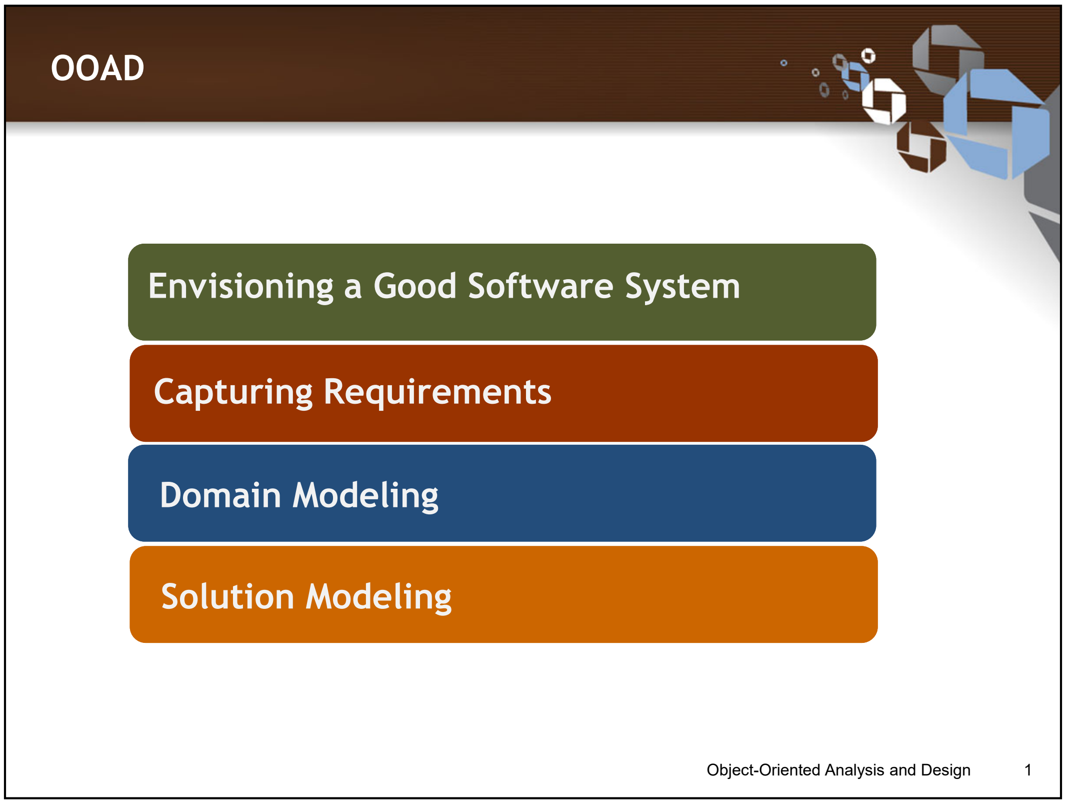


## OOAD

- Envisioning a Good Software System
- Capturing Requirements
- Domain Modeling
- Solution Modeling

## Domain Modeling

## Introduction to OOA – Domain Modeling

- A domain model is a representation of:
  - Entities constituting the problem domain
  - Attributes of the entities
  - Behavior of the entities
  - Interactions
  - Relationship between these entities

## Introduction to OOA – Domain Modeling (Cont'd)

- The advantages of creating a domain model during OOA are:
  - It defines the scope of the system to be developed.
  - It helps the problem to be viewed as one constituting real life concrete, and atomic entities rather than subjective statements.
  - It is a useful way of showing the stakeholders an initial view of what the system would do and gather feedback from them early in the SDLC.
  - It provides a developer's view of the problem domain.
  - It serves as a stepping stone to system design.

## Steps in Domain Modeling

- The task of creating a domain model is termed as domain modeling.
- There are two aspects to domain modeling:
  - Static - Involves modeling the structure of the system
  - Dynamic - Involves modeling the behavior of the system

## Steps in Domain Modeling (Cont'd)

- Static domain modeling consists of the following steps:
  - Identify candidate classes
  - Evaluate the candidate classes
  - Identify the candidate attributes and operations for each class
  - Create the initial static domain model
  - Evaluate the relationship between the final abstractions
  - Refine the static domain model

## Identifying Candidate Classes

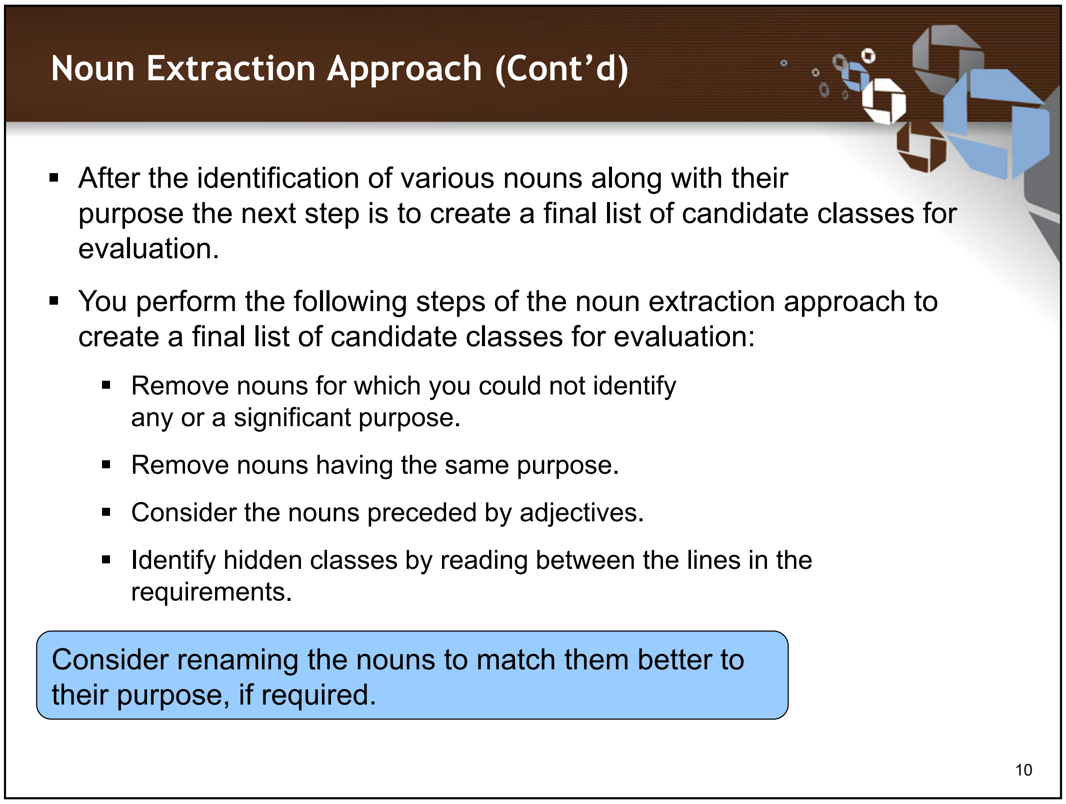
- The real-life entities that are part of the problem domain are identified as candidate classes for the domain model.
- It represents an entity and not exactly a class in a program. It can be considered as a representative of (possibly a single or) a set of classes corresponding to it.
- To identify candidate classes, you can adopt the following approaches:
  - Noun Extraction approach
  - Common Categories approach

## Noun Extraction Approach

- According to the noun extraction approach, you need to perform the following tasks:
  1. Go through the recorded and modeled requirements, and identify nouns in these documents.
  2. Create a list of these nouns and describe the purpose for each noun, assuming you would make it a candidate class.
  3. Remove the nouns for which you are unable to define a purpose.
  4. Ignore nouns that have the same purpose as another noun.
  5. Consider nouns preceded by adjectives. If the same noun is preceded by different adjectives and is expected to behave differently in each case, consider making two separate candidate classes for these.

