2 STATIC UML DIAGRAMS

Class Diagram – Elaboration – Domain Model –Finding conceptual classes and description classes – Associations – Attributes – Domain model refinement – Finding conceptual class Hierarchies – Aggregation and Composition -Relationship between sequence diagrams and use cases – When to use Class Diagrams

1 Class Diagram

- UML includes class diagrams (**or design class diagram (DCD))** to illustrate classes, interfaces and their associations.
- They are used for static object modeling. It represents the static view of an application.
- A UML class box used to illustrate software classes often shows three compartments.
  - ✓first compartment illustrates the class name
  - ✓second compartment illustrates the attributes
  - ✓third illustrates the methods of the class
- Visibility, parameters and compartments are optional
- The class diagram describes the attribute and operations of a class and also the constraints imposed on the system.
- The class diagrams are widely used in the modeling of object oriented systems because they are the only UML diagram which can be mapped directly with object orient languages.
- The class diagram shows the collection of classes, interfaces, associations, collaborations and constraints. It is also known as ***Structural diagram.***

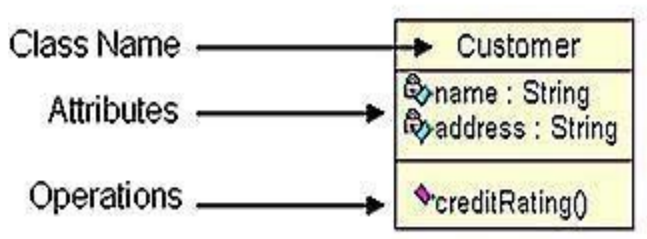


Figure: Class Notation

- Class diagrams can be applied in two perspectives:

1. Conceptual perspective

2. Software perspective

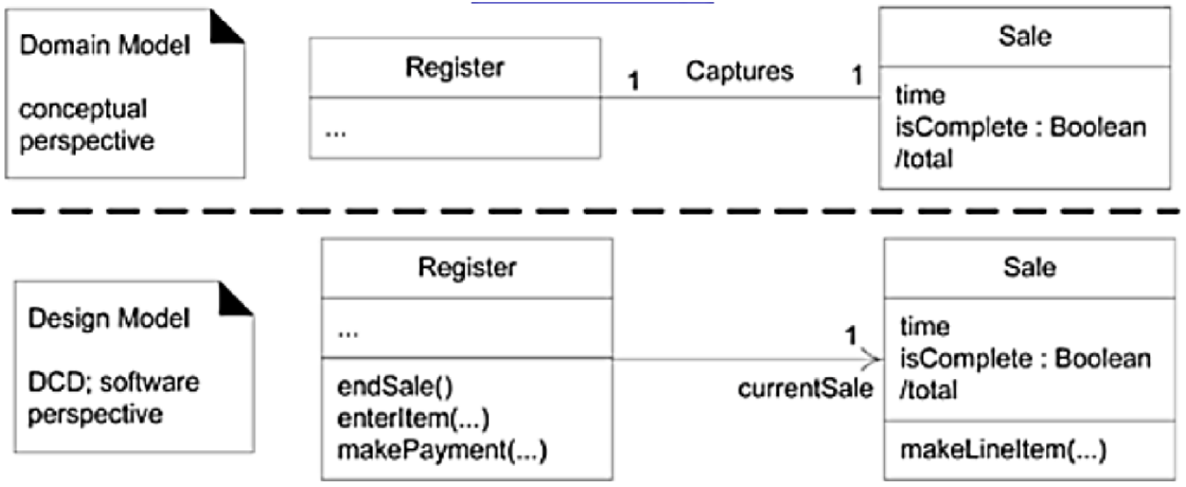
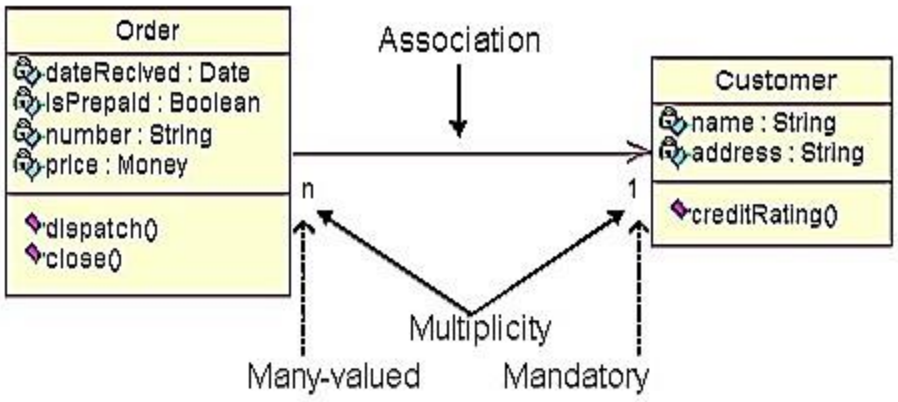


Figure: UML class diagrams in two perspectives



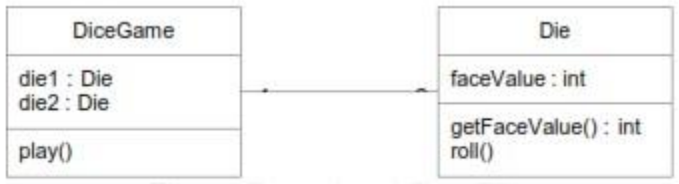


Figure: Examples of Class Diagram

- A UML classifier is a model element that describes behavioral and structure features. They are a generalization of many of the elements of the UML, including classes, interfaces, use cases and actors.

- A UML constraint is a restriction or condition on a UML element. It is visualized in text between braces.

Example: {size>=0}

Ways to show UML attributes

- Attributes of a classifier (also called Structural properties in the UML) are shown several ways:
  - Attribute text notation
  - Association line notation
  - Both together

The full format of the attribute text notation is: ***visibility name: type multiplicity = default { property-string }***

Relationships used in class diagrams:

- Association
- Dependency
- Aggregation
- Composition
- Generalization

Association: represents relationship between instances of classes.

Example: A person works for a company

- Each association has two ends, each end is attached to one of the classes in the association.
- The end should be explicitly named with a label.
- This label is called as a role name.
- This label is called as a role name.
- Association ends are often called as roles
- An association end also has **multiplicity** which is an indication of how many elements participate in the given relationship.
- A **qualified association** has a qualifier that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key.
- An **association class** allows us to treat an association itself as a class, and model it with attributes, operations, and other features.
- The UML **dependency relationship** indicates that one element (of any kind, including classes, use cases, and so on) has knowledge of another element.
- It is illustrated with a dashed arrow line.
- In class diagrams the dependency relationship is useful to depict non-attribute visibility between classes; in other words, parameter, global, or locally declared visibility.

Generalization is the activity of identifying commonality among concepts and defining superclass and subclass relationships.

- For example, the concepts CashPayment, CreditPayment and CheckPayment are all very similar and it is possible to organize them into a generalization, specification, class hierarchy. ***Aggregation*** is a vague kind of association in the UML that loosely suggests Whole -part relationships. It has no meaningful distinct semantics in the UML versus a plain association, but the term is defined in the UML.

Composition, also known as **composite aggregation**, is a strong kind of whole –part aggregation and is useful to show in some models. It is an instance of a part belongs to only one composite instance.

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

Operations & Methods

A UML operation is a declaration, with a name, parameters, return type, exceptions list and possibly a set of constraints of pre and post conditions.

A UML method is the implementation of an operation, if constraints are defined, the method must satisfy them.

2 Elaboration

- Elaboration is the initial series of iterations during which:
  - The majority of requirements are discovered and stabilized.
  - The major risks are mitigated or retired.
  - The core architectural elements are implemented and proven.

Elaboration: builds the core architecture, resolve the high-risk elements, define most requirements, and estimate the overall schedule and resources

- During inception, the Vision summarizes the project idea in a form to help decision makers determine if it is worth continuing, and where to start.
- Through the elaboration iterations, the "vision" and the Vision are refined, based upon feedback from incrementally building parts of the system, adapting, and multiple requirements workshops over several development iterations.
- Elaboration often consists of between two and four iterations ➢ Each iteration is recommended to be between two and six weeks.
- Each iteration is timeboxed, meaning its end date is fixed
- Elaboration is not a design phase or a phase when the models are fully developed in preparation for implementation in the construction step—that would be an example of superimposing waterfall ideas on to iterative development and the UP.
- During the Elaboration phase the project team is expected to capture a healthy majority of the system requirements.
- However, the primary goals of Elaboration are to address known risk factors and to establish and validate the system architecture.
- Common processes undertaken in this phase include the creation of use case diagrams, conceptual diagrams (class diagrams with only basic notation) and package diagrams (architectural diagrams).

- o Refined vision o Core architecture o Resolution of high risk
- o Identification of most requirement and scope

Key ideas and best practices that will manifest in Elaboration include:

- o do short timeboxed risk-driven iterations o start programming early o adaptively design, implement, and test the core and risky parts of the architecture o test early, often, realistically . o adapt based on feedback from tests, users, developers . o write most of the use cases and other requirements in detail, through a series of workshops, once per elaboration iteration.

2.1.1 ELABORATION ARTIFACTS:

1. ***Domain Model:***
   - o Visualization of domain concepts. It is similar to static information model of the domain entities.
2. ***Design Model:***
   - o Set of diagrams that describes the logic design .This include software class diagrams, object interaction diagrams, package diagrams etc.
3. ***Software architecture document:***
   - o Summary of the outstanding design ideas and their motivation in the system.
4. ***Data model:***
   - o Collection of conceptual tools for describing data,data relationships, data semantics and consistency constraints.
5. ***Use case story boards and UI prototypes:***
   - o Description of the user interface, path of investigation, usability models etc.

Artifact	Comment
Software Architecture Document	A learning aid that summarizes the key architectural issues and their resolution in the design. It is a summary of the out-standing design ideas and their motivation in the system.
Data Model	This includes the database schemas, and the mapping strategies between object and non-object representations.
Test Model	A description of what will be tested, and how.
Implementation Model	This is the actual implementation - the source code, executables, database, and so on.
Use-Case Storyboards,	A description of the user interface, paths of navigation, usability models,

UI Prototypes	and so forth.

- Elaboration phase testing is important, to obtain feedback, adapt, and prove that the core is robust.
- The majority of terms will be discovered and elaborated in the Glossary during this phase.

