2 *Unit*

STATIC UML DIAGRAMS

2.1 CLASS DIAGRAMS

The Essentials

A class diagram describes the types of objects in the system and the various kinds of static relationship that exist among them.

There are 2 kinds of static relationships.

- Associations
- Subtypes

The class diagrams show the attributes and operations of a class and the constraints that apply to the way objects are connected.

Perspectives

There are three different perspectives; we use in drawing class diagrams. They are

- Conceptual
- Specification
- Implementation

We take the concepts in the domain under study and represent in diagrams, for the conceptual perspective.

These concepts will relate to the classes that implement them by there us often no direct aping.

A conceptual model should be drawn with little or no regard to the software that implements it, so it is considered language independent.

(a) Conceptual

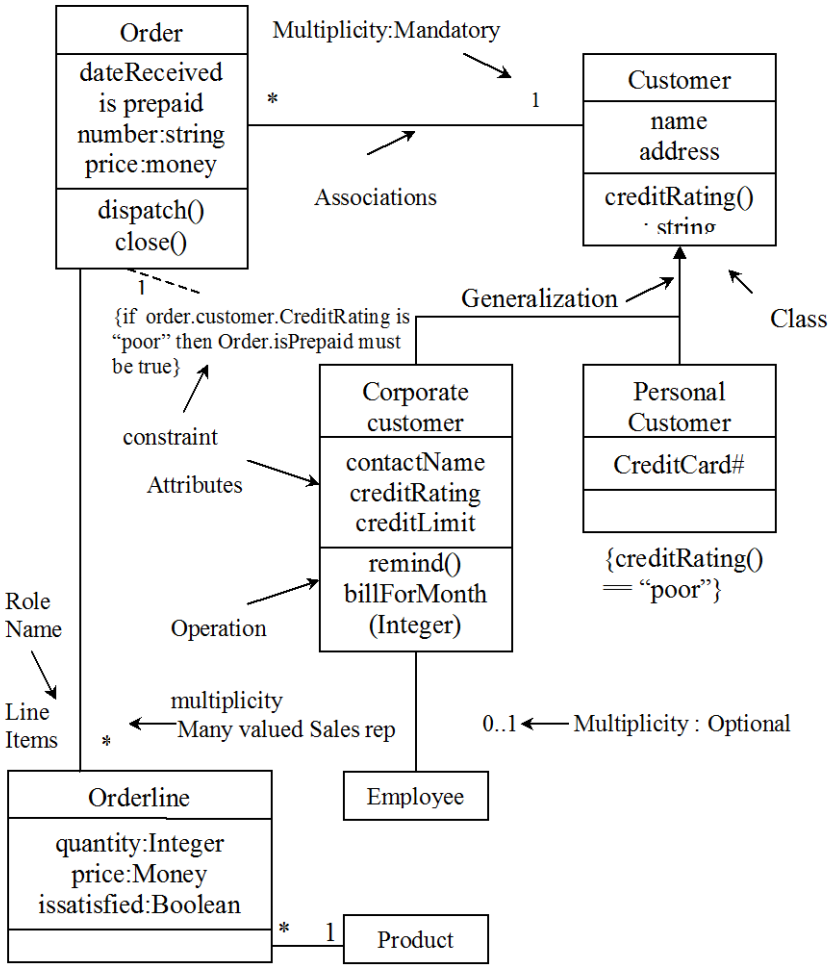


Figure 2.1 Class Diagram

(b) Specification

Here we look at the interface of the software but not its implementation.

The word type is used to talk about an interface of a class.

A type can have many classes that implement it and a classes that implement it an a class implement many types.

(c) Implementation

Here we have classes and we lat the implementation base.

This is the most often used perspective but un many cases the specification perspective is better than implementation.

Classes are marked with << implementation class >> to show the implementation perspective and with <<type>> for the specification and conceptual perspective.

2.2 ELABORATION

Elaboration has the following series of iterations.

1) The core (risky) software architecture is programmed and tested.
2) Majority of the requirements are discovered and stabilized.
3) Major rises are mitigated or retired.

Elaboration normally consists of two or three iterations.

During these iterations the team

- Does serious investigations
- Implements programs and tests for core architecture
- Clarifies requirements tackles high risk issues.

Each iteration is time boxed. i.e. its end date is fixed. It lasts for two to six weeks.

- During elaboration the models are fully developed in preparation for implementation.
- During elaboration, the code and design are in production. It is called "Executable architecture" or "Architectural baseline".

Elaboration is

- Building the core architecture
- Resolving the high risk elements
- Define more requirements and

Estimate overall schedule and resources

Domain model

- Visualization of domain concepts.
- It is similar to static information model.

Design model

It is a set of diagrams that describe logical design.

It includes

- Class diagrams
- Object interaction diagrams
- Package diagrams etc.

Software architecture documents

A summary of outstanding design ideas.

Summarizes key architecture issues and their resolution in design.

Data model

Includes database schemas, mapping strategies between object and non-object representations.

Use case storyboards, UI prototypes.

Description of user interface.

Paths of navigation

Usability models etc.

2.2.1 Key Ideas of Elaboration

- Elaboration is not more than few months
- Has short timeboxed iterations
- Has realistic tests early
- Adapt according to the feedback of users
- Most of the use cases and other requirements are written in detail, through a series of workshops.

2.3 DOMAIN MODELS

A domain model is a visual representation of conceptual classes or real situation objects in a domain.

In object oriented analysis, domain model is the most important.

It illustrates the concepts in a domain.

It acts as a source of inspiration for designing some software objects.

2.3.1 Example

A partial domain model for a visual dictionary is drawn with UML class diagram. The conceptual classes of

- Payment and
- Sale

are shown

The payment is related to sale.

Sale has date and time information.

The conceptual perspective can be had by applying UML class diagram notation for domain model.

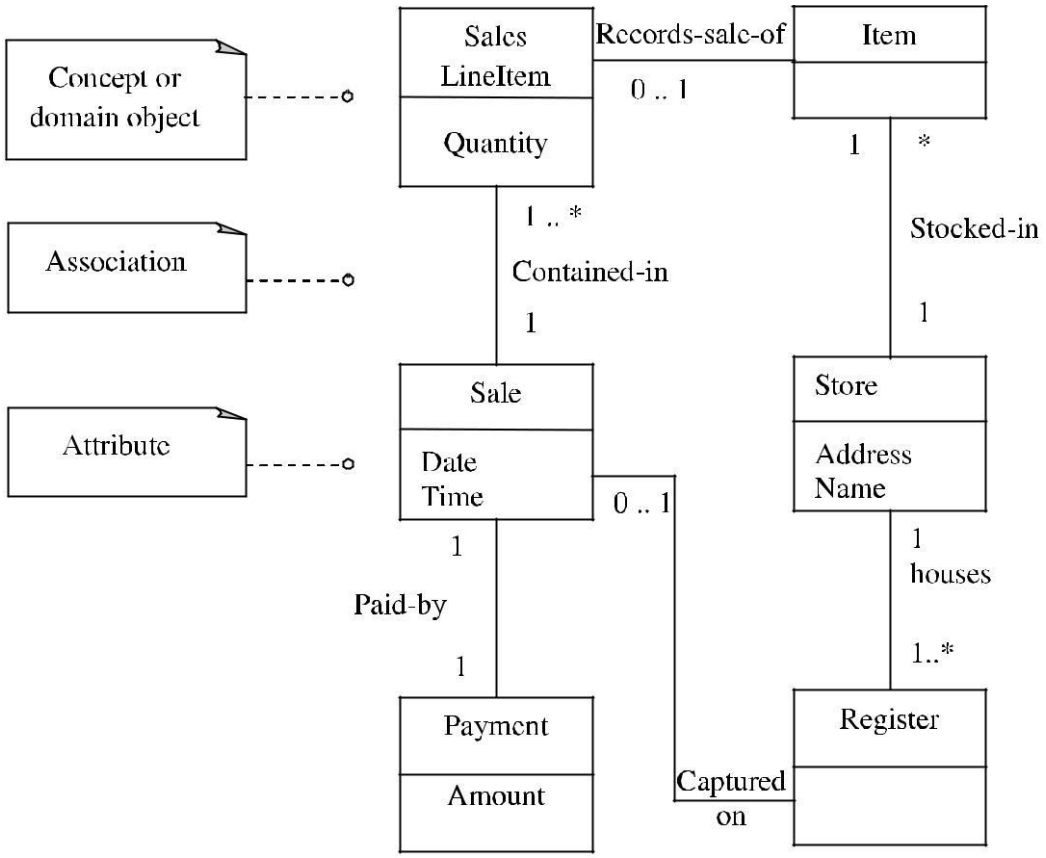


Figure 2.2 Partial Domain Model

Domain models are also called as conceptual models.