## Noun Extraction Approach (Cont'd)

6. Review the requirement descriptions and scenarios in the passive mode, where you try to convert crucial verbs into abstract nouns. Rephrasing the description as such may reveal crucial hidden candidate classes.
7. Finalize the list for detailed evaluation.



## Noun Extraction Approach (Cont'd)

- After the identification of various nouns along with their purpose the next step is to create a final list of candidate classes for evaluation.
- You perform the following steps of the noun extraction approach to create a final list of candidate classes for evaluation:
  - Remove nouns for which you could not identify any or a significant purpose.
  - Remove nouns having the same purpose.
  - Consider the nouns preceded by adjectives.
  - Identify hidden classes by reading between the lines in the requirements.

Consider renaming the nouns to match them better to their purpose, if required.

The customer can search for items from a product catalogue.

The system provides the customer an option to buy products online after logging in.

The customer adds items to a shopping cart.

The customer places an order.

The availability of the order is checked for each SKU added.

The customer can make payment for the order by choosing any one payment mode -- credit card or bank transfer.

An order ID is sent to the customer by the system for tracking.

For each product in the order, the system validates its shipping status and any docket tracking numbers.

The order ID is also sent to the dispatch clerk for managing dispatch of the ordered items.

The dispatch clerk posts a requisition for retrieval from warehouse of the products.

The system generates and prints a shipping invoice.

The warehouse clerk initiates a reconciliation process.

The inventory is reconciled and the last reconciliation time is updated in the system.

After selecting the nouns.

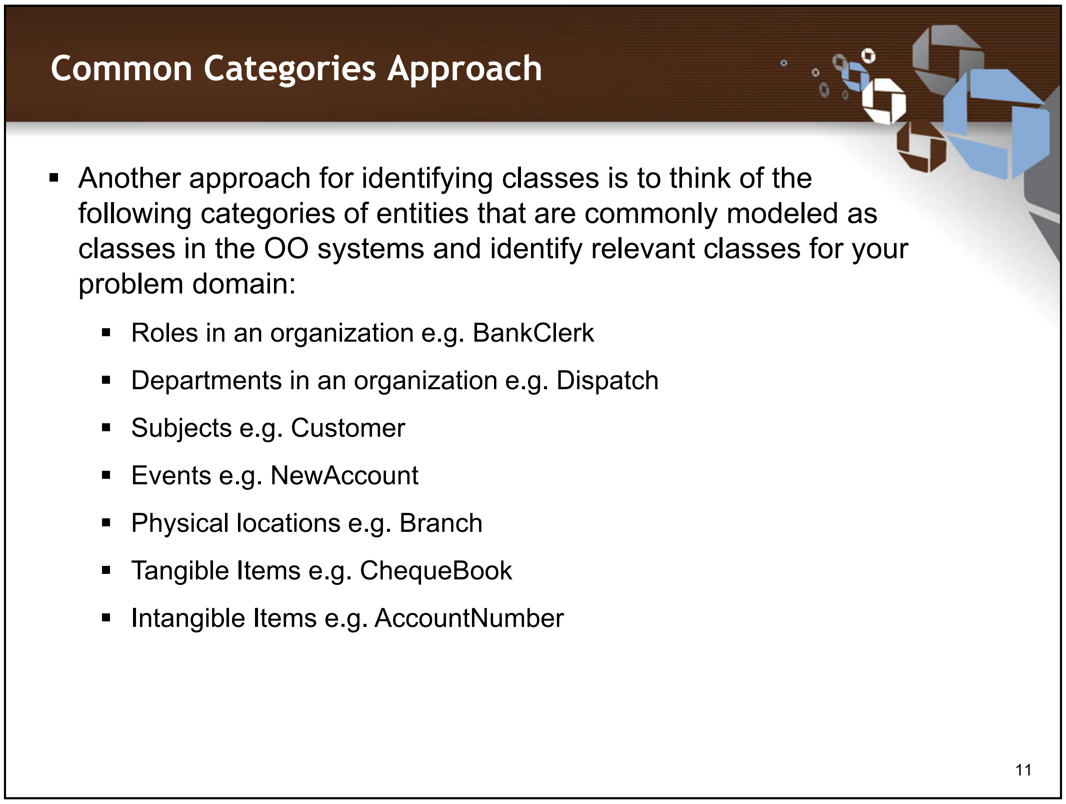
- Remove nouns for which you could not identify any or a significant purpose. For example, Warehouse and Inventory refer to the same abstraction for the purpose of your problem domain, and a specific purpose for Inventory could not be identified. In fact, Inventory can be represented by the Warehouse. Therefore, Inventory can be eliminated from the list of candidate classes.

- Remove nouns having the same purpose. For example, Item and Product are nouns referring to the same thing. As Product is a more appropriate term, you can remove the term, Item. In addition, the System candidate class is the collection of all other classes (because it represents the entire system), and therefore, can be eliminated. It is only a part of the language used for describing the requirements and not really a part of the problem domain.

- Consider the nouns preceded by adjectives. The noun, clerk, behaves differently when preceded by the words, Dispatch and Warehouse. So, you can consider putting these as separate candidate classes in your list.

- Identify hidden classes by reading between the lines in the requirements. For example, if you focus on the requirement: Once the payment is through, the system generates a unique order ID, and the order information is recorded and sent to the warehouse and the customer. From use case UC04, you will identify the need for an OrderReceipt class. The purpose of this class will be to send a confirmation to the customer regarding the order and a receipt of the payment. This information is important for tracking the order. Similarly, consider another requirement: For each product ordered, the system validates its identity on the basis of SKU. If you reconsider this as “for *each product in the order*, the system validates its identity on the basis of SKU, you will feel the need for an OrderItem class that refers to each line item in the order. You can also consider this as a candidate class and further evaluate its need.

Further, some nouns may intuitively appear to be attributes of other nouns. For example, Order ID is obviously a characteristic of Order. You can eliminate these as candidate classes and consider them as candidate attributes.



