

# Software Engineering - I

An Introduction to Software Construction Techniques for Industrial Strength Software

## Chapter 7 – Object Oriented Analysis and Design

## **1. Object Oriented Design - Why?**

Software is primarily used to represent real-life players and processes inside a computer. In the past, software was considered as a collection of information and procedures to transform that information from input to the output format. There was no explicit relationship between the information and the processes which operate on that information. The mapping between software components and their corresponding real-life objects and processes was hidden in the implementation details. There was no mechanism for sharing information and procedures among the objects which have similar properties. There was a need for a technology which could bridge the gap between the real-life objects and their counter-parts in a computer. Object oriented technology evolved to bridge the gap. Object-oriented technology helps in software modeling of real-life objects in a direct and explicit fashion, by encapsulating data and processes related to a real-life object or process in a single software entity. It also provides a mechanism so that the object can inherit properties from their ancestors, just like real-life objects.

A complex system that works is invariably found to have evolved from a simple system that worked. The structure of a system also plays a very important role. It is likely that we understand only those systems which have hierarchical structure and where intra-component linkages are generally stronger than inter component linkages. That leads to loose coupling, high cohesion and ultimately more maintainability which are the basic design considerations. Instead of being a collection of loosely bound data structures and functions, an object-oriented software system consists of objects which are, generally, hierarchical, highly cohesive, and loosely coupled.

Some of the key advantages which make the object-oriented technology significantly attractive than other technologies include:

- Clarity and understandability of the system, as object-oriented approach is closer to the working of human cognition.
- Reusability of code resulting from low inter-dependence among objects, and provision of generalization and specialization through inheritance.
- Reduced effort in maintenance and enhancement, resulting from inheritance, encapsulation, low coupling, and high cohesion.

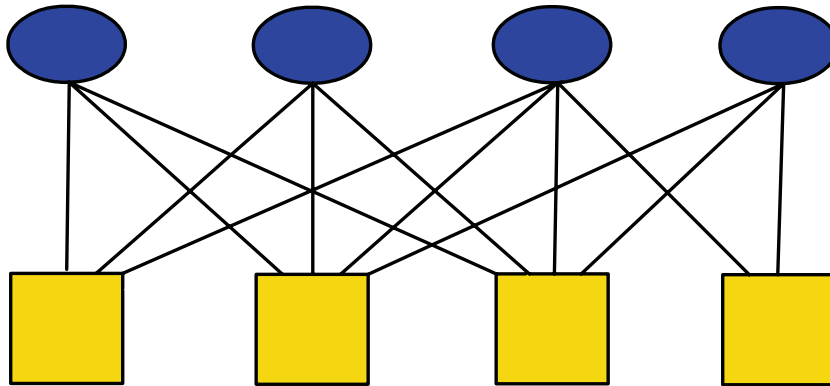
### **1.1. Difference between object-oriented and function-oriented design**

Before talking about how to derive and object-oriented design, we first need to understand the basic difference between object-oriented and function oriented (or action oriented) approach.

In the case of action-oriented approach, data is decomposed according to functionality requirements. That is, decomposition revolves around function. In the OO approach, decomposition of a problem revolves around data. Action-oriented paradigm focuses only on the functionality of a system and typically ignores the data until it is required. Object-oriented paradigm focuses both on the functionality and the data at the same time. The basic difference between these two is decentralized control mechanism versus centralized control mechanism respectively. Decentralization gives OO the ability to handle essential complexity better than action-oriented approach.

This difference is elaborated with the help of the following diagram:

## Functions



## Data

In this diagram, the ovals depict the function while rectangles/squares depict data. Since a function contains dynamic information while data contains only static information, if the function and data are managed separately, the required data components can be found by scanning a function but the functions that use a particular data cannot be found by just looking at the data. That is, the function knows about the data it needs to use but the data do not know about the functions using it. That means, it is easy to make a change in a function since we would know which data components would be affected by this change. On the other hand, changing a data structure would be difficult because it would not be easy to find all the functions that are using this data and hence also need to be modified.

In the case of OO design since data and function are put together in one class, hence, in case of a change, the effected components can be identified easily and the effect of change is localized. Therefore, maintenance becomes relatively easy as compared to function-oriented approach.

## 2. Object Oriented Design Components - What?

### 2.1. The Object and the Class

The basic unit of object oriented design is an object. An object can be defined as a tangible entity that exhibits some well defined behavior. An object represents an individual, identifiable item, unit, or entity, either real or abstract, with a well defined role in the problem domain. An object has state, behavior, and identity.

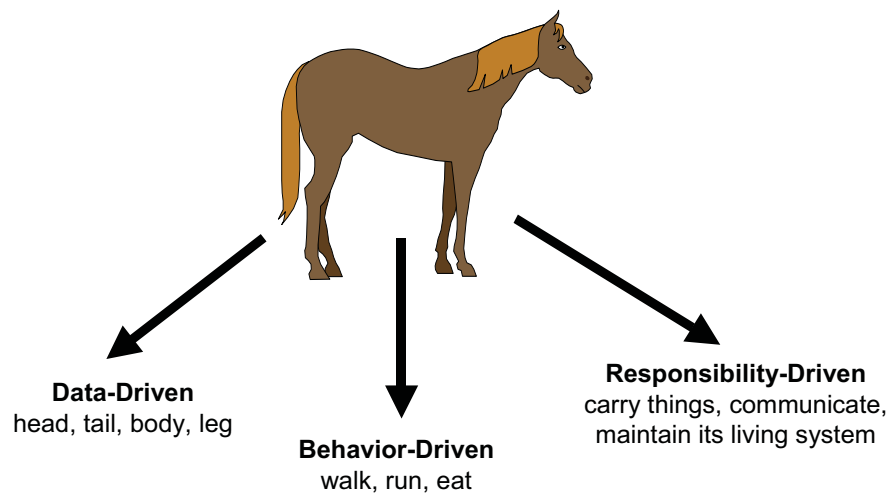
The state of an object encompasses all of the properties of the object and their current values. A property is an inherent or distinctive characteristic. Properties are usually static. All properties have some value. The state of an object is encapsulated within the object.