Domain models are illustrated with a set of class diagrams

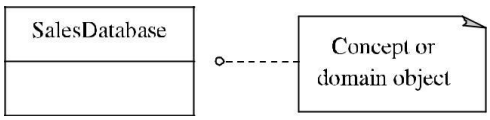
- Domain objects or conceptual classes
- Association between conceptual classes
- Attributes of conceptual classes

  In the above example if all the information were expressed in plain text it is not easy to understand and hence we go for visual representations.

  So the domain model is a visual dictionary of abstractions, domain vocabulary and information content of the domain.

- A domain model shows real-situation conceptual classes and not software classes. [such as Java or C++ classes]
- So a domain model should probably avoid
  - Software artifacts like window or database
  - Methods and responsibilities.

Avoid



Avoid

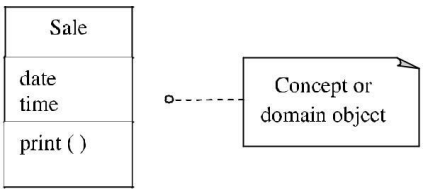


Figure 2.3 A domain model does not show software artifacts or classes

2.3.2 Conceptual classes

A conceptual class may not considered in terms of

1) Symbol – words or images
2) Intension – definition of conceptual class
3) Extension – set of examples to conceptual class

Example: Purchase transaction event

Symbol – sale (English)

Intension – Represents events of purchase transaction, having date and time

Extension – set of all sale instances in the universe.

A conceptual class

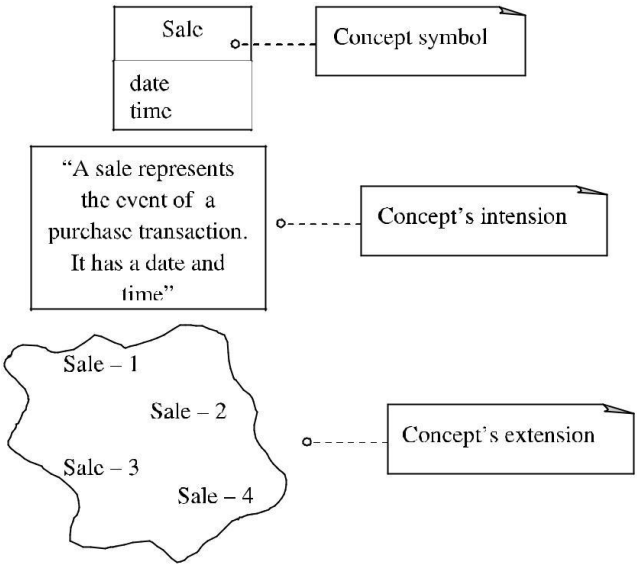


Figure 2.4 A domain model is not a data model

2.3.3 Creation of Domain Model

The steps for creating a domain model are

(i) Find the conceptual classes

(ii) Draw them as classes in UML class diagram

(iii) Add associations and attributes

2.4 FINDING CONCEPTUAL CLASSES AND DESCRIPTION CLASSES

2.4.1 Finding Conceptual Classes

The three strategies to find conceptual classes are

1) Reuse or modify existing models
2) Use a category list
3) Identity noun phrases

2.4.1.1 Reuse or Modify Existing Models

- It is the easiest approach
- There are published and well known domain models and data models like
  - Inventory
  - Finance
  - Health

These can be modified into domain model

2.4.1.2 Use a Category List

Some of the common categories found in

1) POS
2) Monopoly and
3) Airline reservation are listed below

are listed below

Conceptual class category	Examples
1. Business transactions Guideline: They are critical	Sale, payment reservation
2. Transaction line itemsGuideline: Transaction come with related line items	SalesLineItem
3. Product or service relate to a transaction or transaction line item	Item flight, seat, meal

4.	Where is transaction recorded?	Register, ledger
5.	Roles of people; actors in usecase	Cashier, customer, passenger
6.	Place of transaction	Store, airport
7.	Events with time or place to remember	Sale, payment
8.	Physical objects	Item, Register, Airplane
9.	Description of things	ProductDescription, FlightDescription
10.	Catalogs	ProductCatalog, FlightCatalog
11.	Containers of things	Store, Bin, Board, Airplane
12.	Things in container	Item, Square
13.	Collaborating systems	Air Traffic Control
14.	Records of finance, work, contracts, legal matters	Receipt, Ledger
15.	Financial Instruments	Cash, Check, Line of credit, TicketCredit
16.	Schedules, manuals, documents referred regularly	DailyPriceChangeList, RepairSchedule

2.4.1.3 Noun Phrase Identification

Linguistic analysis

Indentify the nouns and noun phrases in textual description of a domain.

Example:

Process Sale

Main success scenario (or Basic Flow)

1) Customer arrives at a POS checkout with goods
2) Cashier starts a new sale
3) Cashier enters item identifier
4) System records SaleLineItem

   Presents item description, price and running total

   Price calculated from a set price rules.
5) System presents total with taxes and asks for payment
6) Cashier tells customer the total and asks for payment
7) Customer pays and system handles payment

8) System logs completed sale and sends sale and payment to external accounting

9) System presents receipt

10) Customer leaves with receipt and goods

Extensions

7.a. Paying by cash

1) Cashier enters amount

2) System presents balance

3) Cashier deposits cash and returns balance

4) System, records cash payment

Some noun phrases are candidate conceptual classes. Some refer to simple attributes of conceptual class.

The disadvantage of this approach is – Different noun phrases may represent same conceptual class or attributes among ambiguities.

2.4.1.4 Examples

POS Domain

A list of candidate classes for the domain is generated.

The list consists of requirements and simplifications of process sale.

Sale	Cashier
Cash payment	Customer
SalesLineItem	Store
Item	ProductDescription
Register	ProductCatalog
Ledger	

Then, construct the UML class diagram for the above classes.

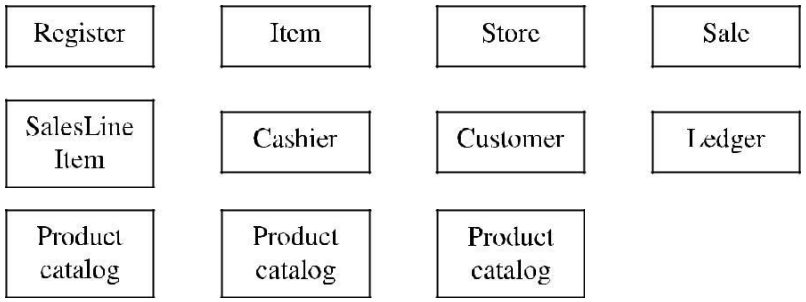


Figure 2.5 Initial POS Domian Model

Then we add attributes and associations.

Monopoly Domain

The list of candidate classes for Monopoly domain are

Player

Piece

Die

Board

Square

The representation of these classes are

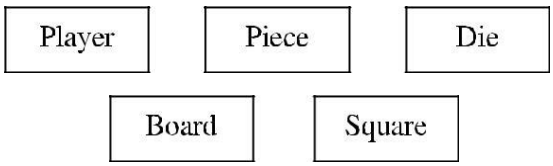


Figure 2.6 Initial Monopoly Domain Model

2.4.2 Finding Description Classes

A description class is a class that contains information that describes something.

Example:

ProductDescription is a description class that records

- Picture
- Price and
- Text description

2.4.2.1 Need for description classes

Consider a store where

- An item is a physical item
- An item has serial no, cost, description, price, itemID etc.
- After selling the item, all these information are lost as the software instance of item is deleted.

Example:

If a store has ‘Reynolds’ pen and in a demand the pen is sold and its instance is lost.