Inception	**Elaboration**
It is about initiating the project	It is about creating a partial but working version of the system-an executable architectural baseline.
Goals includes- ▪ Establishing feasibility ▪ Creating a business case ▪ Capturing essential requirements ▪ Identifying critical risks	Goals includes ▪ Create an executable architectural baseline ▪ Refine the risk assessment ▪ Define quality attributes ▪ Capture use cases to 80% of the functional requirements
It focus on requirements and analysis workflows	It focuses on requirements, analysis, design, implementation and test
Milestone makes use of goal oriented approach which sets certain goals that must be achieved for the milestone to have been reached. The milestone for inception is the life cycle objectives that include defining system scope with the help of use case models, capturing key requirements.	Milestone is life cycle architecture. It includes UML Static model, UML dynamic model, UML case model, revising risk assessment through, etc
Example: University management system: Requirements: • Register courses • Change password • View timetable • Look up course description • Change student personal details • Ability to view exam results	**Architecture baseline:** • Early version of the final system known as architecture baseline. It is a subset of the entire system. It includes the subsystems, components and nodes • The architecture is influenced not only by the architecturally significant use cases, but also platform, legacy system that system needs to be integrated, standard and policies

Identifying stakeholders for business case:	**Capturing use cases:**
1.Stakeholders: **Student:** Want to register for the courses for an upcoming semester	**USECASE 1:**register for courses **Precondition:** Student must be logged on to the system
Program advisor: Wants to ensure that all the prerequisites are met for the student's courses.	
2.Stakeholders: **Students:** Would like to have a detailed description of the course, course requirements, the instructor and the class times	**USE CASE2 :**look up a course in the course directory **Precondition:** Student must be logged into the system.
3.Stakeholders: **Students:** Need to have updated personal information so they can be contacted	**Use case 3:** Change student personal information **Precondition:** Student is logged into the system.

3 Domain Model

- *What is a domain model?*
- *Motivation: why create a domain model?*
- *Guideline: how to create a domain model?*

3.1 What is a domain model?

- A **domain model** is a visual representation of conceptual classes or real-situation objects in a domain.
- Domain models have also been called **conceptual models, domain object models** and **analysis object models.**

Definition: In the UP, ―Domain Model‖ means a representation of real-situation conceptual classes, not of software objects. The term does not mean a set of diagrams describing software classes, the domain layer of a software architecture or software objects with responsibilities.

- The UP Domain model is a specialization of the UP **Business Object Model** [BOM] ―focusing on explaining things and products important to a business domain‖.
- Applying UML notation, a domain model is illustrated with a set of **class diagrams** in which no operations (method signature) are defined. It provides a ***conceptual perspective***. It may show:
  - Domain objects or conceptual classes
  - Associations between conceptual classes
  - Attributes of conceptual classes

Definition: why call a domain model a "visual dictionary"?

- It visualizes and relates words or concepts in the domain.
- It also shows an abstraction of the conceptual classes

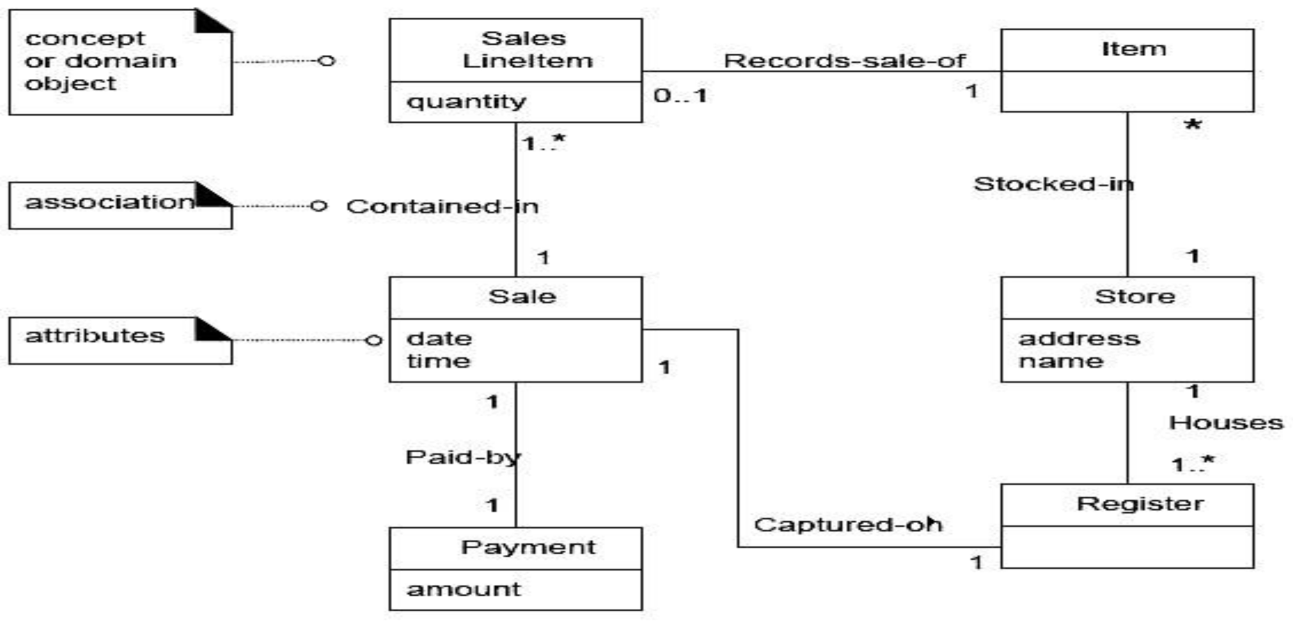


Figure: Partial domain model- a visual dictionary

- ➢ The information it illustrates (using UML notation) could alternatively have been expressed in plain text (in the UP glossary).
- ➢ The domain model is a visual dictionary of the noteworthy abstractions, domain vocabulary, and information content of the domain.

Definition: Is a domain model picture of software business objects?

- ➢ A UP Domain Model is a visualization of things in a real situation domain of interest, not of software objects such as Java or C# classes, or software objects with responsibilities.
- ➢ The following elements are not suitable in a domain model:
  - ▪ Software artifacts, such as a window or a database, unless the domain being modeled are of software concepts, such as a model of graphical user interfaces.
  - ▪ Responsibilities or methods.

Definition: What are two traditional meanings of "domain model"?

- ➢ Domain model is conceptual perspective of objects in a real situation of the world, not a software perspective.
- ➢ It has been used to mean ―the domain layer of software objects‖. That is, the layer of software objects below the presentation or UI layer that is composed of domain objects- software objects that represent things in the problem domain space with related ―business logic‖ or

  ―domain logic‖ methods.
- ➢ For example, a *Board* software class with a *getSquare* method.

Definition: What are conceptual classes?

A conceptual class is an idea, thing, or object.

A conceptual class may be considered in terms of its symbol, intension, and extension.

- **Symbol** – words or images representing a conceptual class.
- **Intension** – the definition of a conceptual class.
- **Extension** – the set of examples to which the conceptual class applies.

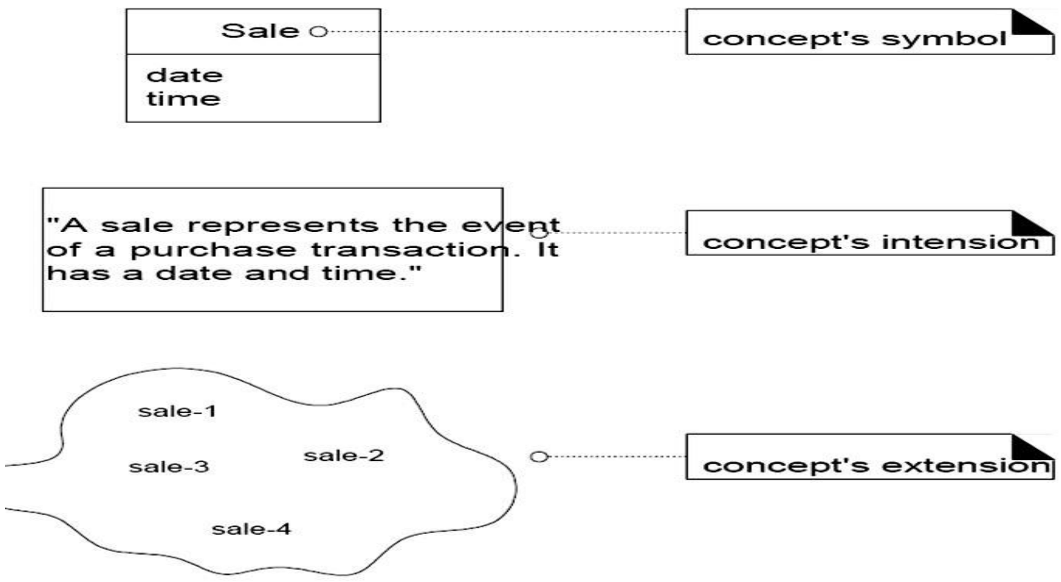


Figure: a Conceptual class has a symbol, intension and extension

Definition: What are Description classes?

- A Description class contains information that describes something else.
- For example, a *ProductDescription* that records the price, picture, and text descriptions of an item. This was first named the *Item-Descriptor pattern.*
- The need for description classes :
  - An item instance represents a physical item in a store; it may have a serial number
  - An item has *description*, *price,* and *itemID*
  - A *ProductDescription* class records information about items
  - Even if all inventoried items are sold and corresponding item software instances are deleted, the Product Description still remains.

Definition: Are domain and data model the same thing?

- A domain model is not a data model, so do not exclude a class simply because the requirements don't indicate any obvious need to remember information about it or because the conceptual class has no attributes.

3.2 Motivation: why create a domain model?

Motivation: lower representational gap with OO modeling

This is a key idea in OO: use software class names in the domain layer inspired from names in the domain model, with objects having domain-familiar information and responsibilities. This supports a low representational gap between our mental and software models. And that's not just a philosophical nicety- it has a practical time and money impact. For example, a source code payroll program written in 1953: 1000010101000111101010110110100010101010101111010101….

As compute science people, we know it runs, but the gap between this software representation and our mental model of the payroll domain is huge, that profoundly affects comprehension (and modification) of the software. OO modeling can lower that gap.

3.3 Guideline: How To Create A Domain Model

Bounded by the current iteration requirements under design:

1. Find the conceptual classes
2. Draw them as classes in a UML class diagram.
3. Add associations and attributes.

3.4 UML DOMAIN MODEL

- A UML domain model will relate objects in the system domain to each other. It will define concepts and terms.
- Objects in the domain model can be:
- Physical objects.
- Abstract concepts.
- A domain modeling process includes the following steps:
  - Model and document business processes first.
  - Select a candidate business process and work with the business domain experts to document it using the ubiquitous language.
  - Identify all the services that are required for the candidate business process. o These services can be atomic orchestrated in nature. o They can also be business or infrastructure. o Identify and document the state behavior of the objects used by services identified in the previous step.

