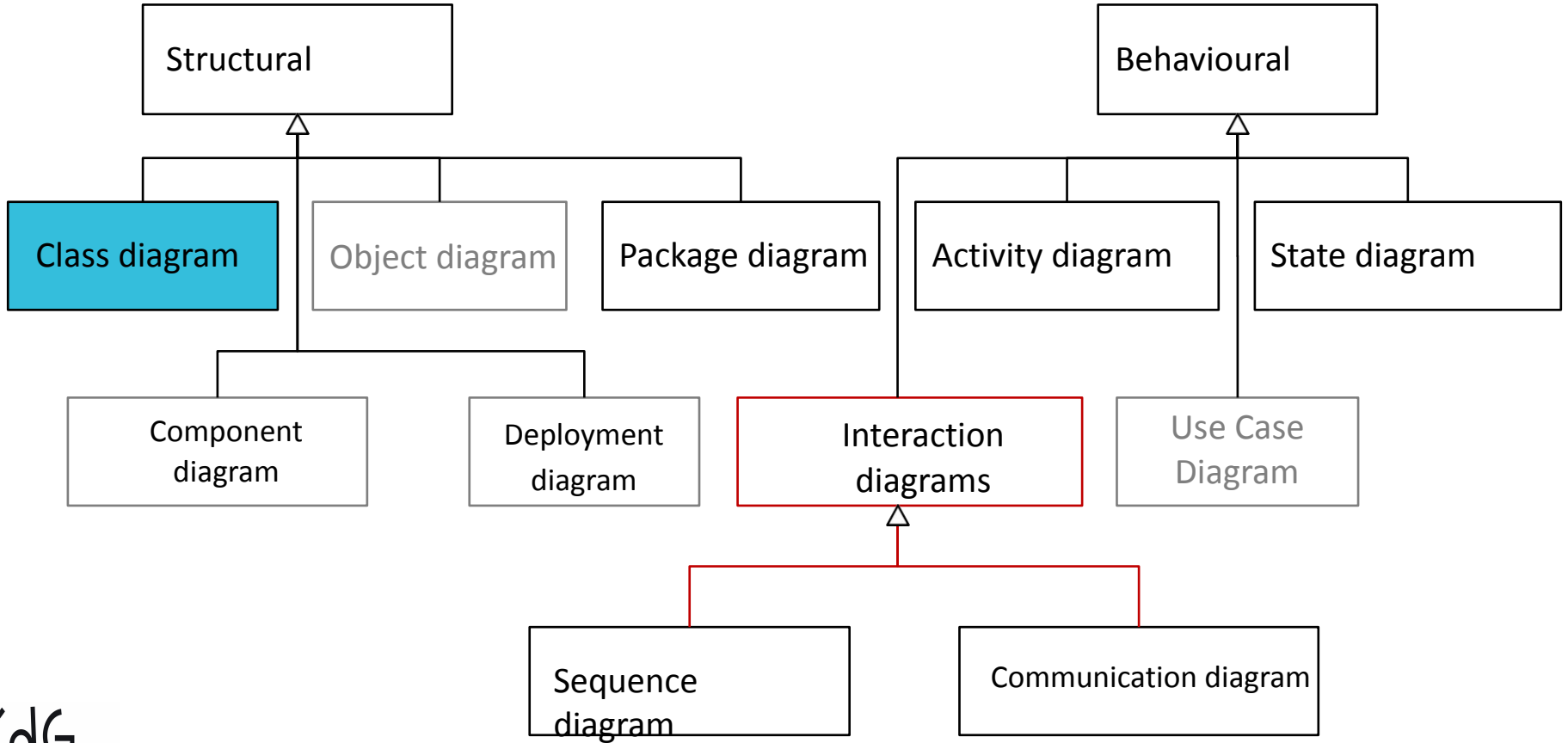