If a customer asks

‘What is the cost of Reynolds Pen’

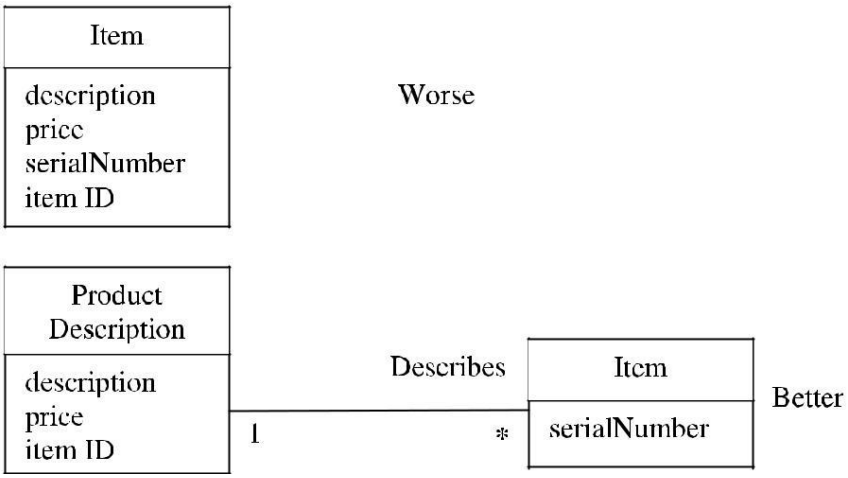
No one can answer because the price of the item is lost, once it is sold.

We also cannot have a duplicate instance of all objects because it causes inefficiency and wastage of memory.

So the solution is to have a 'description class'. For example, here we have got ProductDescription class.

Here ProductDescription would have the price, description of the item, but not the serialNo which is irrelevant.

The ProductDescription class is represented by



2.7 Figure Description classes

Description classes are used when

- Need for description of item, or service required
- Deleting instance of item, leads to loss of information
- Reduces redundancy and duplication

Examples

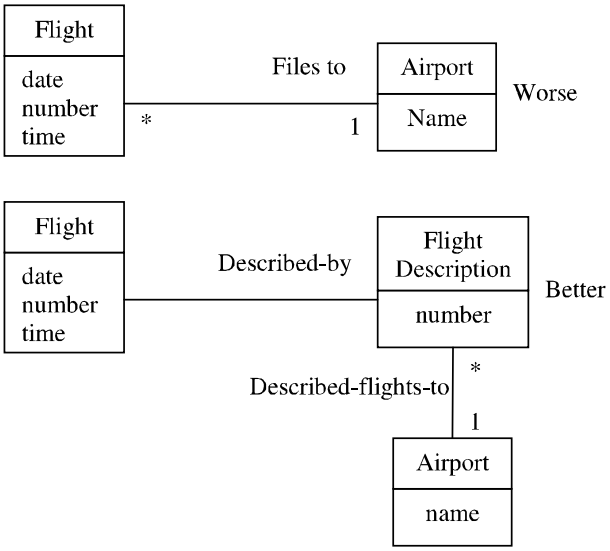
1) Airline company

After a plane crash, all flights were cancelled for six months.

So flight instance software is deleted and all information about the flight services were lost.

So here the description classes must have details on flight services.

So flight description has flight route even if flight is not scheduled.



2) Mobile phone company

It has packages like "gold", "silver", "bronze". Here description classes contains information about the package like

- Rates per minute
- Cost etc…

2.5 ASSOCIATION

An association is a relationship between classes that indicate meaningful and interesting connection.

It is represented as a solid line connecting two class symbols.

Example

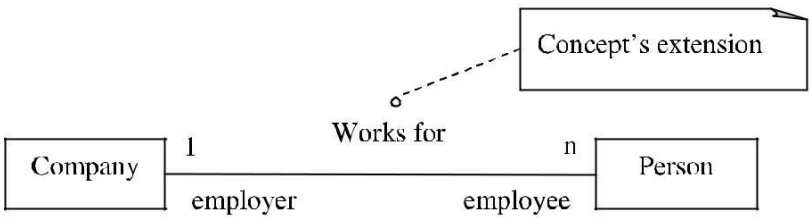


Figure 2.9 Association

Here the association "works for" connects two roles – employee and employer.

A person is an employee of a company and company is an employer of person.

2.5.1 Definition

An association is defined as "Semantic relationship between two or more classifiers that involve connections among their instances".

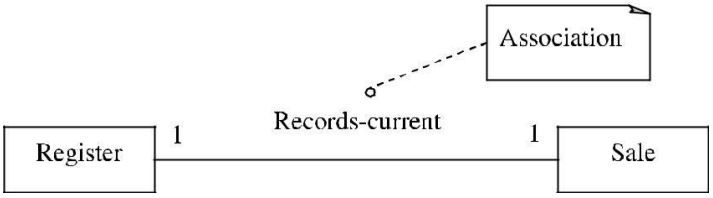


Figure 2.10 Association

Avoid too many associations to a domain model.

Example:

A graph with 'n' nodes can have (n(n-1))/2 associations to other nodes. i.e. 20 classes will have 190 association lines.

So avoid the unnecessary associations.

An association is not statement about data flows during domain modeling.

2.5.2 Association Notation

Binary association is solid line connecting two class symbols with capitalized association name.

The ends of association contains multiplicity expression i.e., numerical instances of the classes.

Example:

Legal formats of compound association name.

- Work-for
- Work for

Each end of associations is called role.

Roles may optionally have

- Multiplicity expression
- Name
- Navigability

2.5.3 Multiplicity

Multiplicity defines how many instances of a class A can be associated with the instance of class B

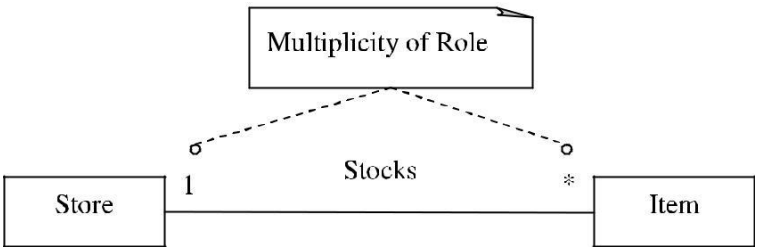


Figure 2.11 Multiplicity on an association

2.5.4 Multiplicity values

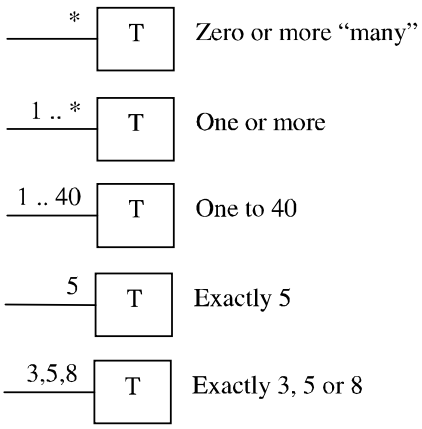


Figure 2.12 Multiplicity values

The multiplicity value depends on the modeler or software developer.

2.5.5 Navigation

UML uses association navigation or navigability to specify a role affiliated with each other at the end of association line.

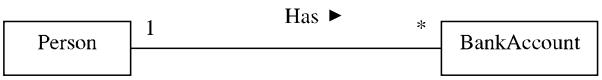


Figure 2.13 Navigation

An optional reading direction arrow indicates direction to read association name.

Here it means association from person to BankAccount (person has BankAccount) and not the reverse.

The convention is to read association from left to right and top to bottom.

2.5.6 Multiple associations between two classes

Two classes may have multiple (more than one) associations between them.

