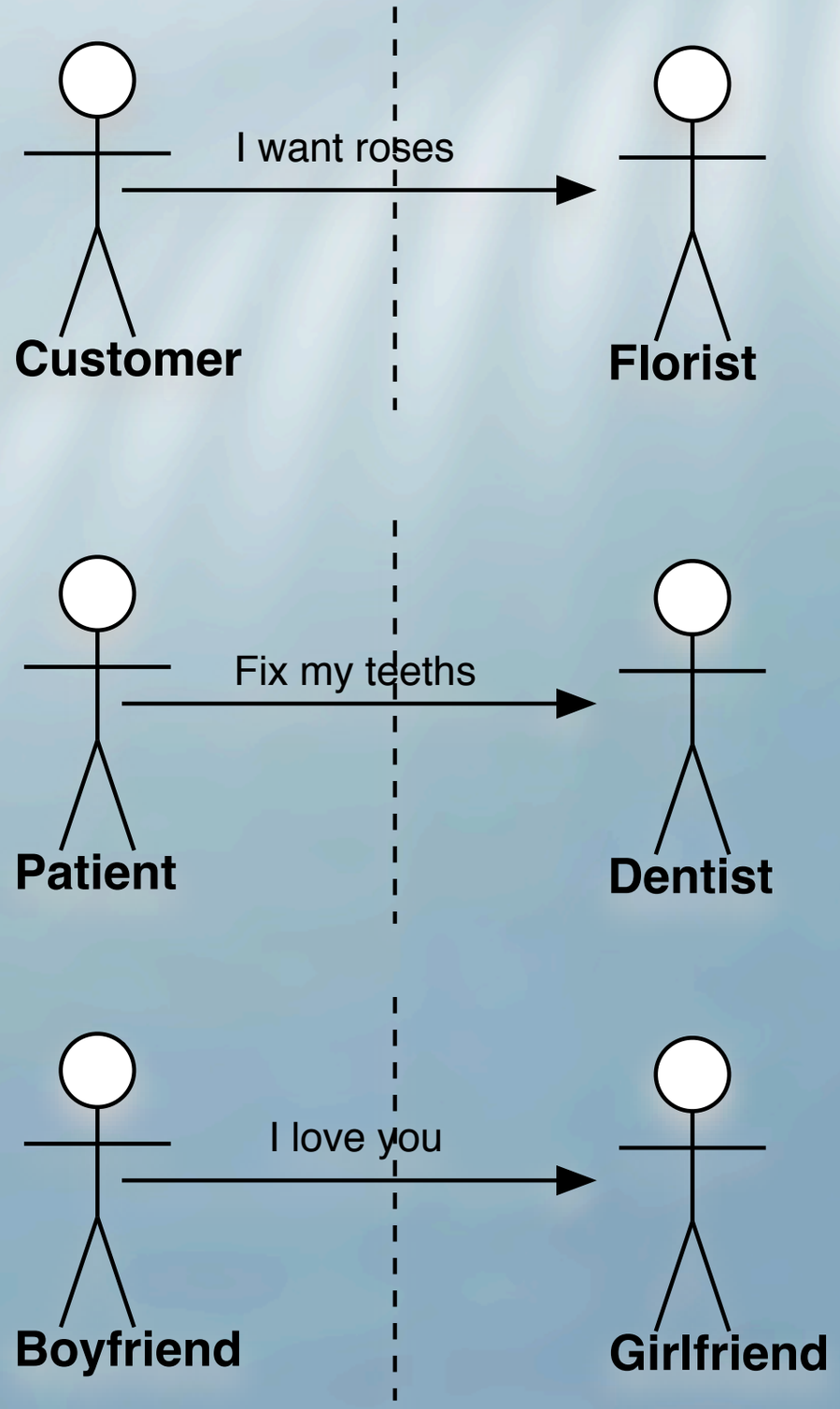
OO-Prinzip: Vererbung (*inheritance*)