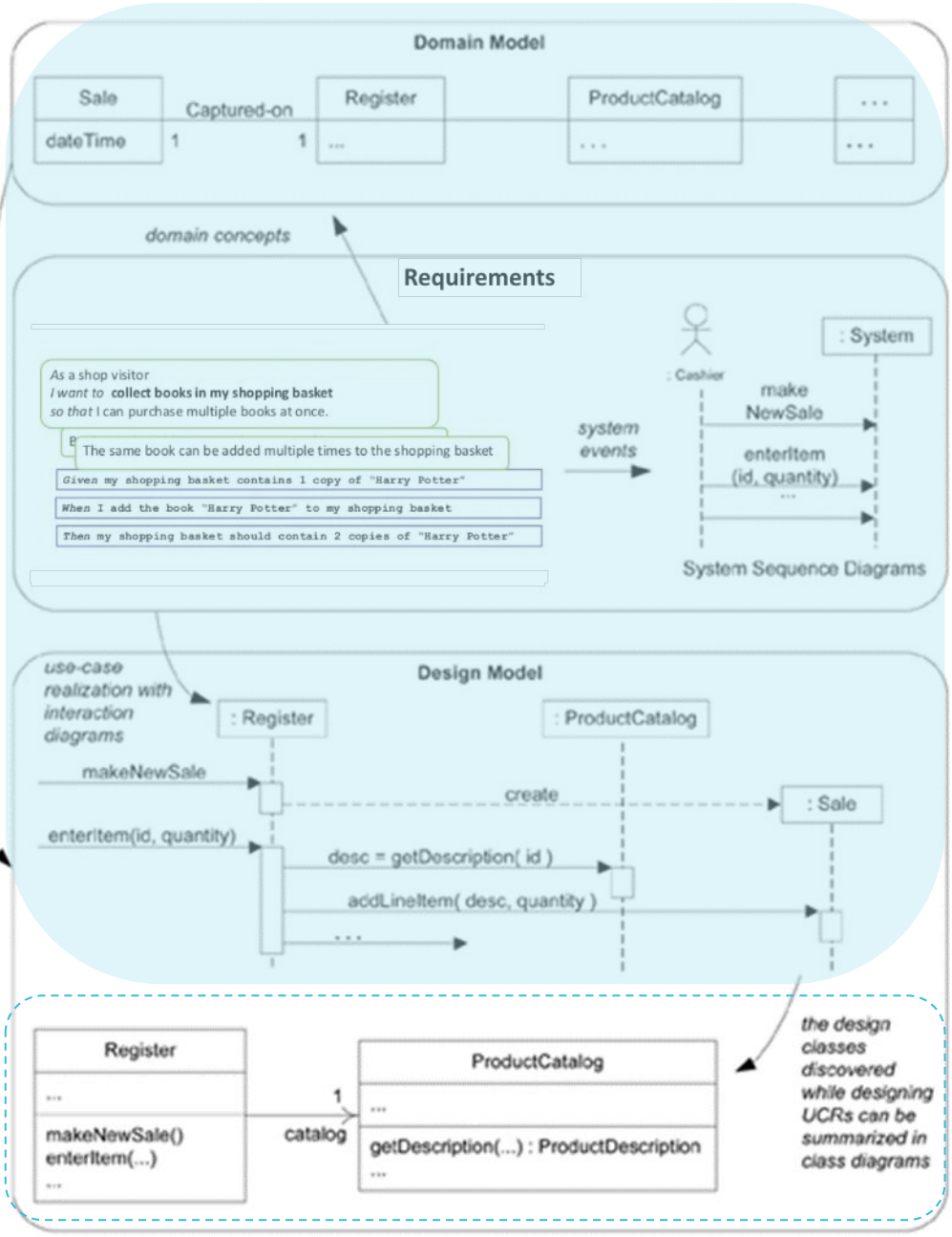
Design Class Diagram

