Zürcher Hochschule
für Angewandte Wissenschaften

School of
Engineering
InIT Institut für angewandte
Informationstechnologie

Bachelor of Science (BSc) in Informatik

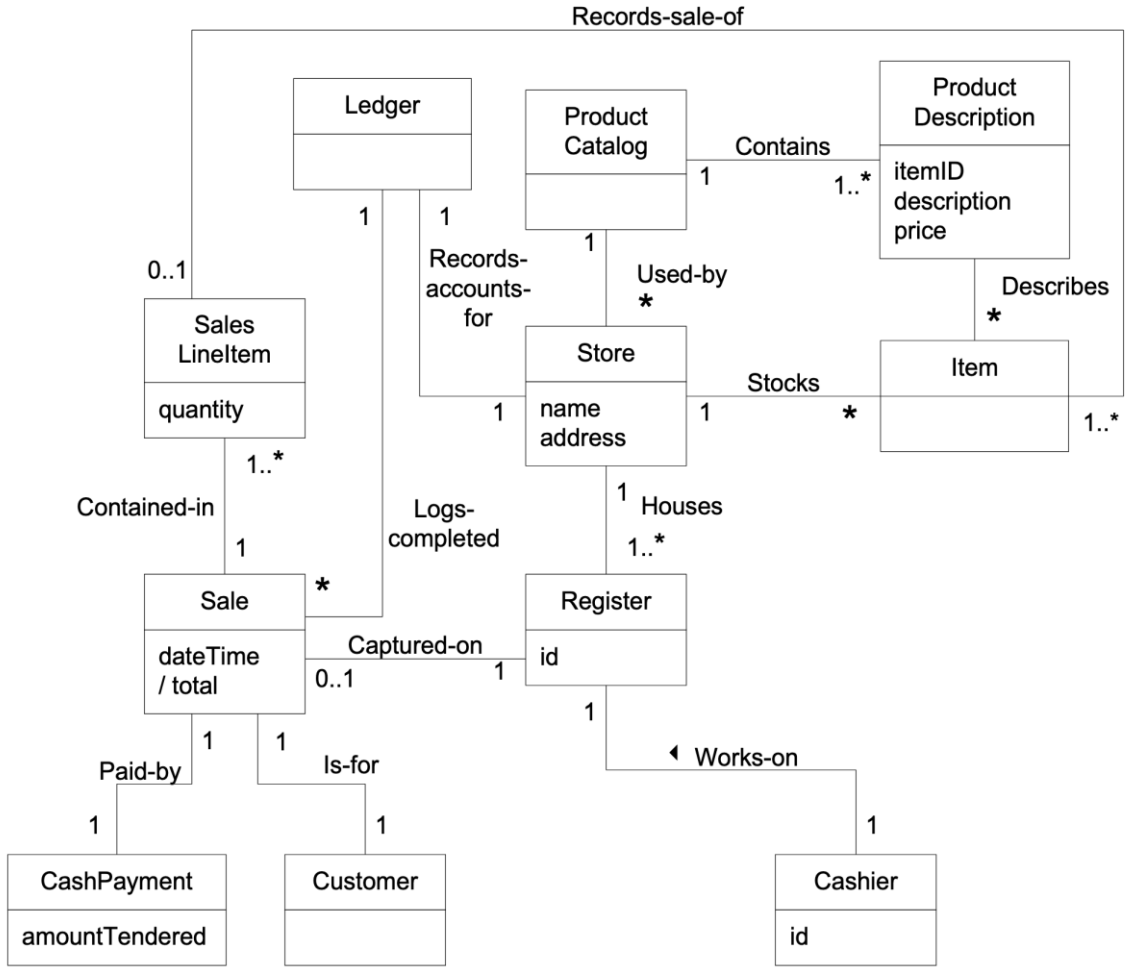
Modul Software-Entwicklung 1 (SWEN1)

