Design Modeling: Design Class Diagram

Instructor: Mehroze Khan

Object-Oriented Design Processes and Models

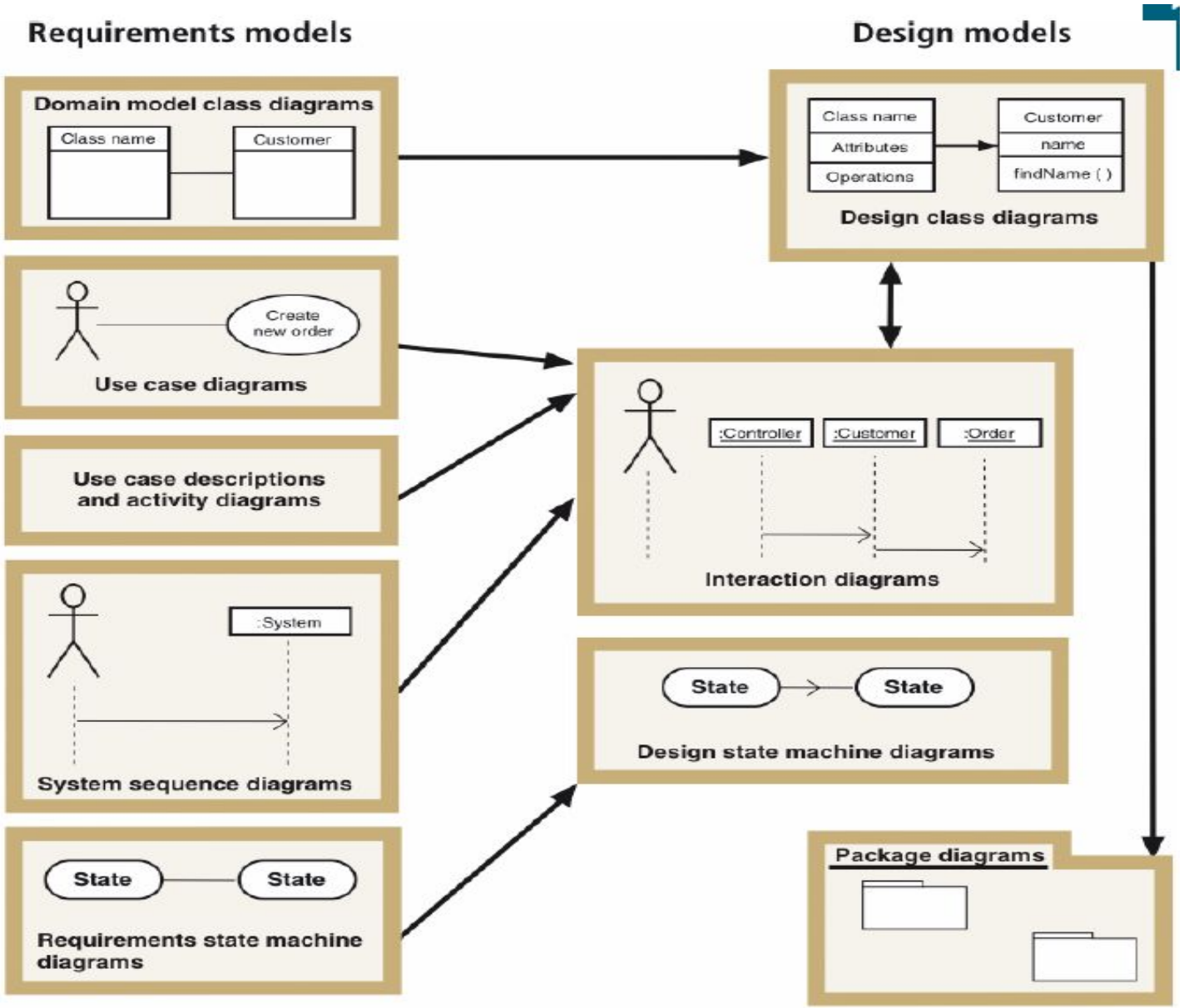
Diagrams developed for analysis/requirements

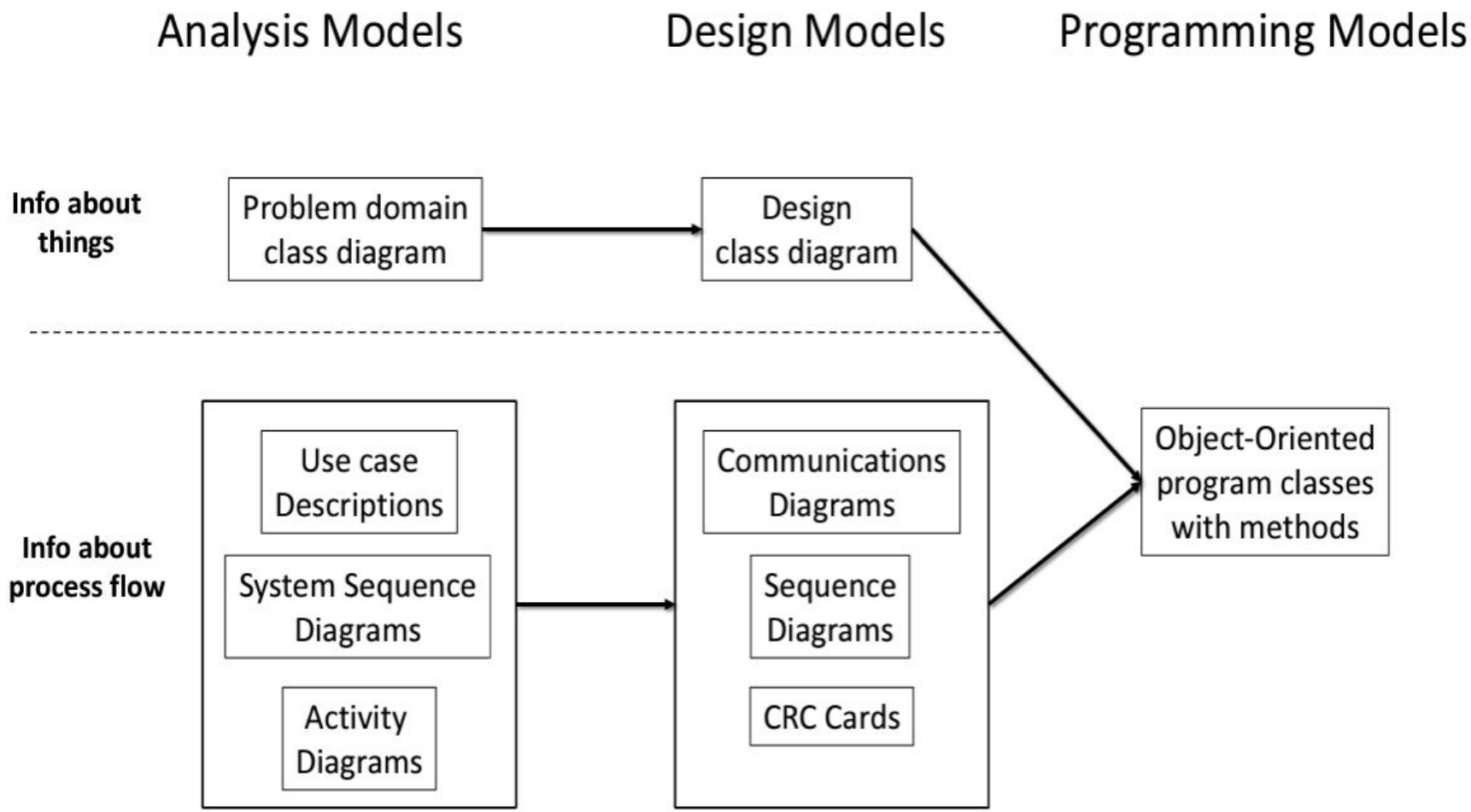
- Use case diagrams and Use case descriptions
- Activity diagrams
- Domain model class diagrams
- System sequence diagrams