Programming 2.2

KdG
University of Applied Sciences and Arts

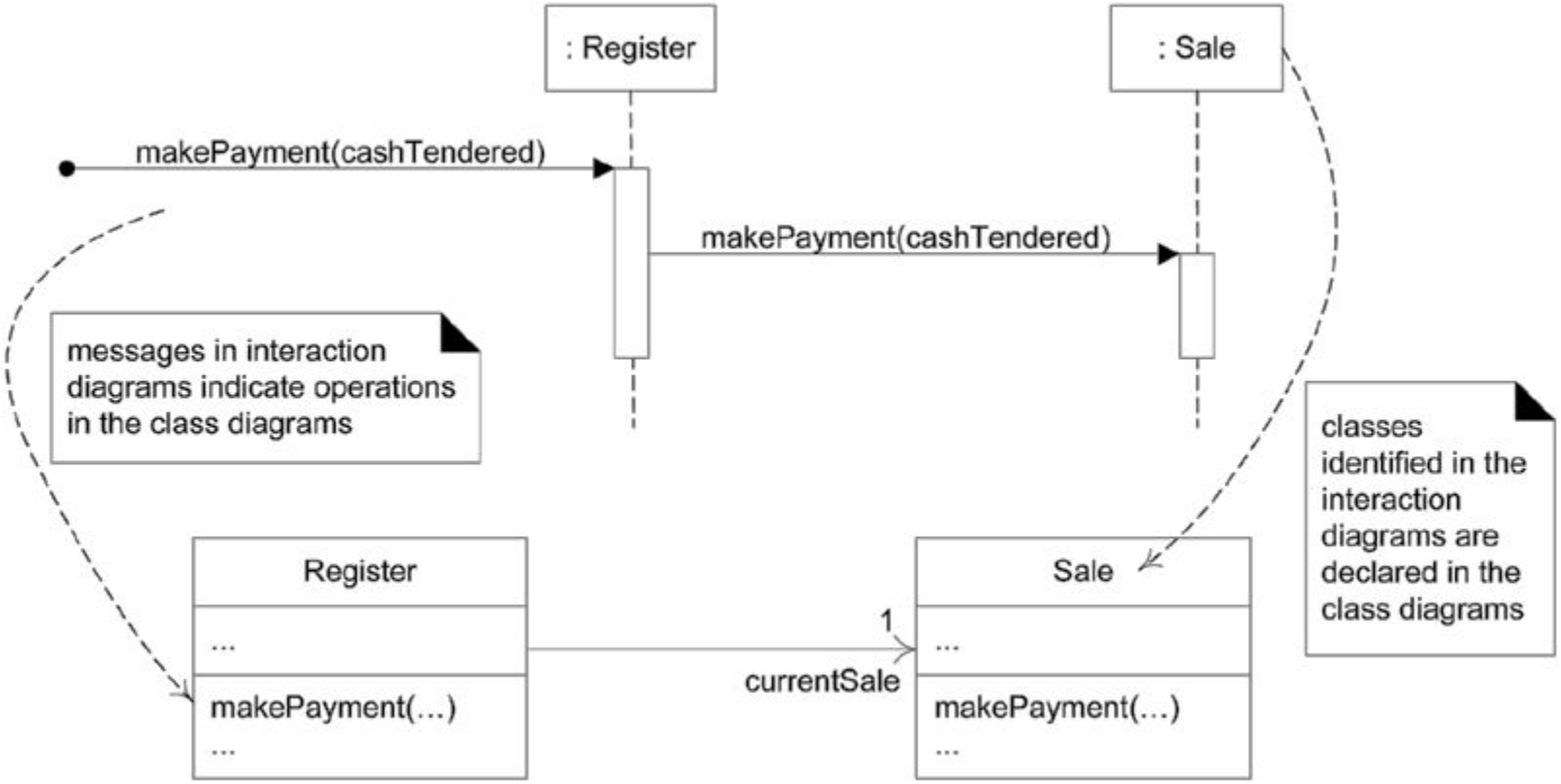
UML diagrams




Domain Model
Sale
dateTime
Captured-on
1
1
Register
...
ProductCatalog
...
...
...
domain concepts
Requirements
As a shop visitor
I want to collect books in my shopping basket
so that I can purchase multiple books at once.
The same book can be added multiple times to the shopping basket
Given my shopping basket contains 1 copy of "Harry Potter"
When I add the book "Harry Potter" to my shopping basket
Then my shopping basket should contain 2 copies of "Harry Potter"
system events
: Cashier
: System
make NewSale
enterItem (id, quantity)
...
System Sequence Diagrams
conceptual classes in the domain inspire the names of some software classes in the design
use-case realization with interaction diagrams
Design Model
: Register
: ProductCatalog
: Sale
makeNewSale
create
enterItem(id, quantity)
desc = getDescription(id)
addLineItem(desc, quantity)
...
Register
...
makeNewSale()
enterItem(...)
...
1
catalog
ProductCatalog
...
getDescription(...) : ProductDescription
...
the design classes discovered while designing UCRs can be summarized in class diagrams