## Common Categories Approach

- Another approach for identifying classes is to think of the following categories of entities that are commonly modeled as classes in the OO systems and identify relevant classes for your problem domain:
  - Roles in an organization e.g. BankClerk
  - Departments in an organization e.g. Dispatch
  - Subjects e.g. Customer
  - Events e.g. NewAccount
  - Physical locations e.g. Branch
  - Tangible Items e.g. ChequeBook
  - Intangible Items e.g. AccountNumber

11

Another approach for identifying classes is to think of the following categories of entities that are commonly modeled as classes in the OO systems and identify relevant classes for your problem domain:

Roles in an organization – Roles in the customer's organization (particularly those related to the problem domain) can be considered for identifying candidate classes. For example, if a dispatch clerk and warehouse clerk are roles (designations) in the organization involved with your domain of interest (selling of products), you can consider them as candidate classes.

Departments in an organization – The departments of the customer's organization, which fall within the scope of your problem domain, can be considered for identifying candidate classes. For example, Dispatch can be considered as a department.

Subjects – The people who use the system and/or the people who are of interest to the system can be considered for identifying candidate classes. For example, customers are people who will use the system. In fact, they are also of interest to the system as the system needs to save information about them. So, you can consider Customer as a candidate class.

Events – An occurrence in the problem domain can be a candidate class. For example, an order is an occurrence in the problem domain. Therefore, it is a candidate class.

Physical locations – Places relevant to your problem domain can be considered for identifying candidate classes. For example, warehouse is a physical location in the problem domain and can be considered as a candidate class.

Tangible items – Devices or equipment used in your problem domain can be considered for identifying candidate classes. For example, shopping cart is an item used in the problem domain and can be considered as a candidate class.

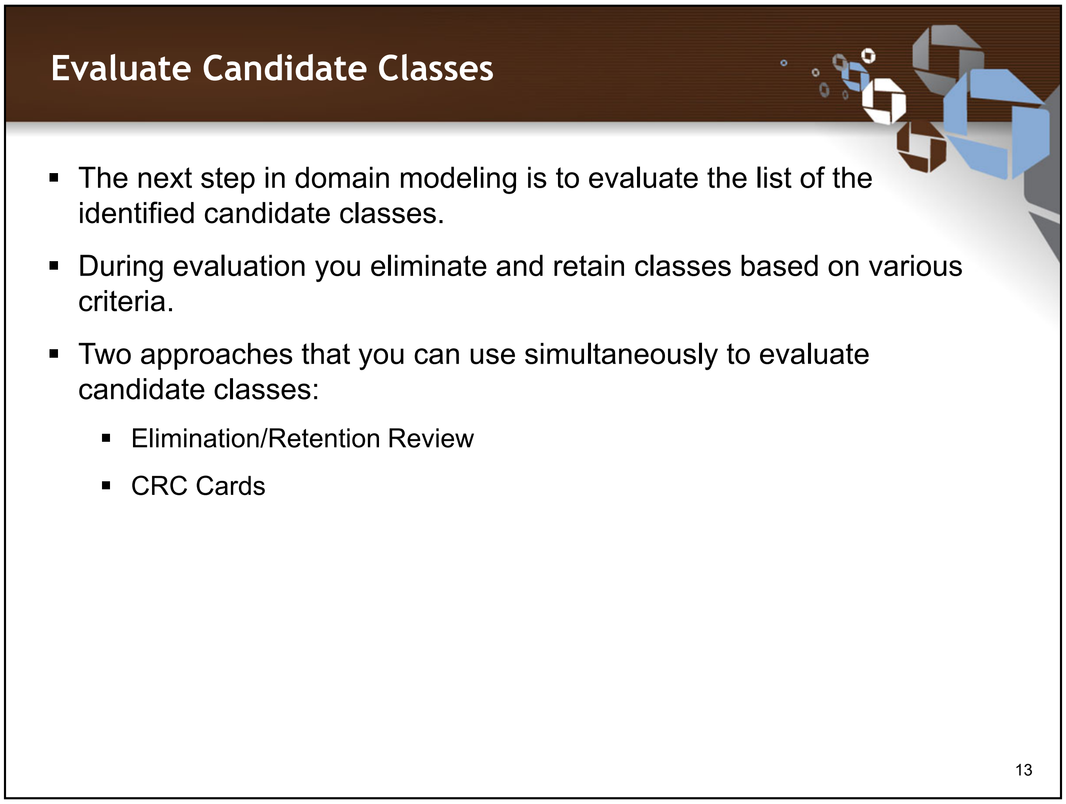
Intangible items – Things that cannot be touched or have a physical existence in the problem domain, but still affect it can be considered for identifying candidate classes. For example, each line item in an order is an intangible item and OrderItem can be considered as a candidate class.

NOTE: You can use either or both approaches of candidate class identification. A useful method is to use one approach to validate the findings of the other approach.

## Steps in Domain Modeling

- Static domain modeling consists of the following steps:

  - Identify candidate classes
  - Evaluate the candidate classes
  - Identify the candidate attributes and operations for each class
  - Create the initial static domain model
  - Evaluate the relationship between the final abstractions
  - Refine the static domain model



## Evaluate Candidate Classes

- The next step in domain modeling is to evaluate the list of the identified candidate classes.
- During evaluation you eliminate and retain classes based on various criteria.
- Two approaches that you can use simultaneously to evaluate candidate classes:
  - Elimination/Retention Review
  - CRC Cards

Output of previous stage:

Customer

Product

Shopping Cart

Product Catalogue

Order

SKU

Payment

Mode

Credit Card

Bank

Shipping status

Docket tracking numbers

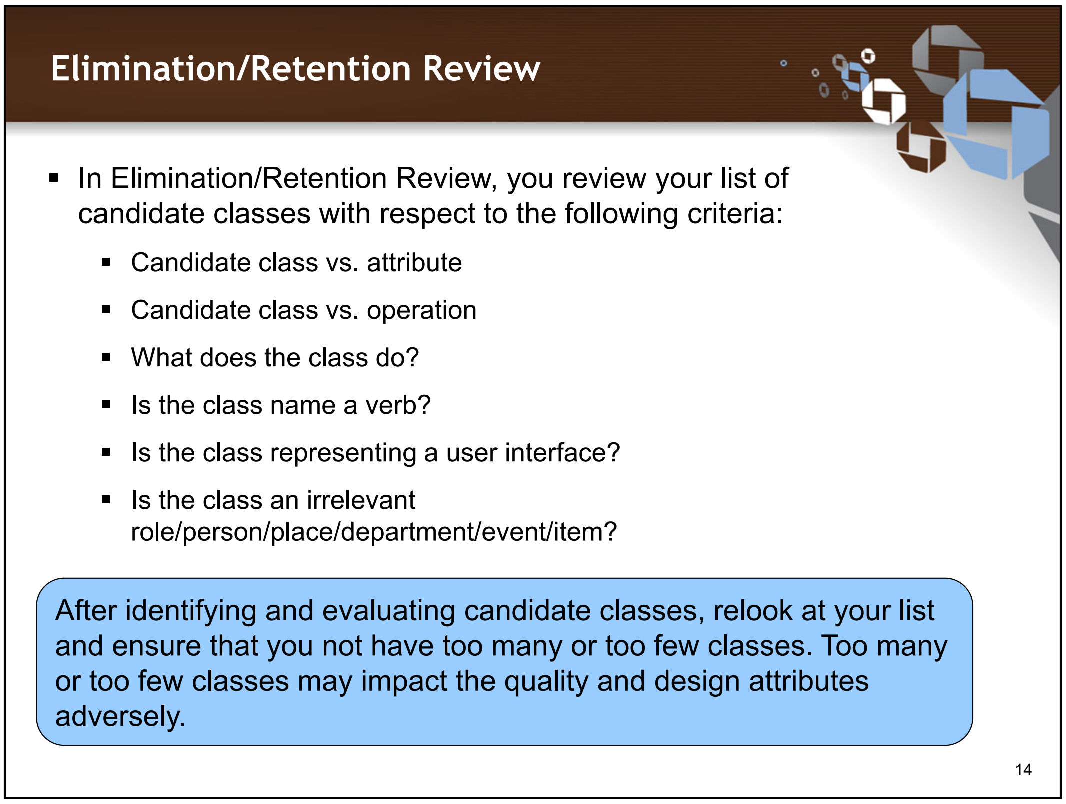
Dispatch Clerk

Warehouse Clerk

Dispatch

Requisition

Warehouse
Shipping Invoice
Reconciliation Time
OrderItem
OrderReceipt



## Elimination/Retention Review

- In Elimination/Retention Review, you review your list of candidate classes with respect to the following criteria:
  - Candidate class vs. attribute
  - Candidate class vs. operation
  - What does the class do?
  - Is the class name a verb?
  - Is the class representing a user interface?
  - Is the class an irrelevant role/person/place/department/event/item?

After identifying and evaluating candidate classes, relook at your list and ensure that you not have too many or too few classes. Too many or too few classes may impact the quality and design attributes adversely.

14

In this approach, you review your list of candidate classes with respect to the following criteria:

- Candidate class vs. attribute – Ask yourself if you actually need a particular candidate class or if it can be modeled as an attribute of another class. If you are able to think of clear attributes and operations for the candidate class, then it may not be an attribute of another class. However, if all its operations and attributes seem to be covered in one or more other candidate classes, then consider eliminating it. For example, the candidate class, Mode, is actually referring to payment mode. This means that it can be considered as an attribute of the Payment candidate class. The Payment candidate class can then be evaluated further for its purpose and you may realize that it can itself be an attribute of the Order class as PaymentMode or PaymentType. This is because the purpose of the Payment class is to identify the mode of payment for the order and provide an interface for paying for the order. The first part of the purpose can be considered an attribute of the order and the latter part can be considered a behavior for order. Similarly, SKU, Order ID, Docket tracking numbers, Reconciliation Time, and Shipping Status, are all candidate attributes rather than candidate classes.

- Candidate class vs. operation – Ask yourself if a candidate class actually represents a process rather than an entity. You can check this by trying to convert the noun to a verb. If a candidate class does represent a process, consider making it an operation rather than a separate class. For example, requisition is a noun in the requirements. However, you can also consider it as a verb ‘requesting’ or ‘requiring’. In terms of our system, it can be associated as a process with the warehouse or the order. Therefore, it can be eliminated as a candidate class. However, that does not mean that all processes are operations. Some processes are abstractions in themselves and therefore need to be represented as separate classes, such as Dispatch.

- What does the class do? – Evaluate the purpose of the class. If the class only offers only a single operation, you can consider eliminating it. This is because a class is not supposed to perform one single operation. It usually has a deeper purpose of providing a set of services to other entities in the problem domain. For example, the Credit Card class is only specifying the number and expiration details on the credit card. This can be eliminated and the details stored in the Order candidate class.

- Is the class name a verb? – If the name of any candidate class is a verb, it is either not named correctly or is not a candidate for retention. A candidate class named as a verb would, typically, perform only one single operation in most cases. So, you can eliminate it.

- Is the class simple and consistent? – If the class has too many characteristics or a complicated purpose covering many features of the problem domain, it may need to be eliminated and replaced with multiple classes, each mapping to a real abstraction in the problem domain. For example, a single Order may have more than one item, with each item being ordered in a specific quantity. All processing related to each ordered products, will then be handled by this class. To simplify this, having an OrderItem class (that uniquely represents one item in the order with it s quantity and price) would be a better approach. Multiple objects of the OrderItem class can then be combined to form the Order class.

- Is the class representing a User Interface? – At this stage, do not identify candidate classes representing UI details. UI classes are part of the solution model and not the domain model. For example, if you use nouns such as ‘Login page’ and try to identify candidate classes for them, you would add unnecessary implementation complexity to the domain model. In fact, ideally, a use case or requirement specification should itself not refer to UI elements.

- Is the class an irrelevant role/person/place/department/event/item? Any role/person/place/department/event/item is irrelevant if the system does not process information about it. Therefore, any candidate class that represents such an entity can be eliminated. For example, the shipping company is an irrelevant role that does not need to be tracked by the system.

After identifying and evaluating candidate classes, relook at your list and ensure that you not have too many or too few classes. Too many or too few classes may impact the quality and design attributes adversely. For example, having a very small number of big classes can cause high coupling between these classes. Reiterate through the identification and evaluation steps to generate a balanced list

Discuss each criteria referring to an example from the Drummix system.

- Candidate class vs. attribute – For example, the candidate class, Mode, is actually referring to payment mode. This means that it can be considered as an attribute of the Payment candidate class. The Payment candidate class can then be evaluated further for its purpose and you may realize that it can itself be an attribute of the Order class as PaymentMode or PaymentType. This is because the purpose of the Payment class is to identify the mode of payment for the order and provide an interface for paying for the order. The first part of the purpose can be considered an attribute of the order and the latter part can be considered a behavior for order. Similarly, SKU, Order ID, Docket tracking numbers, Reconciliation Time, and Shipping Status, are all candidate attributes rather than candidate classes.

- Candidate class vs. operation – For example, requisition is a noun in the requirements. However, you can also consider it as a verb ‘requesting’ or ‘requiring’. In terms of our system, it can be associated as a process with the warehouse or the order. Therefore, it can be eliminated as a candidate class. However, that does not mean that all processes are operations. Some processes are abstractions in themselves and therefore need to be represented as separate classes, such as Dispatch.

- What does the class do? – For example, the Credit Card class is only specifying the number and expiration details on the credit card. This can be eliminated and the details stored in the Order candidate class.

- Is the class name a verb? – If the name of any candidate class is a verb, it is either not named correctly or is not a candidate for retention. A candidate class named as a verb would, typically, perform only one single operation in most cases. So, you can eliminate it.