Example:

Airline system

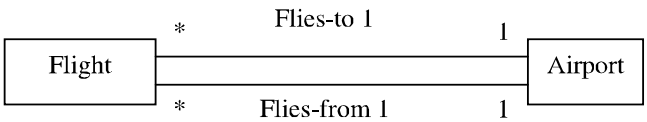


Figure 2.14 Multiple Associations

2.5.7 Common associations list

1) POS 2) Monopoly 3) Airline reservation domains

Category	Examples
A is a transaction related to another transaction B	CashPayment – sale cancellation – reservation
A is a line item of a transaction B	SalesLineItem – Sale
A is a product or service for a transaction (or line item) B	Item – SalesLineItem Flight – reservation
A is a role related to a transaction B	Customer – PaymentPassenger – Ticket
A is physical or logical part of B	Drawer – RegisterSquare – BoardSeat – Airplane
A is physically or logically contained in/on B	Register – storeItem – shelfPassenger – Airplane
A is a description of B	ProductDescription – ItemFlightDescription – Flight
A is known/logged/recorded/reported/ captured/ in B	Sale – RegisterPiece – Square
A is a member of B	Cashier – storePlayer – monopoly gamePilot – Airline
A is an organizational subunit of B	Department – StoreMaintenance – Airline
A uses or manages or owns	Cashier – RegisterPlayer – PiecePilot – Airplane
A is next to B	SalesLineItem – SalesLineItemSquare – SquareCity – City

2.5.8 Examples – Associations in Domain Models:

CaseStudy:

(i) NextGenPOS:

In POS the association are given as

1) Transaction related to another transaction:

   Sale paid – by CashPayment

2) Line items of transaction – Sale contains SalesLineItem

3) Product for transaction – SalesLineItem records – sale-of-item

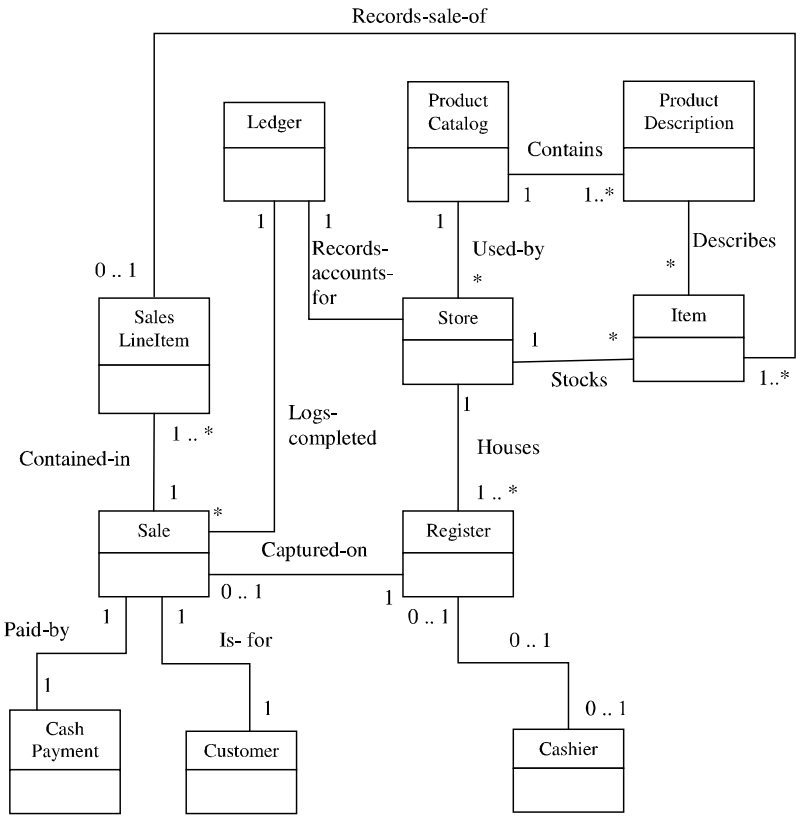


Figure 2.15 NextGen POS partial Domain Model

(ii) Monopoly

The associations are

1) A is contained in or on B – Board contains square
2) A owns B – Players own piece
3) A is known in/on B – Piece Is-on square
4) A is member of B – Player member-of monopoly game

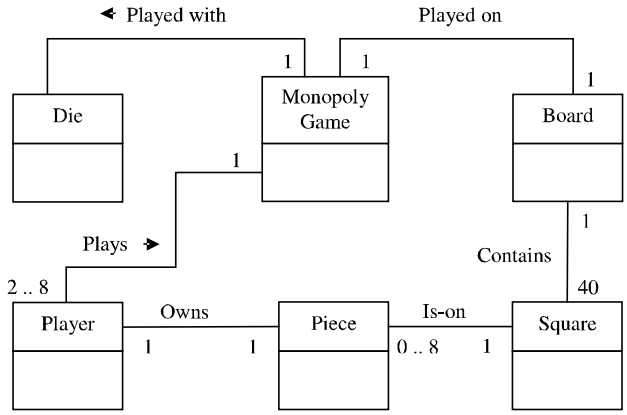


Figure 2.16 Monopoly partial Domain Model

2.5.9 Association

- Associations represents relationship between instances of classes. [A person works for a company; a company has a number of offices]
- The associations represent conceptual relationship between classes.
- Each association has two association ends, each end is attached to one of the classes in the association. An end can be explicitly named with a label. This label is called a role name. Association ends are often called roles.
- An association end also has multiplicity which is an indication of how many elements participate in the given relationship.
- Multiplicity indicates lower and upper bounds for the participating objects.
- The * represents the range 0…. Infinity.
- The 1 stands for 1…..1
- The most common multiplicaties are 1, * and 0…1.
- For general multiplicity we can have a single number (11 for players in crickets team) a range (such as 2….4 for players of canasta game)

  or discrete combinations of numbers and ranges (such as 2,4 for doors in a car before the onset of minivans)

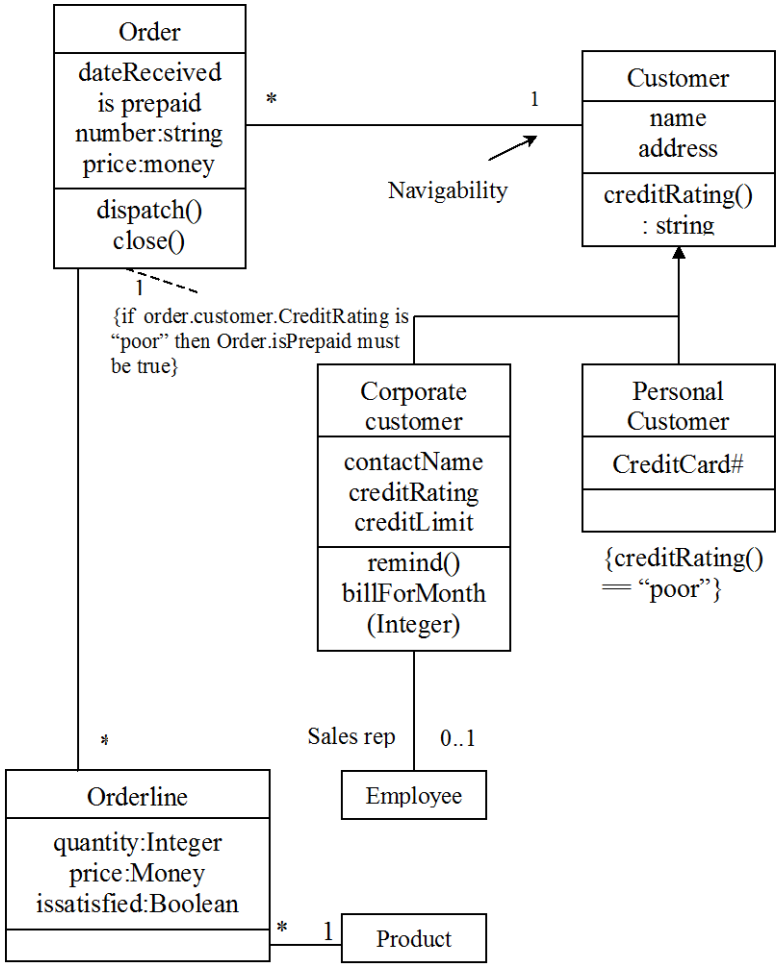
Within the specification perspective, associations represent responsibilities.

- In the specification perspective, associations represent responsibilities.
- Single valued relationship are implemented with a method that returns the related objects.
- Multivalued relationship are implemented with an iterate into a collection of related objects.
- An association implies some responsibility for updating the relationship.
- In implementation model there is pointer in both directions between related classes.
- **Example:** Association between order and customer. Arrows are indicated on the association lines. They indicate navigability.
- Here order has a responsibility to tell which customer it is for, but a customer has no corresponding ability to tell which orders it has. Here instead of symmetric responsibilities, we now have responsibilities on only one end of the line.
- But in implementation diagram, order contains a pointer to the customer but customer would not have a pointer to order.
- Navigability is an important part of implementation and specification diagrams.
- Navigability's are added as part of the shift to the specification and implementation perspective.
- If navigability exists in only one direction, we call the associations as unidirectional association.