3.5 ADVANTAGES OF DOMAIN MODELLING

- Domain driven design (DDD) is a powerful concept that will change the way modelers, architects, developers and testers will look at the software once the team is trained in DDD.
- A domain model creates q web of interconnected objects, where each object represents some meaningful individual, whether as large as a corporation or as small as a single line on an order form.

4 Finding conceptual classes and description classes

4.1 Finding conceptual classes

Three strategies to find conceptual classes:

1. Reuse or modify existing models.
2. Use a category list 3. Identify noun phrases.

1. **Reuse or modify existing models**
   This is the first, best, and usually easiest approach.
   There are published and well known domain models and data models like
   - Inventory
   - Finance
   - Health

   These can be modified into domain model.
   Reusing existing models is excellent, but outside our scope.
2. **Use a category list**
   The second method, using a category list, is also useful. Some of the categories found in
   - POS
   - Airline Reservation are listed below

Conceptual Class Category	**Example**
Physical or tangible objects	*Item, Register Airplane*
Specifications, designs, or descriptions	*ProductSpecification FlightDescription*
Place of transaction, Place of service	*Store Airport, Plane, Seat*
Business Transactions	*Sale, Payment Reservation*
Transaction line items	*SalesLineItem*
Where is the transaction recorded?	*Register, Ledger FlightManifest*
Catalogs	*ProductCatalog FlightCatalog*
Roles of people	*Cashier pilot*
Containers of other things	*Store,Bin Airplane*
Things in a container	*Item Passenger*
Other computer or electro-mechanical	*CreditPaymentAuthorizationSystem Air traffic Control*
Abstract noun concepts	*Hunger Acrophobia*
organizations	*SalesDepartment ObjectAirline*

 Noun Phrase Identification

➢ Identify the nouns and noun phrases in textual descriptions of a domain, and consider them as candidate conceptual classes or attributes.

➢ The fully dressed usecases are an excellent description to draw from for this analysis. For example, the current scenario of the *ProcessSale* usecase can be used.

Main Success Scenario (or Basic Flow):

1. **Customer** arrives at a **POS checkout** with **goods** and/or **services** to purchase.
2. **Cashier** starts a new **sale.**
3. **Cashier** enters **item identifier.**
4. System records **sale line item** and presents **item description, price,** and running **total.** Price calculated from a set of price rules. Cashier repeats steps 2-3 until indicates done.
5. System presents total with **taxes** calculated.
6. Cashier tells Customer the total, and asks for **payment.**
7. Customer pays and System handles payment.
8. System logs the completed **sale** and sends sale and payment information to the external **Accounting** (for accounting and **commissions)** and **Inventory** systems (to update inventory).
9. System presents **receipt.**
10. Customer leaves with receipt and goods (if any).

Extensions (or Alternative Flows):

7a.Payingbycash:

1. Cashier enters the cash **amount tendered**
2. System presents the **balance due,** and releases the **cash drawer.**
3. Cashier deposits cash tendered and returns balance in cash to Customer.
4. System records the cash payment.

➢ The domain model is a visualization of noteworthy domain concepts and vocabulary. In the use cases, thus they are a rich source to mine via noun phrase identification.

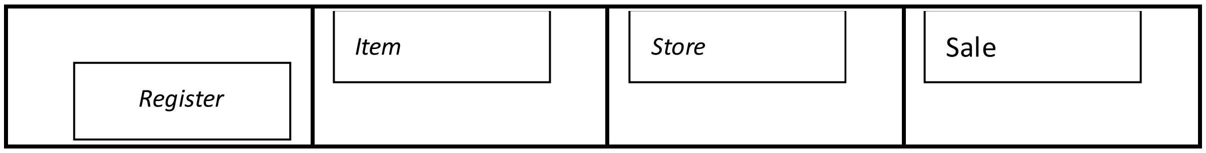
➢ A weakness of this approach is the imprecision of natural language; different noun phrases may represent the same conceptual class or attribute, among other ambiguities.

➢ It is recommended in combination with the Conceptual Class Category List technique.

4.1.1 Example: Find and Draw Conceptual Classes - POS domain

As an example the following are identified list of conceptual classes for Process Sale scenario:

Sale	Cashier
CashPayment	Customer
SalesLineItem	Store
Item	ProductDescription
Register	ProductCatalog
Ledger	



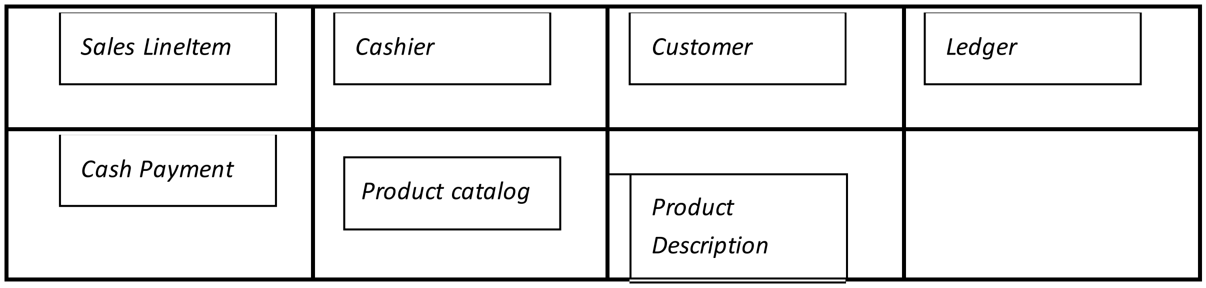


Fig: Initial POS Domain Model

Draw them in UML Class Diagrams – As in below figure

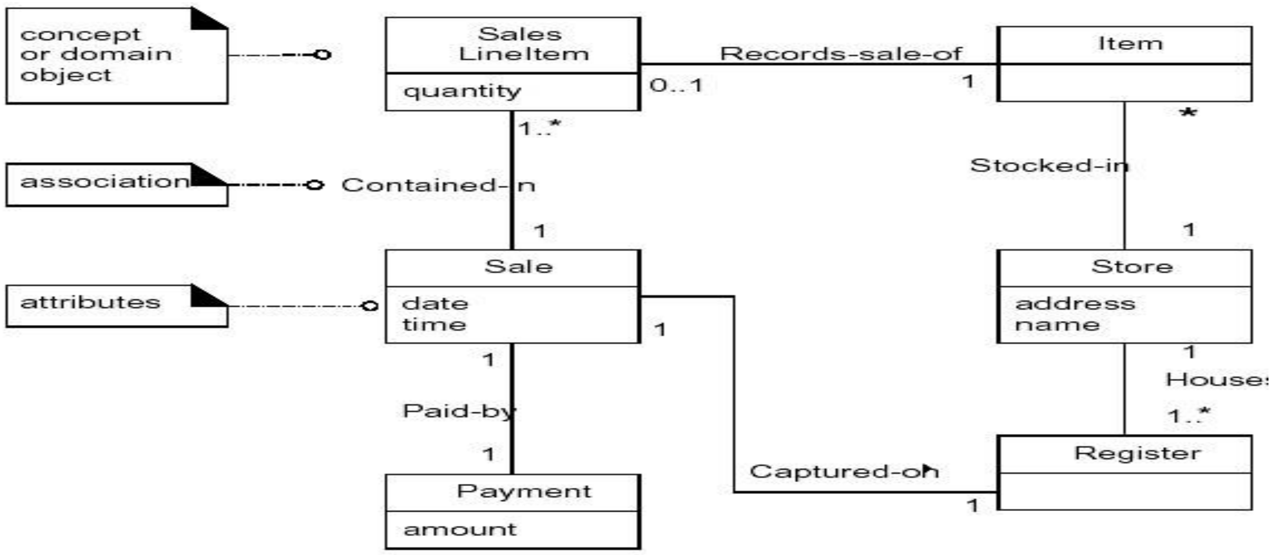


Figure: Example of domain model with UML class Diagram notation

4.2 DESCRIPTION CLASSES

4.2.1 Guideline: When to model with Description classes ➢ A

description class contains information that describes something.

- ➢ For example, a product Description of an item, the price, picture and text description of an item.
- ➢ Test is a description class which is for the description of the tests conducted for some patient. This class contains the information such as name of the test, date on which it is conducted, description of the test, and test results.

4.2.2 Why Description classes are used:

- ➢ The need for description classes is common in many domain models.
- ➢ Following are the situations in which the description class needs to be maintained:
  - Whenever there is a need for describing some item or service then the description class must be defined.
  - When there is a need to reduce the redundant or duplicated information then the description class is used.
  - If there exists some class in such a manner that deleting of this class may loss some other important information which needs to be maintained for the implementation of the system.

4.2.3 Example:

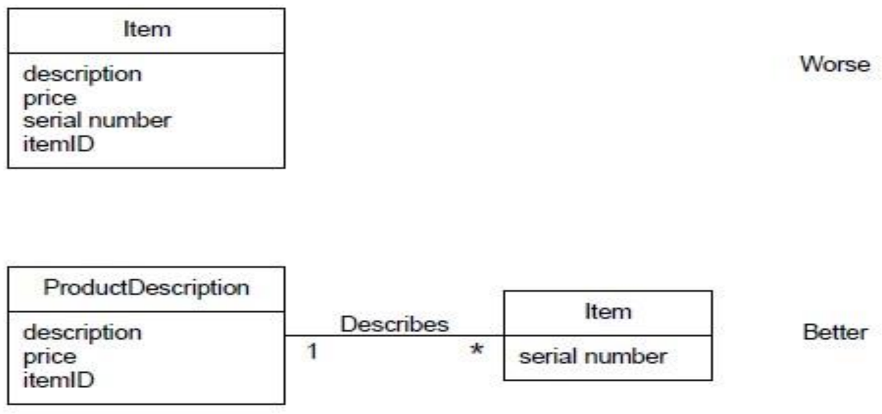


Figure Descriptions about other things. The * means a multiplicity of "many." It indicates that one *ProductDescription* may describe many (*) *Items*.

4.2.4 Guideline-When Are Description Classes Useful

Add a description class when

- There needs to be a description about an item or service, independent of the current existence by any items or services.
- Deleting instances of things they describe results in a loss of information that needs to be maintained, but incorrectly associated with deleted thing.
- It reduces redundancy or duplication of information.