Relation with interaction diagrams

- Interaction diagrams and design class diagram are typically designed simultaneously

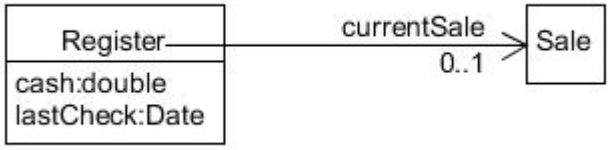


Evolve the domain model

- The design class diagram is an evolved version of the domain model
  - Add design/implementation detail
  - Stick with domain concepts (ubiquitous language)

Design associations

- Role name implies navigability
- Role name = name of the attribute that is used to implement navigability




```
public class Register {
  private double cash;
  private Date lastCheck;
  private Sale currentSale;
}

public class Sale {
}
```

```
public class Register {
  private double cash;
  private Date lastCheck;
}

public class Sale {
  private Register kassa;
}
```

Design associations: bidirectional

- Implement this association

Design associations: bidirectional

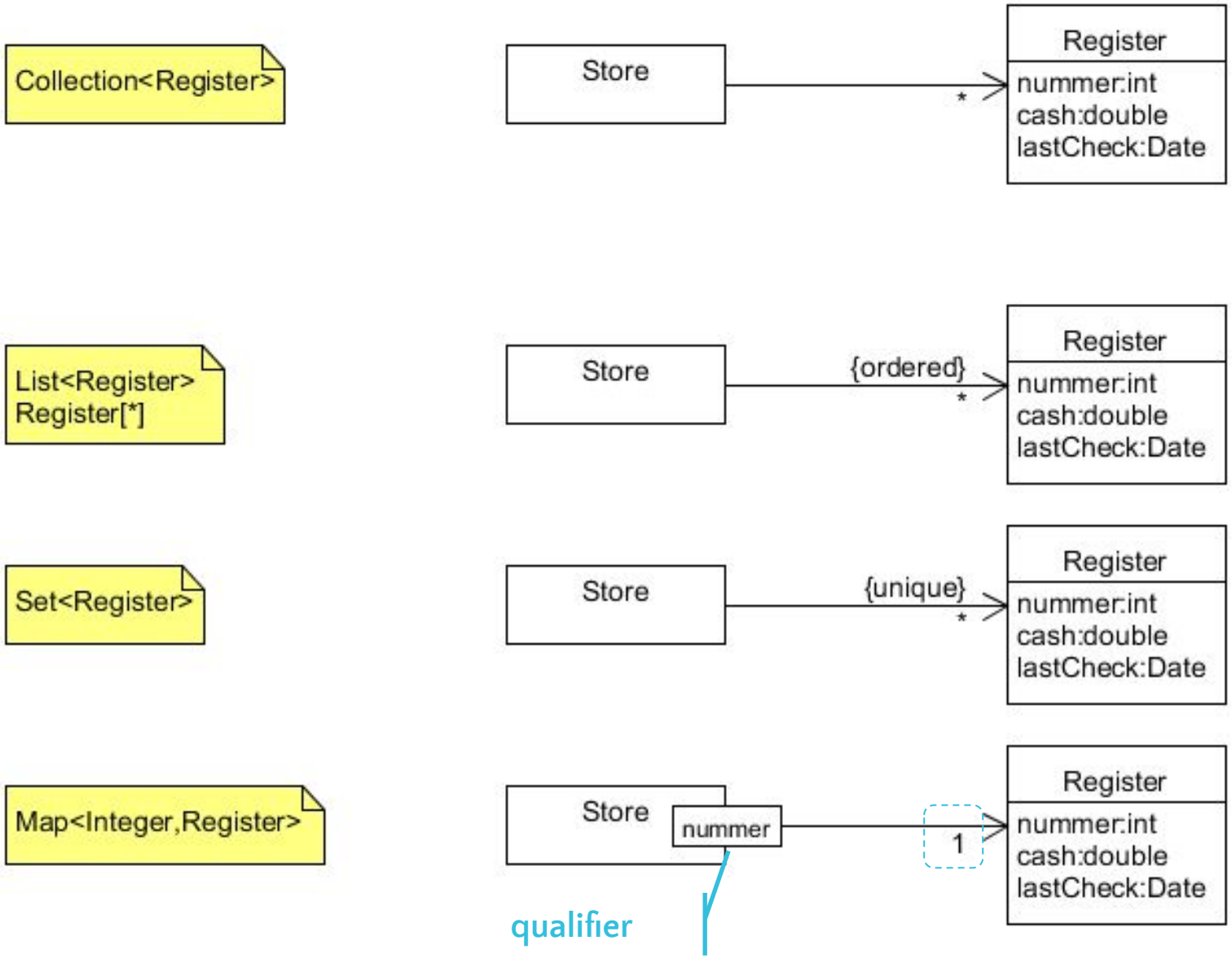

```
public class Register {
   private double cash;
   private Date lastCheck;
   private Sale sale;
}

public class Sale {
   private Register register;
}
```