Diagrams developed for design

- Interaction diagrams and package diagrams
- Design class diagrams – include object-oriented classes, navigation between classes, attribute names, method names, and properties needed for programming

Design Models with Their Respective Input Models

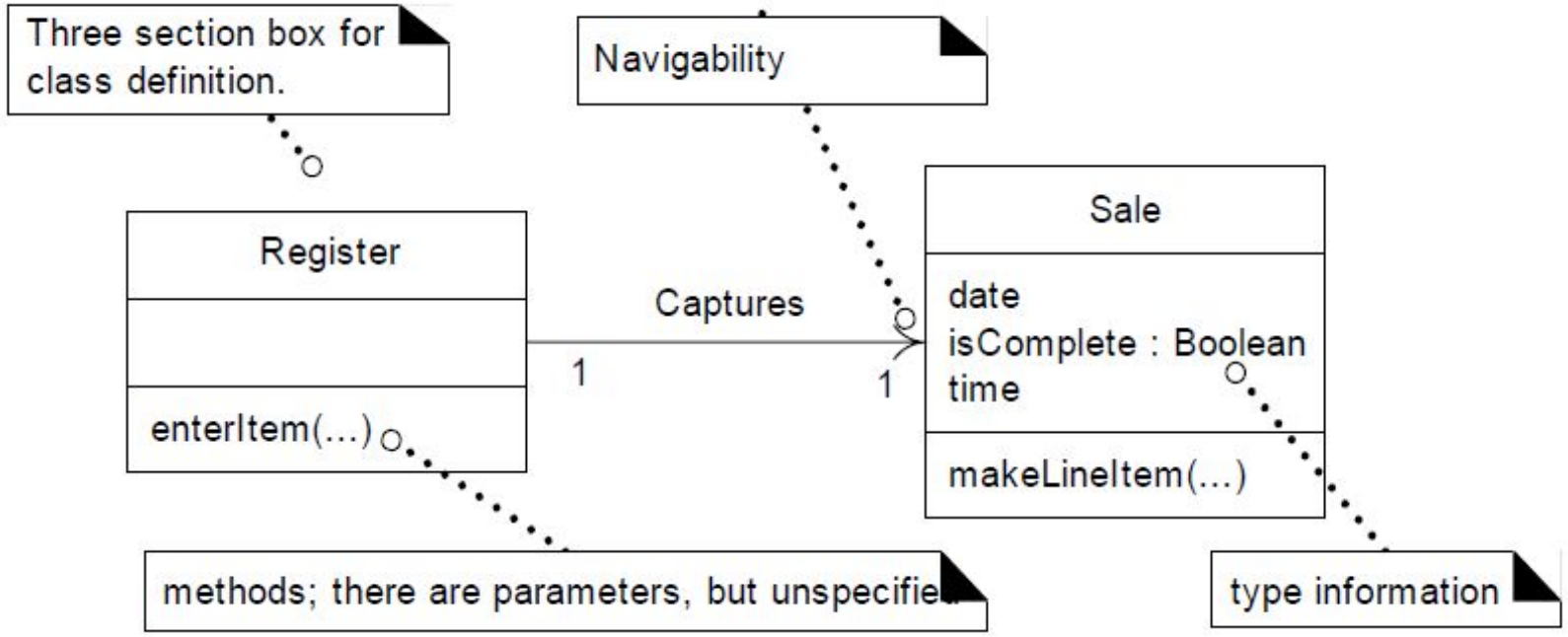



Analysis Models
Design Models
Programming Models
Info about things
Problem domain class diagram
Design class diagram
Info about process flow
Use case Descriptions
System Sequence Diagrams
Activity Diagrams
Communications Diagrams
Sequence Diagrams
CRC Cards
Object-Oriented program classes with methods