- Kapselung
  - kann für Unterklassen aufgehoben werden (*open for extension, closed for modification*)
- Abstrakte Klassen
- *final* Klassen
- Schnittstellen (*interface*)
- Mehrfachvererbung (*multiple inheritance*)

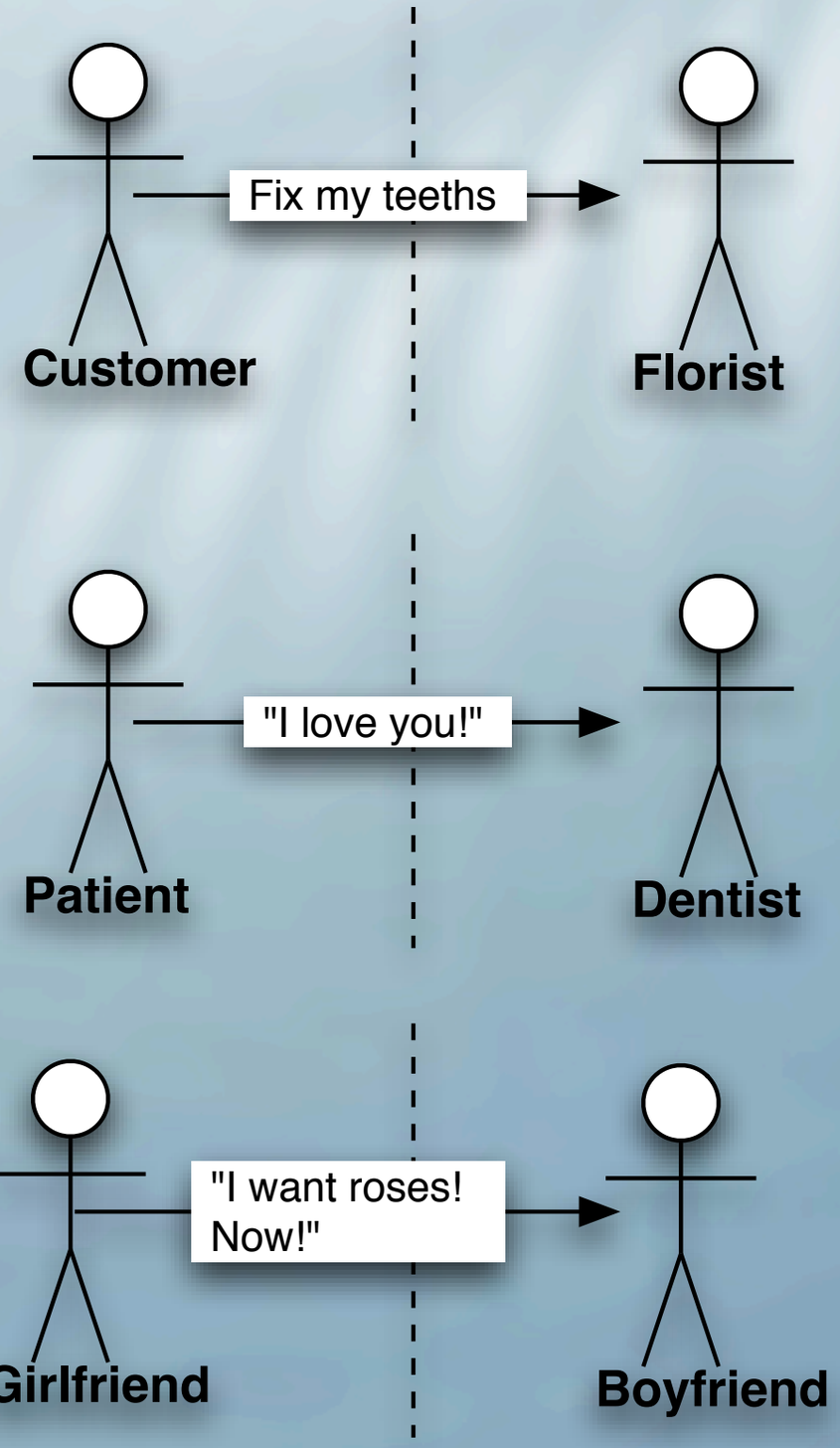
Schnittstellen

(*interfaces*)