5 Associations

- An association is a relationship between classes (more precisely, instances of these classes) that indicates some meaningful and interesting connections.
- Associations are defined as semantic relationship between two or more classifiers that involve connections among their instances.
- It is represented as a solid line connecting 2 class. Avoid too many associations to a domain model.

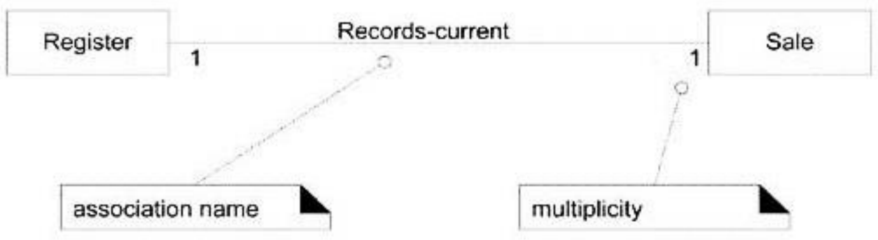


Figure: Association Relationship **How to name an association in UML:**

- Association names should start with a capital letter, since an association represents a classifier of links between instances; in the UML, classifiers should start with a capital letter.

- Association defines the relationship between two or more classes in the System. These generally relates to the one object having instance or reference of another object inside it.

Associations in UML can be implemented using following ways:

A. Multiplicity
B. Aggregation
C. Composition

Roles may optionally have

- Multiplicity expression
- Name
- Navigability

Multiplicity

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

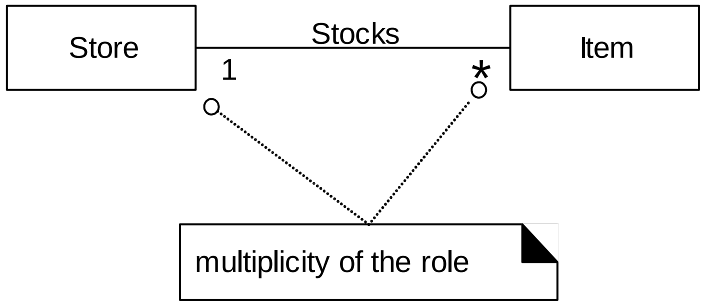
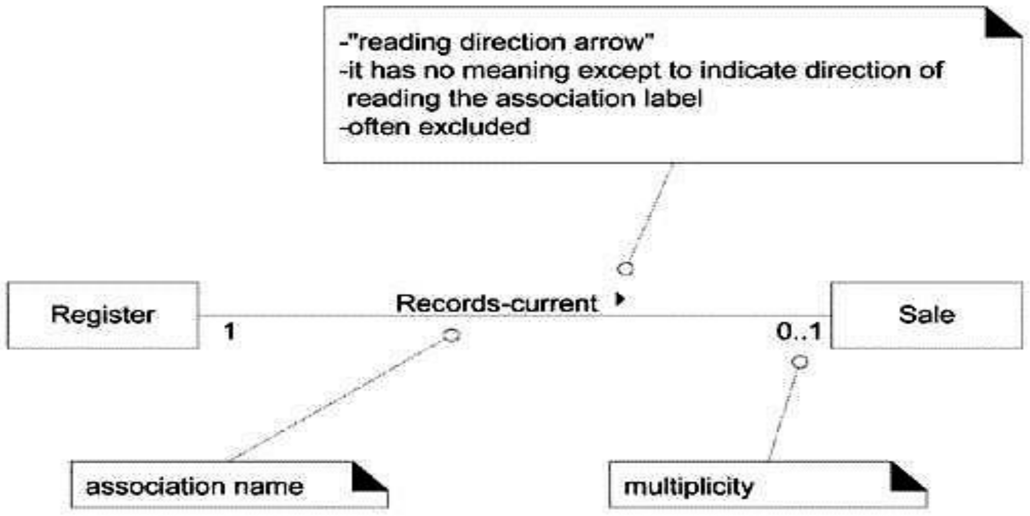


Figure: Multiplicity of an Association

Notation	**Description**
1 0..1	Only One Instance Zero or One Instance
*	Many Instance
0..*	Zero or Many Instance
1..*	One or Many Instance

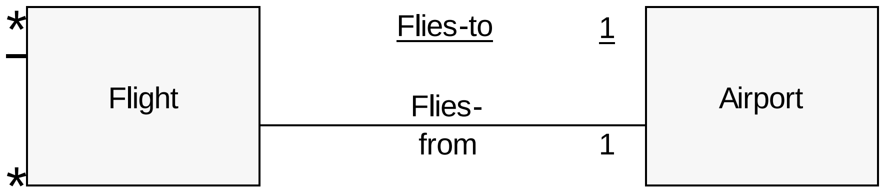
Navigation

UML uses association navigation or navigability to specify a role affiliated with each other at the end of association line. The reading direction arrow indicates the direction to read the association name called navigability.


-"reading direction arrow"
-it has no meaning except to indicate direction of reading the association label
-often excluded
Register
Records-current ▸
Sale
1
0..1
association name
multiplicity

Multiple associations between 2 classes

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



Method of finding Associations using Common Association List

1. POS
2. Monopoly
3. Airline Reservation System

Finding Association With A Common Association List

Category	**Examples**
A is a transaction related to another transaction	CashPayment – Sale Cancellation - Reservation
A is a line item of a transaction B	SalesLineItem - Sale
A is a product or service for a transaction B(or	Item – SalesLineItem
A is a role related to a transaction B	Customer – Payment Passenger - Ticket
A is a physical or logical part of B	Drawer-Register Square-Board Seat-Air plane

A is physically or logically contained in /or B	Register – Store Item-Shelf Square-Board Passenger - Airline
A is a description for B	ProductDescription – Item FlightDescription - Flight
A is known /logged/recorded/reported/captured in B	Sale – Register Piece – Square Reservation - Flightmanifest
A is a member of B	Cashier – Store Player – monnopolyGame Pilot - Airline
A is an organizational Subunit of B	Department – Store Maintenance - Airline
A uses or manages or owns B	Cashier – Register Player – Piece Pilot - Airplane
A is next to B	SalesLineItem – SalesLineItem Square – Square City - City

High-Priority Associations

Some of the high-priority association categories that are in variably useful to include in a domain model:

- A is a *physical or logical part* of B.
- A is *physically or logically contained* in/on B.
- A is *recorded in* B.

Naming Association

Association names should start with a capital letter, since an association represents a classifier of links between instances; in the UML, classifiers should start with a capital letter. Two common and equally legal formats for a compound association name are:

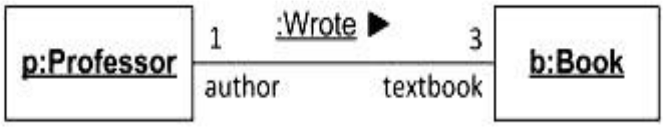
- *Paid-by*
- *PaidBy* **Association Class**

➢ An association may be refined to have its **own set of features**, that is, features that do not belong to any of the connected classifiers but rather to the association itself. Such an association is called an **association class**.

➢ It is both an association, connecting a set of classifiers and a class, and as such could have features and might be included in other associations.

➢ An association class can be seen as an association that also has class properties, or as a class that also has association properties.

- An association class is shown as a class symbol attached to the association path by a dashed line.
- The association path and the association class symbol represent the same underlying model element, which has a single name.
- The association name may be placed on the path, in the class symbol, or on both, but they must be the same name. **Link**
- **Link** is an instance of an association. It is a tuple with one value for the each end of the association, where each value is an instance of the type of the end.
- Association has at least two ends, represented by properties (**end properties**).
- Link is rendered using the same notation as for an association. Solid line connects instances rather than classifiers.
- Name of the link could be shown underlined though it is not required. End names (roles) and navigation arrows can be shown.



Link **Wrote** between instance **p** of **Professor** playing **author** role and instance **b** of **Book** in the **textbook** role.

6 Attributes

An attribute is a logical data value of an object. It is useful to identify those conceptual classes that are needed to satisfy the information requirements of the current scenarios under development.

When to show attributes?

Include attributes that the requirements suggest or imply a need to remember information

UML Notation

In the class representation, attributes are shown in the second compartment. Attribute type is optional.

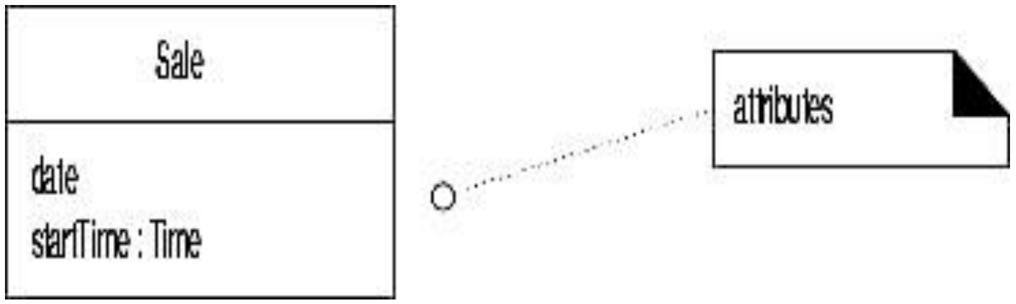


Figure :Attribute Notation

The full syntax for an attribute in the UML is:

Visibility name: type multiplicity=default {property-string}.

Derived attribute:

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

➢ A quantity that can be calculated from other values, such as role multiplicities, is a derived attribute, designated in UML by a leading slash symbol.

➢ The total attribute in the Sale can be calculated or derived from the information in the *SalesLineItem*.

