10

Object-Oriented Analysis and Modeling Using the UML

Overview

This is the first of two chapters on object-oriented tools and techniques for system development. This chapter teaches students the important skill of object modeling during systems analysis. The students will learn about the various unified modeling language (UML) diagrams and build the ones that apply to systems analysis. Object-oriented design is covered in Chapter 18.

Chapter to Course Sequencing

Depending on the intent of the instructor and the structure of the course, this chapter can follow a number of chapters. For those courses that want to focus on object-oriented tools and techniques, it should follow Chapter 7 and can even be used in place of Chapters 8 an 9. If the intent is to expose the student to data, process, and object modeling techniques, this module can be used in conjunction with Chapters 8 and 9 in any order. For those courses that treat object-oriented development as advanced material this chapter should be covered last if time permits.

What's Different Here and Why?

The following changes have been made to this chapter in the seventh edition:

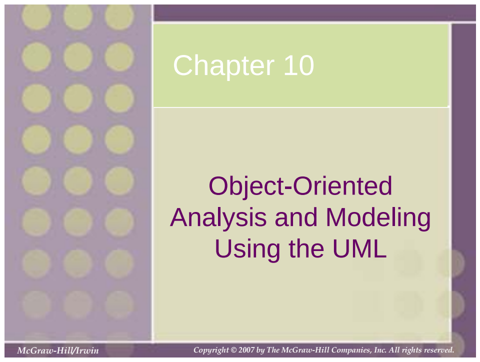
1. The biggest change is that this chapter as moved from chapter 11 to chapter 10. This places all three analysis methods chapters (8, 9, and 10) together prior to the chapter on the system proposal.
2. As with all chapters, we have streamlined the SoundStage episode into a quick narrative introduction to the concepts presented the chapter.
3. The chapter has been revised for UML 2.0.
4. We intentionally used the terms object class and object instance as consistently as possible throughout the chapter. In teaching we have found that students get confused with the terms class, object, and instance. By focusing on just the difference between object class and object instance and being more consistent in what they are called, we hope to avoid confusion.

5. We specifically relate the concept of multiplicity to the concept of cardinality from data modeling.
6. While still distinguishing between composition and aggregation, we note that UML 2.0 has dropped the notation for aggregation, and we explain why.
7. We have included a table listing every UML 2.0 diagram and a list of which of the diagrams are covered in which chapter of the text.
8. We have added expanded the discussion of activity diagrams, including partitions, and have revised the coverage for UML 2.0 notation. We have also added a "Guidelines for Constructing Activity Diagrams" section.
9. We have added coverage for system sequence diagrams, which is another way to model a specific use case and lays the groundwork for design-level sequence diagrams in chapter 18.
10. We added a section noting differences between class diagrams and data diagrams. This was based on confusion we see in students in the classroom.

Lesson Planning Notes for Slides

The following instructor notes, keyed to slide images from the PowerPoint repository, are intended to help instructors integrate the slides into their individual lesson plans for this chapter.

Slide 1



Chapter 10

Object-Oriented Analysis and Modeling Using the UML

McGraw-Hill/Irwin

Copyright © 2007 by The McGraw-Hill Companies, Inc. All rights reserved.

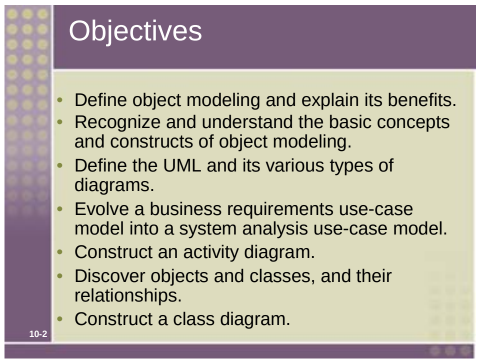
slide appearance after initial mouse click in slide show mode

This repository of slides is intended to support the named chapter. The slide repository should be used as follows:
Copy the file to a unique name for your course and unit.
Edit the file by deleting those slides you don't want to cover, editing other slides as appropriate to your course, and adding slides as desired.
Print the slides to produce transparency masters or print directly to film or present the slides using a computer image projector.

Each slide includes instructor notes. To view those notes in PowerPoint, click-left on the View Menu; then click left on Notes View sub-menu. You may need to scroll down to see the instructor notes.

The instructor notes are also available in hard-copy as the Instructor Guide to Accompany Systems Analysis and Design Methods, 6/ed.