- A bidirectional association contains navigability's in both direction.
- In UML, if the associations are without arrows it means the navigability is unknown or the association is bidirectional.
- Bidirectional associations include an extra constraint, that the two navigations are inverses of each other.

2.5.10 Naming Associations

There are many ways of naming associations. Traditional data modules like to name an association using a verb phrase, so that the relationship can be used in a sentence. In object modelers nouns are used to name the role of one or the other of the ends. An association represents a permanent link between two objects. The link exists during the whole lives of the objects, although the instances that are connected may change over time. So a parameters reference creation of an object, doesn't imply an association. They are modeled with dependencies.



2.6 ATTRIBUTES

- Attributes are very similar to associations.

- At conceptual level, an attribute customer's name indicates that customer have names.
- At specification level, the attributes name indicates the customer name and has a way of selecting a name.

At the implementation level, a customer has a field (an instance variable or a data member) for its name.

The notation for an attribute shows the attributes name, type and default value.

The UML syntax is

Visibility name : type = defaultValue

where visibility is the same as for operations.

2.6.1 Difference between Attribute and Association:

- From conceptual perspective, there is no difference between attribute and association.
- An attribute carries another kind of notation.
- Attributes are usually single valued.
- From the diagram we cannot infer whether the attribute is optional or mandatory.
- Multiplicity is put after the attribute name in square brackets

  Example: dateReceived [0….1]: Date

  The difference occurs at the specification and implementation levels.

  Attributes are small, simple classes such as strings, dates, money objects and non object values such as **int** and **real**.

2.6.2 Operations

The processes carried out but a class are called operations.

At specification level, operation correspond to public methods on a type. We have to indicate whether the object is read only or frozen.

At implementation model, we have to show private and protected operations.

2.6.3 UML syntax for operations

Visibility name (parameter_list) : return _type_ expression {property_string}

Where,

Visibility is + (public), # (protected) or – (private)

name is a string

Parameter list contains comma separated parameters.

Whose syntax is

Direction name: type = default value

Direction shows whether parameter used for input (in), output (out) or both (inout).

By default the direction value is in.

Return type expression is a comma- separated list of return types. Most people use single return type but multiple return types are also allowed.

UML defines a query as an operation that gets a value from a class without changing the system state. Such operations are marked with constraint {query}

Operations that do change state are called modifier.

Queries can be executed in any order but sequence of modifiers is more important Other terms are

- getting method
- setting method
- A getting method returns a value from a field.
- A setting method puts a value into a field.
- A client from outside should not tell whether

A query is getting method or a modifier is a setting method knowledge of getting and setting method is entirely internal to the class.

Other distribution is between operation and method.

An operation is something that is invoked on an object [procedure call] A method is the body of the procedure.

In C++, operation are called ‘member function’

Smalltalk calls operation as ‘methods’

UML uses the term ‘feature’ for an attribute or an operation.

2.6.4 Attributes

An attributes is a logical data value of an object.

Attributes of conceptual classes are identified to satisfy information requirements of current scenarios under development.

Example:

A receipt in processSale includes dateTime.

So,

- Sale requires ‘dateTime’ attribute
- Store requires ‘name’ and ‘address’

- Cashier requires an 'ID'

2.6.5 Notation

In the class representation, Attributes are shown in the second compartment.

Example:

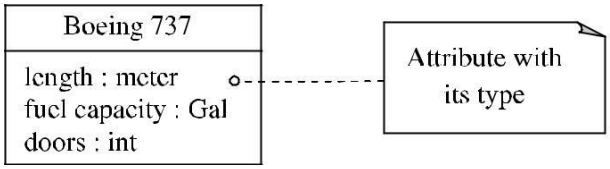


Figure 2.18 Attributes

The types of attribute is optional.

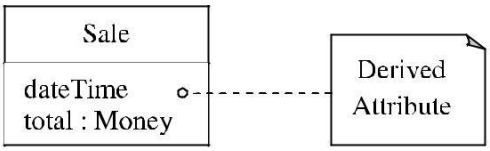


Figure 2.19 Derived Attribute

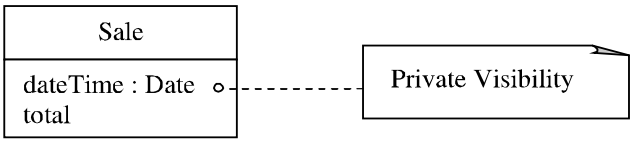
2.6.6 Syntax

The syntax of attribute in UML is

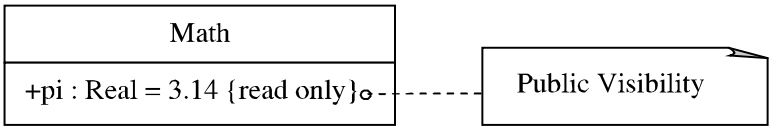
visibility name : type multiplicity = default {property-string}

Examples

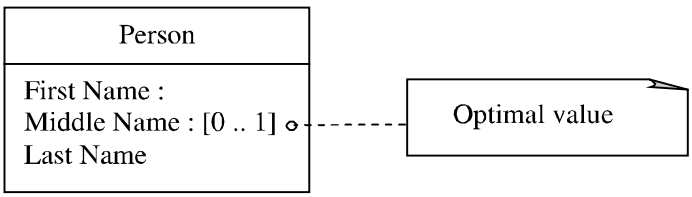
1)



2)



3)



Conventionally, all attributes have private visibility.

{read only} – Property of string for attributes

[0..1] - Optional multiplicity values 0 or 1

2.6.7 Derived Attributes

If an attribute is derived from some other information it is called derived attribute.

Example:

Total attribute in Sale is calculated or derived from 'SalesLineItem'.

Quantity calculated from actual multiplicity value of the association.

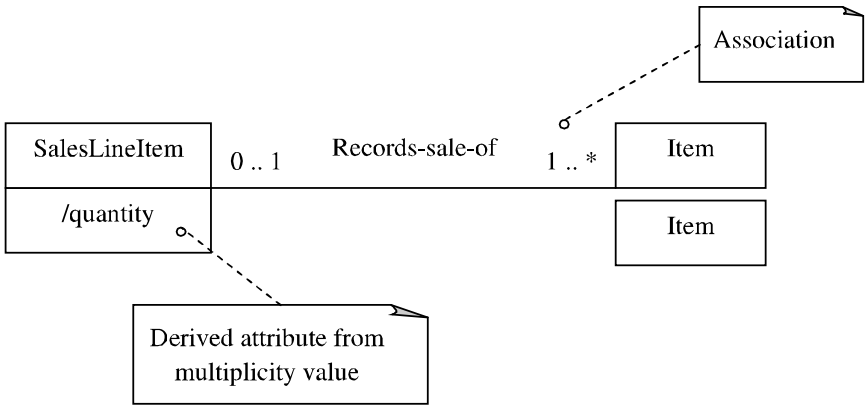


Figure 2.21 Derived attributes

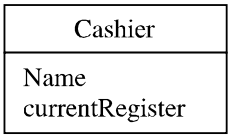
2.6.8 Datatype Attribute in Domain Model

Most of the attributes types are primitive datatype like

- Numbers
- Booleans
- Characters
- Date(or DateTime)
- Time
- String(Text)

Other common types are

- Address
- Color
- Geometrics
- Phone number
- ZIP or postal code
- Enumerated datatypes and so on.
- In place of complex concepts, associations is used instead of attributes.



is replaced by

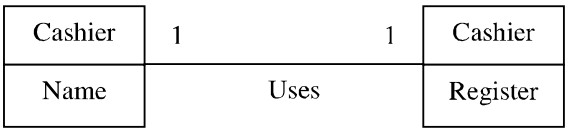


Figure 2.22 Relate with associations not attributes

2.6.9 Guidelines for Modeling Datatypes

In a domain model an attribute which may be initially considered as a number or string may be assigned a new datatype in conditions like the following

1) Data is composed of separate sections

   Phone no – Mobile, Landline, Office

   Name of person – First Name, Middle Name, Last Name

2) There are operations associated with it like validation

   SSN – Social Security Number

3) It has other attributes

   Price of promotion can have start-date and end-date

4) It is a quantity with a unit

   Payment amount has unit of currency

5) Abstraction of one or more types with some quantities.