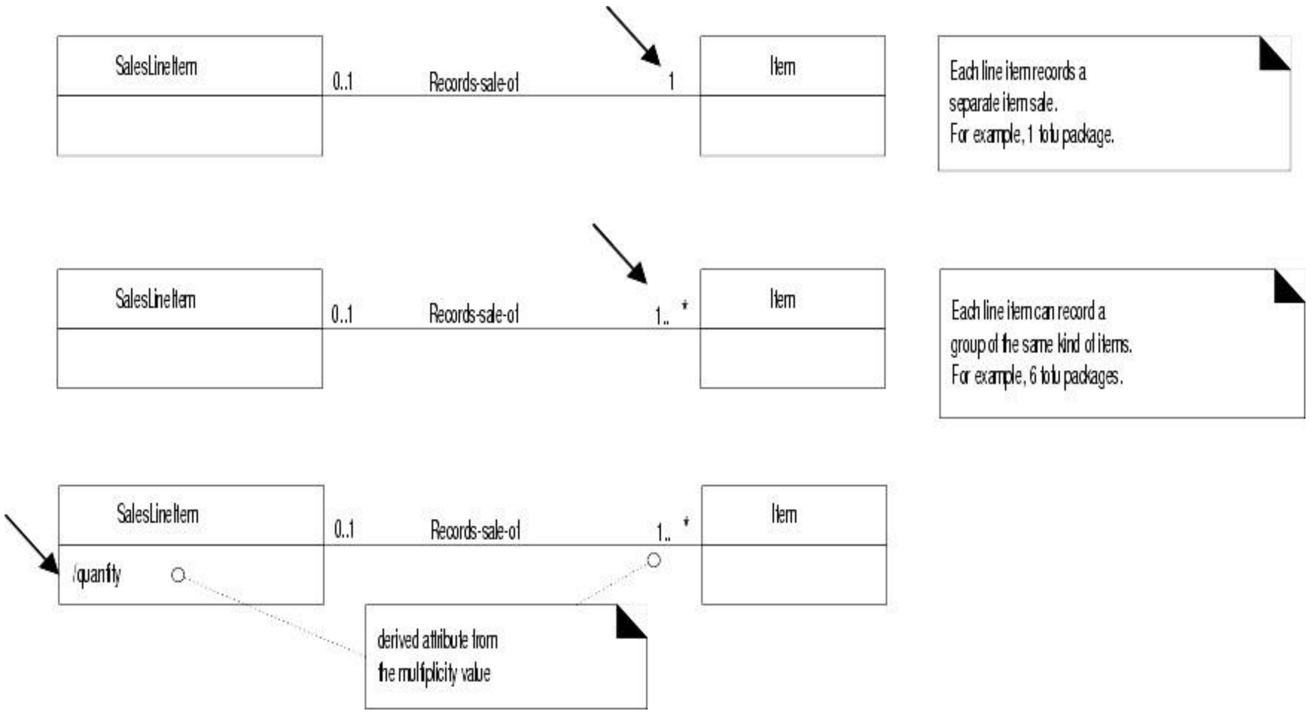

SalesLineItem
0..1
Records-sale-of
1
Item
Each line item records a separate item sale. For example, 1 tofu package.
SalesLineItem
0..1
Records-sale-of
1..*
Item
Each line item can record a group of the same kind of items. For example, 6 tofu packages.
SalesLineItem
/quantity
0..1
Records-sale-of
1..*
Item
derived attribute from the multiplicity value

Figure: Derived Attribute - Recording the quantity of items sold in a line item.

Data type attribute:

It is a primitive data type such as

- Numbers
- Character
- Boolean
- String
- Date(or DateTime)
- Time

Other types used are:

- Address
- Color
- Geometrics
- Phone number   ZIP or postal code

- Enumerations

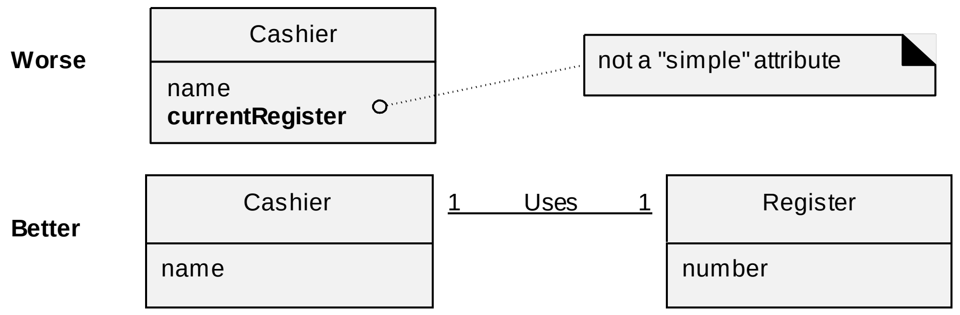


Figure: Relate with association not with attribute

In place of complex concepts, associations are used instead of attributes.

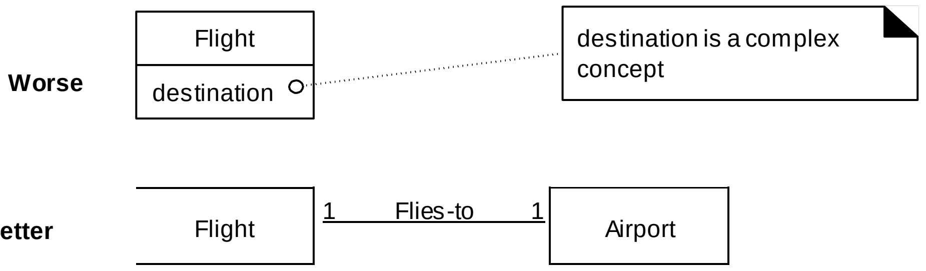
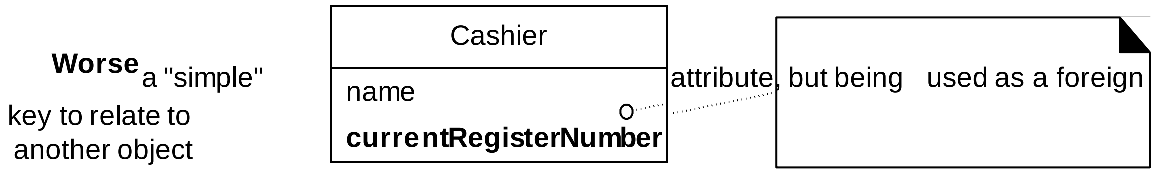


Figure: Don't show complex concept as attributes, use associations

➢ An attribute should be what the UML standard calls a data type: a set of values for which unique identity is not meaningful. Numbers, strings, Booleans, dates, times, phone numbers, and addresses are examples of data types. Values of these types are called value objects.

Relating Types

- Conceptual classes in a domain model should be related by associations, not attributes In particular, an attribute should not be used as a kind of **foreign key**.



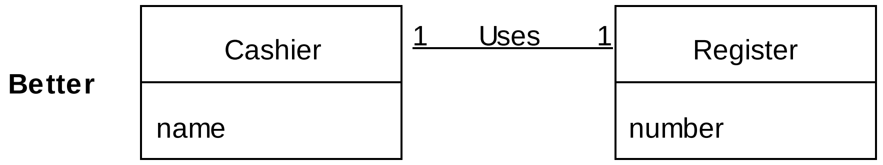
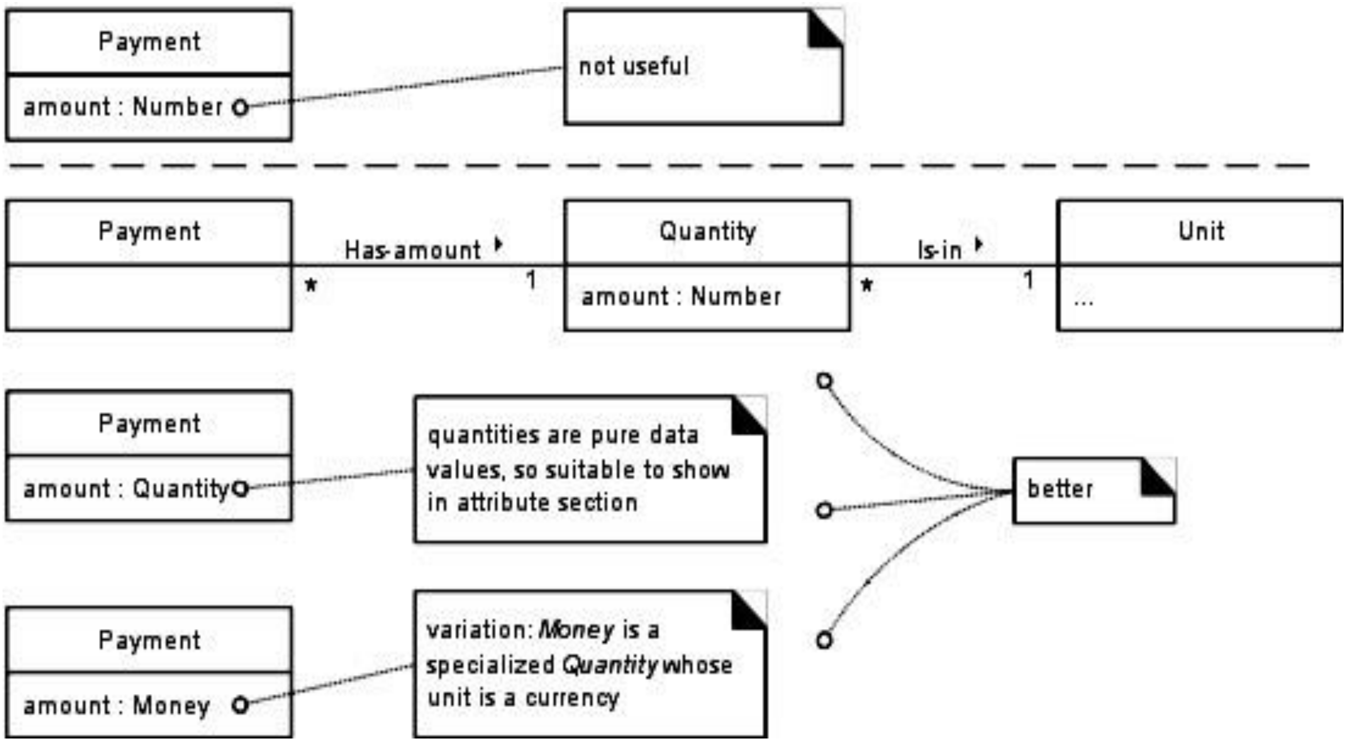


Figure: Don't use attributes as Foreign Key

Quantities and Units

Quantities with associated units should be represented either as conceptual classes or as attributes of specialized types that imply units (e.g. Money or Weight).


Payment
amount : Number
not useful
Payment
Has-amount
*
1
Quantity
amount : Number
Is-in
*
1
Unit
...
Payment
amount : Quantity
quantities are pure data values, so suitable to show in attribute section
better
Payment
amount : Money
variation: Money is a specialized Quantity whose unit is a currency

Guidelines:

Represent what may initially be considered a primitive data type (such as a number or string) as a non-primitive class if:

1. It is composed of separate sections.
   - phone number, name of person
2. There are operations usually associated with it, such as parsing or validation.
   - Social security number
3. It has other attributes.
   - Promotional price could have a start (effective) date and end date
4. It is a quantity with a unit.
   - Payment amount has a unit of currency
5. It is an abstraction of one or more types with some of these qualities.