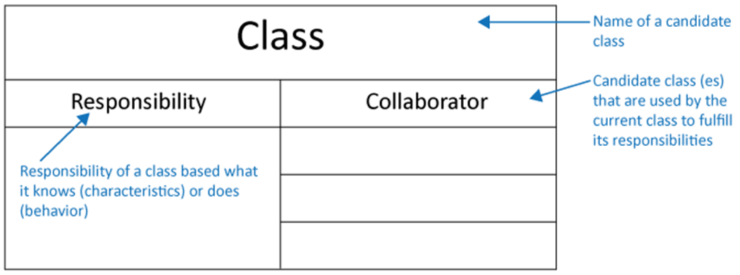
- Is the class simple and consistent? – For example, a single Order may have more than one item, with each item being ordered in a specific quantity. All processing related to each ordered products, will then be handled by this class. To simplify this, having an OrderItem class (that uniquely represents one item in the order with it s quantity and price) would be a better approach. Multiple objects of the OrderItem class can then be combined to form the Order class.

- Is the class representing a User Interface? – For example, if you use nouns such as ‘Login page’ and try to identify candidate classes for them, you would add unnecessary implementation complexity to the domain model. In fact, ideally, a use case or requirement specification should itself not refer to UI elements.

- Is the class an irrelevant role/person/place/department/event/item? For example, the shipping company is an irrelevant role that does not need to be tracked by the system.

## CRC Cards

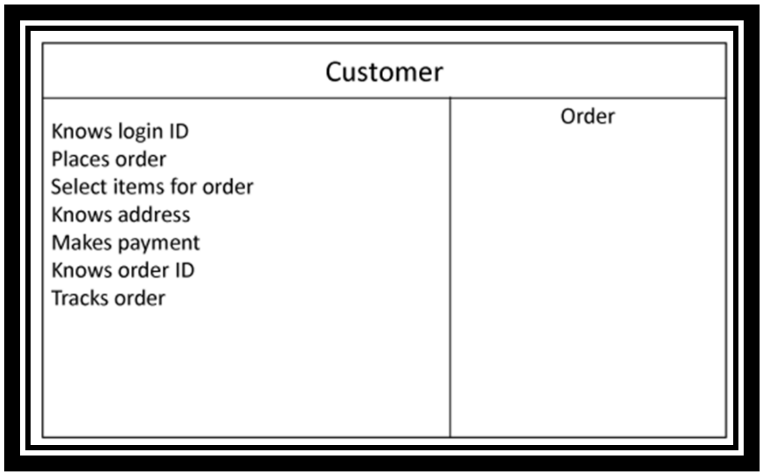
- Class Responsibility Collaborator (CRC) cards is a technique used for candidate class identification as well as evaluation.
- In this technique, you create small cards, created by hand for each candidate class that you identify.


Class
Responsibility
Collaborator
Name of a candidate class
Candidate class (es) that are used by the current class to fulfill its responsibilities
Responsibility of a class based what it knows (characteristics) or does (behavior)

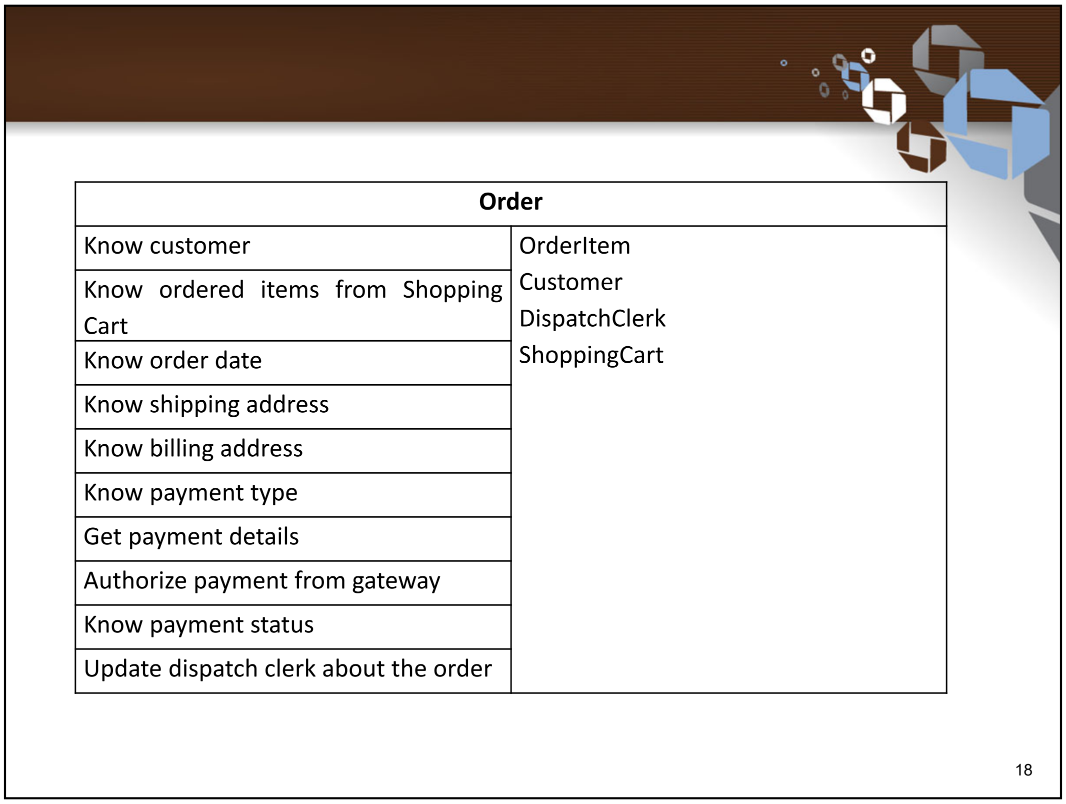
## CRC Cards (Cont'd)

- For every candidate class identified, you can identify:
  - Responsibilities and write them on the card
  - Collaborators for the candidate class

## CRC Cards (Cont'd)



| Customer | |
|---|---|
| Knows login ID<br>Places order<br>Select items for order<br>Knows address<br>Makes payment<br>Knows order ID<br>Tracks order | Order |



| Order | |
|---|---|
| Know customer | OrderItem |
| Know ordered items from Shopping Cart | Customer |
| Know order date | DispatchClerk |
| Know shipping address | ShoppingCart |
| Know billing address | |
| Know payment type | |
| Get payment details | |
| Authorize payment from gateway | |
| Know payment status | |
| Update dispatch clerk about the order | |