   ItemIdentifier is a generalization of

   Universal Product Code (UPC)

   European Article Number (EAN)

Applying the guidelines to POS domain attributes

1) 'Item' is having common coding schemes like

   UPC – A

   UPC – E

EAN etc

These schemes have manufacturer, Product, Check-sum digit etc So datatype ItemID class may be present.

2) 'Price' and 'amount' attributes will be of type 'money' because they refer to currency.

3) The address attribute should be of type 'Address' because it has got three separate sections.

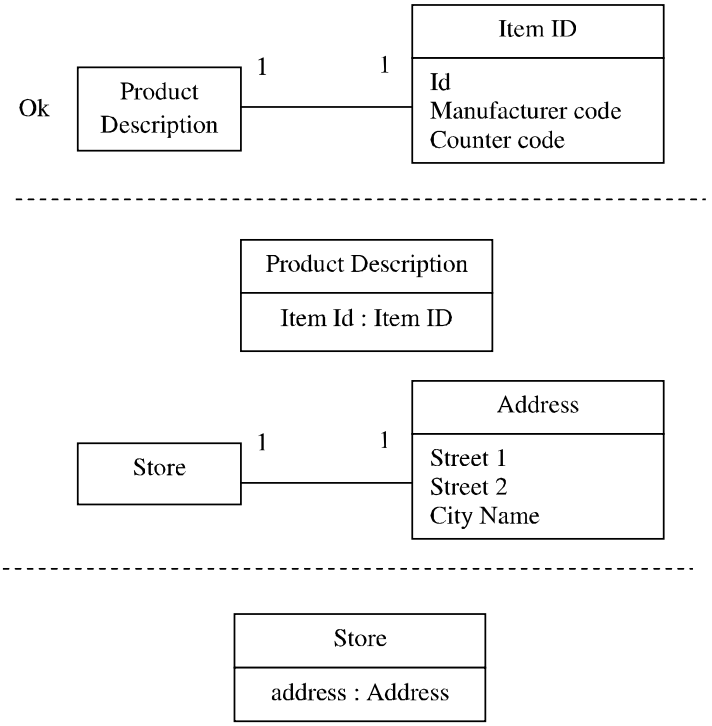


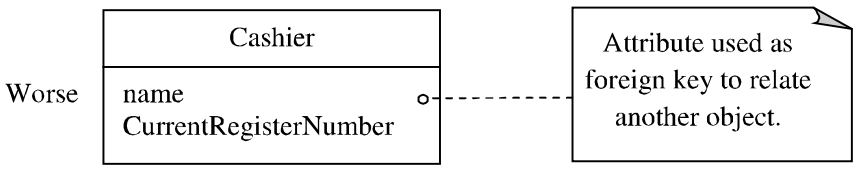
Figure 2.23 Two ways to indicate a datatype property of an object

Foreign keys (like relational databases) can be expressed with associations, instead of attributes.

Example:

If the 'cashier' class has a 'current Register No' attribute it relates cashier to Register object.

Here association can be used to relate cashier to Register object



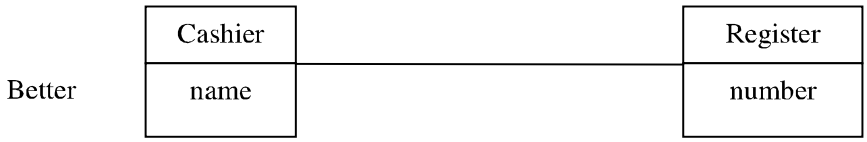


Figure 2.24 Don't use attributes as foreign keys

There are quantities with associated units.

Example:

Money is a quantity with 'currencies' as units.

Weight is a quantity with 'kilograms' or 'pounds' as units

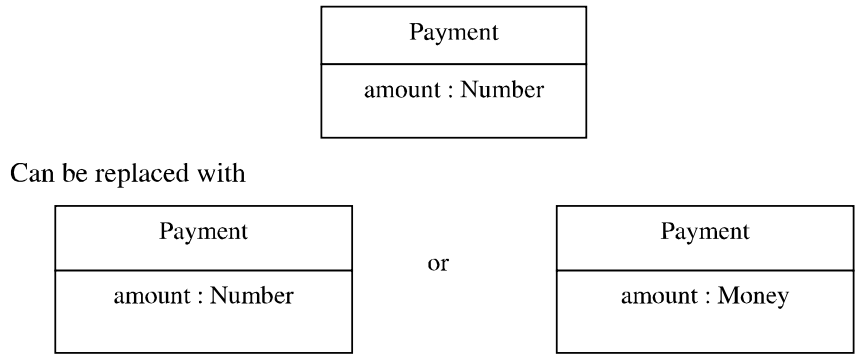


Figure 2.25

2.6.10 Example: Attributes in Domain Model

Case Study 1:

1) NextGenPOS:

Class	**Attributes**	**Description**
CashPayment	Amount Tendered	Determines if sufficient payment is done
ProductDescription	Description itemID price	Shows description display or receipt looks up Looks up ProductDescription Calculates salestotal. Shows lineitemprice
Sale	dateTime	Shows date and time of sale Useful for sales analysis
SalesLineItem	Quantity	Records quantity entered
Store	Address, name	Name and address of store

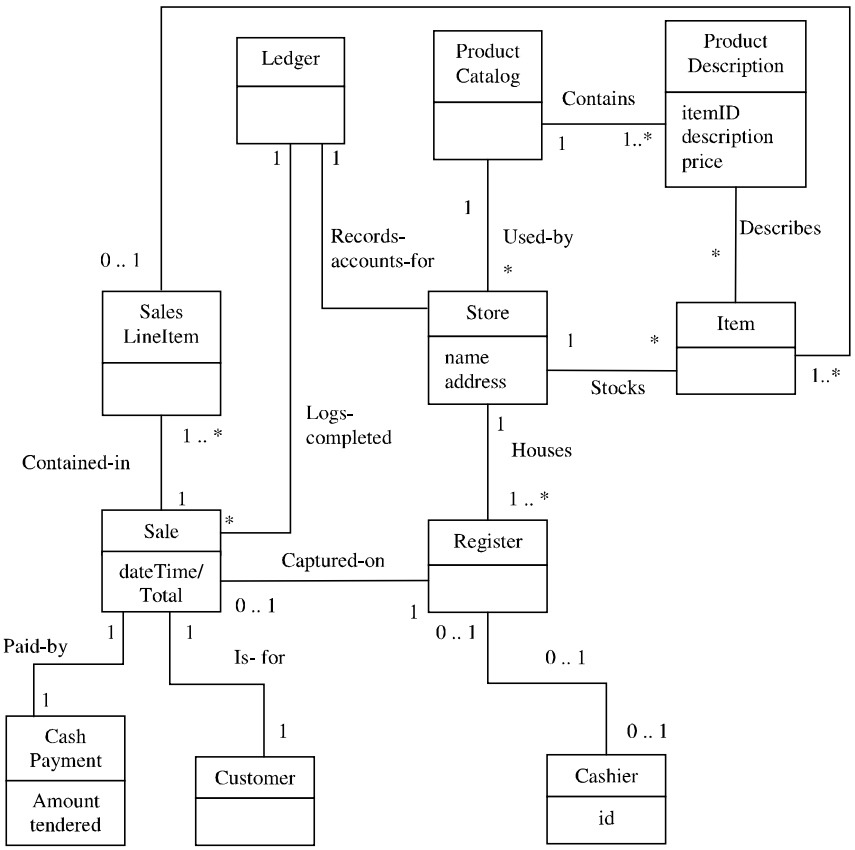


Figure 2.26 NextGen POS Partial Domain Model showing Attributes

2) Case Study 2:

Monopoly game

Class	Attributes	Description
Die	faceValue	After rolling die, calculate distance of move
Square	Name	Print output desired
Player	Name	Name of player
Piece	Name	Name of piece

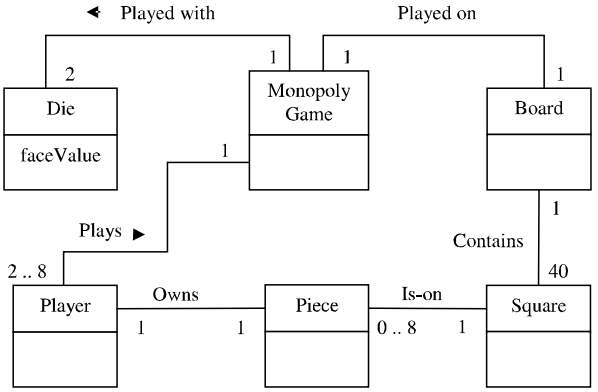


Figure 2.27 Monopoly Partial Domain Model

2.7 DOMAIN MODEL REFINEMENT

In the domain model refinement the fundamental concepts are

(i) Generalization

(ii) Specialization and

(iii) Conceptual class hierarchies

2.8 FINDING CONCEPTUAL CLASS HIERARCHIES

Generalization has to be understand to identify conceptual super and subclasses.