Fallstudie NextGenPos: OO Design

Prof. Dr. H.-P. Hutter

Fallbeispiel NextGenPos: OO Design

- Resultate aus OO Analyse
  - Domänenmodell (siehe rechte Seite)
  - Systemoperationen
    - makeNewSale()
    - enterSalesLineItem(itemID, quantity)
    - endSale()
    - makePayment(amount)

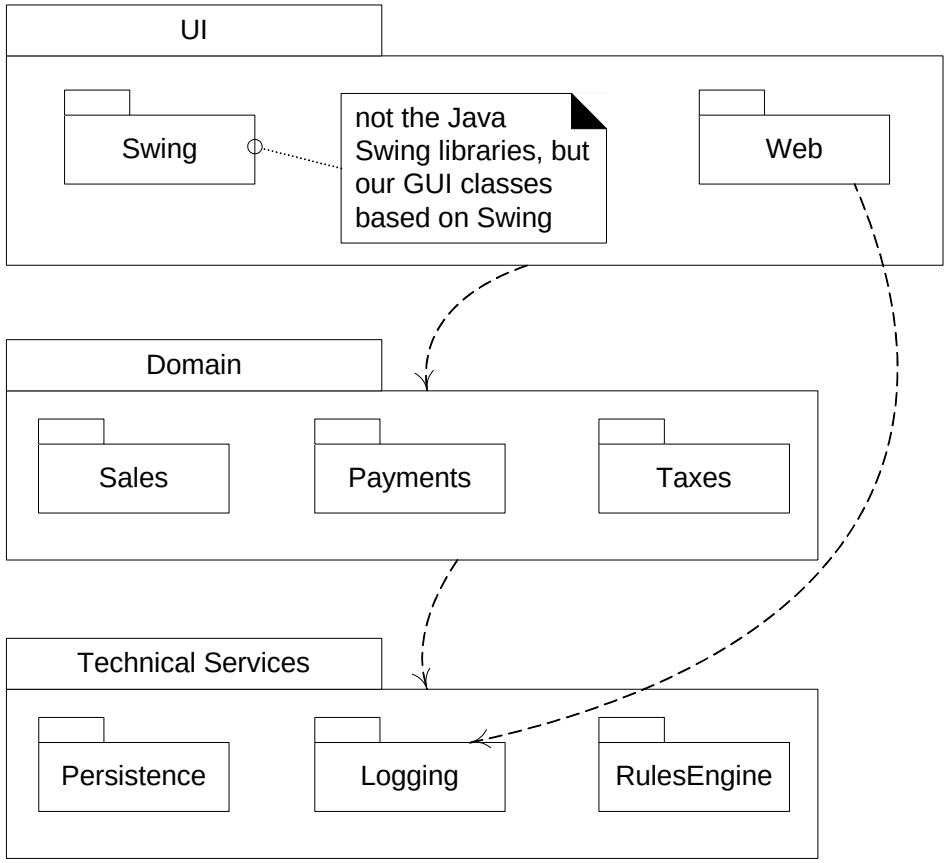


Fallbeispiel NextGenPos: OO Design

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

1. Schritt: Architektur
   - 3-Schichten-Architektur (UI, Domäne, Techn. Services)
   - Domänenlayer
     - Fassaden-Controller?
     - Session-Controller pro UC?