Slide 2



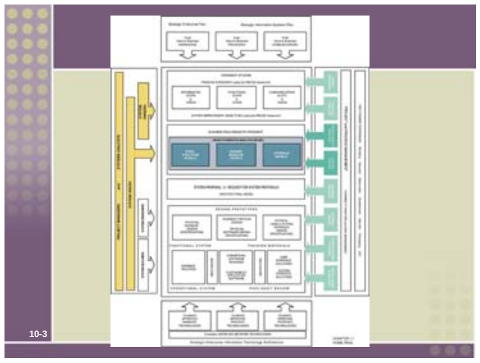
Objectives

- Define object modeling and explain its benefits.
- Recognize and understand the basic concepts and constructs of object modeling.
- Define the UML and its various types of diagrams.
- Evolve a business requirements use-case model into a system analysis use-case model.
- Construct an activity diagram.
- Discover objects and classes, and their relationships.
- Construct a class diagram.

10-2

No additional notes.

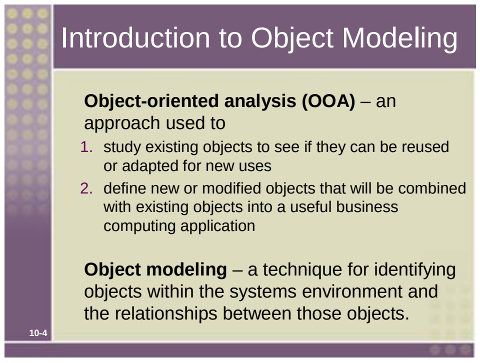
Slide 3



10-3

Teaching Notes
This slide shows the how this chapter's content fits with the building blocks framework used throughout the textbook. Since the object-oriented approach attempts to unify DATA, PROCESSES, and COMMUNICATION, the emphasis of this chapter is upon all three. It also reflects the fact that OO analysis is done during the Requirements Analysis and Logical Design phases and that it involves not only systems analysts…but owners and users.

Slide 4



Introduction to Object Modeling

Object-oriented analysis (OOA) – an approach used to

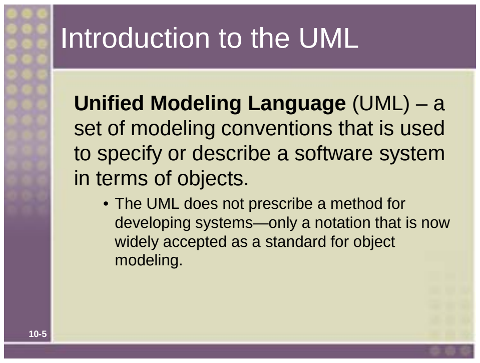
1. study existing objects to see if they can be reused or adapted for new uses
2. define new or modified objects that will be combined with existing objects into a useful business computing application

Object modeling – a technique for identifying objects within the systems environment and the relationships between those objects.

10-4

Teaching Notes
The object modeling technique prescribes the use of methodologies and diagramming notations that are completely different from the ones used for data modeling and process modeling.
In the late 1980s and early 1990s many different object-oriented methods were being used throughout the industry. The most notable of these were Grady Booch's *Booch Method,* James Rumbaugh's *Object Modeling Technique (OMT),* and Ivar Jacobson's *Object-Oriented Software Engineering (OOSE)*.

Slide 5



Introduction to the UML

Unified Modeling Language (UML) – a set of modeling conventions that is used to specify or describe a software system in terms of objects.

- The UML does not prescribe a method for developing systems—only a notation that is now widely accepted as a standard for object modeling.

10-5

Teaching Notes

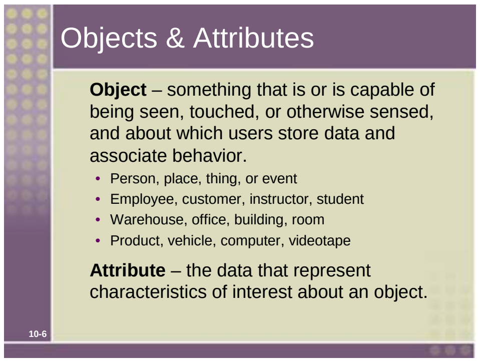
In 1994 Grady Booch and James Rumbaugh joined forces to merge their respective object-oriented development methods with the goal of creating a single, standard process for developing object-oriented systems.

Ivar Jacobson joined them in 1995 and the three altered their focus to create a standard object modeling language instead of a standard object-oriented approach or method.

Referencing their own work as well as countless others in the OO industry, the Unified Modeling Language (UML) version 1.0 was released in 1997. UML version 2.0 is expected to be released in late 2000.

At the time of this writing, Booch, Rumbaugh, and Jacobson have developed and marketed an object modeling methodology called the Unified Method or Objectory.