Design Class Diagram

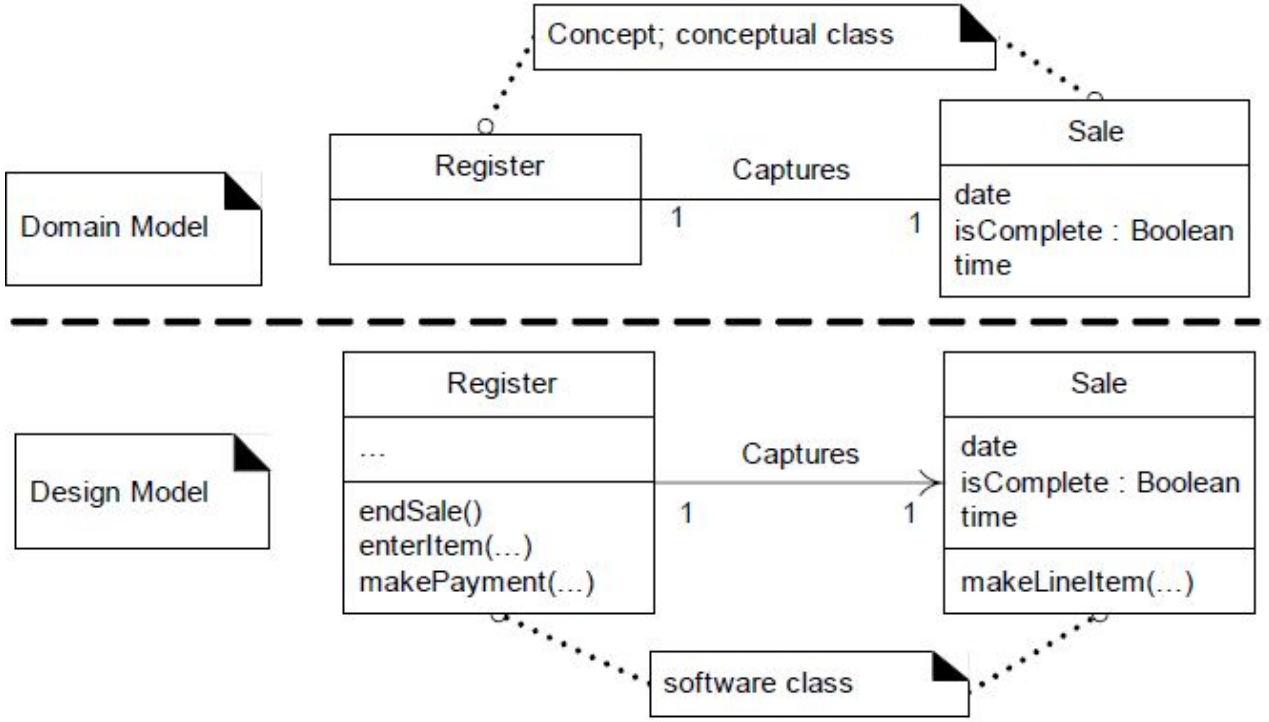
In addition to basic associations and attributes, the diagram is extended to illustrate

- Methods of each class
- Attribute type information
- Attribute visibility
- Navigation between objects



Domain Model Vs Design Model Classes

- In Domain Model, a ***Sale*** does not represent a software definition; rather, it is an **abstraction** of a real-world concept about which we are interested in making a statement.
- By contrast, DCDs express—for the software application—the definition of classes as software components. In these diagrams, a Sale represents a **software class**



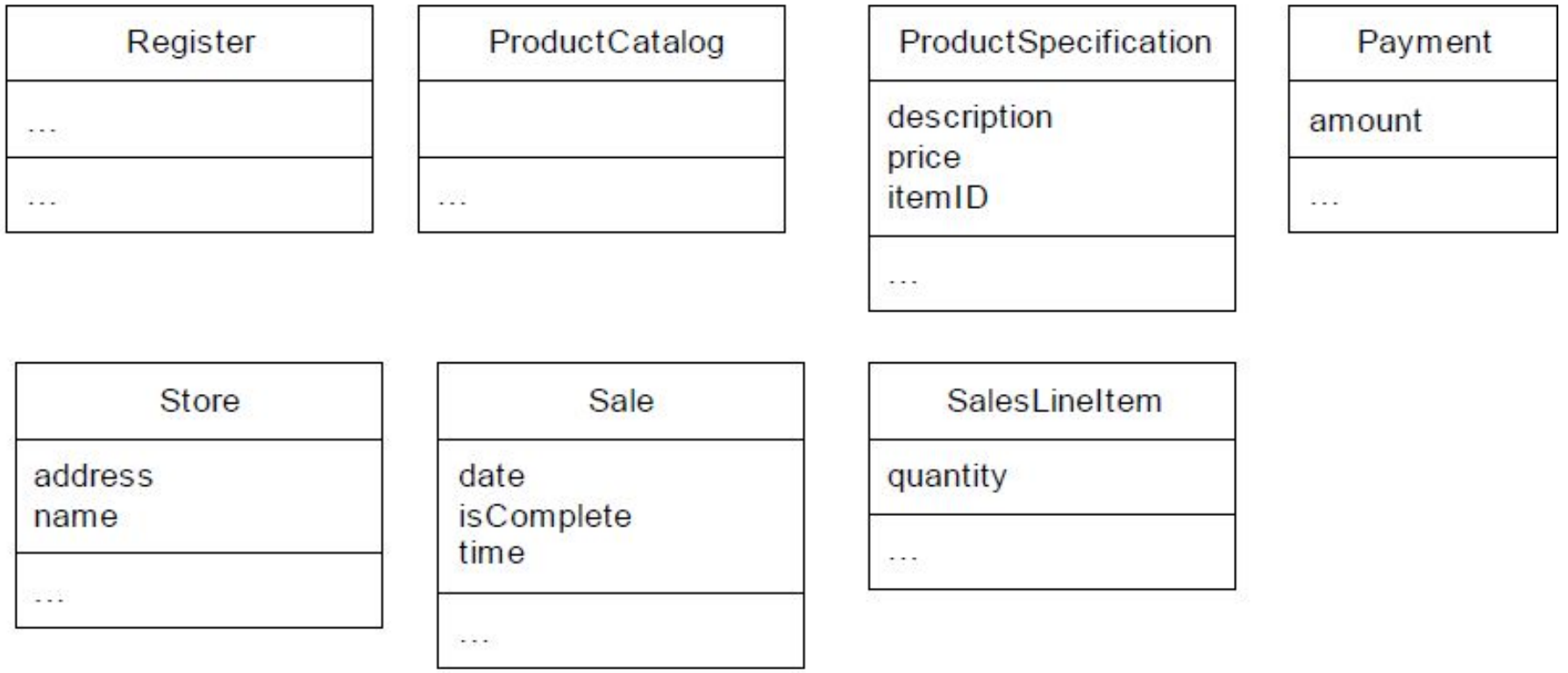
Creating DCD

Identify Software Classes and Illustrate Them

- The first step in the creation of DCDs as part of the solution model is to **identify those classes that participate in the software solution**.
- These can be found by scanning all the **interaction diagrams** and **listing the classes mentioned**.
- For the POS application, these are some classes:
  - Register
  - ProductCatalog
  - Store Payment
  - Sale
  - ProductSpecification
  - SalesLineItem