- Entscheid:
  - Fassaden-Controller
    - weil nur wenige Systemoperationen

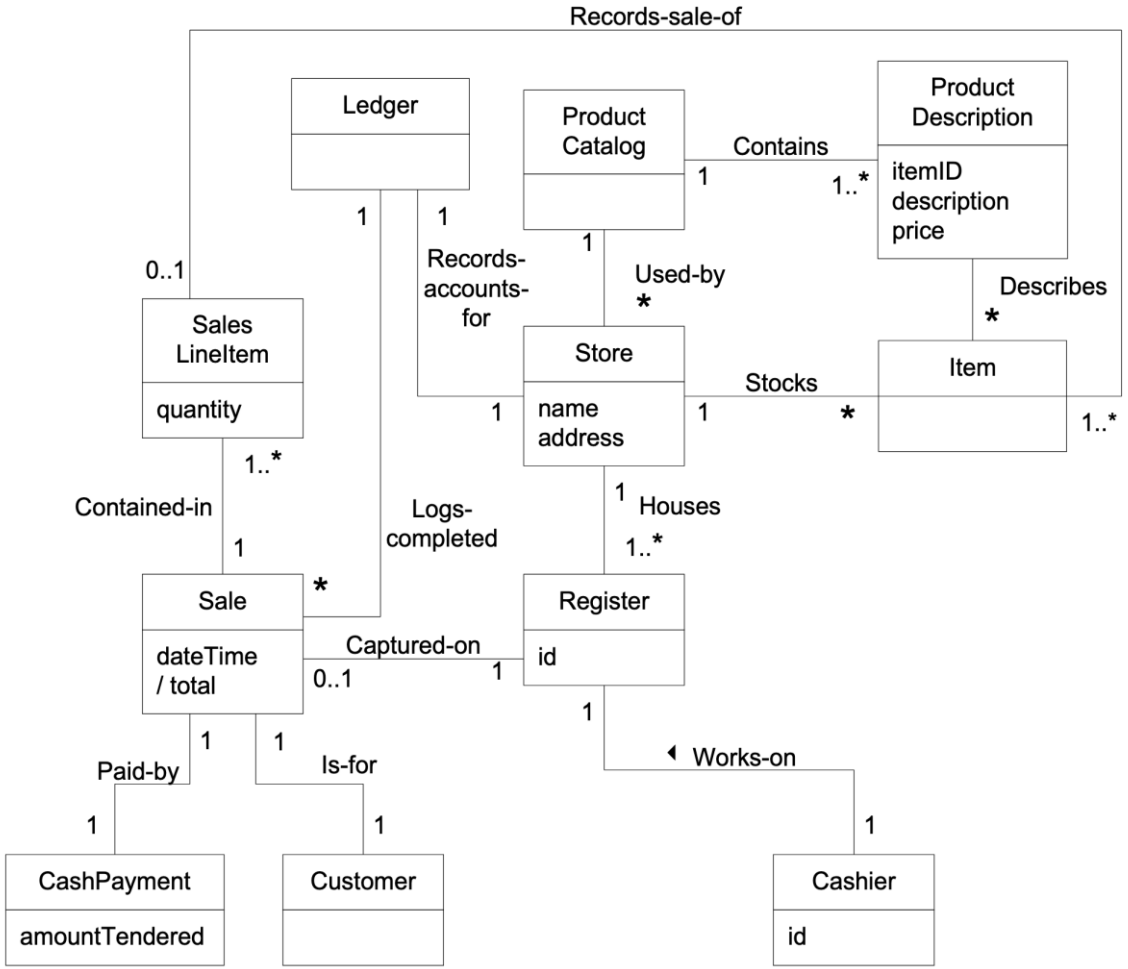


Fallbeispiel NextGenPos: OO Design

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

2. Schritt

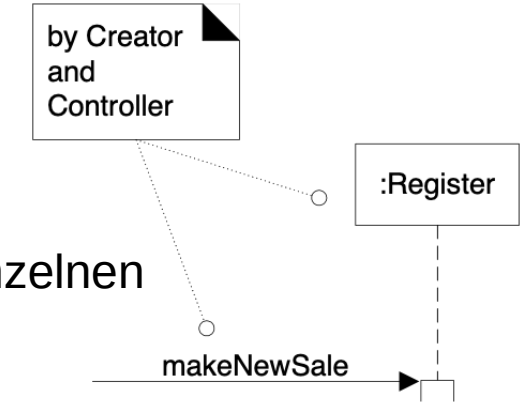
- Wer soll Verantwortung haben für das gesamte System?
  - Kandidaten aus Domänenmodell
    - Store, Register, Sale?
- Entscheid
  - Klasse `Register`
    - Verantwortung:
      „Verwaltet **eine** Kasse“
    - Knowing:
      - Kennt alle Verkäufe der Kasse
    - Doing:
      - Koordiniert alle Systemoperationen