Slide 6



Objects & Attributes

Object – something that is or is capable of being seen, touched, or otherwise sensed, and about which users store data and associate behavior.

- Person, place, thing, or event
- Employee, customer, instructor, student
- Warehouse, office, building, room
- Product, vehicle, computer, videotape

Attribute – the data that represent characteristics of interest about an object.

10-6

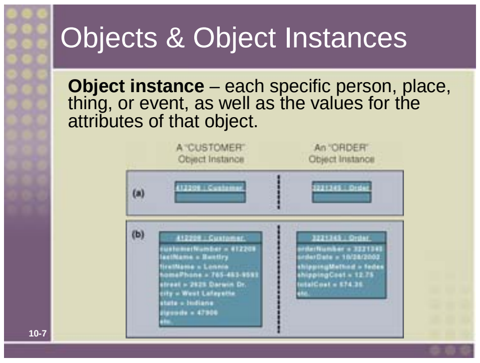
Teaching Notes

There are several concepts that object-oriented analysis is based on. Some of these concepts require a totally new way of thinking about systems and the development process. These concepts have presented a formidable challenge to veteran developers who must relearn how they have traditionally viewed systems.

Object-oriented approaches to systems development are concerned with identifying attributes that are of interest regarding an object. It is important to note that with advances in technology, attributes have evolved to include more than simple data characteristics. Today, objects may include newer attribute types, such as a bitmap or a picture sound, or even video.

Have students provide examples of objects, instances of objects, and their attributes that exist in the classroom. For instance, pen is an object, the pen I use is an instance of that object, color of ink is an attribute.

Slide 7



Objects & Object Instances

Object instance – each specific person, place, thing, or event, as well as the values for the attributes of that object.

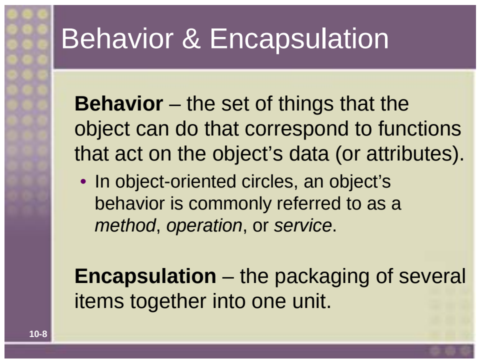
10-7

Teaching Notes

This figure depicts the symbol for representing an object instance using the UML modeling notation. An object is represented using a rectangle. The name of the object instance and its classification are underlined and appear at the top of the symbol. The attribute values for the object instance are optionally recorded within the symbol and are separated from the object name with a line.

The name of an object instance is the value of the attribute that uniquely identifies it. The attribute customer number, whose value is 412209, uniquely identifies that instance of customer. Thus, 412209 is the name of the object instance and customer is its classification.

Slide 8



Behavior & Encapsulation

Behavior – the set of things that the object can do that correspond to functions that act on the object’s data (or attributes).

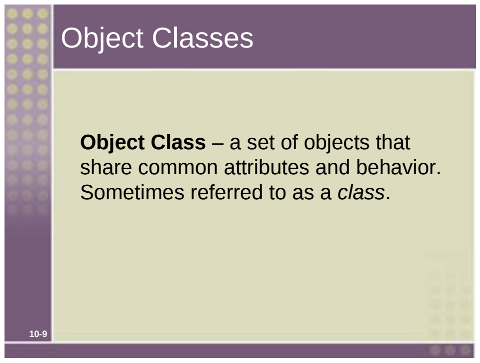
- In object-oriented circles, an object’s behavior is commonly referred to as a *method*, *operation*, or *service*.

Encapsulation – the packaging of several items together into one unit.

10-8

Teaching Notes
In encapsulation, both attributes and behavior of the object are packaged together. The only way to access an object's attributes is through that object’s behaviors. No other object may perform that object’s behavior.
Have students identify the behaviors of a door, window, or VCR.

Slide 9



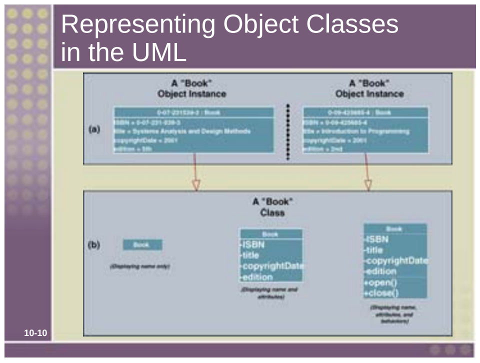
Object Classes

Object Class – a set of objects that share common attributes and behavior. Sometimes referred to as a *class*.