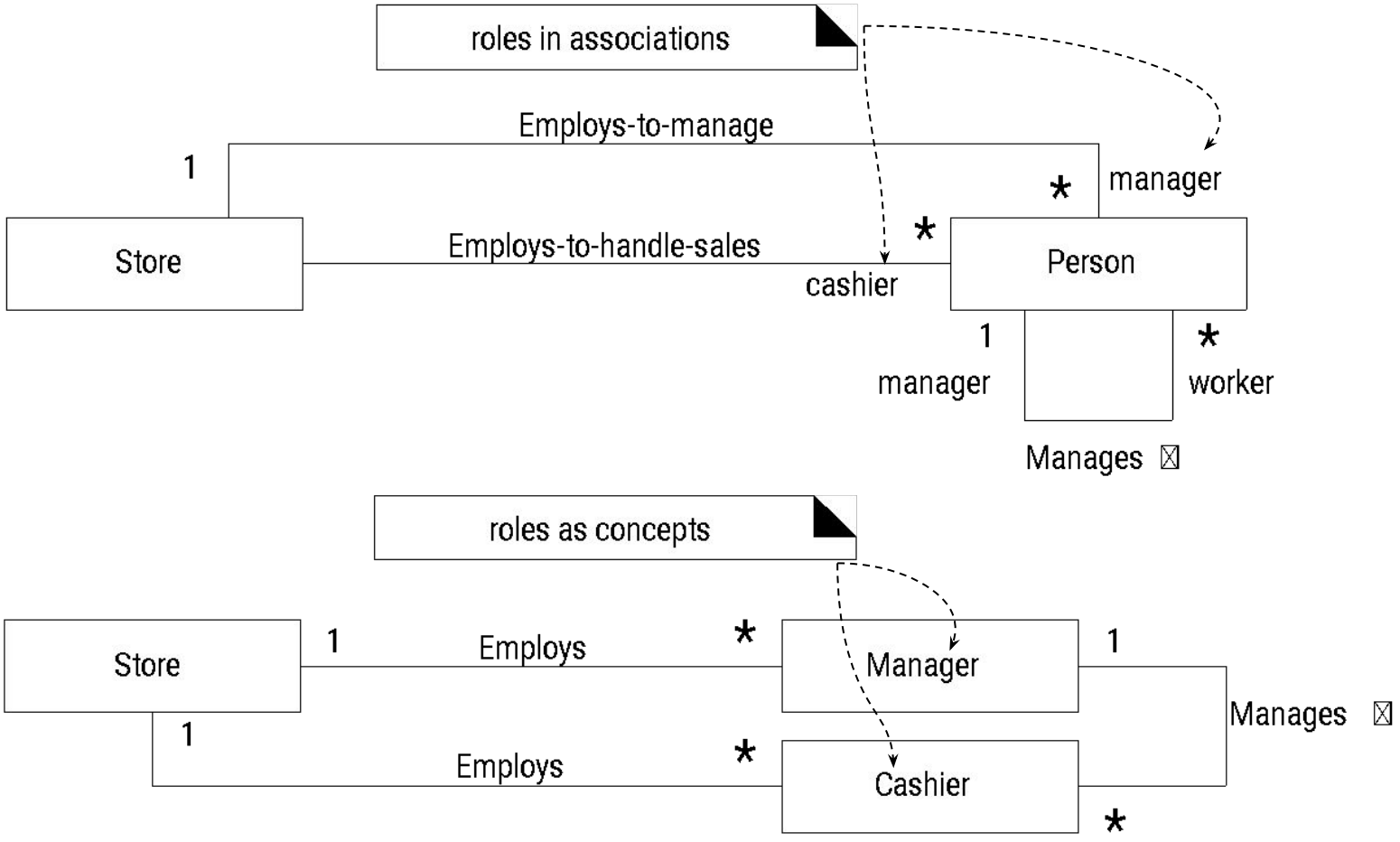
Associations: navigation design

- Try keeping associations simple. Is there complexity in the real world that can be avoided or is not used in the interaction diagrams?
  - Associations that can be removed?
  - to-many associations that can be reduced to to-one associaties?