Fallbeispiel NextGenPos: OO Design

3. Schritt

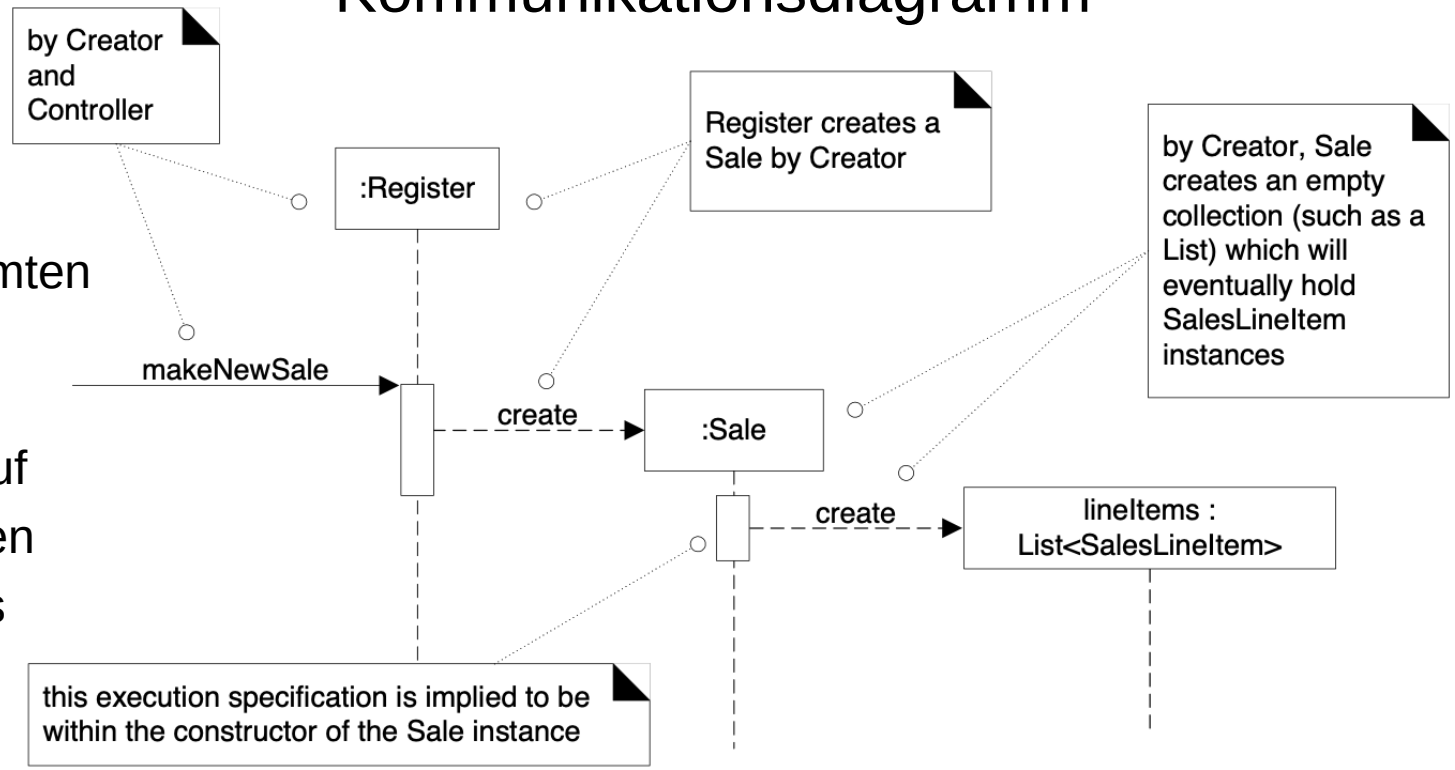
- Design der 1. Systemoperation
  - `makeNewSale()`
- Frage:
  - Wer soll Verantwortung für einen einzelnen Verkauf haben?
    - Register?
    - Objekt einer neuen Klasse?
      - Welche?