Behavior is how an object acts and reacts in terms of its state changes and message passing. The behavior of an object is completely defined by its actions. A message is some action that one object performs upon another in order to elicit a reaction. The operations that clients may perform upon an object are called methods.

The structure and behavior of similar objects are defined in their common class. A class represents an abstraction - the essence or the template of an object. A class specifies an interface (the outside view - the public part) and defines an implementation (the inside view - the private part). The interface primarily consists of the declaration of all the operations applicable to instances of this class. The implementation of a class primarily consists of the implementation of all the operations defined in the interface of the class

## 2.2. Classification

The most important and critical stage in the OOA and OOD is the appropriate classification of objects into groups and classes. Proper classification requires looking at the problem from different angles and with an open mind. When looked at from different perspectives and analyzed with different set of characteristics, same object can be classified into different categories. Let us try to understand this with the help of an example.



Here, we can take a data-driven, behaviour driven, or responsibility driven perspective and will categorize the horse accordingly.

## 2.3. The Object Model

The elements of object oriented design collectively are called the Object Model. The object model encompasses the principles of abstraction, encapsulation, and hierarchy or inheritance.

Abstraction is an extremely powerful technique for dealing with complexity. Unable to master the entirety of a complex object, we ignore its essential details, dealing instead with generalized, idealized model of the object. An abstraction focuses on the outside view of an object, and hence serves to separate an objects external behavior from its implementation. Deciding upon the right set of abstractions for a given domain is the central problem in object oriented design.

Abstraction and encapsulation are complementary concepts. Abstraction provides the outside view to the client and encapsulation prevents clients from seeing its inside view. For abstraction to work, implementation must be encapsulated. Encapsulation hides the details of the implementation of an object. Intelligent encapsulation localizes design decisions that are likely to change. The ability to change the representation of an object without disturbing any of its clients is the essential benefit of encapsulation.

### 2.3.1. Relationship Among Objects

The object model presents a static view of the system and illustrates how different objects collaborate with one another through patterns of interaction. Inheritance, association and aggregation are the three inter-object relationships specified by the object model.

Inheritance defines a “kind of” hierarchy among classes. By inheritance, we specify generalization/specialization relationship among objects. In this relationship, a class (called the subclass) shares the structure and behavior defined in another class (called the superclass). A subclass augments or redefines the existing structure and behavior of its superclass. By classifying objects into groups of related abstractions, we come to explicitly distinguish the common and distinct properties of different objects, which further help us to master their inherent complexity. Identifying the hierarchy within a complex system requires the discovery of patterns among many objects.

In an association relationship, when object A “uses” object B, then A may send messages to B. The relationship defines visibility among objects.

The aggregation relationship defines part-of structure among objects. When object A is part of the state of object B, A is said to be contained by B. There are some tradeoffs between aggregation and association relationships. Aggregation reduces the number of objects that must be visible at the level of enclosing objects and may lead to undesirable tighter coupling among objects.

### 2.3.2. Aggregation and Association - Conceptual and Implementation Issues and Differences

#### 2.3.2.1. Association and Aggregation - Some basic differences

Objects do not exist in isolation. They rather collaborate with one another in many different ways to achieve an overall goal. The different types of relationships in which these objects are involved include association, aggregation, and inheritance. Briefly, inheritance denotes a “kind of” relationship, aggregation denotes a “part of” relationship, and association denotes some semantic connection among otherwise unrelated classes. Any further elaboration on inheritance relationship is beyond the scope of this discussion and therefore we shall concentrate on aggregation and association relationships only.

As mentioned earlier, aggregation is the “part-whole” or “a-part-of” relationship in which objects representing the components of something are encapsulated within an object representing the entire assembly. In other words, the whole is meaningless without its parts and the part cannot exist without its container or assembly. Some properties of the assembly propagate to the components as well, possibly with some local modifications. Unless there are common properties of components that can be attached to the assembly as a whole, there is little point in using aggregation. Therefore, as compared to association, aggregation implies a tighter coupling between the two objects which are involved in this relationship. Therefore, one way to differentiate between aggregation and association is that if the two objects are tightly coupled, that is, if they cannot exist independently, it is an aggregation, and if they are usually considered as independent, it is an association.

#### 2.3.2.2. Object Creation and Life Time

From the object creation and life time point of view, when an object is instantiated, all of its parts must also be instantiated at the same time before any useful work can be done and all of its part die with it. While in the case of association, the life time of two associated object is independent

of one another. The only limitation is that an object must be alive or has to be instantiated before a message can be sent to it.