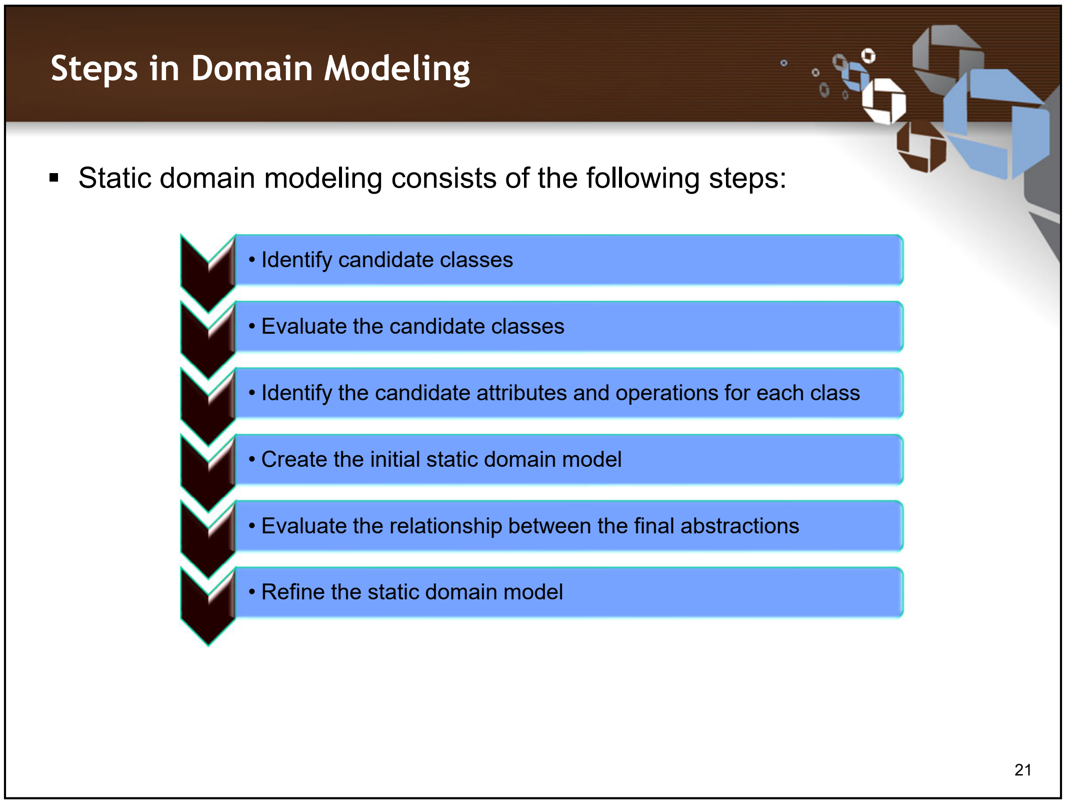
18

## CRC Cards (Cont’d)

- Following is the specific CRC criteria:
  - If you are unable to define any responsibilities for a candidate class, consider eliminating it.
  - If a candidate class has only 1 responsibility, identify another candidate class under which you can place the responsibility and eliminate the original candidate class.
  - If a class has too many responsibilities, consider breaking it down into separate classes.
  - If a candidate class has various collaborators, relook at the candidate classes that they represent and evaluate them.
  - If most collaborators and responsibilities of a candidate class are being handled by more than one class, consider clubbing them.

## CRC Cards (Cont'd)

- If you identify a responsibility from the requirements description and models, which does not fall under any candidate class, consider creating a new candidate class.
- Do not keep a candidate class that does not have collaborating role or whose responsibilities are also fulfilled by another class, partly or completely.
- If a candidate class does not have any collaborators, consider distributing the information in a class to other associated classes.



## Steps in Domain Modeling

- Static domain modeling consists of the following steps:
  - Identify candidate classes
  - Evaluate the candidate classes
  - Identify the candidate attributes and operations for each class
  - Create the initial static domain model
  - Evaluate the relationship between the final abstractions
  - Refine the static domain model

Output of previous stage

Customer

Product

Shopping Cart

Product Catalogue

Order

Dispatch Clerk

Warehouse Clerk

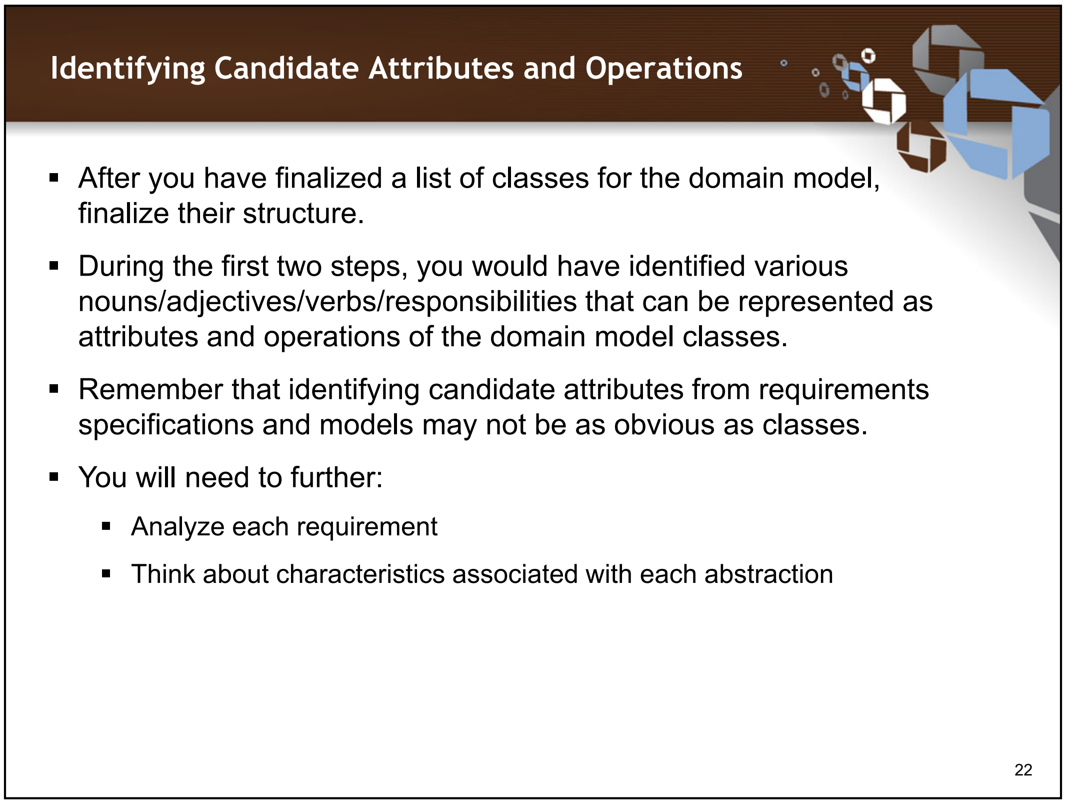
Dispatch

Warehouse

Shipping Invoice

OrderItem

OrderReceipt



## Identifying Candidate Attributes and Operations

- After you have finalized a list of classes for the domain model, finalize their structure.
- During the first two steps, you would have identified various nouns/adjectives/verbs/responsibilities that can be represented as attributes and operations of the domain model classes.
- Remember that identifying candidate attributes from requirements specifications and models may not be as obvious as classes.
- You will need to further:
  - Analyze each requirement
  - Think about characteristics associated with each abstraction

22

Adhere to the following guidelines when identifying attributes

- Attributes are identified from the information that every real life entity (represented by a candidate class) needs to maintain and manage. So, think of information associated with every entity.