Fallbeispiel NextGenPos: OO Design

3. Schritt
   - Entscheid:
     - Neue Klasse `Sale`
       - Verantwortung:
         Ist verantwortlich für einen gesamten Verkauf
       - Knowing
         - Alle Details zu einem Verkauf
         - Kennt alle Verkaufspositionen
           - Liste von SalesLinItems
       - Doing
         - Einen Verkauf abwickeln
     - `Register` erzeugt `Sale`-Objekt (Creator)

- Kommunikationsdiagramm



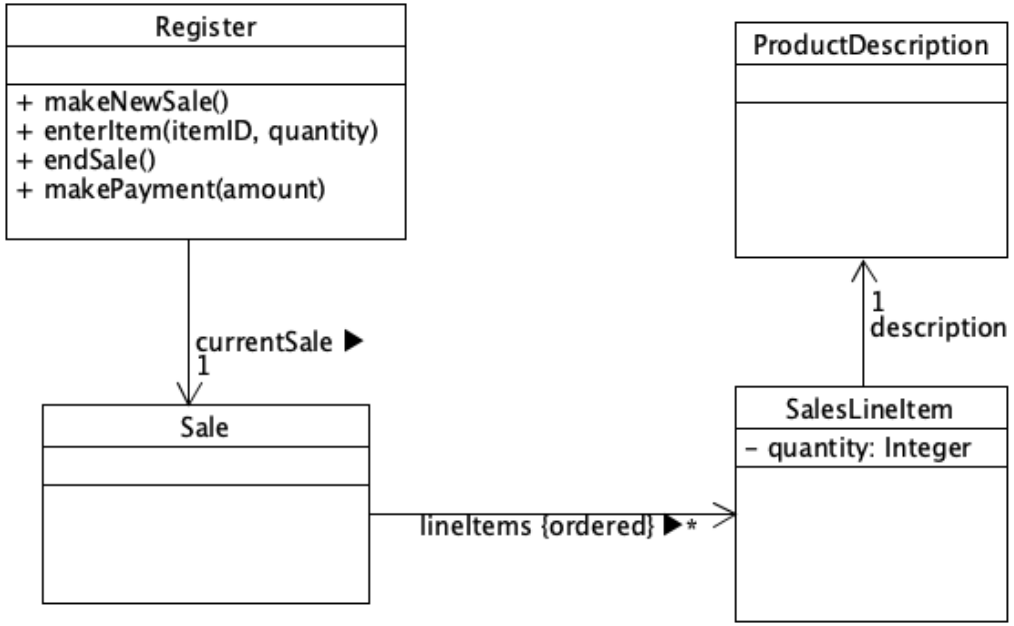
Fallbeispiel NextGenPos: OO Design

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

3. Schritt

- Wir brauchen Neue Klasse `SalesLineItem`
  - Verantwortung: “Verwaltet 1 SalesLineItem“
  - Knowing
    - „Produktbeschreibung“
    - Anzahl
  - Doing
    - ...