### 2.3.2.3. Coupling and Linkages

As mentioned earlier, aggregation implies a much tighter coupling than association. In case of aggregation, the links between the whole and its part are permanent while in case of association the links may be maintained only just for the period an object requires the services of its associated object and may be disconnected afterwards.

### 2.3.2.4. Ownership and visibility

Another way of differentiating among the two is to look at them from the ownership and sharing point of view. In case of aggregation, since the whole contains the part, the part is encapsulated or hidden within the whole and is not accessible from outside while in case of association, the associated object may be used directly by other objects also. That is, in case of aggregation, only the whole is supposed to send a message to its parts while in case of association, anyone who holds a reference to it can communicate with it directly. In other words, in case of aggregation, the whole owns its parts and the part becomes a private property of the whole. For all practical purposes, any other object does not even need to know about its existence. On the other hand, an associated object may be shared among many different objects. That is, many different object may hold reference to the same object simultaneously.

### 2.3.2.5. Database persistence

From a database perspective, when an object is persisted or stored in the database, all of its components (all parts of the whole) must also be persisted in their entirety along with the “whole” for future reference while only a reference to the associated object may be stored in the database. Note that a normalized database would also enforce the above restriction.

## **3. Object Oriented Analysis**

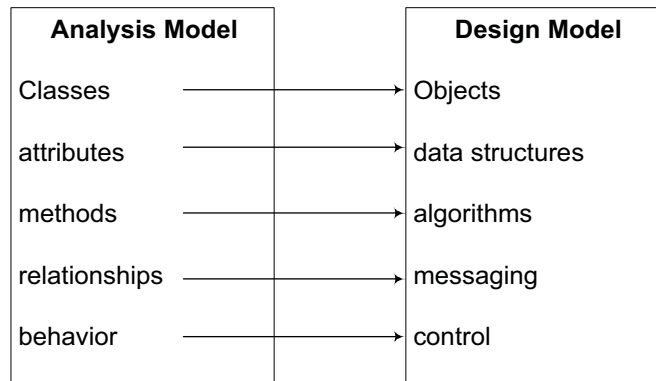
The intent of OOA is to define all classes, their relationships, and their behavior. A number of tasks must occur:

- 1) Static Model
  - a) Identify classes (i.e. attributes and methods are defined)
  - b) Specify class hierarchy
  - c) Identify object-to-object relationships
  - d) Model the object behavior
- 2) Dynamic Model
  - a) Scenario Diagrams

#### 4. Object Oriented Design

OOD transforms the analysis model into design model that serves as a blueprint for software construction. OOD results in a design that achieves a number of different levels of modularity. The four layers of the OO design pyramid are:

- 1) **The subsystem layer.** Contains a representation of each of the subsystems that enable the software to achieve its customers defined requirements and to implement the technical infrastructure that supports customer requirements.
- 2) **The class and object layer.** Contains the class hierarchies that enable the system to be created using generalization and increasingly more targeted specializations. The layer also contains design representations for each object.
- 3) **The message layer.** Contains the details that enable each object to communicate with its collaborators. This layer establishes the external and internal interfaces for the system.
- 4) **The responsibility layer.** Contains the data structures and algorithmic design for all attributes and operations for each object.



Translating the analysis model into a design model during object design

## 5. Object-Oriented Analysis using Abbot's Textual Analysis

The first object-orientation technique that we will study is one of the oldest techniques to identify objects and their relationships. This technique is called *Textual Analysis*. It was initially developed by Abbot and then extended by Graham and others. In this technique different parts of speech are identified within the text of the specification and these parts are modeled using different components. The following table shows this scheme.

<b>Part of speech</b>	<b>Model component</b>	<b>Example</b>
<b>proper noun</b>	<b>instance</b>	<b>Mehdi Hassan</b>
<b>improper noun</b>	<b>class/type/role</b>	<b>student, teacher</b>
<b>doing verb</b>	<b>operation</b>	<b>buy</b>
<b>being verb</b>	<b>classification</b>	<b>is a horse, is a book</b>
<b>having verb</b>	<b>composition</b>	<b>fan has wings</b>
<b>adjective</b>	<b>attribute value or class</b>	<b>this ball is green</b>
<b>adjective phrase</b>	<b>association</b>	<b>the customer with children</b>
	<b>operation</b>	<b>the customer who bought the kite</b>

Once all the model components have been identified, we will eliminate the redundant or irrelevant components by again analyzing the text and the context of the problem. Let's now try to understand this with the help of an example:

### Problem Statement:

A simple cash register has a display, an electronic wire with a plug, and a numeric keypad, which has keys for subtotal, tax, and total. This cash storage device has a total key, which triggers the release on the drawer. The numeric buttons simply place a number on the display screen, the subtotal displays the current total, the tax key computes the tax, and the total key adds the subtotal to the tax.

Our task now is to:

- Identify all the classes in this problem statement.
- Eliminate the unnecessary classes.

We are now going to use nouns to find classes.

