Design: GRASP

OO Design

- Definition Objektorientiertes Design:
  - *"After identifiying your requirements and creating a domain model, then **add methods** to the software classes, and **define the messaging between the objects** to fulfill the requirements"*
- Kritische, nicht-triviale Entscheidung:
  - welche Methoden in welcher Klasse sind, und
  - diese Interagieren

Verantwortlichkeiten

- Verantwortlichkeit (*responsibility*)
  - Wissen (*knowing*)
    - Wissen über private, gekapselte Daten
    - Wissen über assoziierte Objekte
    - Wissen über Dinge, die abgeleitet oder berechnet werden können
- doing
  - Etwas selbst tun (Exemplare erzeugen, Berechnungen durchführen)
  - Aktionen in einem anderen Objekt initiieren
  - Aktionen in einem anderen Objekt kontrollieren oder koordinieren

Verantwortlichkeiten

- Verantwortlichkeiten werden während des Designs zugewiesen
- Eine Methode wird Implementiert, um eine Verantwortlichkeit zu erfüllen
- Verantwortlichkeiten können aus Interaktionsdiagrammen abgelesen werden
  - d.h. Zeitpunkt der Zuweisung einer Verantwortung

Entwurfsmuster (*Patterns*)

- Entwurfsmuster:
  - Bewährtes/Erprobtes Prinzip/Idiom
  - Benanntes Paar aus Problem und Lösung
  - Kann in neuen Fällen angewandt werden
  - Benennung: Kommunikation unter Experten

GRASP Patterns

- Beschreiben fundamentale Prinzipien des objektorientierten Designs über zu Zuordnung von Verantwortlichkeiten
- GRAPS: General Responsibility Assignment Software Patterns
- Information Expert
- Creator
- High Cohesion
- Low Coupling
- Controller