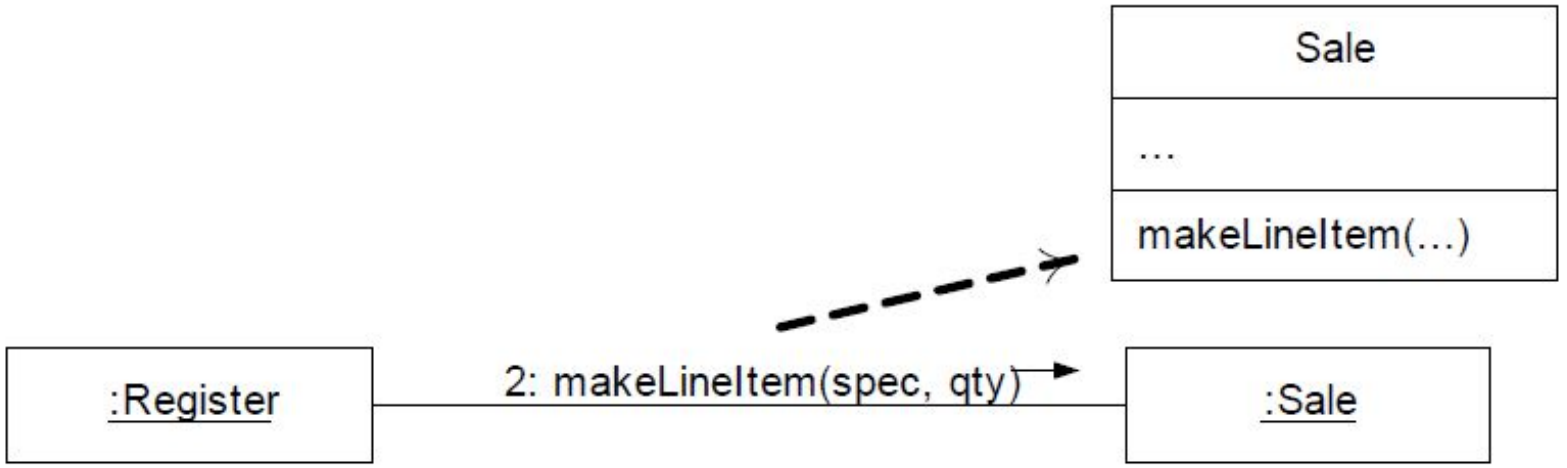
Draw Classes and Add Attributes

- The next step is to **draw a class diagram** for these classes and **include the attributes** previously identified in the Domain Model that are also used in the design.



Add Method Names

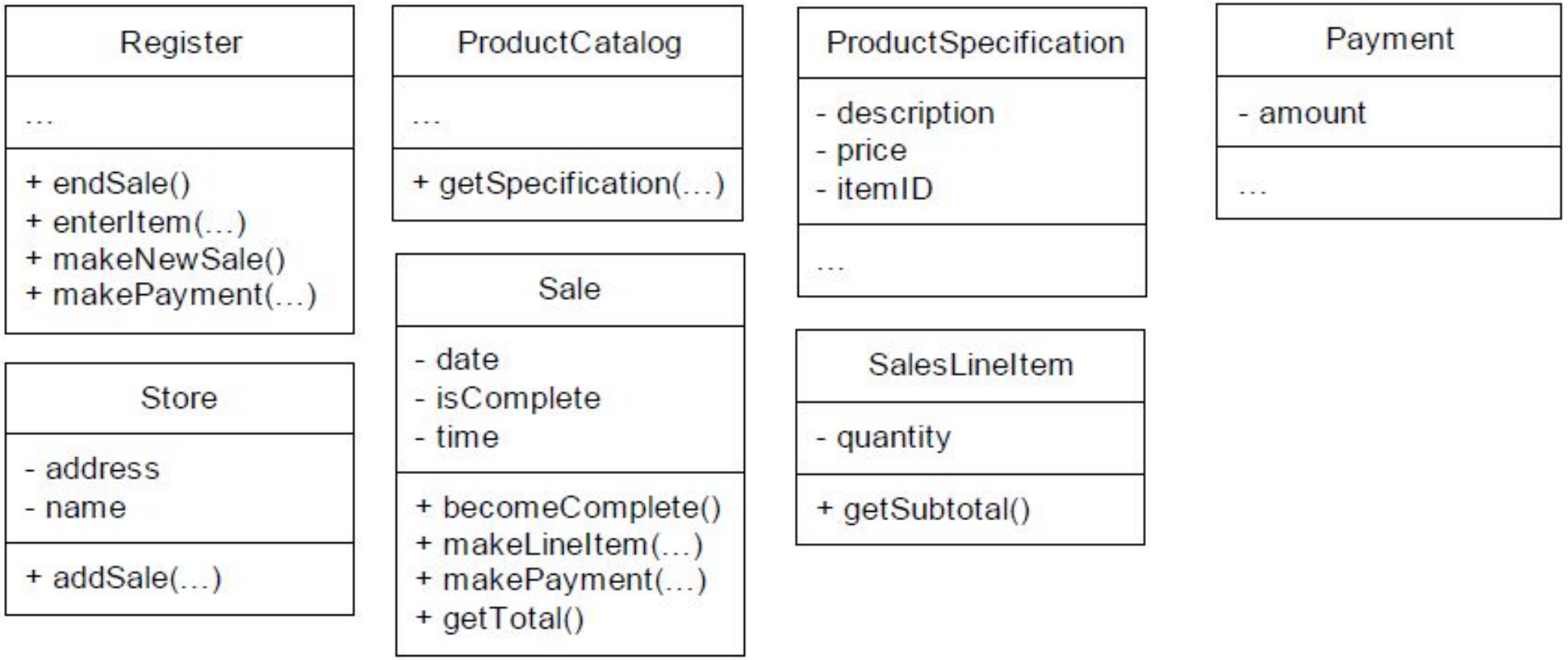
- The methods of each class can be identified by analyzing the interaction diagrams.
- For example, if the message ***makeLineItem*** is sent to an instance of class ***Sale***, then class Sale must define a makeLineItem method.
- In general, the set of all messages sent to a class X across all interaction diagrams indicates the majority of methods that class X must define.