- Attributes are usually written in the form of nouns followed by possessive phrases in requirements specifications, such as ‘cost of the product’. Here cost is an attribute of product.

- Omit derived attributes such as total cost of an order because it can be derived from other attributes. Instead these are candidates for operations. For example, the Order class should have an operation to calculate the total cost.

- Ensure that your candidate attributes do not refer to the same characteristic. Remove the duplicates.

- Do not identify and include characteristics that are irrelevant for the problem domain. For example, the number of family members in a customer’s family is a characteristic associated with a customer. However, it is irrelevant for your problem domain.

- If you identify a set of attributes that themselves represent an entity, consider moving them to a separate class. For example, the separation of Order and OrderItem classes represents this case. If you would only retain the Order class, it would have too many attributes and a set of which would represent a separate entity – a line item in the order.

## Identify Candidate Attributes and Operations (Cont'd)

- Adhere to the following guidelines when identifying attributes:
  - Attributes are identified from the information that every real life entity needs to maintain and manage.
  - Attributes are usually written in the form of nouns followed by possessive phrases in requirements specifications.
  - Omit derived attributes.
  - Ensure that your candidate attributes do not refer to the same characteristic. Remove the duplicates.
  - Do not identify and include characteristics that are irrelevant for the problem domain.
  - If you identify a set of attributes that themselves represent an entity, consider moving them to a separate class.

## Identify Candidate Attributes and Operations (Cont'd)

- Operations can be identified from the purpose in the noun extraction technique and responsibilities in the CRC card technique. In addition, operations are identified by thinking about the attributes (data) that need to be managed.
- Domain models do not need to have a detailed list of operations. In fact, the main elements of a static domain model are the class, attributes, and their relationships. However, you can list all possible operations evident from your analysis of the requirements.

## Steps in Domain Modeling

- Static domain modeling consists of the following steps:
  - Identify candidate classes
  - Evaluate the candidate classes
  - Identify the candidate attributes and operations for each class
  - Create the initial static domain model
  - Evaluate the relationship between the final abstractions
  - Refine the static domain model

## Creation of the Static Domain Model

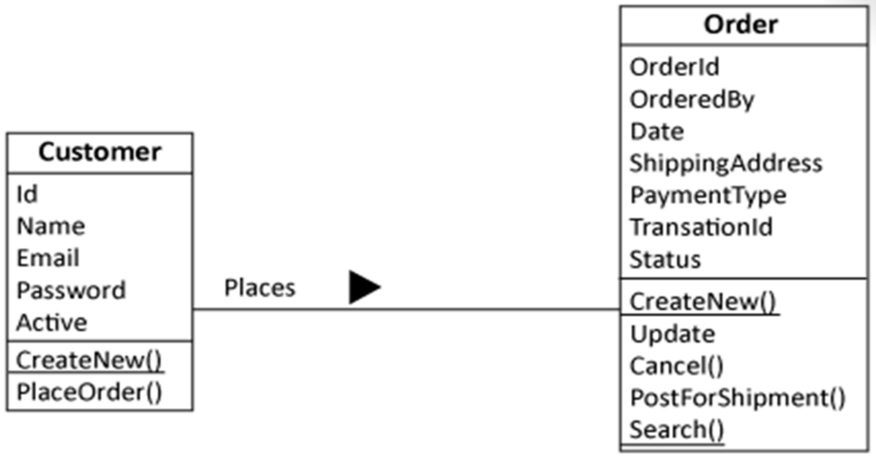
- After you finalize a list of candidate classes, their purpose, responsibilities, and collaborators, you can create the initial domain model.
- The static domain model for a problem domain can be created in UML in the form of a class diagrams.

## Association

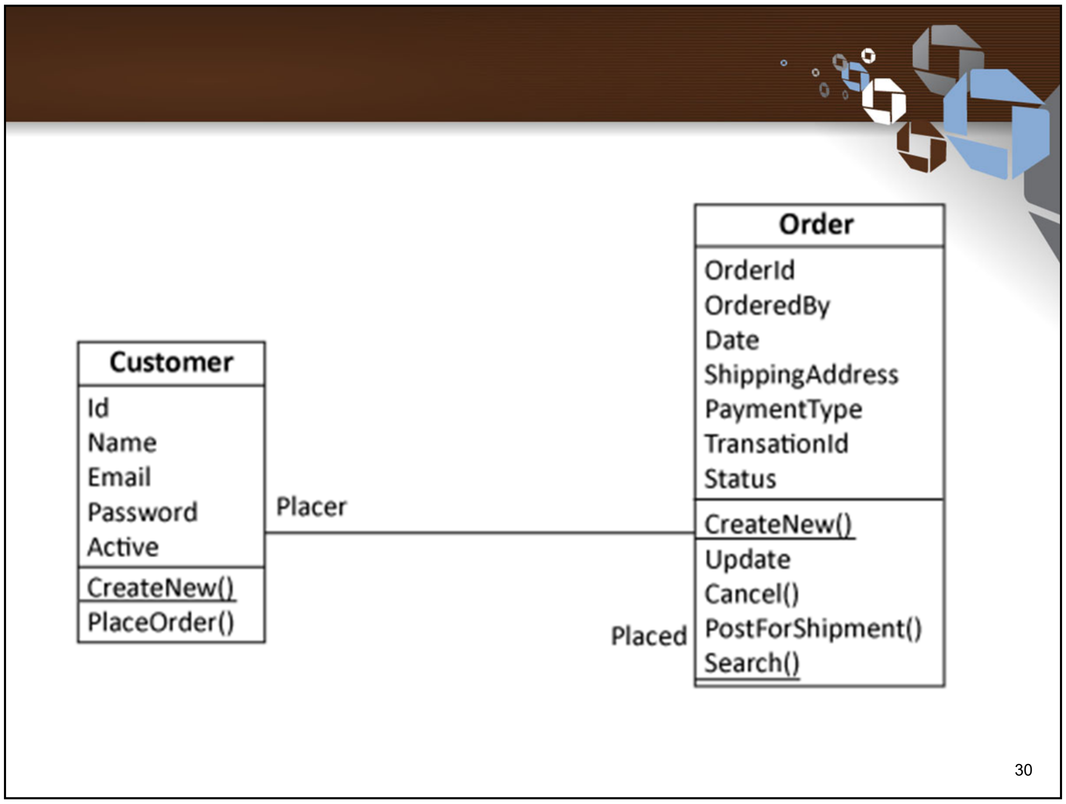
- The association relationship in a model:
  - Emulates the acquaintance relationship in real life
  - Indicates that the instances of the two classes are related
  - Indicates that one (or both) class(es) involved in the association relationship know about the existence of the other and will eventually communicate or collaborate with the other to share its data
  - Is not specific to classes and can also exist between other elements in UML diagrams, such as use cases, actors, and components