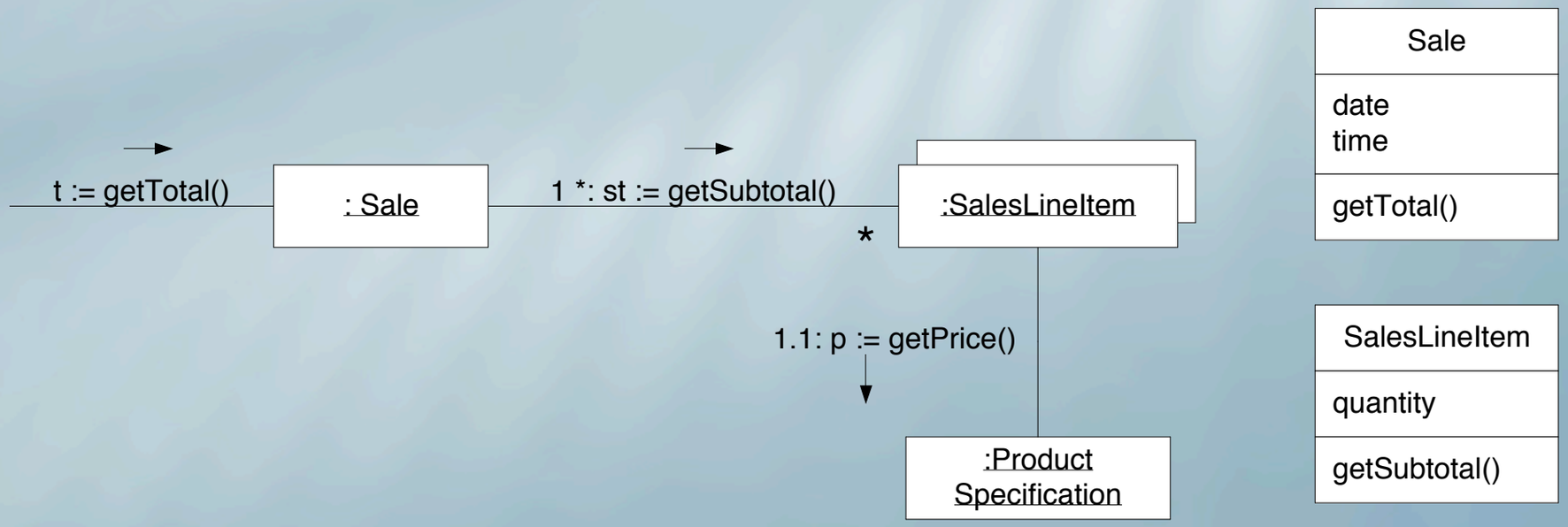
GRASP: Information Expert

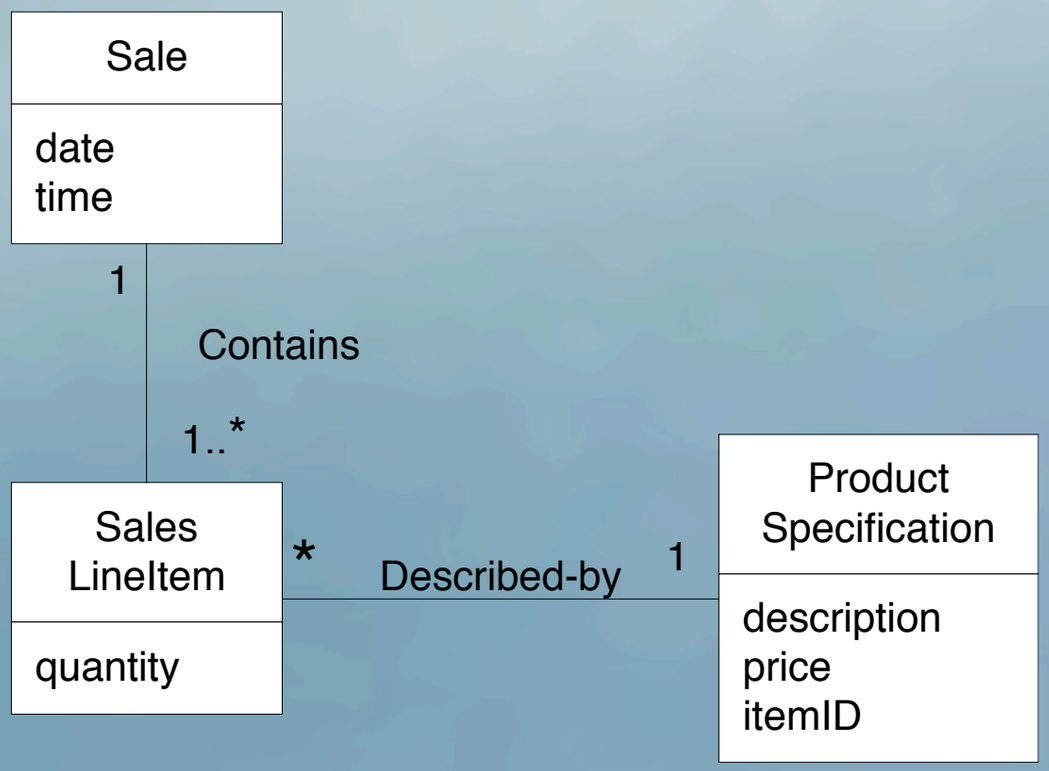
- **Problem**: Nach welchem grundlegenden Prinzip sollen Verantwortlichkeiten zugeordnet werden?
- **Lösung**: Ordne die Verantwortlichkeit dem *information expert* zu; d.h. derjenigen Klasse, die die notwendigen Informationen hat, um die Verantwortung zu übernehmen

Beispiel

- Klare Formulierung der Verantwortung
  - "Wer ist dafür verantwortlich, die Gesamtsumme zu wissen?"
- Analyse:
  - Welche Klasse hat die notwendigen Informationen?
  - Im Design Model?
  - Im Domain Model?