  - bidirectionele associations that can be made unidirectional
  - Sometimes associations can be simplified in code by looking up data externally (database…) instead of having associations in memory
  - …

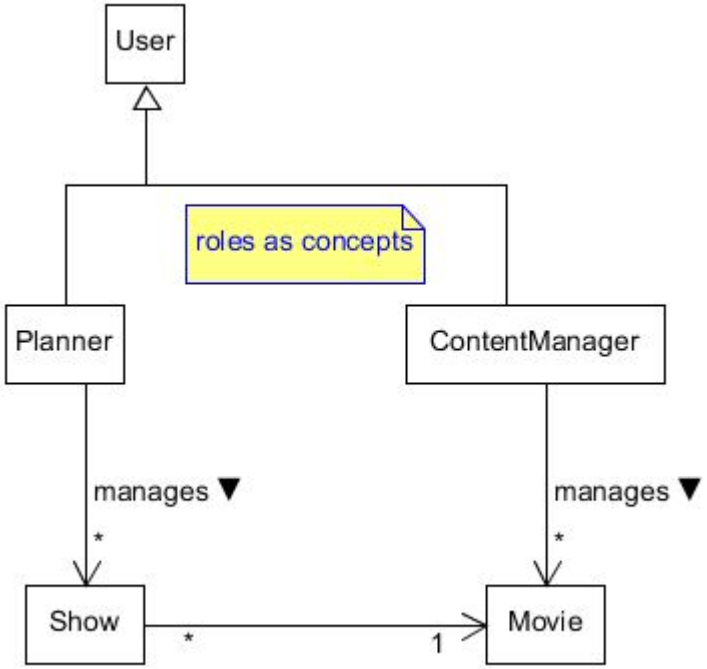
To Many associations



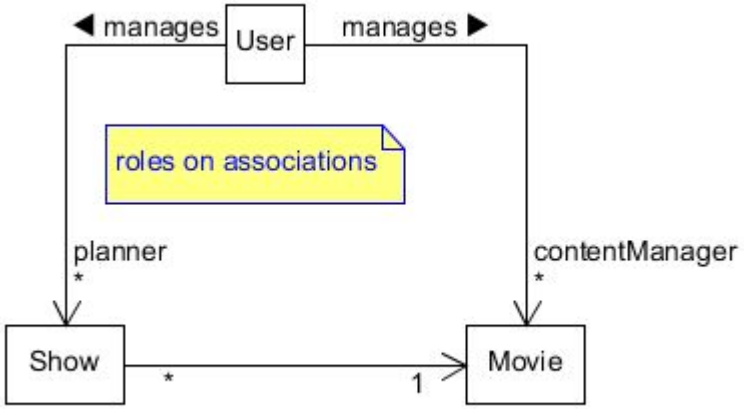
Roles: associations or concepts?