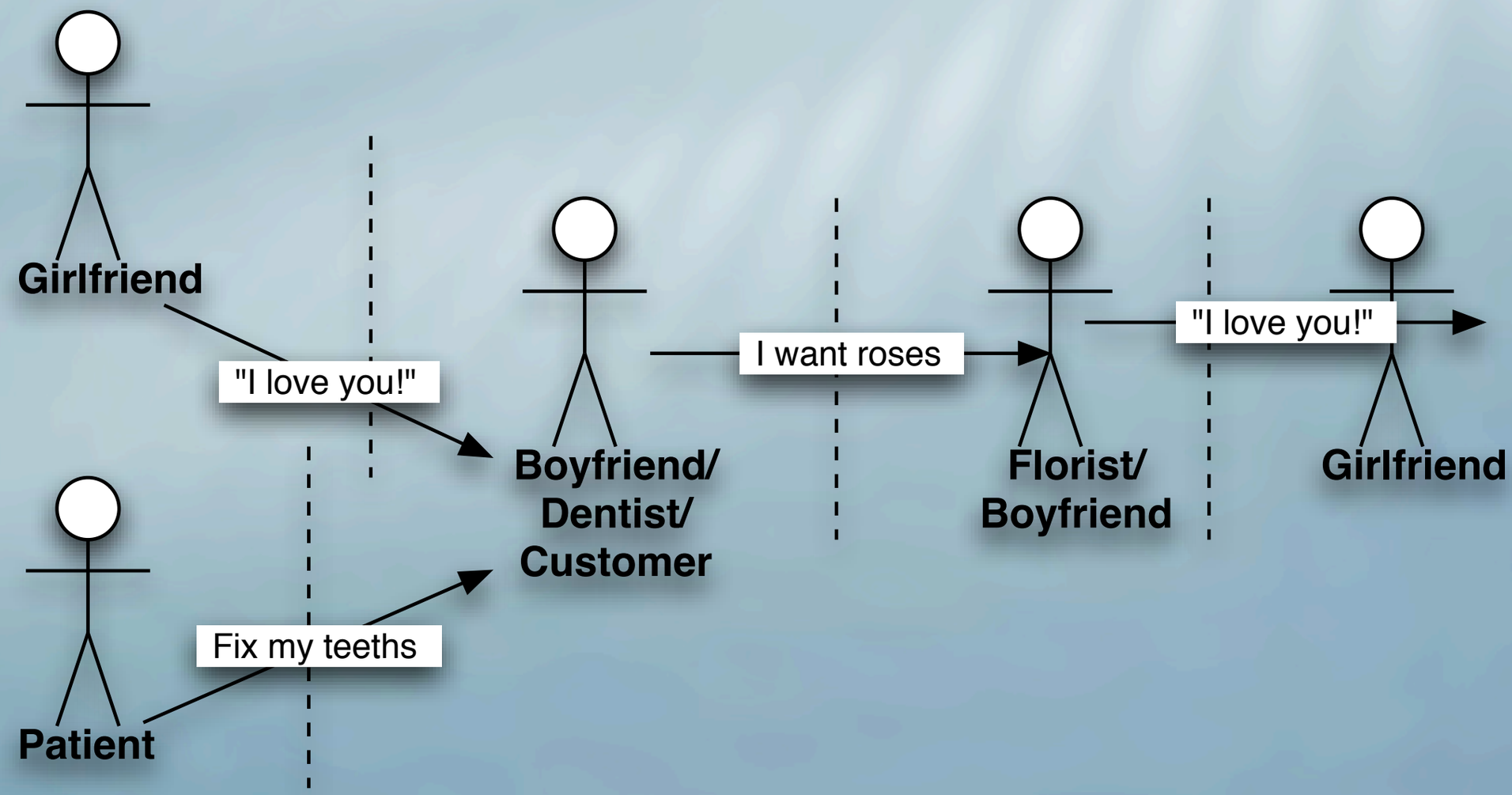
- Rolle = *Interface*



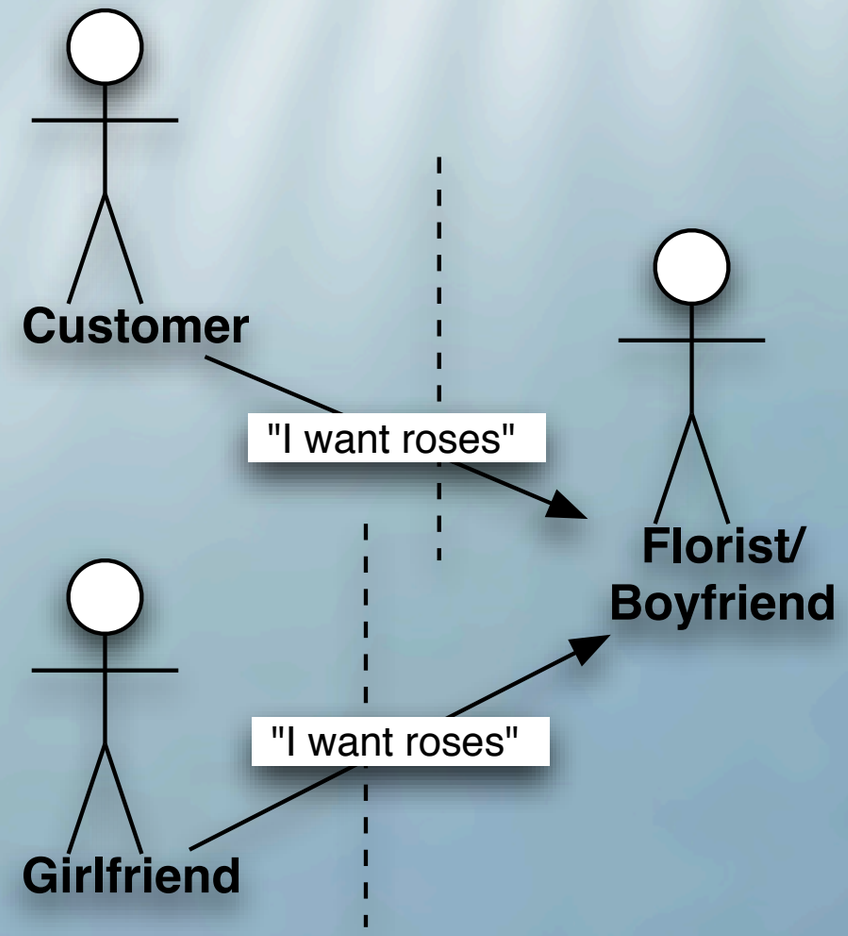
- Schnittstellen müssen passen



- Ein Exemplar kann mehrere Schnittstellen haben, d.h. mehrere Rollen spielen



- Same message, same object, different role, different behaviour



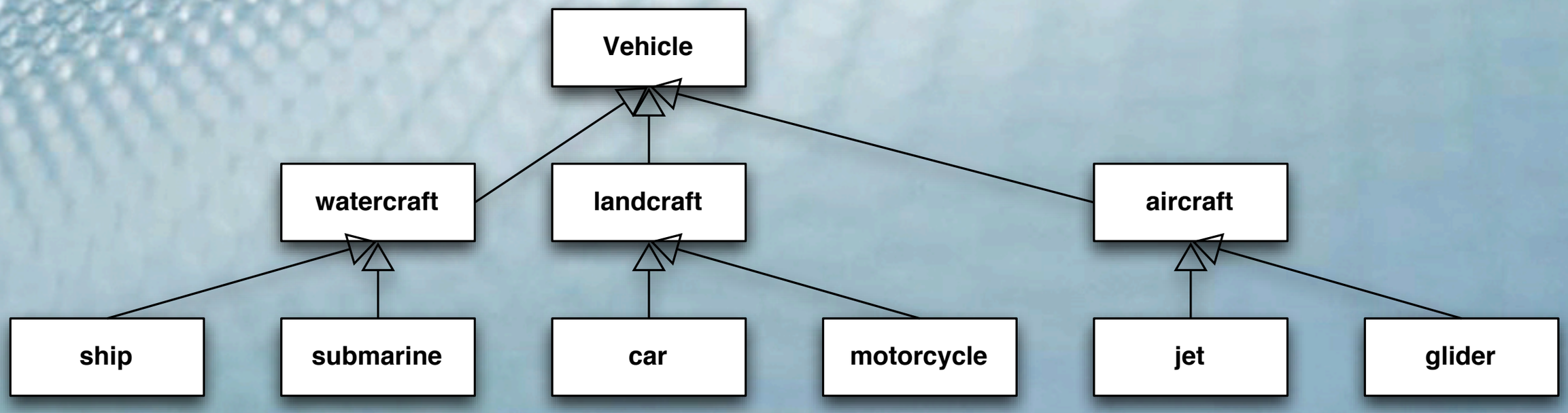