### Nouns (initial)

Register	Display	Wire
Plug	Keypad	Keys
Devices	Release	Drawer
Buttons	Screen	Number
Total	Tax	

### Nouns (General Knowledge)

0-9 keys	Money	Subtotal Key
Tax Key	Total Key	



### Eliminating Irrelevant/Redundant Nouns

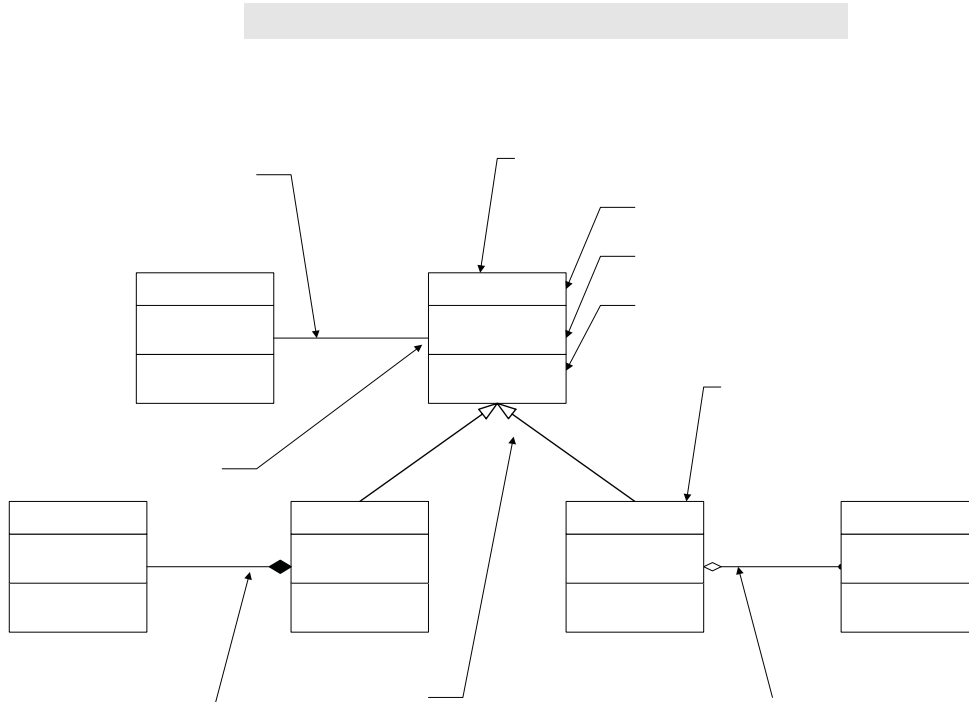
We now analyze the identified nouns and try to establish whether they would be stand-alone classes in our domain or not. Outcome of this analysis is shown below.

Register		
Display		
Wire	→	Irrelevant
Plug	→	Irrelevant
Keypad		
Keys		
Devices	→	Vague
Release	→	Irrelevant
Drawer		
Buttons	→	Redundant
Screen	→	Redundant
Number	→	Attribute
Total	→	Attribute
Tax	→	Attribute
0-9 Key		
Value	→	Attribute
Money		
Subtotal Key		
Tax Key		
Total Key		

We will continue with technique to identify all the constituent components of the model and derive our object-oriented design.

## 6. The Notation

Many different notations are used for documenting the object oriented design. Most popular of these include, Rumbaugh, Booch, and Coad, and UML(Unified Modeling Language). We will be using UML to document our design. Although the notation is very comprehensive and detailed, but the key features of this notation are presented in the following diagram.



## **7. Derivation of the Object Model – The Coad Methodology**

An object model of a system captures the static structure of a system by showing the objects in the systems, their relationships, their attributes, and their services. To stream line the derivation of the object model, Peter Coad has divided the process into 5 activities, each being further subdivided into a number of steps. Following is the description of these activities.

### **7.1. Select Objects – who am I?**

We have used an approach that divides the objects into different categories to make it easier to find them and establish their attributes, services, and collaborations. This activity, consisting of 6 steps, can help you find objects and categorize them. These steps are:

#### **7.1.1. Select actors**

Actors are people and organizations that take part in the system under consideration. Examples of actors are: person, organization (agency, company, corporation, foundation). Note that we are talking about actors and not their “roles”. e.g. a customer is a role that a person plays, so if we have a customer in our problem domain, we will also add a person as actor in the model.

#### **7.1.2. Select Participants**

A participant is a role that each actor plays in the system under consideration. Examples of participants are: agent, applicant, buyer, cashier, clerk, customer, dealer, distributor, donor, employee, investor, member, officer, owner, policy holder, recipient, student, supervisor, supplier, teacher, worker. It may be noted that the same person may play different roles at different times in the system. That means that if we model this behavior using Generalization-Specialization instead of Actor-Participant, we may end up with multiple inheritance.

#### **7.1.3. Select Places**

Places are where things come to rest or places that contain other objects. Examples of places are: airport, assembly-line, bank, city, clinic, country, depot, garage, hanger, hospital, plant, region, sales outlet, service center, shelf, station, store, warehouse, zone.

#### **7.1.4. Select Transactions**

Transactions are the “events” that must be remembered through time. These are entries that must be maintained in a historical record or log which may be used to answer questions or perform assessments. These transactions usually come from a window (GUI), some object which monitors for significant event and logs that information, or a another system that interacts with the system under consideration and logs some information. Examples of transactions are: agreement, assignment, authorization, contract, delivery, deposit, incident, inquiry, order, payment, problem report, purchase, refund, registration, rental, reservation, sale, shift, shipment, subscription, withdrawal. Note that nearly all transactions consist of a number of transaction line items.

#### **7.1.5. Select Container Objects**

Containers are objects that hold other objects. Note the similarity of definition between container and places. The difference is that a place is a place in the literal sense while a container is a any object that can hold other objects, e.g. bin, box, cabinet, folder, locker, safe, shelf, etc. Therefore a place is also a container but every container need not be a place.