Information Expert Beispiel





GRASP: Information Expert

- Kontraindikation:
- Information kann ggfls. *Low Coupling* und *High Cohesion* verletzen
    - Bsp.: Persistenz, d.h. SQL und JDBC in allen Business Klassen (folgt aus Information Expert)
- Designentscheidung, was ist wichtiger

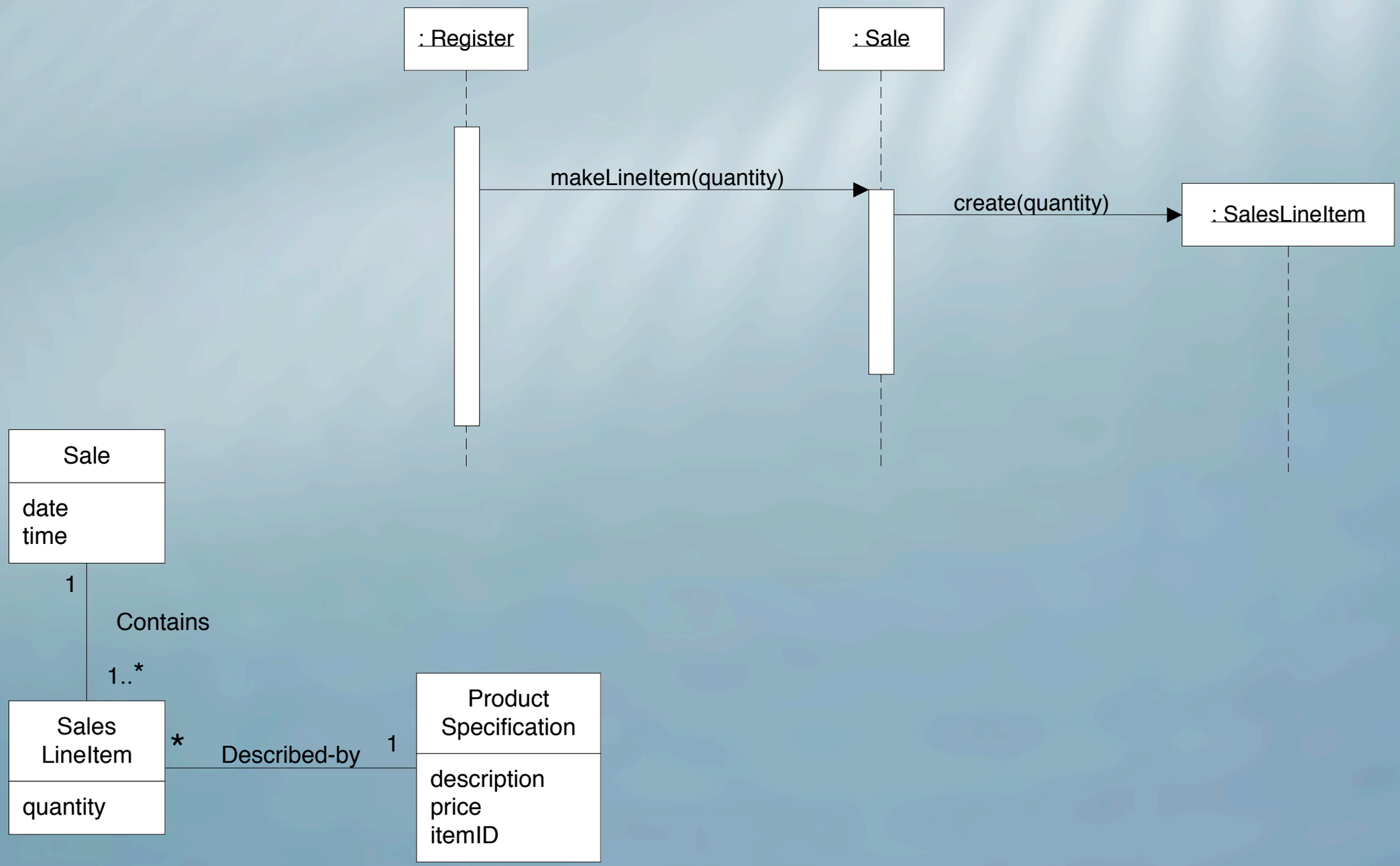
GRASP: Information Expert

- Vorteile:
  - Kapselung: Informationen verbleiben in *einem* Objekt: Führt zu *low coupling*.
  - Leichtgewichtige Klassen: Verhalten ist über die Klassen verteilt, die über die Informationen verfügen. Unterstützt *high cohesion*.