Visibility Symbols

Notation	Visibility
+	Public
-	Private
#	Protected
~	Package

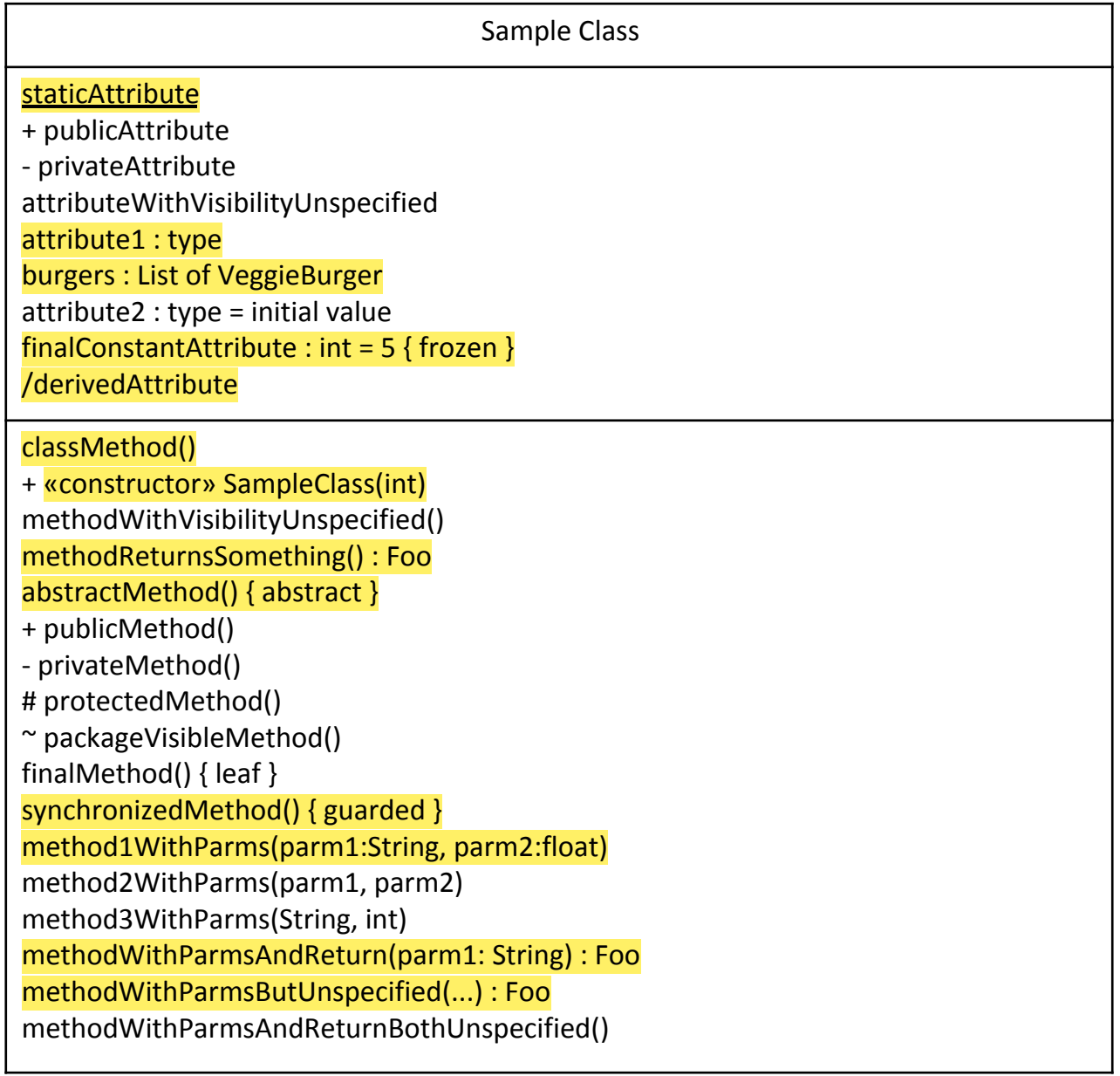
Method Names from Interaction Diagrams



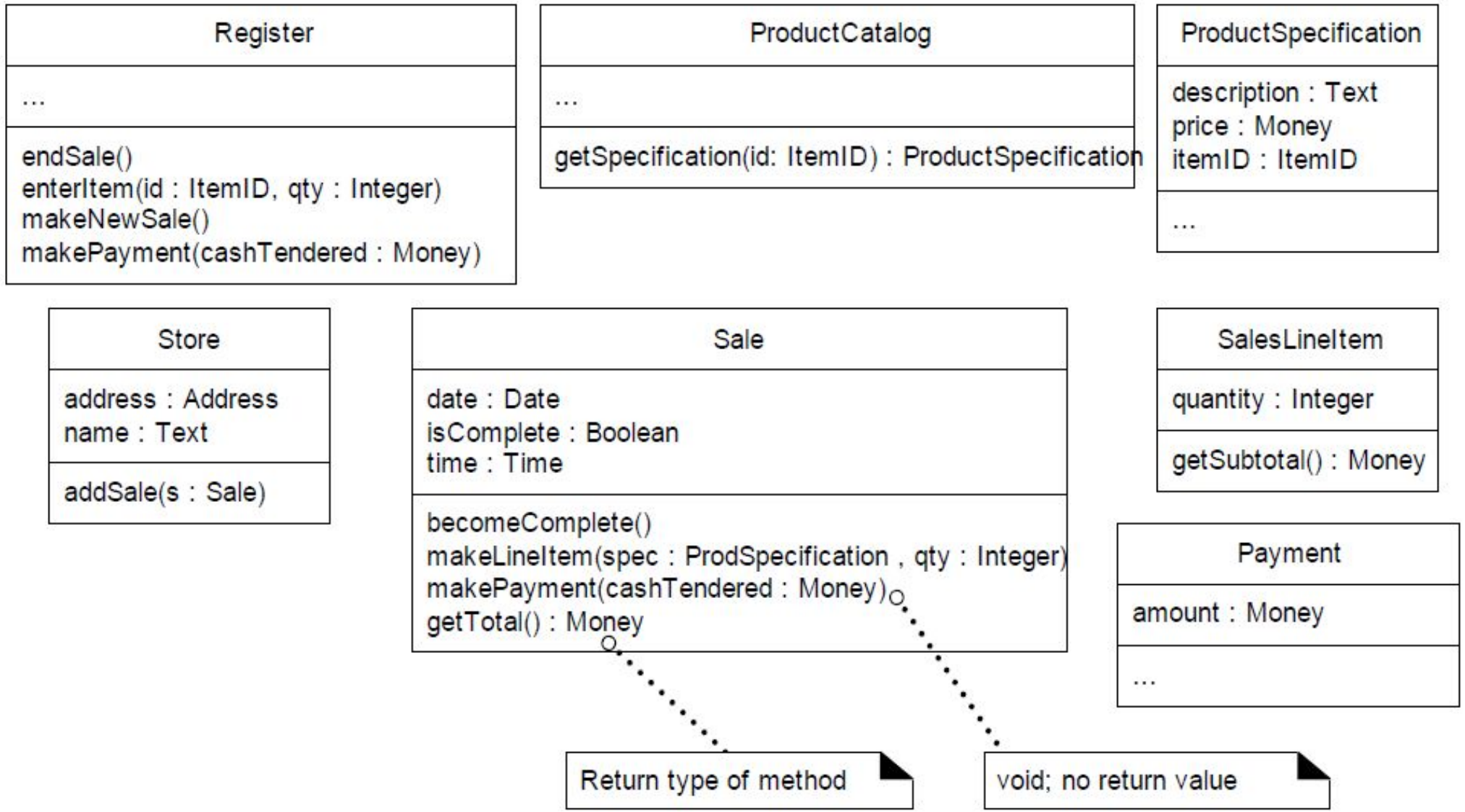
Add More Type Information

- The types of the ***attributes, method parameters, and method return values*** may all optionally be shown.
- The question as to whether to show this information or not should be considered in the following context:
- The DCD should be created by considering the audience.
  - If it is being created in a CASE tool with automatic code generation, full and exhaustive details are necessary.
  - If it is being created for software developers to read, exhaustive low-level detail may adversely affect the noise-to-value ratio.