Item identifier in the sales domain is a generalization of types such as Universal Product Code (UPC) or European Article Number (EAN).

7 Domain model refinement

- ➢ Domain model a representation of real world conceptual classes, not of software components. Domain model is a tool of communication.
- ➢ Domain modeling is driven by use cases as they become known.
- ➢ A report object of other information in domain model is not useful since all its information is derived from other resources.
- ➢ Domain model can be refined by adding the attributes to the class and showing the associations among these classes.

In the domain model refinement, the fundamental concepts are:

1. Generalization
2. Specialization
3. Conceptual class hierarchies **Generalization:**

- ➢ Generalization is the activity of identifying commonality among concepts and defining superclass and subclass relationships.
- ➢ It is a way to construct taxonomic classifications among concepts which are then illustrated in class hierarchies.

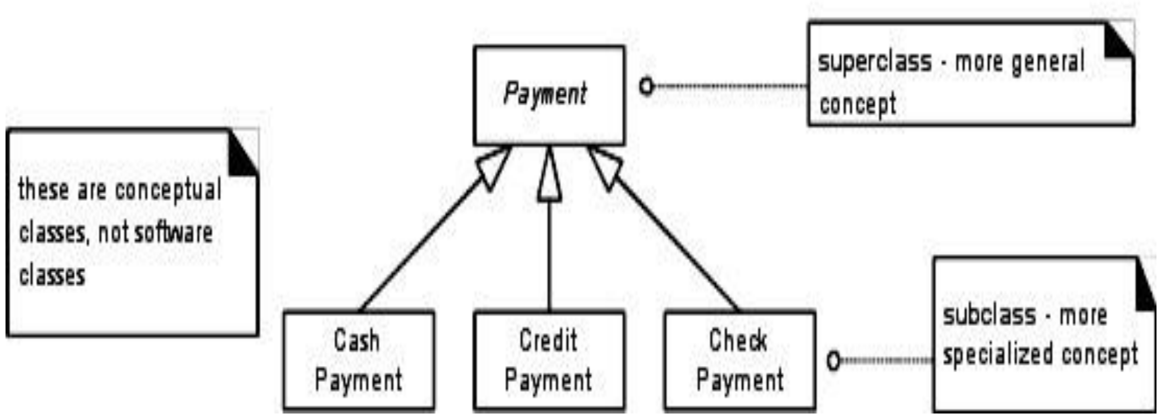

Payment
superclass - more general concept
these are conceptual classes, not software classes
Cash Payment
Credit Payment
Check Payment
subclass - more specialized concept

Figure: Generalization – Specialization Hierarchy

- ➢ For example, the concepts *CashPayment*, *CreditPayment* and *CheckPayment* are all very similar and it is possible to organize them into a **generalization - specification class hierarchy** in which the **superclass** *Payment* represents a more general concept, and the subclasses more specialized ones.
- ➢ Identifying a superclass and subclasses is of value in a domain model because their presence allows us to understand concepts, refined and abstract terms.
- ➢ A conceptual superclass definition is more general or encompassing than a subclass definition. ➢ **Superclass:** *Payment*
- ➢ **Subclass:** *Cash Payment*, *Credit Payment*, *Check payment*

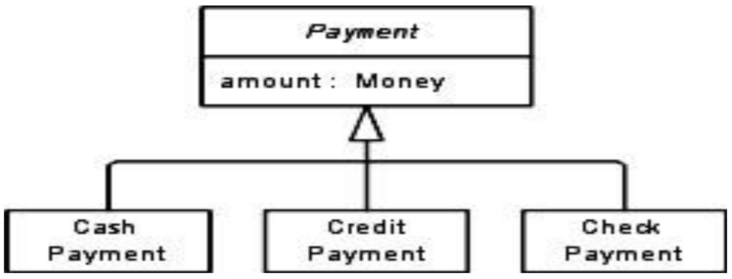


Figure: Payment class hierarchy

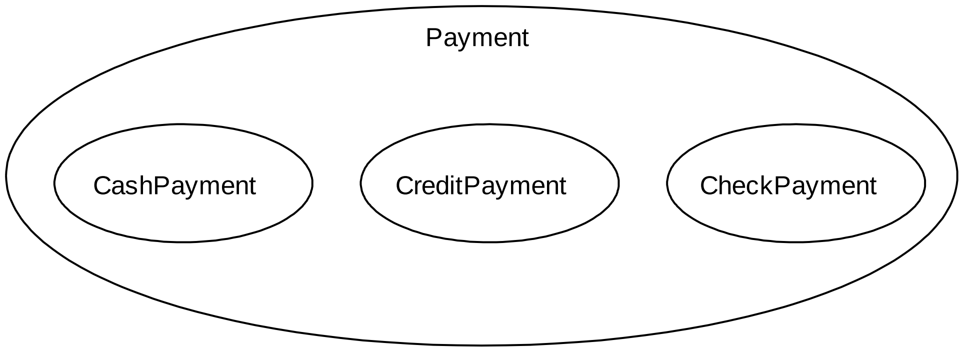


Figure: Venn diagram of set relationships

8 Finding conceptual class Hierarchies

When a class hierarchy is created, statements about super classes that apply to subclasses are made.

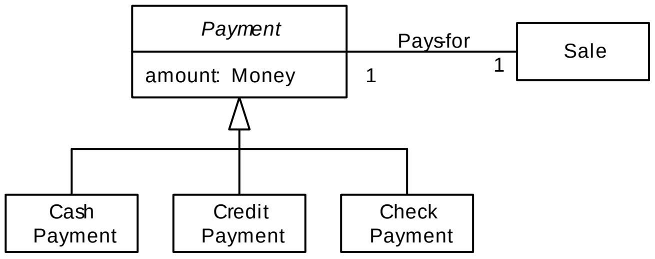


Figure: Subclass conformance

Guideline: 100% Rule

100% of the conceptual superclass's definition should be applicable to the sub-class. The subclass must conform to 100% of the superclass's. ▪ *Attributes*

▪ *associations*

Guideline: Is a Rule

All the members of a subclass set must be members of their superclass set.

*In natural language, this can usually be informally tested by forming the statement: Subclass **is a** Superclass*

WHEN TO DEFINE A CONCEPTUAL SUBCLASS?

A **Conceptual Class Partition** is a division of a conceptual class into disjoint subclasses.

8.1.1 Motivations to partition a conceptual class into subclasses

Create a conceptual subclass of a superclass when:

1. The subclass has additional attributes of interest.
2. The subclass has additional associations of interest.
3. The subclass concept is operated on, handled, reacted to, or manipulated differently than the superclass or other subclasses, in ways that are of interest.
4. The subclass concept represents an animate thing that behaves differently than the superclass or other subclasses, in ways that are of interest.

WHEN TO DEFINE A CONCEPTUAL SUPERCLASS CLASS?

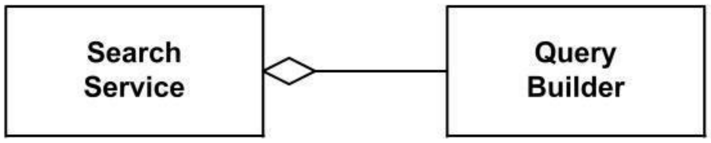
8.1.2 Motivations to generalize and define a superclass:

1. The potential conceptual subclasses represent variations of a similar concept.
2. The subclasses will conform to the 100% and Is-a rule.
3. All subclasses have the same attribute that can be factored out and expressed in the super class.
4. All subclasses have the same association that can be factored out and expressed in the super class.

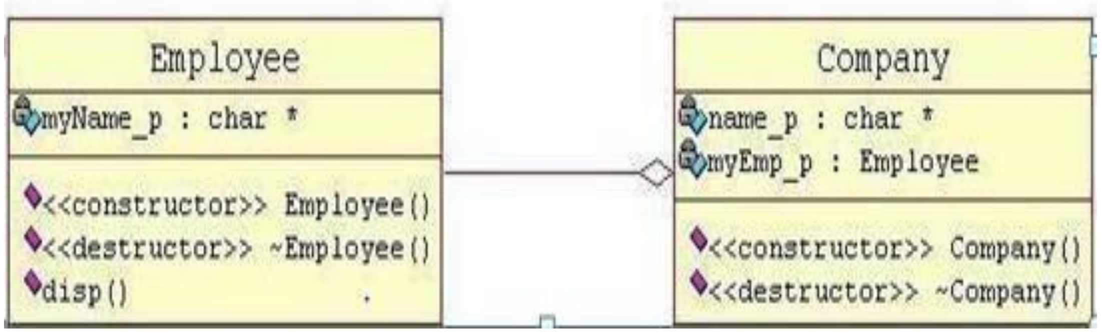
9 Aggregation and Composition

Aggregation

- Aggregation is a vague kind of association in the UML that loosely suggests whole/part relationship. Aggregation type could be either: ▪ shared aggregation (aggregation) or
  - ▪ composite aggregation (composition).
- Aggregation is a specialize form of Association where all object have their own life cycle but there is an ownership like parent and child. Child object cannot belong to another parent object at the same time.
- It is similar to "has-a" relationship.
- Aggregation (shared aggregation) is a "weak" form of aggregation when part instance is independent of the composite:
- The same (shared) part could be included in several composites, and if composite is deleted, shared parts may still exist.
- Shared aggregation is shown in the UML with a **hollow or filled diamond symbol** at the aggregate end of the association line.

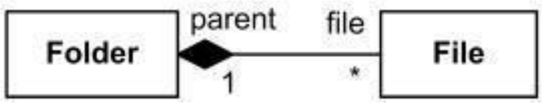


Employee class has Agregation Relatioship with Company class

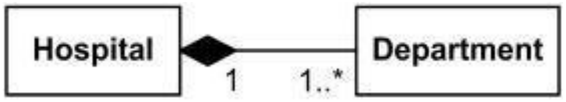

Employee
myName_p : char *
<<constructor>> Employee()
<<destructor>> ~Employee()
disp()
Company
name_p : char *
myEmp_p : Employee
<<constructor>> Company()
<<destructor>> ~Company()

Composition

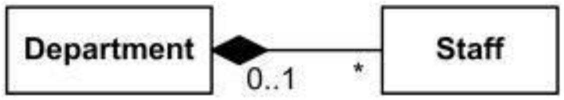
- Composition is again specialize form of Aggregation. It is a strong type of aggregation. **Composition** (**composite aggregation**) is a "strong" form of aggregation. Composition requirements/features listed in UML specification are
  - it is a **whole/part relationship**,
  - it is binary association,
  - part could be included in **at most one** composite (whole) at a time, and
  - if a composite (whole) is deleted, all of its composite parts are "normally" deleted with it.
  - It shows string ownership to represent the component of complex object.
- Note, that UML does not define how, when and specific order in which parts of the composite are created. Also, in some cases a part can be removed from a composite before the composite is deleted, and so is not necessarily deleted as part of the composite.
- Composite aggregation is depicted as a binary association decorated with a **filled black diamond** at the aggregate (whole) end.