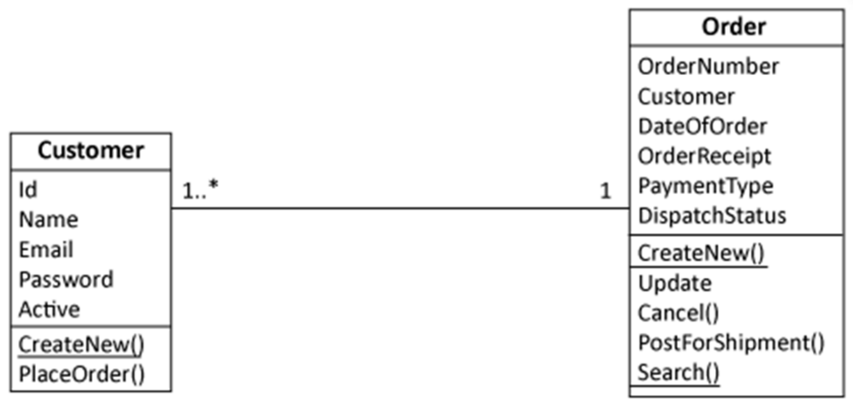
## Annotating Associations

- You can define the following annotations for associations in a class diagram:
  - Name – It represents name of the relationship that association represents.
  - Role – A role name can be assigned to each class for the role that it plays in the relationship.
  - Multiplicity – It represents the number of instances of an object that can participate in the relationship.


Customer
Id
Name
Email
Password
Active
CreateNew()
PlaceOrder()
Places
Order
OrderId
OrderedBy
Date
ShippingAddress
PaymentType
TransationId
Status
CreateNew()
Update
Cancel()
PostForShipment()
Search()

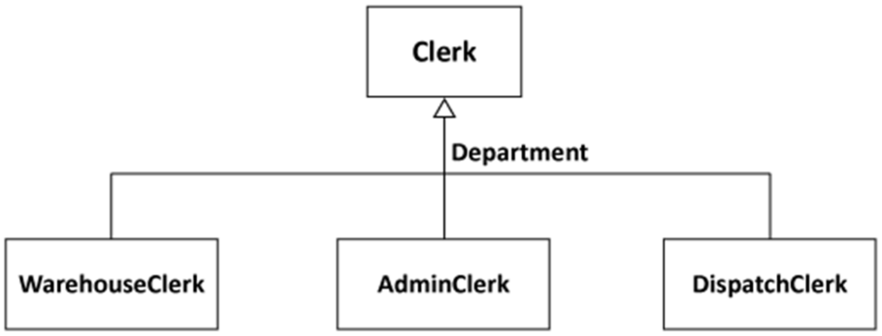
29


Customer
Id
Name
Email
Password
Active
CreateNew()
PlaceOrder()
Placer
Placed
Order
OrderId
OrderedBy
Date
ShippingAddress
PaymentType
TransationId
Status
CreateNew()
Update
Cancel()
PostForShipment()
Search()
30


Customer
Id
Name
Email
Password
Active
CreateNew()
PlaceOrder()
1..*
1
Order
OrderNumber
Customer
DateOfOrder
OrderReceipt
PaymentType
DispatchStatus
CreateNew()
Update
Cancel()
PostForShipment()
Search()

## Guidelines for Identifying and Modeling Associations

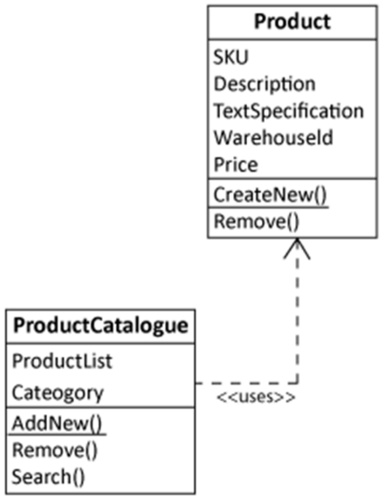
- Refer to the following guidelines for identifying and modeling associations:
  - Read the requirements specifications and identify which entities are described by using prepositions.
  - Do not consider verbs that are candidates for being operations of a class as associations.
  - Try to simplify and make the multiplicity of the association finite rather than using a multiplicity of many.
  - Eliminate derived or redundant associations, which can be expressed in terms of other associations.
  - Do not use aggregations if you are not sure of them during domain modeling.


Clerk
Department
WarehouseClerk
AdminClerk
DispatchClerk

33

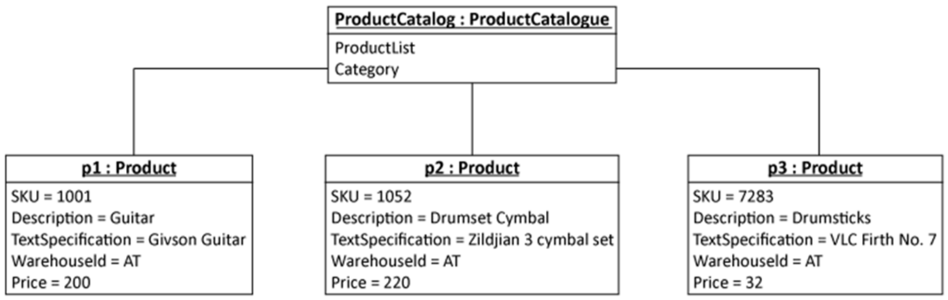
## Dependency

- When a usage dependency exists between a supplier and a client, any change to the supplier may require the functionality of the client to be changed.
- This dependency can be shown as a dashed line with an open arrowhead at the end of the supplier and the <<uses>> stereotype


Product
SKU
Description
TextSpecification
WarehouseId
Price
CreateNew()
Remove()
ProductCatalogue
ProductList
Cateogory
AddNew()
Remove()
Search()
<<uses>>

## Object Diagrams

- An object diagram:
  - Is an instance of the class diagram.
  - Helps you view a snapshot of the class diagram optionally with real life values for its attributes and relationships between the instances of the classes at a particular point of time
  - May contain a subset of the classes in the class diagram, which are relevant for the snapshot.
  - To draw an object diagram, you need to perform the following tasks:
    1. Identify the important objects that need to participate in the snapshot.
    2. Create objects for the object diagram.
    3. Define the relationships between these objects.
    4. Define the runtime state of the object in terms of values of crucial attributes in each object.


ProductCatalog : ProductCatalogue
ProductList
Category
p1 : Product
SKU = 1001
Description = Guitar
TextSpecification = Givson Guitar
WarehouseId = AT
Price = 200
p2 : Product
SKU = 1052
Description = Drumset Cymbal
TextSpecification = Zildjian 3 cymbal set
WarehouseId = AT
Price = 220
p3 : Product
SKU = 7283
Description = Drumsticks
TextSpecification = VLC Firth No. 7
WarehouseId = AT
Price = 32