Notation for Member Details



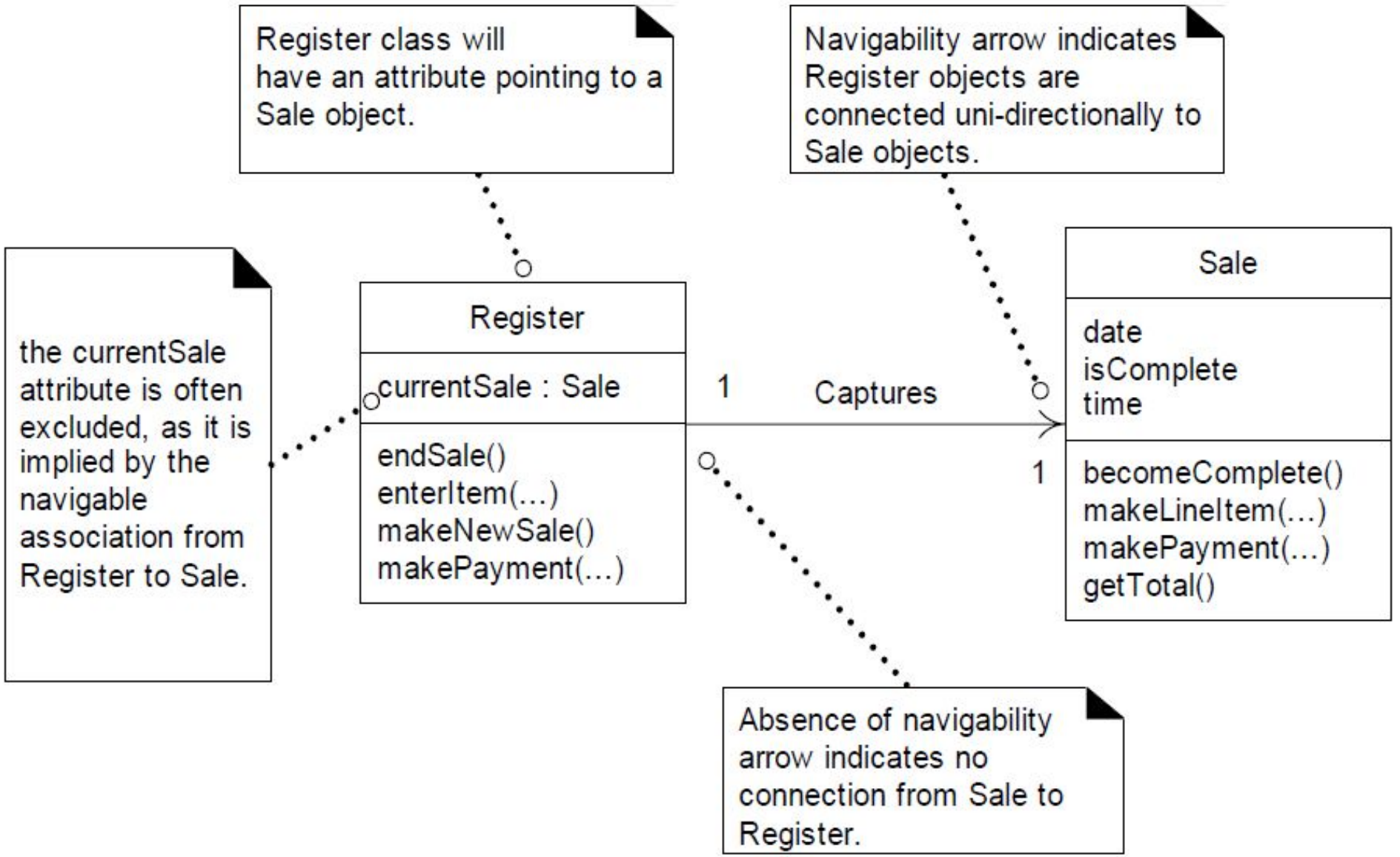
Add More Type Information



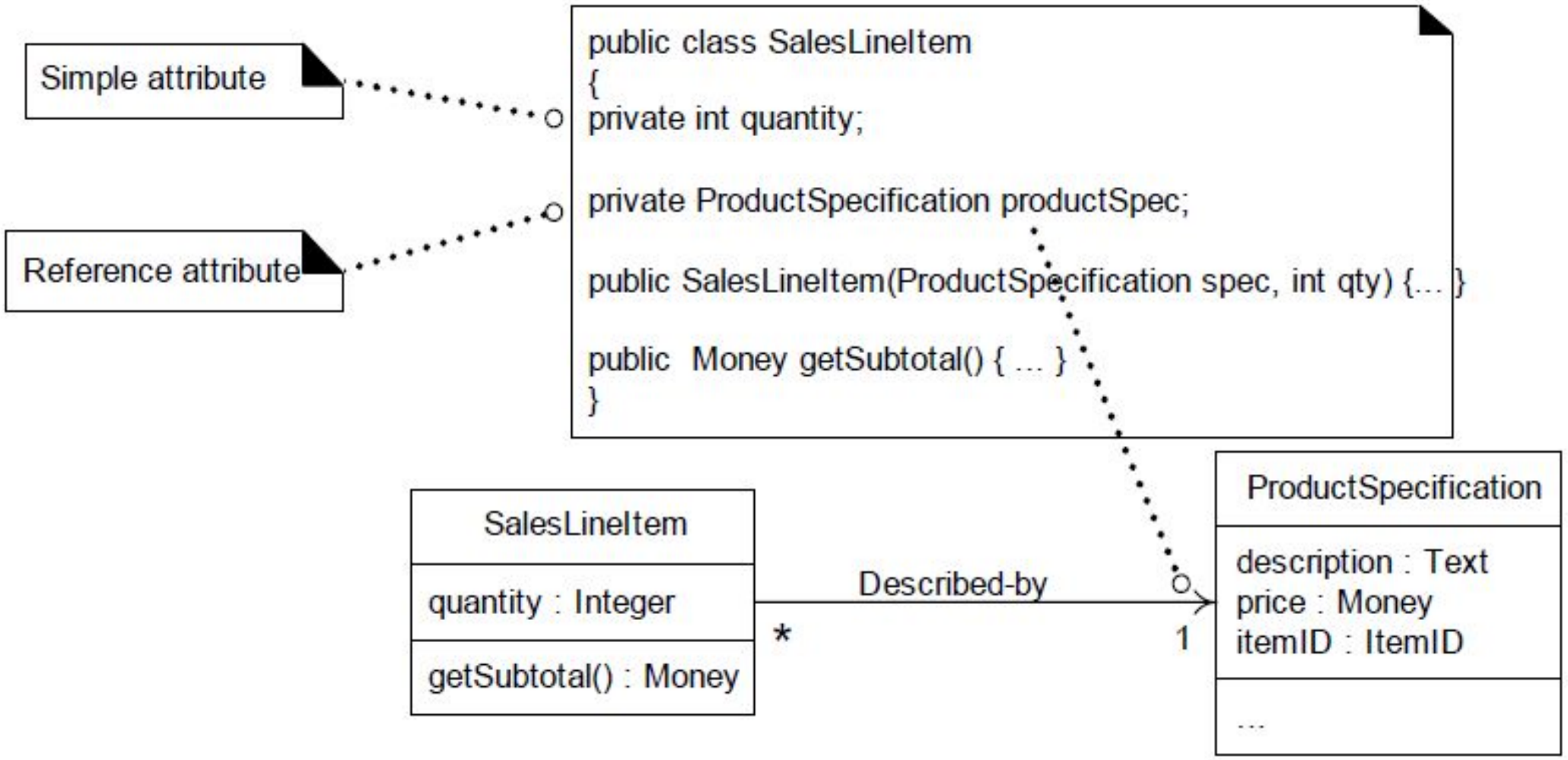
Adding Associations and Navigability

- Each end of an association is called a ***role***, and in the DCDs the role may be decorated with a ***navigability arrow***.
- Navigability is a property of the role that indicates that it is possible to ***navigate uni-directionally*** across the association from objects of the source to target class.
- Navigability implies visibility—usually attribute visibility.

Adding Associations and Navigability



Adding Reference Attributes