- Design-Klassendiagramm

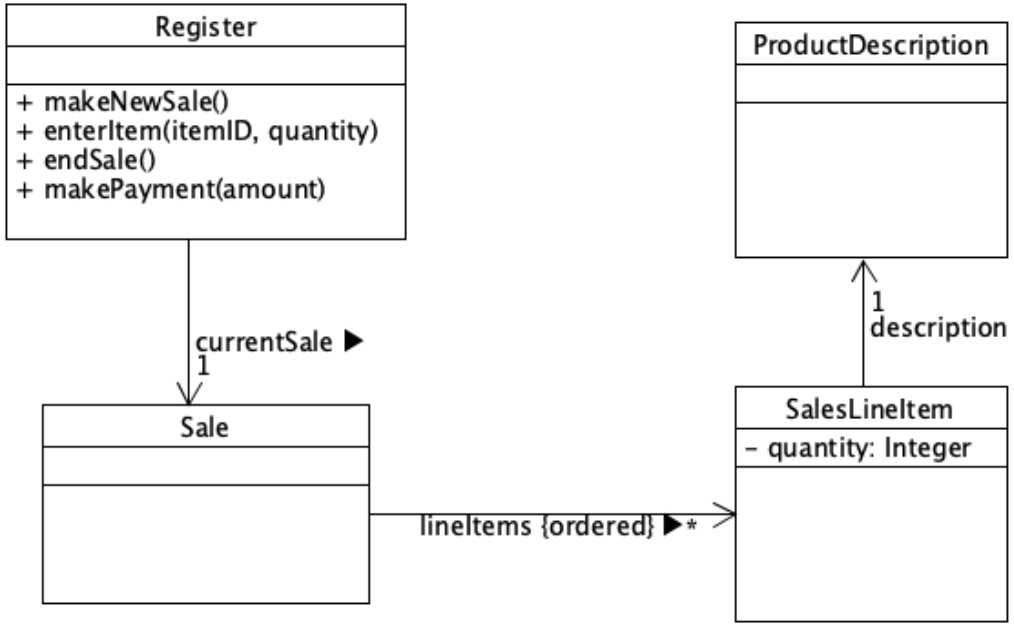


Fallbeispiel NextGenPos: Implementation

Zürcher Hochschule
für Angewandte Wissenschaften
School of Engineering
InIT Institut für angewandte Informationstechnologie

4. Schritt
   - Implementation/Test der Systemoperation `makeNewSale()` gemäss
     - Interaktionsdiagramm
     - Klassendiagramm

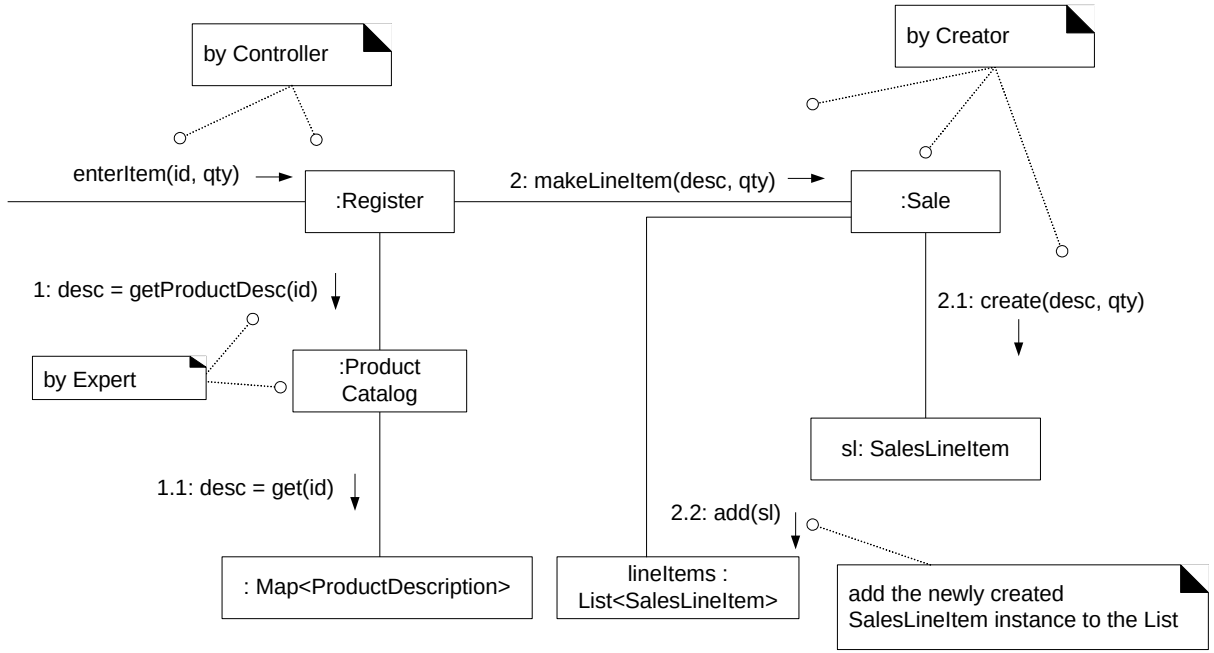
- Design-Klassendiagramm



Fallbeispiel NextGenPos: OO Design

- Nächster Schritt
  - Design und Implementation der nächsten SystemOperation
    - `enterItem(idemID, quantity)`
    - Wir brauchen neuen Klassen (siehe DM)
      - `ProductCatalog`
        - Verwaltet *alle* Produktbeschreibungen
      - `ProductDescription`
        - Verwaltet *eine* Produktbeschreibung
        - Kennt
          - ItemID
          - Produktpreis