### 7.1.6. Select Tangible things

Take a “walk” through the system and select “tangible” things around you used in the problem domain. These may be characterized as all the remaining (not yet selected) “nouns” that make up the problem domain. Examples are: account, book, calendar, cash box, cash drawer, item, plan, procedure, product, schedule, skill, tool, etc.

While selecting objects, the following considerations should be kept in mind for a simpler (and better) object model.

1. Every object that you put in your object model should have some responsibility or role to play in the problem domain. You need to know each object, its attributes, and services. If there is no way to know about the object, remove it from the object model.
2. Avoid having controller objects because controllers usually end up with functionality that’s better done by other objects themselves, making the message passing more complicated, and resulting in higher coupling. Use delegation instead. Note the difference between controlling and delegation; a controller wants to do every thing by himself (doesn’t trust anyone), while a good manager delegates responsibility (and takes credit).
3. In large systems several objects are likely to have similar or even identical responsibilities. Look for such objects and seek a common name to simplify the object model.
4. Use meaningful class names, names that describe objects in that class. Try to use names from the domain vocabulary to avoid confusion.

## 7.2. Identify Structures

A structure is a manner of organization which expresses a semantically strong organization within the problem domain. There are two type of structures: Generalization-Specialization (Gen-Spec) and whole-part. This activity covers the identification of these structures in the following 2 steps:

### 7.2.1. Identify Gen-Spec Structures (Hierarchy)

Consider each class that you have identified as a specialization and then look for its generalization and vice versa.

### 7.2.2. Identify Whole-Part structures (Aggregations) - What are my components?

For each object that you have identified, consider it as a whole and then try to find out its parts - objects that make up this object.

## 7.3. Define Attributes - What I Know?

The first two activities would identify most of the objects (classes) in the problem domain. Now is the time to think about the role and responsibilities of these objects. The first thing to consider is their attributes, i.e., what it knows.

For each object include the attributes that come to mind when you first think about the object. The criteria for the inclusion of an attribute is that it should be included if the system needs to know its value and it cannot get it any other way. Don not add an attribute for an association or aggregation. Examples of attributes are: number, name, address, date, time, operational state, phone, status, threshold, type, etc. In particular, consider the following attributes for different types of objects.

1. For actors consider name, address, phone.
2. for participants consider number, date and time, password, authorization level.
3. for place/location consider number, name, address (perhaps latitude, longitude, altitude).
4. for transaction consider number, date, time, status.
5. for line item consider quantity, status.
6. for item consider name, description, dimension, size, UPC, weight.

Like object selection, there are a number of issues that every designer must be aware of while defining attributes of an object. These are:

1. An attribute that varies over time, e.g., price of an item, should be replaced by an additional class with an effective date and value.
2. An attribute that may have a number of values should be replaced by a new class and an object connection.
3. Replace “yes/no” type attributes with “status” type attributes for flexibility.
4. If there are classes with common attributes and generalization-specialization makes good sense, then add a generalization class and factor out the commonality.

#### **7.4. Show Collaborations (associations and aggregations) - Who I know?**

The second step in establishing each object’s responsibility is to identify and show how this object collaborates with other objects, i.e., who it knows. These collaborations can be identified with the help of the following 8 steps:

1. For an actor, include an object connect to its participants (association).
2. For a participant, include an object connection to its actor (already established) and its transactions (association).
3. For a location, include object connections to objects that it can hold (association), to its part objects (aggregation), and to the transactions that are taking place at that location (association).
4. For transactions, include object connections to its participants (already established), its line items (aggregation), and its immediate subsequent transaction (aggregation).
5. For a transaction line item, include object connections to its transaction (already established), its item (association), a companion “item description” object (association), and a subsequent line item (association).
6. For an item, include object connections to transaction line item (already established), a companion “item description” object (association).
7. For a composite object, include object connections to its “part” object (aggregation).
8. For all objects (including all of the above) select connecting objects to which the object under consideration sends a message (within one or more scenarios) to get some information or to answer a query about objects directly related to it (association).

#### **7.5. Define Services - What I do?**

The third and last step in establishing each object’s responsibility is to define what services does each object in the problem domain provide, i.e., what it does. Putting the right service with the right object is also very important since any mistake in judgment will increase coupling and reduce cohesion. The verbs in your problem domain usually indicate some of the services required of the associated object.

Software objects do things that the system is responsible to do with regard to that object. By putting the services with the attributes they work on results in lower coupling and stronger

cohesion, and increased likelihood of reuse. The basic principle is to keep data and action together for lower coupling and better cohesion. The basic services, done by all (such as get, set, create, initialize), are not shown in the object model. While establishing the services for an object, the following fundamental questions should be asked:

1. Why does the system need this object any way?
2. What useful questions can it answer?
3. What useful action can it perform?
4. What this object can do, based upon what it knows?
5. What this object can do, based upon whom it knows?
6. What calculations can it do?
7. What ongoing monitoring could it do?
8. What calculations across a collection could it make (letting each worker do its part)?
9. What selections across a collection could it make (letting each worker do its part)?