Generalization and conceptual class definition:

Definition:

A conceptual superclass is more general or encompassing than subclass

Example:

Superclass – Payment

Subclasses – CashPayment

Credit Payment

CheckPayment

Representation

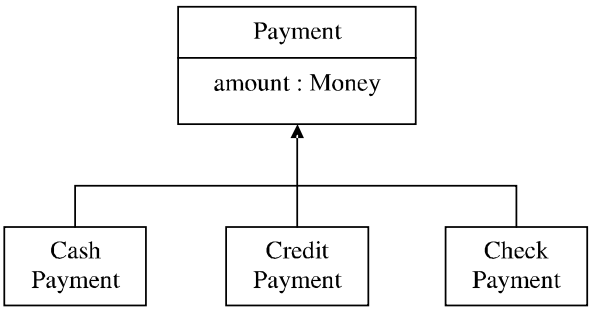


Figure 2.28 Class Hierarchies

Here cash payment is transferring money from one party to another.

Credit payment is transfer of money via credit institution which to be authorized.

2.8.1 Generalization and class sets

All members of a conceptual subclass set are members of their superclass set.

Example:

In terms of set membership, all instances of set ‘CreditPayment’ are also members of set ‘Payment’.

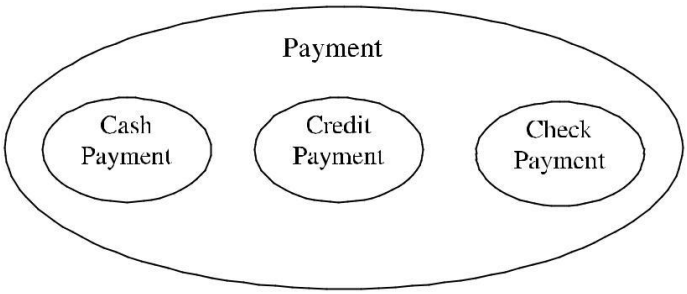


Figure 2.29 Venn Diagram of set relationships

2.8.2 Conceptual subclass Definition conformance

During creation of class hierarchy, statements about superclass that apply to subclass are made.

Example:

All 'payments' have an attribute 'amount' which are associated with 'Sale'.

2.9 AGGREGATION AND COMPOSITION

Aggregation is a form of association.

A hollow diamond is attached to the end of the path to indicate aggregation.

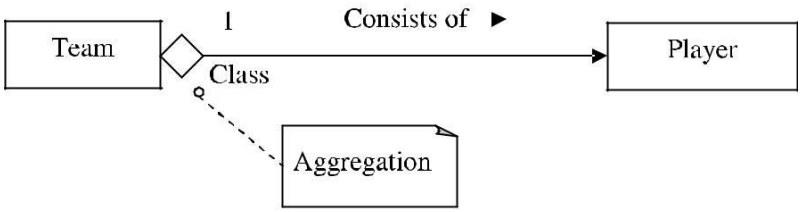


Figure 2.30 Aggregation

Composition (or a part of) is a form of aggregation.

It shows string ownership to represent the component of complex object.

Composition is also called as part whole relationship.

Composition is denoted as solid diamond at the end of the path.

A composition relationship implies that

1) An instance of the part (like square) belongs to only one composite instance (such as board) at a time.
2) The part must always belongs to composite.
3) The composition is responsible for creation and deletion of its parts.

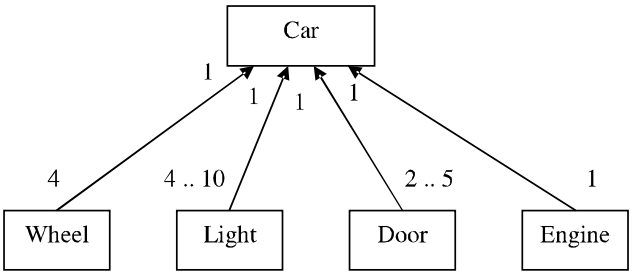


Figure 2.42 composition

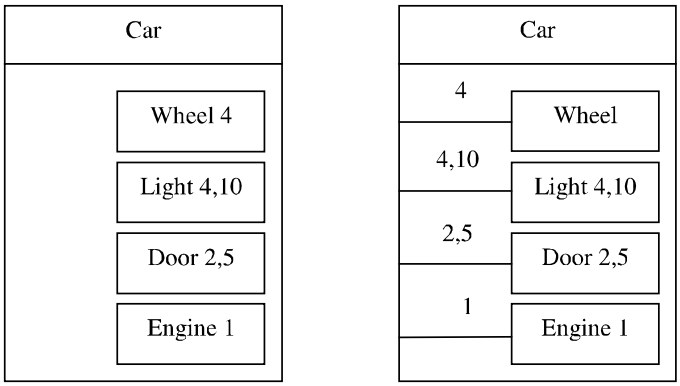


Figure 2.31 Nested composition

2.9.1 Guidelines for composition

- The life time of a part is bound within the lifetime of its composite.
- There is obvious whole-part physical or logical assembly.
- Composition sometimes refers to
- Operations like destruction, movement, recording parts such as location.

Benefits of composition

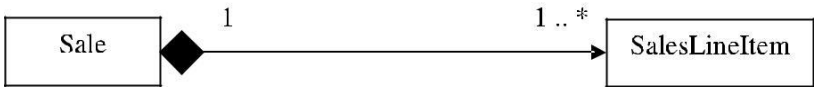
Composition is very useful in design than analysis.

1) Composition classifies domain constraint for the existence of the part independent of the whole.
2) It assists in identification of creator using GRASP.
3) Operations like copy, delete apply to parts.

Composition in NextGen Domain Model

In POS domain

(i) SalesLineItem is a part of composite sale



(ii) ProductCatalog is a composite of ProductDescription

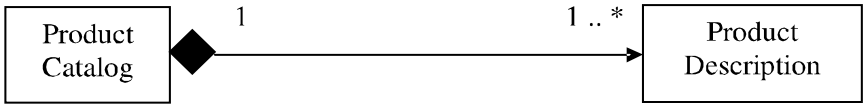


Figure 2.32

2.10 RELATIONSHIP BETWEEN SSD AND USE CASES

SSD shows systems events for one scenario of a use case.

The SSD is generated from inspection of a use case.

SSDs are derived from use cases.