GRASP: Creator

- **Problem**: Wer ist für die Erzeugung neuer Instanzen verantwortlich?
- **Lösung**: Eine Klasse *B* ist Verantwortlich für die Instanziierung von Objekten der Klasse *A*, wenn
  - B Objekte der Klasse A aggregiert, enthält, verwaltet, häufig verwendet, oder
  - B über die Initialisierungsparameter verfügt

Creator Beispiel



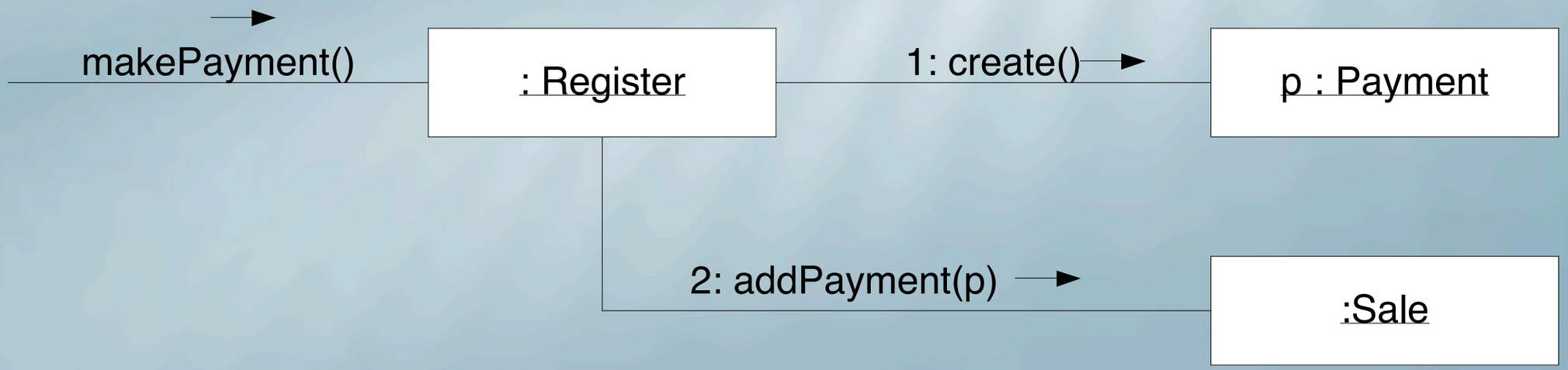
GRASP: Creator

- Vorteile:
  - Führt zu low coupling
- Gegenanzeige:
  - Falls Instanziierung komplex ist (Proxies, Recycling, etc.), sollte eher *Factory* verwendet werden