While establishing services of certain specific types of objects, the following should be considered:

1. For an actor, consider: calculate for me, rate me, is <value>, rank participants, calculate over participants.
2. For a participant, consider: calculate for me, rate me, is <value>, rank transactions, calculate over transactions.
3. For a place, consider: calculate for me, rate me, is <value>, rank transactions, calculate over contents, calculate over container line items.
4. For a Transaction, consider: calculate for me, rate me, is <value>, how many, how much, rank transaction line items, rank subsequent transactions, calculate over transaction line items, calculate over subsequent transactions.
5. For a line item, consider: calculate for me, rate me.
6. For an item, consider: calculate for me, rate me, is <value>, how many, how much, rank, calculate over specific items.

## **8. CASE STUDY: Connie's Convenience Store - A point of Sale System**

### **8.1. The System**

#### **8.1.1. Identify the purpose of the system**

- develop an overall purpose statement in 25 words or less. Why this system? Why now?
- Keep the overall goal, the critical success factor, always before you.
- "To support, to help, to facilitate, ..."

##### **8.1.1.1. Connie's Wish List**

- scan items and automatically price them
- know whether an item is on sale
- automatically total the sale and calculate tax
- handle purchases and returns
- handle payments with cash, check, or charge
- authorize checks and cards
- calculate change when working with cash or checks
- record all of the information about a customer transaction

- balance the cash in the drawer with the amount recorded by the point-of-sale system.

#### 8.1.1.2. Why ?

- speed up checkout time
- reduce the number of pricing errors
- reduce the labour required to ticket the item with a price, originally and when prices change.

#### 8.1.1.3. Summary

*to help each cashier work more effectively during checkout, to keep good records of each sale, and to store more efficient store operations.*

### 8.1.2. Identify system features

Be certain to include features that cover the following

1. log important information
2. conduct business
3. analyze business results
4. interact with other systems

#### 8.1.2.1. Identify features for logging important information

- to maintain prices based upon UPC
- to maintain tax categories (categories, rates, and effective dates)
- to maintain the authorized cashiers
- to maintain what items we sell in a store
- to log the results of each sale in a store

#### 8.1.2.2. Identify features for conducting business

- to price each item, based upon its UPC
- to subtotal, calculate tax, and total
- to accept payment by cash, check, or charge

#### 8.1.2.3. Identify features for analyzing business results

- to count how many of each item sold
- to count how much we received in cash, check, or credit card sales
- to assess how each cashier is performing
- to assess how each store is performing

#### 8.1.2.4. Identify features for working with interacting systems

- to obtain authorization from one or more credit (or check) authorization system

## 8.2. SELECTING OBJECTS

### 8.2.1. Select Actors

the actor is:

- person

### 8.2.2. Select Participants

the Participants are:

- cashier
- head cashier
- customer

#### Cashier and Head Cashier

Is there a difference between head cashier and cashier in terms of their behavior and knowledge?. If no then we don not need a separate class for head cashier.

#### Customer

customer. You must have a way to know about customer objects; otherwise it should not be put in the domain model.

### 8.2.3. Select Places

The places are:

- store
- shelf

#### Shelf

The system does not keep track of the shelves.

### 8.2.4. Select Transactions

Significant Transactions are:

- sale
- every sale is a collection of sale line items
- return
- payment
- session

### 8.2.5. Select Container Classes

The store is a container class.

a store contains

- cashiers
- registers
- items

### 8.2.6. Select Tangible Things

Tangible things in store:

- item
- register
- cash drawer
- Tax Category (Descriptive things)

#### Session

Is it important? It is important in order to evaluate a cashier's performance.

## 8.3. Identify Structures

### 8.3.1. Identify Gen-Spec Structures

Kinds of stores:



A store is a kind of sales outlet. Perhaps over time, Connie will expand to other kinds of sales outlets. Stores might be specialized into kinds of stores. For now on leave store as it is.

Kinds of sales:

- sales, returns
- only different is that the amount is positive or negative. Is there any other difference?

Prices:

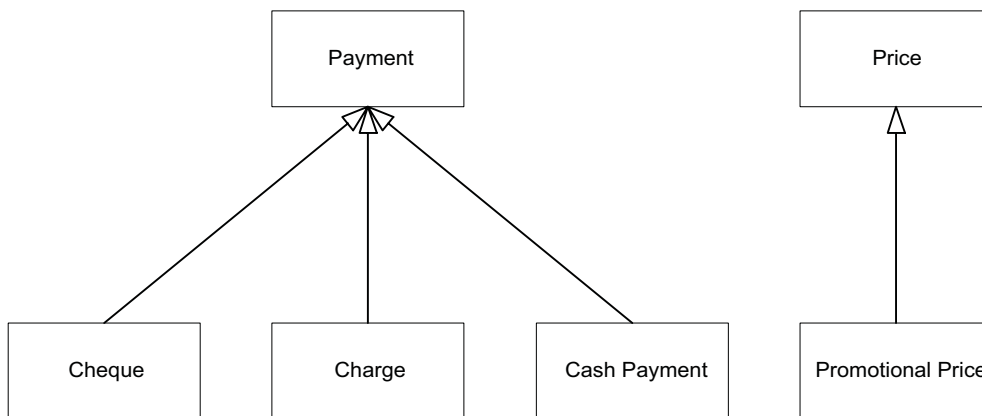
- regular price, and promotional (sales) price

Payment:

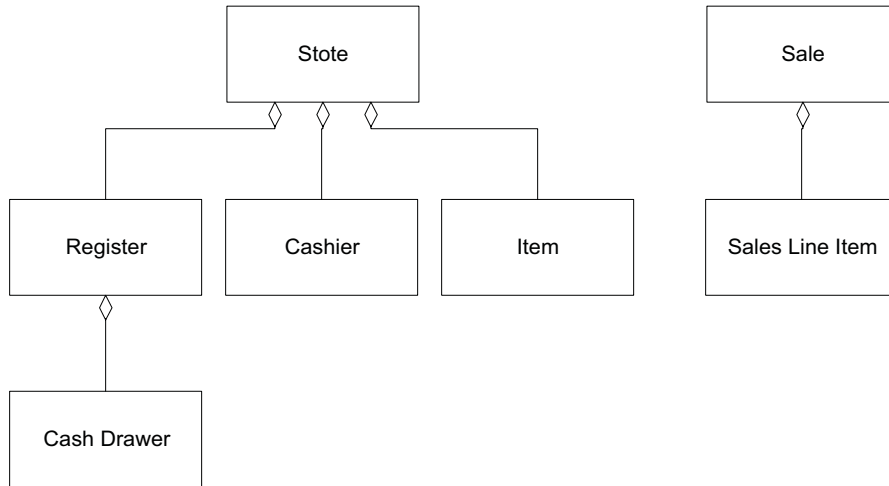
- cash, check, and charge are kind of payments

### 8.3.2. Identify Whole-Part Structures

- A store as a whole is made up of cashiers, registers, and items.
- A register contains a cash drawer.
- A sale is constituted of sale line items.



Object Hierarchy

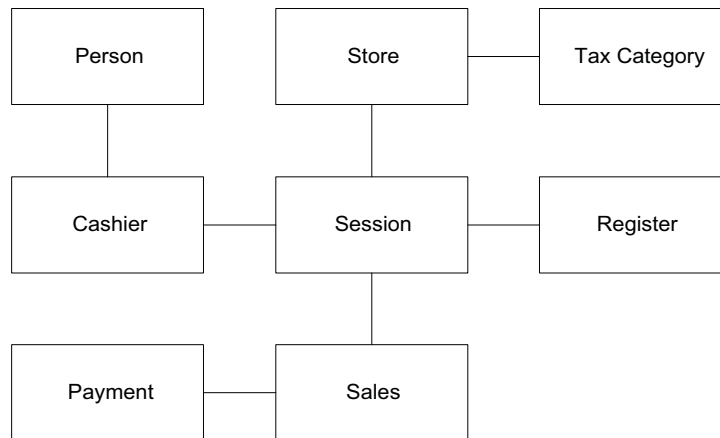


Whole-Part Structures

## 8.4. Establishing Responsibilities

### 8.4.1. Who I Know - Rules of Thumb

- an actor knows about its participants  
person knows about cashier
- a transaction knows about its participants  
a session knows about its register and cashier
- A transaction contains its transaction line items  
sale contains its sales line items
- A transaction knows its sub transactions  
session knows about its sales  
sale knows about its payments
- A place knows about its transactions  
store knows about its sessions
- A place knows about its descriptive objects  
store knows about its tax categories
- A container knows about its contents  
a store knows about its cashiers, items, and registers



Association Relationships

### 8.4.2. Define Attributes, Services, and Links - What I know, What I do, and Who I know?

Actors:

person

Attributes: name, address, phone

Services:

Participants:

cashier

Attributes: number, password, authorization level, current session

Services: isAuthorized, assess Performance

Places:

store

Attributes: name

Services: get item for UPC, get cashier for number

Tangible things:

item

Attributes: number, description, UPCs, prices, taxable  
**attributes with repeating names - create new objects**  
**UPC, Price (specialization - promotional price)**

Services: get price for a date, how much for quantity

Who I Know? UPC, Price, tax category, sale line item

register

Attributes: number

Services: how much over interval, how many over interval

Who I know? store, session, cash drawer (part of register)

cash drawer

Attributes: balance, position (open, close), operational state  
Services: open  
Who I know? register

Tax Category

Attributes: category, rate, effective date  
Services: just the basic services - get, add, set - don't show  
Who I know? items?

Transactions:

sale

Attributes: date and time  
Services: calculate subtotal, calculate total, calculate discount, calculate tax, commit  
Who I Know? session, payment, SLIs

sale line item

Attributes: date and time ?, quantity, tax status ( regular, resale, tax-exempt)  
Services: calculate sub total  
Who I Know? item, sale

sale line item - how do you handle returns and sales

sale - you have control  
return - more difficult  
- return to a different store  
- purchased for a different price  
- returns an item no longer in the inventory