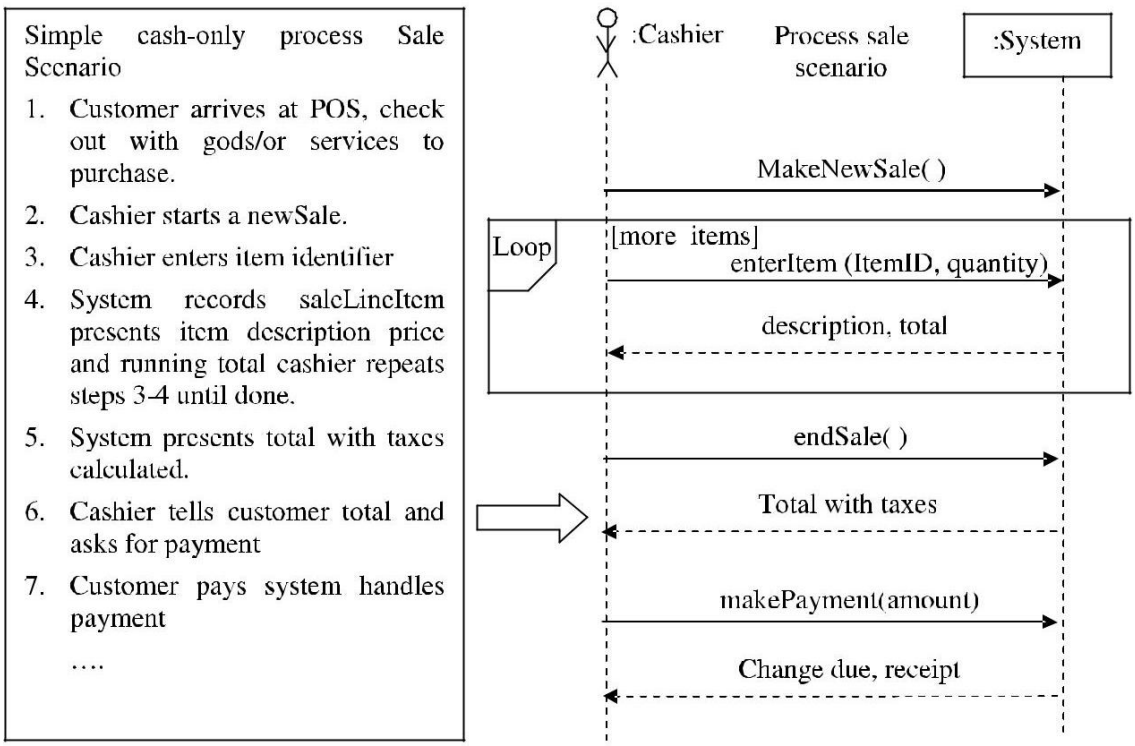


Figure 2.33

While naming the system events, use the abstract level of representation rather than physical input device.

Example:

"enterItem" is better than "Scan".

Because the entry could be through scanner, keyboard, voice input or anything.

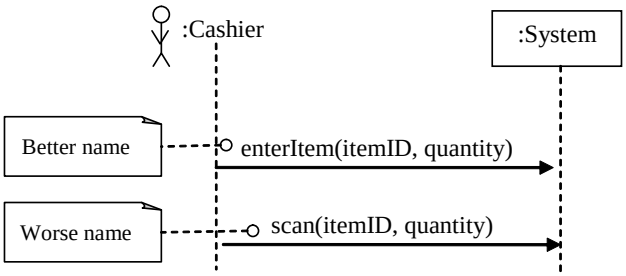


Figure 2.34

Always start the system event with a verb like

- add…
- enter…
- end…
- make…

to improve clarity

SSDs are also used to collaborate between systems.

Example:

NextGenPOS and external credit payment authorizer.

The elements of SSD are

- operation name
- parameters
- return data

Glossary is used to maintain the information of what is coming in and what is going out during design.

Example 1:

In the previous diagram, the return line has the description "change due, receipt".

Here glossary has a receipt entry and shows receipt (digital picture) and layouts and contents.

Example 2:

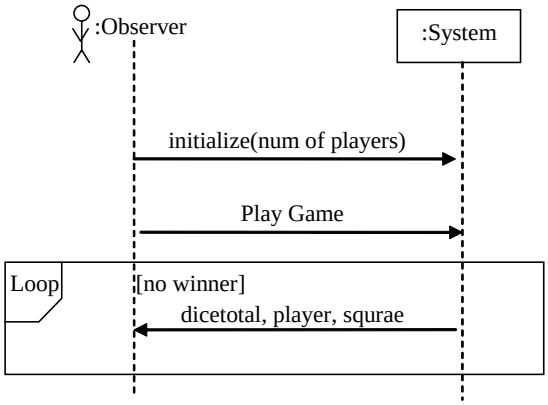


Figure 2.35 the SSD for Monopoly game

Here the events are

1) Observing person initializes number of players
2) He then request simulation of play
3) The output is watched until there is a winner.

SSDs are highly useful

- To understand the interface
- Collaborations of existing systems
- To document the architecture.

Normally it is not advisable to create SSDs for all scenarios.

SSDs can be drawn for scenarios of next iteration.

SSDs are part of the Use Case Model.

SSDs are used to represent analysis and design artifacts.

SSDs are not normally motivated during inception. They are created during elaboration.

Their purpose is to

1) Identify details of system events
2) Clarify major operations the system is designed to handle
3) Write system operation contracts
4) Support estimation

(Example: Macro estimation with unadjusted function points and cocomo II)

2.11 WHEN TO USE CLASS DIAGRAMS

2.11.1 Class Diagrams

The Essentials

A class diagram describes the types of objects in the system and the various kinds of static relationship that exist among them.

There are 2 kinds of static relationships.

- Associations
- Subtypes

The class diagrams show the attributes and operations of a class and the constraints that apply to the way objects are connected.

Perspectives

There are three different perspectives; we use in drawing class diagrams. They are

- Conceptual
- Specification
- Implementation

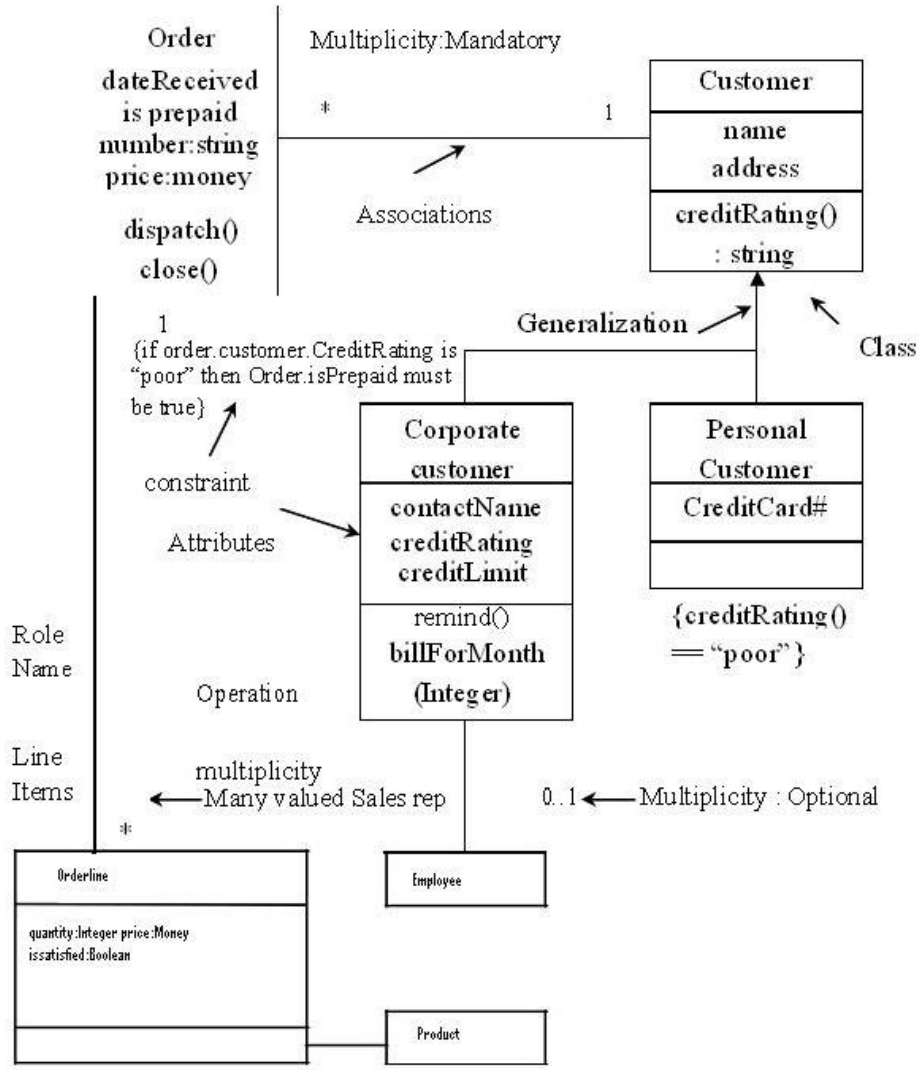


Figure 2.36 Class Diagram

We take the concepts in the domain under study and represent in diagrams, for the conceptual perspective.

These concepts will relate to the classes that implement them by there us often no direct aping.

A conceptual model should be drawn with little or no regard to the software that implements it, so it is considered language independent.

Specification

Here we look at the interface of the software but not its implementation.

The word type is used to talk about an interface of a class.

A type can have many classes that implement it and a classes that implement it an a class implement many types.

Implementation

Here we have classes and we lat the implementation base.

This is the most often used perspective but un many cases the specification perspective is better than implementation.

Classes are marked with << implementation class >> to show the implementation perspective and with <<type>> for the specification and conceptual perspective