Adding Associations and Navigability

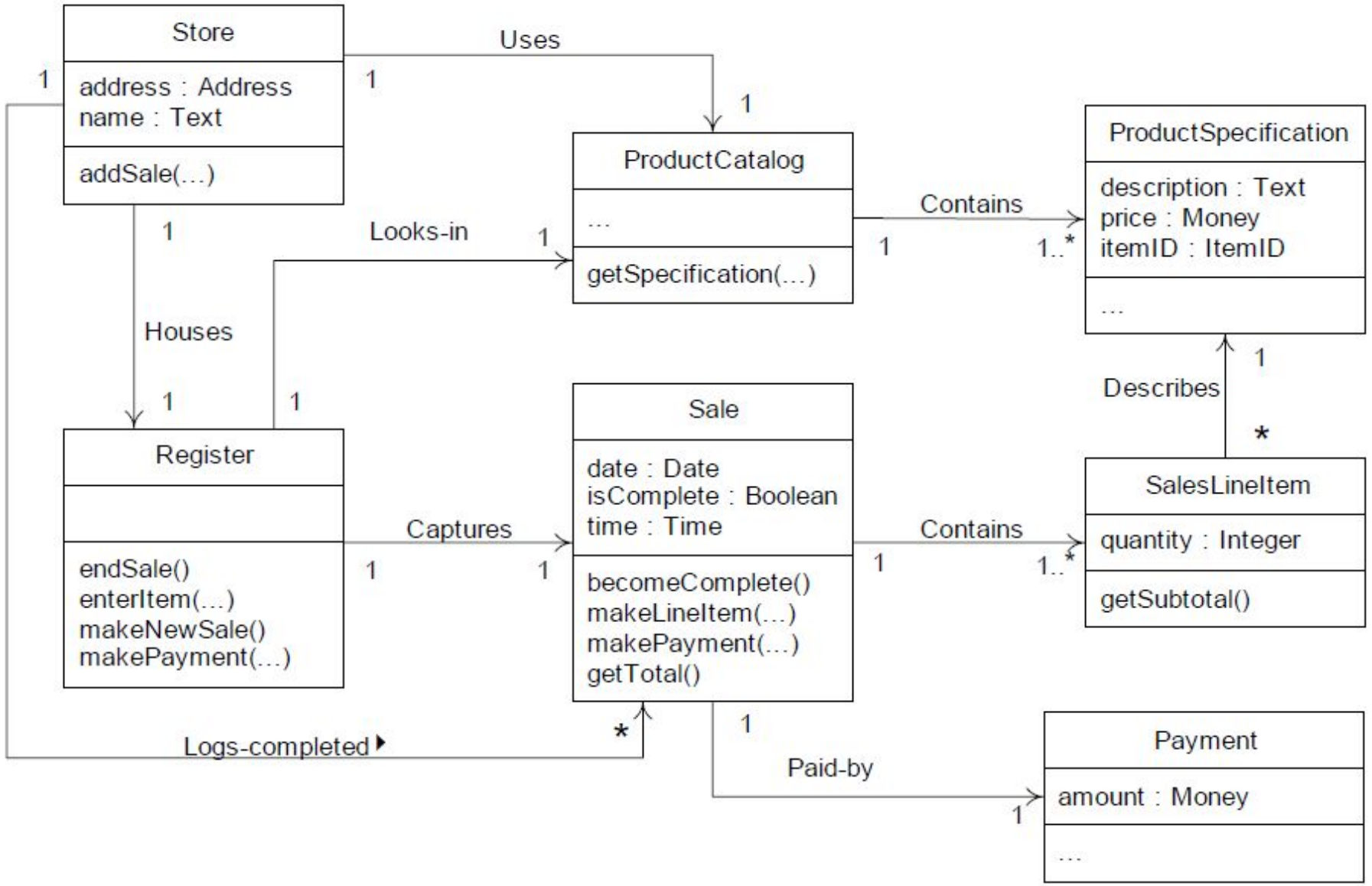
- The usual interpretation of an association with a navigability arrow is ***attribute visibility*** from the ***source*** to ***target*** class.
- During implementation in an object-oriented programming language, it is usually translated as the source class having an attribute that refers to an instance of the target class.
- For instance, the **Register class** will define an **attribute** that **references a Sale instance**.
- Most, if not all, associations in DCDs should be adorned with the **necessary navigability arrows.**