- Folder could contain many files, while each File has exactly one Folder parent. If Folder is deleted, all contained Files are deleted as well.
- When composition is used in **domain models**, both whole/part relationship as well as event of composite "deletion" should be interpreted figuratively, not necessarily as physical containment and/or termination. UML specification needs to be updated to explicitly allow this interpretation



- Hospital has 1 or more Departments, and each Department belongs to exactly one Hospital. If Hospital is closed, so are all of its Departments.
- Note, that though it seems odd, multiplicity of the composite (whole) could be specified as **0..1** ("at most one") which means that part is allowed to be a "stand alone", not owned by any specific composite.



- Each Department has some Staff, and each Staff could be a member of one Department (or none). If Department is closed, its Staff is relieved (but excluding the "stand alone" Staff).

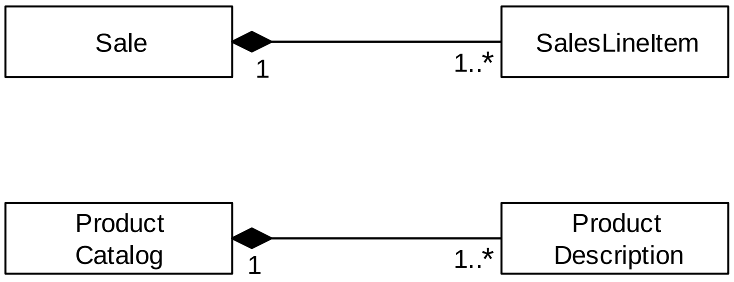


Figure: Composition

Benefits of showing Composition

- It clarifies the domain constraints regarding the eligible existence of the part independent of the whole. In composite aggregation, the part may not exist outside of the lifetime of the whole.
- During design work, this has an impact on the create-delete dependencies between the whole and part software classes and database elements (in terms of referential integrity and cascading delete paths).
- It consists in the identification of a creator (the composite) using the GRASP Creator Pattern.
- Operations – such as copy and delete – applied to the whole often propagate to the parts.

Problems in composition:

- The definition for composition is quite strict, and involves the following rules:
  - whole-part relationship (it is an aggregation).
  - strong (not shared) ownership of part.
  - coincident lifetimes of whole and part.
  - addition/removal of parts allowed for non-fixed multiplicity.
- The first rule is from the definition of an aggregation. A composition association is an aggregation, and so a composition must still represent a whole-part relationship.
- The second rule concerns the ownership of the parts by the whole.

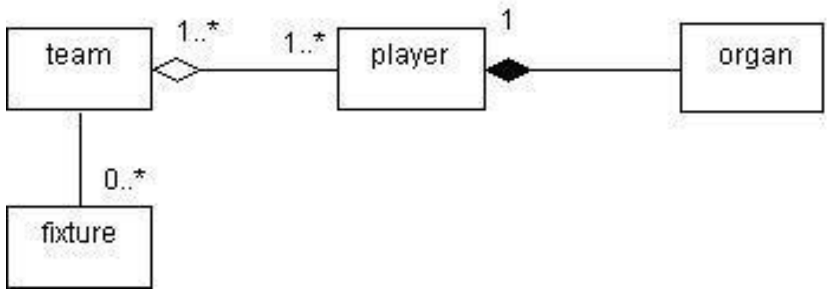


Fig: Association, aggregation and composition

Composition in NextGen POS Domain Model

- In the POS domain, the *SalesLineItem* may be considered as a part of a composite sale.
- Transaction line items are viewed as parts of an aggregate transaction.
- There is a create delete dependency of the line items on the sale their life time is bound within the lifetime of the sale.
- Similarly, *ProductCatalog* is a composite of *ProductDescriptions*.
- No other relationship is a compelling combination that suggests whole-part semantics, a create-delete dependency, and ―If in doubt, leave it out‖

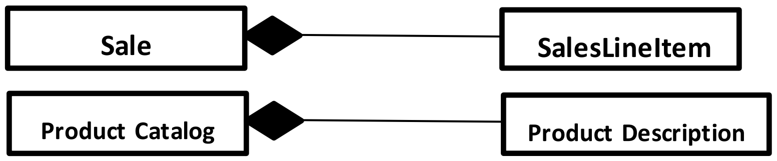


Figure: Aggregation in POS application

10 Relationship between Sequence Diagrams and Use Cases

- Sequence diagram is used to represent one scenario of the use case. Hence it can be derived by observing the use cases.
- A use case diagram just reflects the interaction of user with the system components, whereas the sequence diagram has a set of messages that are exchanged between the objects in the system.
- Sequence diagrams cannot completely describe the system behavior, but they can specify typical use cases for the system, small details in its behavior, and simplified overviews of its behavior.
- The use case diagrams are transformed into object sequence diagrams that define the interactions in terms of public method.
- An SSD –**System Sequence Diagram** shows system events for a scenario of a use case, therefore it is generated from inspection of a use case.

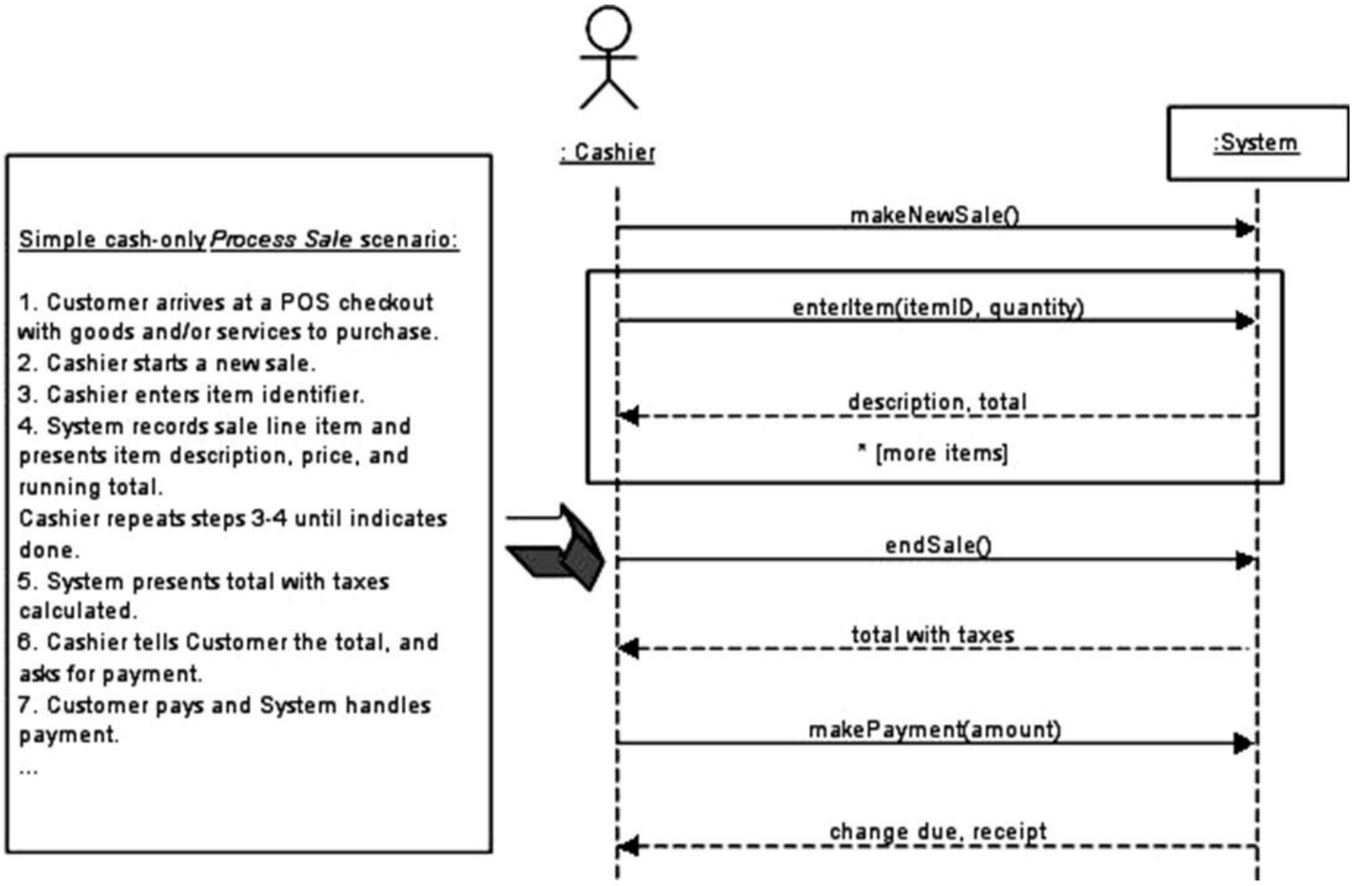

: Cashier
:System
Simple cash-only Process Sale scenario:
1. Customer arrives at a POS checkout with goods and/or services to purchase.
2. Cashier starts a new sale.
3. Cashier enters item identifier.
4. System records sale line item and presents item description, price, and running total.
Cashier repeats steps 3-4 until indicates done.
5. System presents total with taxes calculated.
6. Cashier tells Customer the total, and asks for payment.
7. Customer pays and System handles payment.
...
makeNewSale()
enterItem(itemID, quantity)
description, total
* [more items]
endSale()
total with taxes
makePayment(amount)
change due, receipt

Figure: Relationship between SSD and Use Case

- To identify system events, it is necessary to be clear on the choice of system boundary. For the purposes of software development, the system boundary is usually chosen to be the software system itself; in this context, a system event is an external event that directly stimulates the software.
- System events (and their associated system operations) should be expressed at the level of intent rather than in terms of the physical input medium or interface widget level. It also improves clarity to start the name of a system event with a verb Thus "enter item" is better than "scan" (that is, laser scan).