Roles : associations or concepts?



Classification of concepts is static: a user can never be a Planner and ContentManager at the same time.

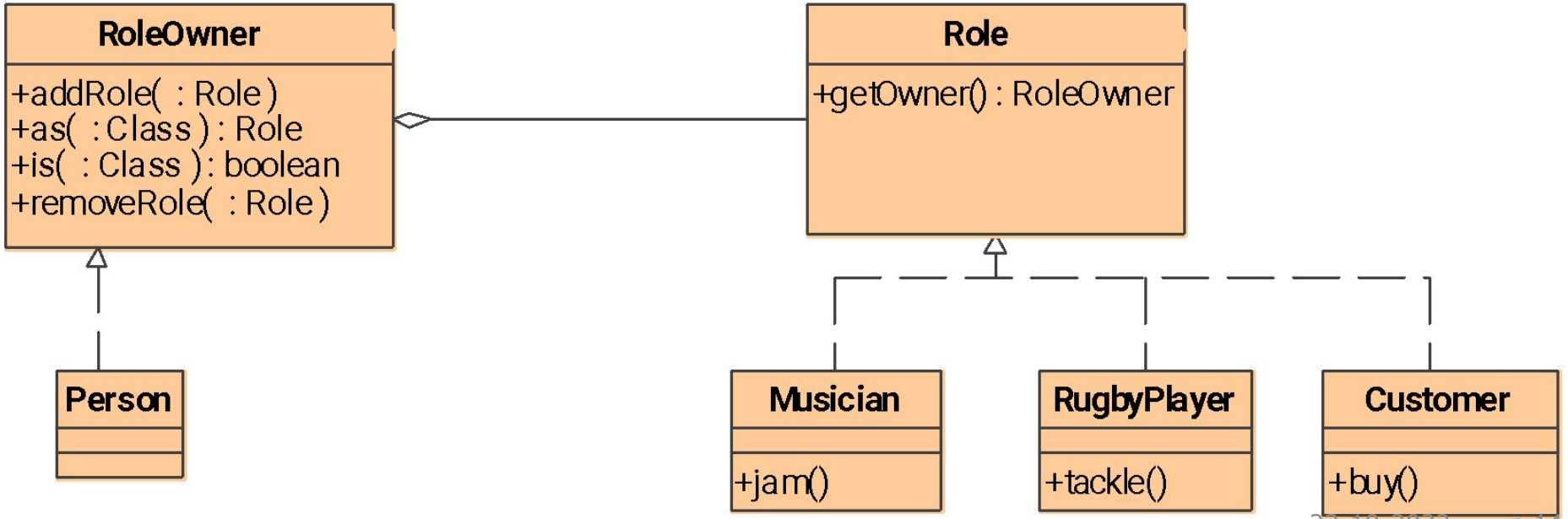


Roles : associations or concepts?

- **Roles on associations**
  - A user can have multiple roles
  - A user can change roles
  - simpler
- **Roles as concepts (separate classes)**
  - Roles can have separate attributes, associations, behaviour and additional semantics
  - complexer

The role design pattern

- Combination of association (RoleOwner -> Role) and concept (Role hierarchy)
  - Combines flexibility of both approaches (highest flexibiliteit)
  - Combines complexity of both approaches (highest complexity)



Roles: which design is the best™?

- The simplest design that meets the requirements is the best
  - Use rolls on associations except when roles have different attributes methods…
  - In the latter case, use roles as classes
  - in the uncommon case where a class can have multiple roles or can change roles, use the role design pattern

Design Class diagram conventions

- No need to specify (defaults)
  - + (public) method accessibility
  - - (private) attribute accessibility
  - simple attribute getter/setter methods
  - constructor without parameters
  - abstract methods overridden in concrete class

Encapsulation

- When adapting a class, you also need to adapt code that uses this class
- Limit the code that can be used by another class
  - Make attributes by default private (AKA data hiding)
  - Provide access through methods, limit the interface
    - Design which methods are needed by other classes. This is called the class interface.
    - Make all other methods private
    - Any change that does not modify the interface remains local to the class
    - The class is encapsulated in the interface