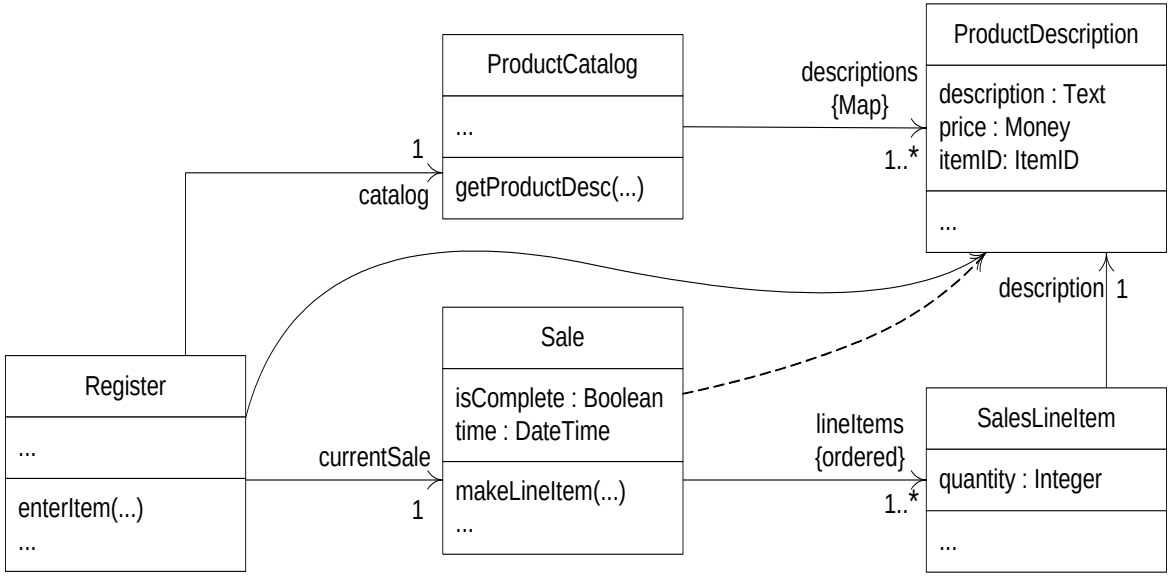
- Kommunikationsdiagramm



Fallbeispiel NextGenPos: OO Design

- Nächster Schritt
  - Design und Implementation der nächsten SystemOperation
    - `enterItem(idemID, quantity)`
    - Wir brauchen neuen Klassen (siehe DM)
      - `ProductCatalog`
        - Verwaltet *alle* Produktbeschreibungen
      - `ProductDescription`
        - Verwaltet *eine* Produktbeschreibung
        - Kennt
          - ItemID
          - Produktpreis
- Usw...

- Design-Klassendiagramm



Fallbeispiel NextGenPos: OO Design

- Wichtig beim Design
  - Verantwortlichkeiten verteilen gemäss GRASP-Patterns
  - Klassendiagramm und Interaktionsdiagramme parallel weiterentwickeln und konsistent halten
  - Bei jeder neu eingeführten Klasse
    - Verantwortlichkeit festlegen und festhalten (1 Satz)
    - Überprüfen, ob Klassenname präzise und zur Verantwortlichkeit passend (Nomen Einzahl)
  - Bei jeder Erweiterung/Anpassung einer Klasse
    - Überprüfen ob Änderung kompatibel mit der Verantwortlichkeit der Klasse ist (High Cohesion)
    - Ob Verantwortlichkeit angepasst werden muss
    - Überprüfen, ob Name noch passt