11 When to Use Class Diagrams

- Class diagram is a static diagram and it is used to model the static view of a system. The static view describes the vocabulary of the system.
- Class diagram clearly shows the mapping with object oriented language such as Java, C++ etc. Practically class diagram is generally used for construction purpose.
- Class diagram is also considered as the foundation for component and deployment diagrams.
- In a nutshell it can be said, class diagrams are used for:
  - Describing the static view of the system
  - Showing the collaboration among the elements of the static view.

- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

Part -A

1) Define UML class diagram.

UML class diagrams illustrate classes, interfaces and their associations. They are used for static object modeling.

2) Write the common uses of class diagram.

Class diagrams are used for:

- ✓ Describing the static view of the system.
- ✓ Showing the collaboration among the elements of the static view.
- ✓ Describing the functionalities performed by the system.
- ✓ Construction of software applications using object oriented languages.

3) List the relationships used in class diagram.[May 2011][Nov 2013][May 2014][Nov 2014] [Nov 2015]

- ✓ Association
- ✓ Dependency
- ✓ Aggregation
- ✓ Composition
- ✓ Generalization

4) Define classifier. A UML classifier is ―a model element that describes behavioral and structure feature‖. Classifiers can also be specialized. They are a generalization of many of the elements of the UML, including classes, interfaces, use cases and actors.

5) Define attribute.

Attributes are also called as **structural properties** in the UML. Attributes of a classifier are shown as:

- ✓ Attribute text notation
- ✓ Association line notation
- ✓ Both together

6) What is the syntax for an attribute in UML? The full syntax of the attribute text notation is: **visibility name: type multiplicity = default { property-string }**

7) What is generalization?

Generalization is the activity of identifying commonality among concepts and defining superclass and subclass relationships. For example, the concepts *CashPayment, CreditPayment* and *CheckPayment* are all very similar and it is possible to organize them into a generalization, specification, class hierarchy.

8) Define aggregation. [May 2014] [Nov 2013][May 2013][Nov 2014] [May 2019]
Aggregation is a vague kind of association in the UML that loosely suggests Whole -part relationships. It has no meaningful distinct semantics in the UML versus a plain association, but the term is defined in the UML.

9) Define composition. [May 2014] [Nov 2013][May 2013][Nov 2014][Nov 2017]
Composition, also known as **composite aggregation**, is a strong kind of whole –part aggregation and is useful to show in some models. It is an instance of a part belongs to only one composite instance. **For Example:** Steering and car are the two classes those can be associated by composite relationship.

10) What is UML constraint?
A UML constraint is a restriction or condition on a UML element. It is visualized in text between braces. **Example: {size>=0}**

11) Define qualified association.
A qualified association has a qualifier that is used to select an object (or objects) from a larger set of related objects, based upon the qualifier key.

12) Define association class.
An association class allows us to treat an association itself as a class, and model it with attributes, operations, and other features.

13) What is Navigability?
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

14) What is a dependency relationship?
✓ The UML **dependency relationship** indicates that one element (of any kind, including classes, use cases, and so on) has knowledge of another element.
✓ It is illustrated with a dashed arrow line.
✓ In class diagrams the dependency relationship is useful to depict non-attribute visibility between classes; in other words, parameter, global, or locally declared visibility.

15) What is class?
A class is an encapsulation of attributes and methods. It is a collection of objects. It encapsulates both data and behaviour.

16) Provide the two perspectives to Design class diagram.
1. Conceptual Perspective – class diagram is used to visualize a domain model 2.
Software Perspective – class diagram is used to visualize a Design Model **17)**
Define Refinement. [May 2019]

Refinement is the counterpart of abstraction. Abstracting adesign means leaving outirrelevant details of that design. In that case a higher level design is obtained. By refininga design, on the other hand, a lower level design is obtained,that has more detail than theoriginal design.

18) Difference between Aggregation and composition.

Aggregation	Composition
„Has a' relationship	„Part of' relationship
In aggregation there exhibit a relationship where a child can exist independently of the parent.	In composition they cannot exist independently of the parent.
Hollow Diamond	Solid Diamond

19) When to use class diagram?

Class diagrams are used for:

- Describing the static view of the system
- Showing the collaboration among the elements of the static view.
- Describing the functionalities performed by the system.
- Construction of software applications using object oriented languages.

20) Define Elaboration. (or)What do you understand from the term Elaboration? [May-2012, 2013, 2014] [Nov 2014]

Elaboration is the initial series of iterations during which the team does serious investigation, implements (programs and tests) the core architecture, clarifies most requirements, and tackles the high - risk issues. It Build the core architecture, resolve the high-risk elements, define most requirements, and estimate the overall schedule and resources.

21) What are the tasks performed in elaboration? [Nov-2015][May 2018]

- the core, risky software architecture is programmed and tested
- the majority of requirements are discovered and stabilized
- the major risks are mitigated or retired

22) Define Domain Model.

In the UP, ―Domain Model‖ means a representation of real-situation conceptual classes, not of software objects. The term does not mean a set of diagrams describing software classes, the domain layer of a software architecture or software objects with responsibilities.

23) why call a domain model a "visual dictionary"? [Nov 2016] ✓ It visualizes and relates words or concepts in the domain.

- ✓ It also shows an abstraction of the conceptual classes
- ✓ The information it illustrates (using UML notation) could alternatively have been expressed in plain text (in the UP glossary).

- ✓ The domain model is a visual dictionary of the noteworthy abstractions, domain vocabulary, and information content of the domain.

24) What is conceptual class?

A conceptual class is an idea, thing, or object.

A conceptual class may be considered in terms of its symbol, intension, and extension.

- **Symbol** – words or images representing a conceptual class.
- **Intension** – the definition of a conceptual class.
- **Extension** – the set of examples to which the conceptual class applies.

25) When to create a subclass of a superclass? [Nov 2017] Create a conceptual subclass of a superclass when:

1. The subclass has additional attributes of interest.
2. The subclass has additional associations of interest.
3. The subclass concept is operated on, handled, reacted to, or manipulated differently than the superclass or other subclasses, in ways that are of interest.
4. The subclass concept represents an animate thing that behaves differently than the superclass or other subclasses, in ways that are of interest.

26) What is the relationship of a conceptual super class to a subclass?[May 2017]

Definition: A conceptual superclass definition is more general or encompassing than a subclass definition.

For example, consider the superclass ***Payment***and its subclasses ***(CashPayment,*** and so on). Assume the definition ***of Payment*** is that it represents the transaction of transferring money (not necessarily cash) for a purchase from one party to another, and that all payments have an amount of money transferred. The model corresponding to this is shown in Figure.

27) Specify the three strategies to find conceptual classes.

Three strategies to find conceptual classes:

1. Reuse or modify existing models.
2. Use a category list
3. Identify noun phrases.

28) How to create a domain model? [Nov-2015][Nov 2016] Bounded by the current iteration requirements under design:

1. Find the conceptual classes
2. Draw them as classes in a UML class diagram.
3. Add associations and attributes.

29) Define Association.

An association is a relationship between classes (more precisely, instances of these classes) that indicates some meaningful and interesting connections.

Associations are defined as semantic relationship between two or more classifiers that involve connections among their instances.

30) What is the purpose of association relationship? [May 2015]

Association is a relationship between classifiers which is used to show that instances of classifiers could be either linked to each other or combined logically or physically into some aggregation.

31) What is an association rule?

Association names should start with a capital letter, since an association represents a classifier of links between instances; in the UML, classifiers should start with a capital letter.

32) Define Multiplicity.[Nov 2017]

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

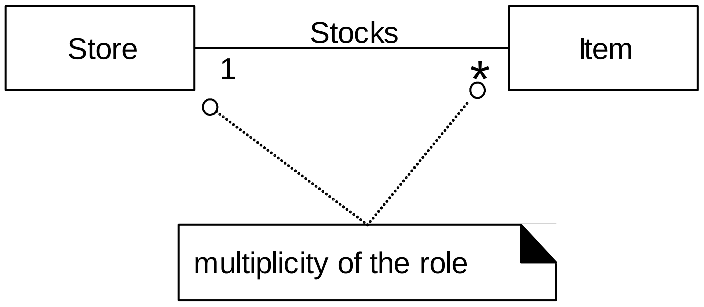


Figure: Multiplicity of an Association

33) What is the relationship between sequence diagram and use case?

An SSD –**System Sequence Diagram** shows system events for a scenario of a use case, therefore it is generated from inspection of a use case.

34) Define System Events and the System Boundary. [Nov 2016]

To identify system events, it is necessary to be clear on the choice of system boundary. For the purposes of software development, the system boundary is usually chosen to be the software system itself; in this context, a system event is an external event that directly stimulates the software.

35) How to Name System Events and Operations? [Nov 2016]

System events (and their associated system operations) should be expressed at the level of intent rather than in terms of the physical input medium or interface widget level. It also improves clarity to start the name of a system event with a verb Thus "enter item" is better than "scan" (that is, laser scan).

PART - B

1. Describe UML notation for class diagram with an example. Explain the concept of link, association and inheritance. [Nov-2014]

   (or)

   Describe the UML notation for class diagram with an example. [May 2013]

   (or)

   With a suitable example, explain how to design a class. Give all possible representation in a class (such as: name, attribute, visibility, methods, and responsibilities). [Nov 2012]
2. Write about elaboration and discuss the differences between differences between Elaboration and Inception with suitable diagram for university domain. [May 2017][Nov 2015][Nov 2013]
3. Construct design for Library Information System which comprises and following notations. i. Aggregations
   ii. Compositions
   iii. Associations [Nov 2015]
4. Discuss in detail about the three strategies to find conceptual classes. [Nov 2016-8M]
5. Explain association, aggregation and composition relationships in detail. [Nov 2016 - 8M] [May 2017 -6M]
6. Explain the guidelines for finding conceptual classes with neat diagram. [May 2017 -10M
7. Illustrate with example the relationship between sequence diagram and use cases. [May 2014 , Nov 2014]

   (or)
8. What is the relation between sequence diagrams and use cases. Take an example to show the relationship, highlighting the advantages. [May 2019][May 2015][Nov 2016]
9. Explain with an example aggregation and composition. [May 2019-8M] [Nov 2017 -6M]
10. What is multiplicity of an association? Explain with an example the different types of multiplicities. [Nov 2017 -7M]
11. Explain with an example generalization and specialization and write a note on abstract class and abstract operation. [Nov 2017 -8M]
12. List and describe the three strategies to identify conceptual classes. [Nov 2017 -16M]
13. Write a short note on domain models. [May 2019-6M]
14. Explain in detail about conceptual classes and description classes. [May 2019-8M]