OO-Prinzip: Polymorphismus (*polymorphism*)

- Überladen (*overloading*)
  - von Operatoren (Typen der Operanden)
  - von Methoden (Typen der Parameter)
- Überschreiben (*overriding*)
  - von Methoden
    - replacement:
    - refinement
- Austauschbarkeit...

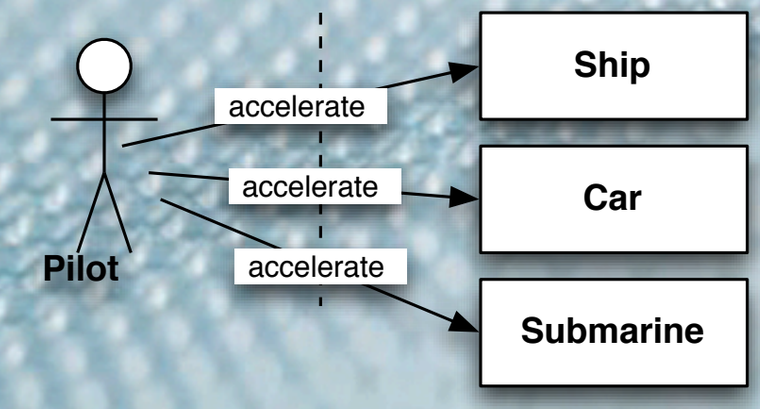
OO-Prinzip: Polymorphismus (*polymorphism*)