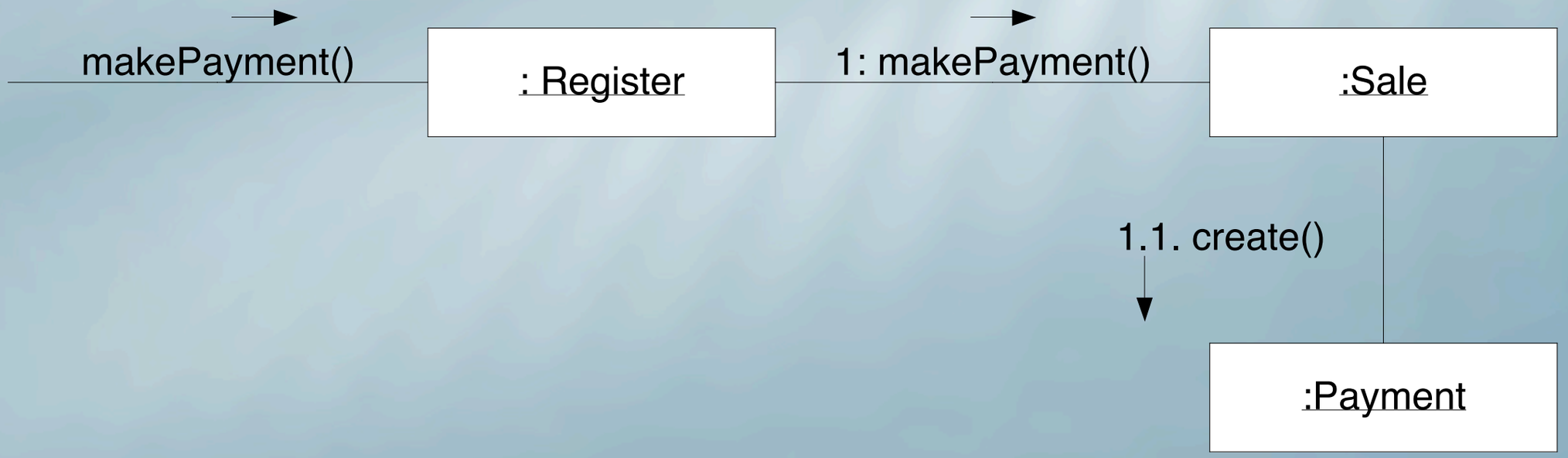
GRASP: Low Coupling

- **Problem**: Wie wird geringe Abhängigkeit, geringe Auswirkungen von Änderungen und hoher Wiederverwendbarkeit erreicht?
- **Lösung**: Ordne Verantwortung so zu, dass die Kopplung niedrig bleibt
  - Klassen sind gekoppelt, wenn diese Methoden aufrufen oder voneinander abhängig sind.