return

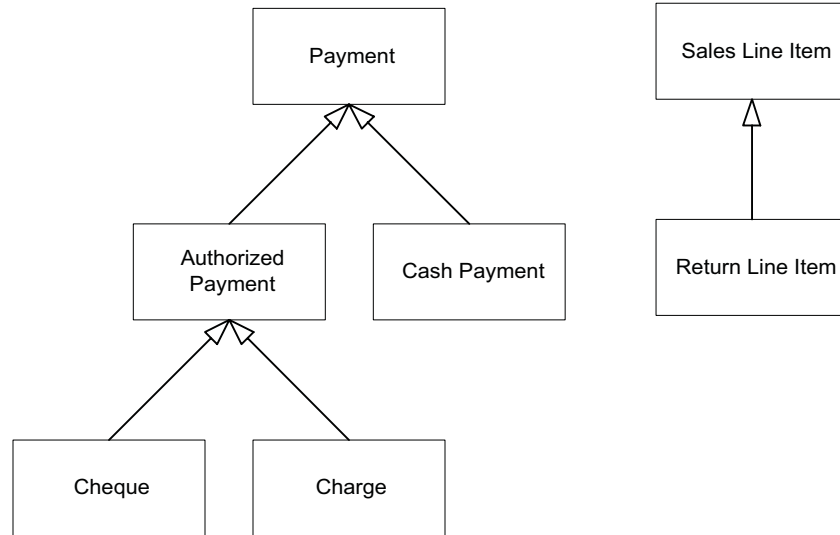
Attributes: return price, reason code, sale date, sale price  
Services:  
Who I Know?

is it a case for gen-spec, what's same, what's different

payment - we have types of payments

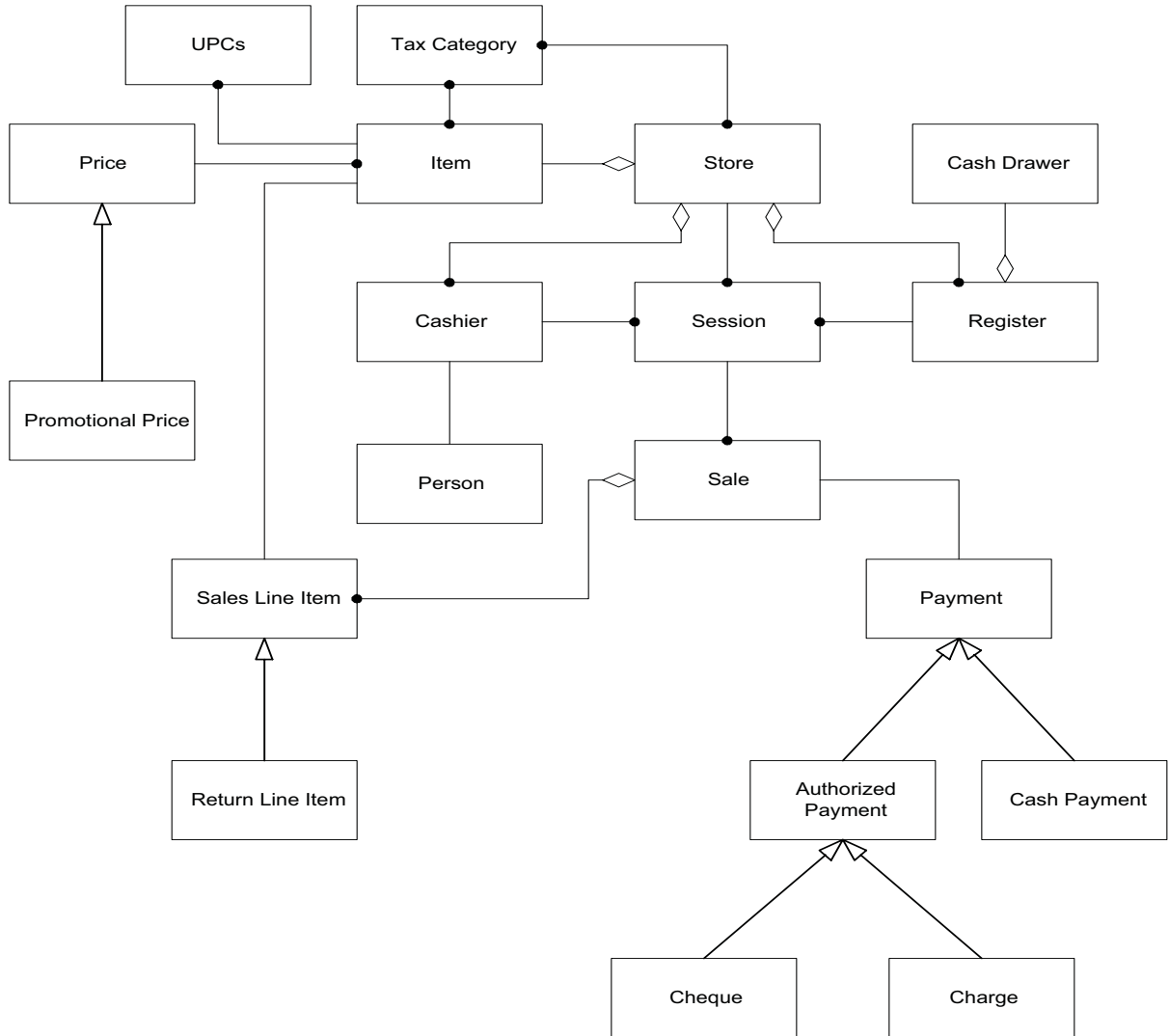
Attributes:  
each payment knows about its  
amount paid, cash tendered  
a check object knows its  
bank, account number, amount tendered, authorization code  
a credit object knows about its  
card type, card number, expiration date, authorization code  
common attributes among check and credit - use gen-spec  
hierarchy becomes:  
payment  
cash payment  
authorized payment  
check  
card

Services:  
who I know: sale



session

Attributes: start date, end date, start time, end time  
Services: how much money collected over interval, how many sales  
Who I know? register, cashier, store, sales



Object Model Diagram for Connie's Convenience Store