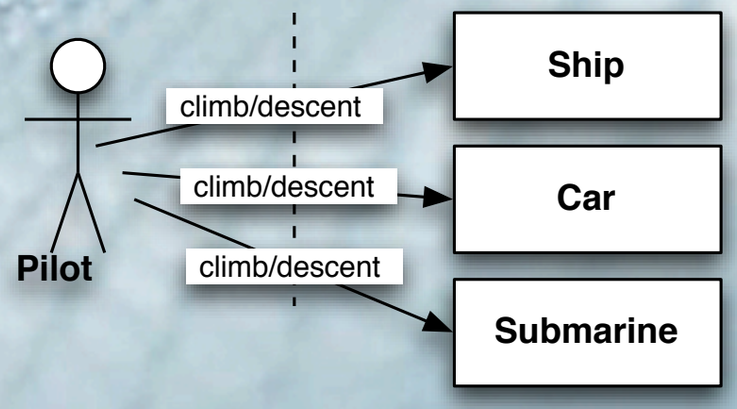
- Austauschbarkeit (*substitutability*)
  - Eine Instanz eine Unterklasse kann verwendet werden, als wäre sie eine Instanz einer Oberklasse
- The Liskov Substitution Principle: *Methods that use references to base classes must be able to use objects of derived classes without knowing it.*