Adding Associations and Navigability

Identify:

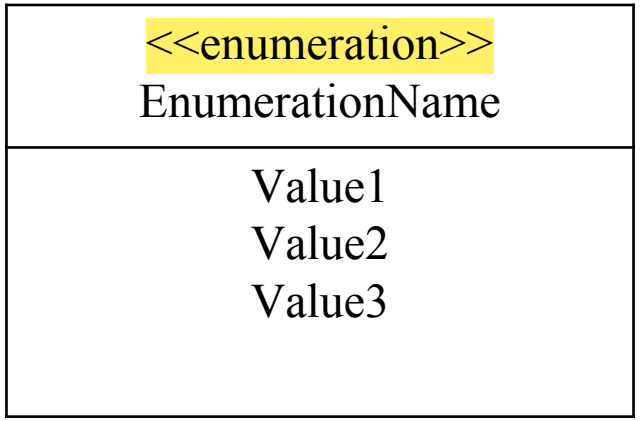
- 1 to many relationships
  - If it is a **superior/subordinate relationship**, add visibility from superior to subordinate
  - Ex. From Sale to SaleItem
- For **Mandatory associations** where one can't live without the other
  - Add navigation from independent to dependent:
  - Ex. From Customer to Sale

DCD for NextGen POS



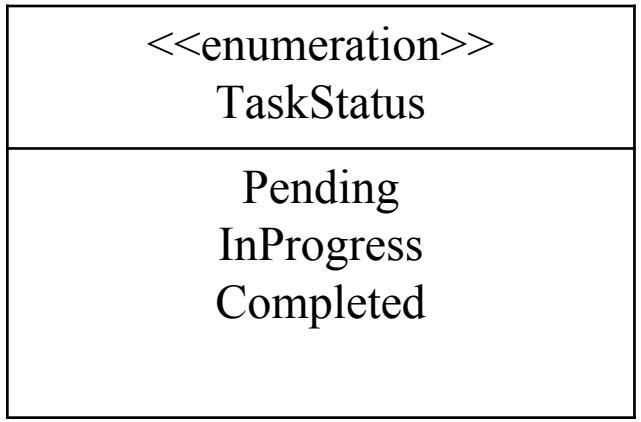
Enumeration Class

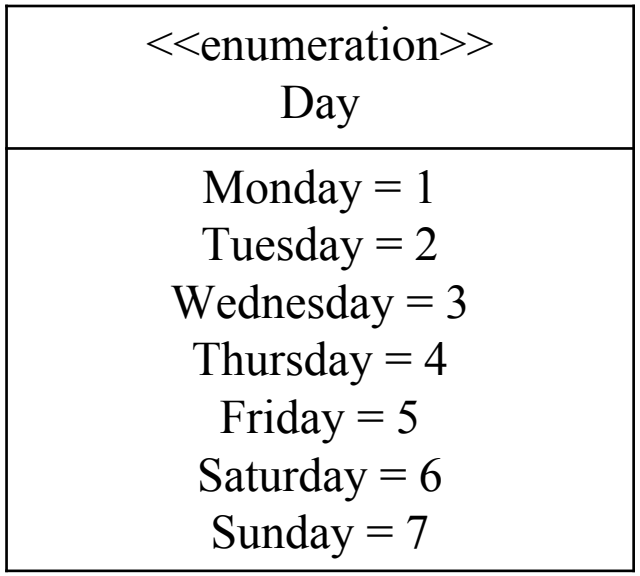
- An ***enum class (or enumeration class)*** is a special type of class in object-oriented programming that defines a ***group of named constant values***.
- Enum class allows you to ***define your own type*** that can only have certain values, like a list of options.



Enumeration Class

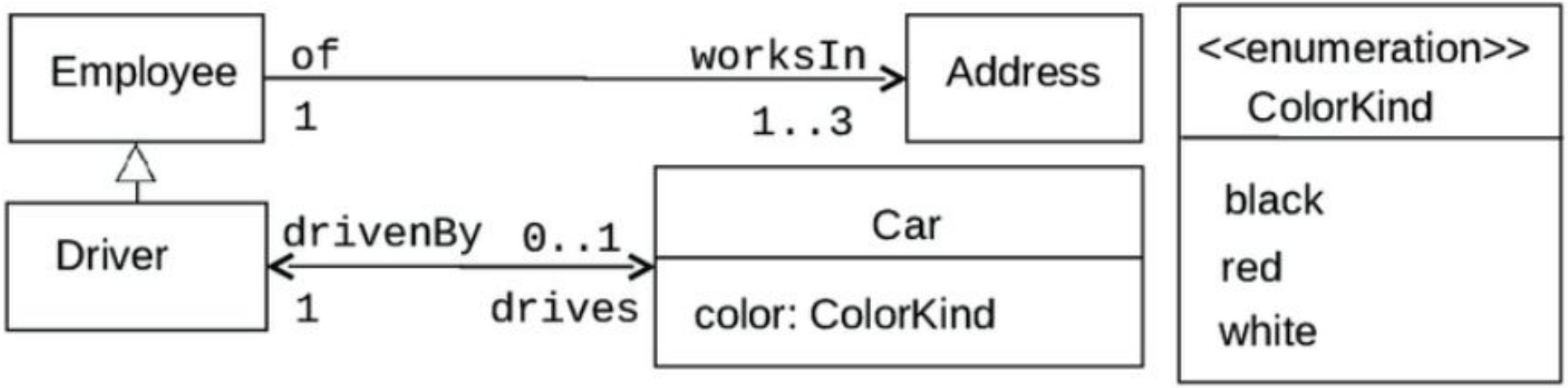
- An ***enum class (or enumeration class)*** is a special type of class in object-oriented programming that defines a ***group of named constant values***.
- Enum class allows you to ***define your own type*** that can only have certain values, like a list of options.





Enumeration Class

- An ***enum class (or enumeration class)*** is a special type of class in object-oriented programming that defines a ***group of named constant values***.
- Enum class allows you to ***define your own type*** that can only have certain values, like a list of options.



References

- Satzinger, John W., Robert B. Jackson, and Stephen D. Burd. Systems analysis and design in a changing world. 4th Edition.
- Larman, Craig. Applying UML and patterns: an introduction to object oriented analysis and design and interative development. Pearson Education India, 2012.
- Object-Oriented Modeling and Design with UML, Michael R. Blaha and James R. Rumbaugh, 2nd Edition, Pearson, 2005.