Low Coupling:
Beispiel 1



Low Coupling: Beispiel 2



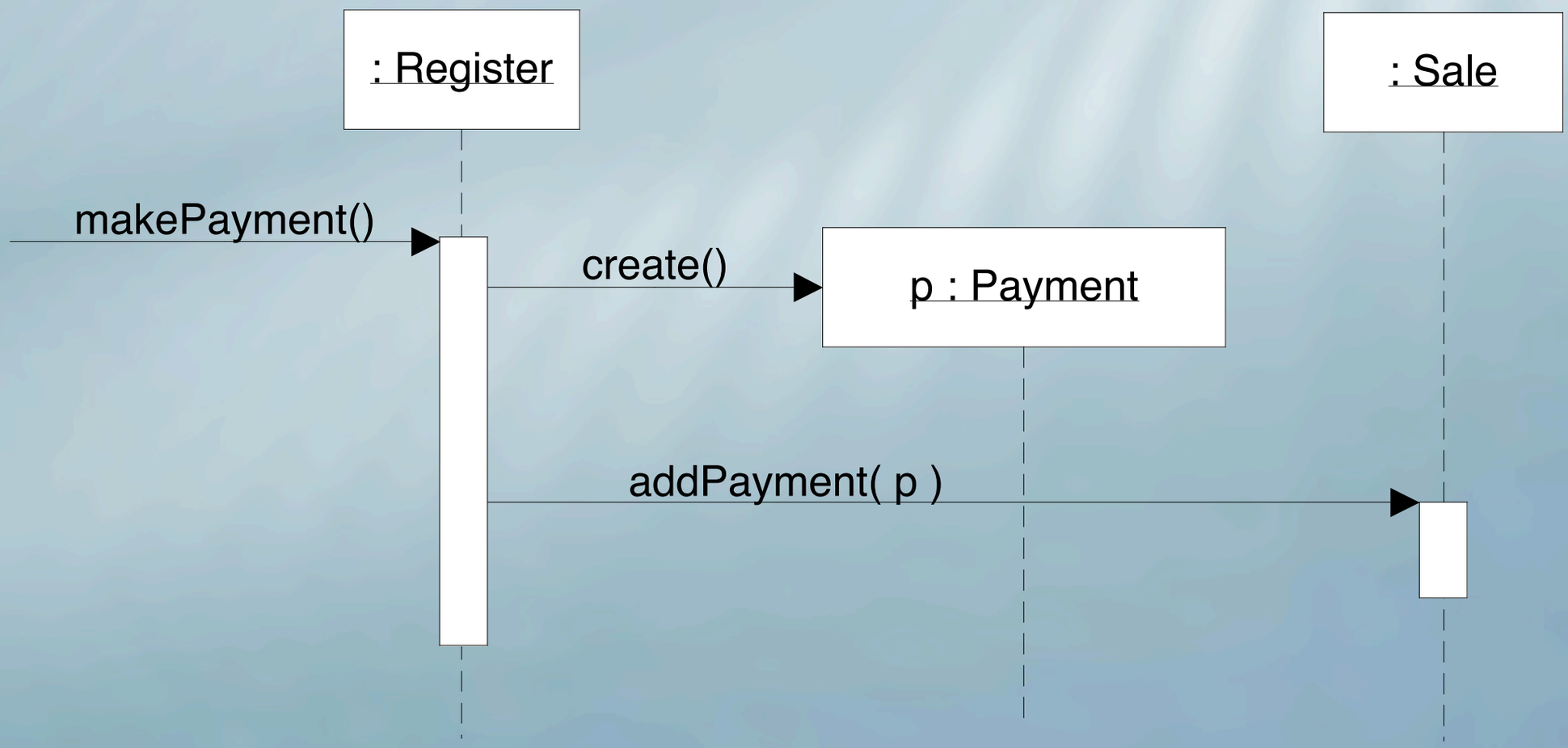
GRASP: Low Coupling

- Grundlegendes Prinzip:
  - Begünstigt Stabilität, Änderungsfreundlichkeit, Wiederverwendbarkeit,
- Extremfall: keine Kopplung: System besteht aus wenigen “Klötzen” kaum interagierender Objekte
- Ausdrücklich erlaubt: Enge Kopplung an stabile Klassen, z.B. java.util.*; J2EE, etc.
- Nur Kopplung an instabile Elemente ist “schlecht”.