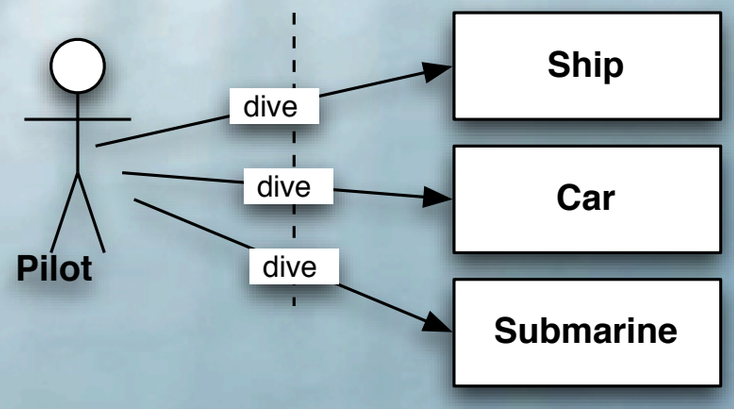
Vererbungshierarchie

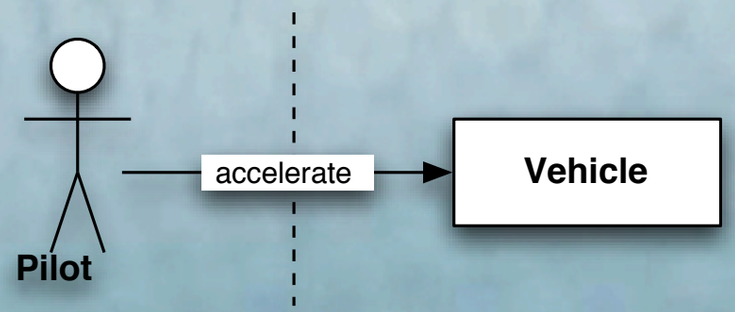


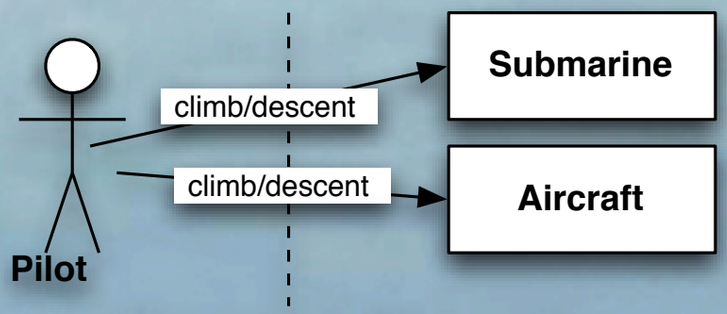
Polymorphismus

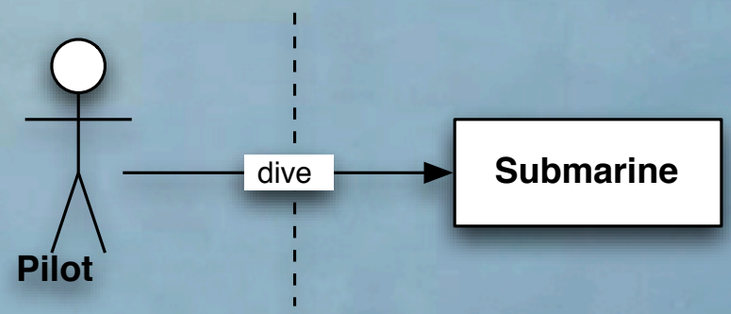












OO Prinzipien Zusammenfassung

- Abstraktion (*abstraction*)
  - Ignorieren irrelevanter Details
- Kapselung (*encapsulation*)
  - Geheimnisprinzip *(information hiding)*
  - Trennung von Implementation und Schnittstelle
- Vererbung (*inheritance*)
  - Erweiterung und Spezialisierung
- Polymorphismus (*polymorphism*)
  - Austauschbarkeit (*substitutability*)
  - Gleiche Schnittstelle, anderes Verhalten