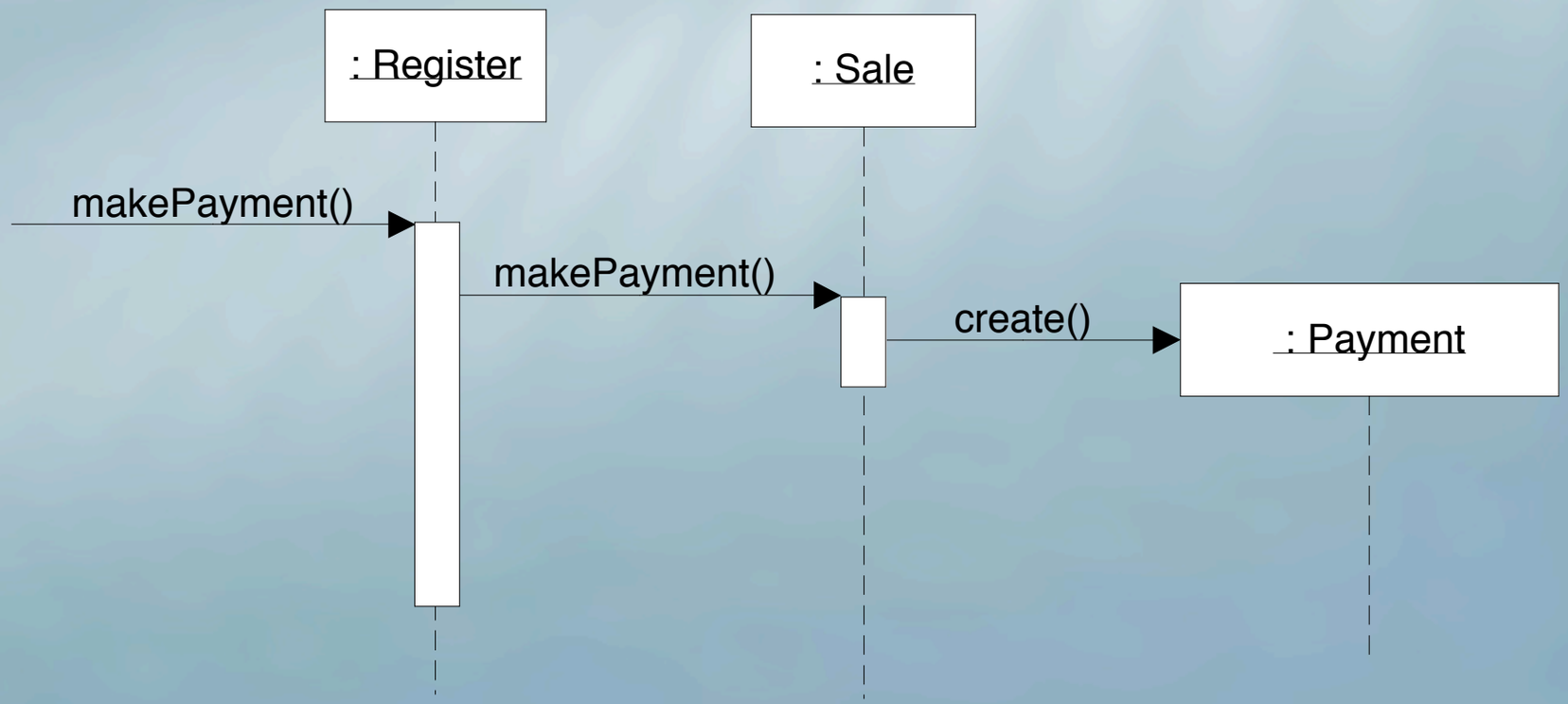
GRASP: High Cohesion

- **Problem**: Wie bleibt Komplexität zu bewältigen?
- **Lösung**: Ordne Verantwortung so zu, dass hohe Kohäsion entsteht
  - Kohäsion ist Kennzeichen, wie stark zusammengehörig und fokussiert ein Element ist.
- Klassen mit geringer Kohäsion sind
  - schwer zu verstehen/begreifen
  - schwer wiederverwendbar
  - schwer zu warten
  - Änderungsanfällig

High Cohesion:
Beispiel: Low Cohesion



High Cohesion:
Beispiel: High Cohesion & Low Coupling



GRASP: High Cohesion

- Grundlegendes Prinzip:
  - "*all work together to provide some well-bounded behavior*"
- *Very low cohesion*: Eine Klasse ist verantwortlich für viele Dinge in verschiedenen Bereichen: *RDB-RPC-Interface*
- *Low cohesion*: Eine Klasse ist verantwortlich für viele Dinge in einem Bereich: *RDB-Interface*
- *High cohesion*: ... überschaubare Verantwortung in einem Bereich, delegiert Verantwortung an weitere Klassen
- *Moderate cohesion*: ... in einigen Bereichen, die zwar zu der Klasse gehörig sind, aber nicht untereinander

Low Coupling, High Cohesion

- Klassisches Prinzip **Modularität**; folgt aus low coupling und high cohesion:
  - "Modularity is the property of a system that has been decomposed into a set of cohesive and loosely coupled modules." [Booch94]
- Low Coupling und High Cohesion bedingen einander

GRASP: Controller

- **Problem**: Wer ist für die Abwicklung von Systemereignissen verantwortlich?
- **Lösung**: Die Verantwortlichkeit für Systemereignisse wird zugewiesen an:
  - eine Klasse, die das Gesamtsystem repräsentiert (*facade controller*)
  - eine Klasse, die einen Use-Case repräsentiert, <UseCaseName>Handler, <UseCaseName>Controller oder <UseCaseName>Session.

GRASP: Controller

- Controller...
  - ist *kein* Objekt der UI-Schicht
  - erscheint *nicht* im Domain-Model, da reines Lösungskonzept (*pure fabrication*)
- erledigt wenig Arbeit selbst, sondern
  - koordiniert den Ablauf, Reihenfolge
  - delegiert Arbeit an Business Objekte
- dient dem UI-Layer als Abstraktion des Domain-Layer

GRASP: Controller

- Vorteile:
  - Austauschbarkeit der UI: UI, Use-Case Logik und Anwendungslogik sind entkoppelt
  - Vereinfachte Sicht von der UI-Schicht auf die Anwendungs-Schicht
  - Zustand eines Use-Case

Controller Beispiel