10-9

Conversion Notes
This is a new slide in the seventh edition

Slide 10

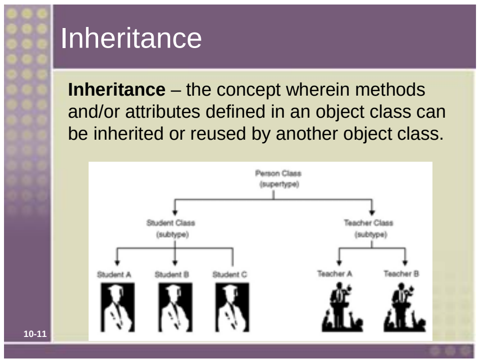


Representing Object Classes in the UML

10-10

Teaching Notes
In UML an object class is represented using a rectangle symbol.
The object rectangle is divided into three portions.
The top portion contains the name of the class.
The middle portion contains the names of the attributes.
The lower portion contains the behaviors (or methods).
To simplify the appearance of a diagram, or to specify more details about a class, class symbols can be drawn without methods or attributes, or the attribute portion can be expanded to include data types, lengths, etc. The appearance depends on what the author of the diagram is intending to communicate.

Slide 11



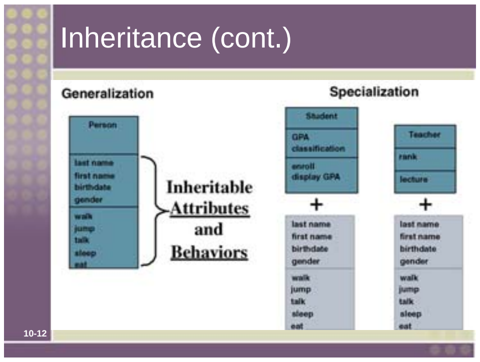
Inheritance

Inheritance – the concept wherein methods and/or attributes defined in an object class can be inherited or reused by another object class.

10-11

Teaching Notes
Inheritance is a critical concept of OO. In fact for a programming language or DBMS to be considered OO, it must support inheritance. In inheritance, the child class inherits the attributes and methods from its parent class(s).

Slide 12

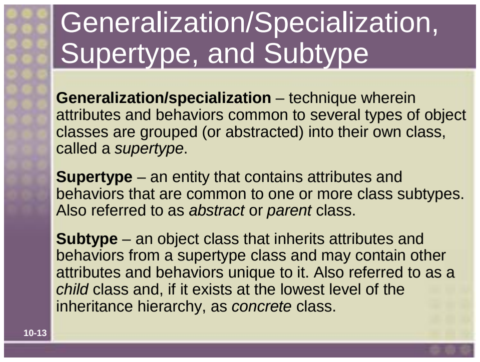


Inheritance (cont.)

10-12

Teaching Notes
The terms Generalization and Specialization will be defined on the next slide.
Walk the students through this diagram.
The Person object has an attribute last name. Therefore the Student and Teacher objects that are based on Person also have an attribute last name as well as their own attributes (GPA or rank).
The Person object has a method walk. Therefore Student and Teach also have a method walk as well as their own methods (enroll or lecture).

Slide 13



Generalization/Specialization, Supertype, and Subtype

Generalization/specialization – technique wherein attributes and behaviors common to several types of object classes are grouped (or abstracted) into their own class, called a *supertype*.

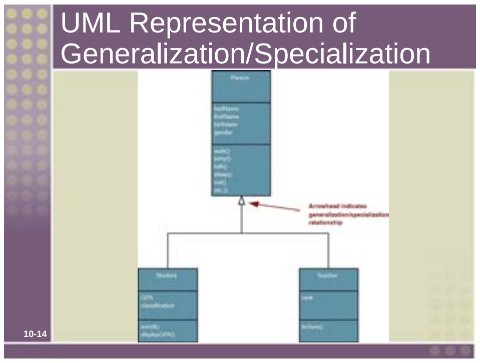
Supertype – an entity that contains attributes and behaviors that are common to one or more class subtypes. Also referred to as *abstract* or *parent* class.

Subtype – an object class that inherits attributes and behaviors from a supertype class and may contain other attributes and behaviors unique to it. Also referred to as a *child* class and, if it exists at the lowest level of the inheritance hierarchy, as *concrete* class.

10-13

Teaching Notes
The previous slide illustrates these terms.
The class supertype will have one or more *one-to-one* relationships to object class *subtypes*.
These relationships are sometimes called "IS A" relationships (or "WAS A" or "COULD BE A") because each instance of the supertype "is also an" instance of one or more subtypes.

Slide 14

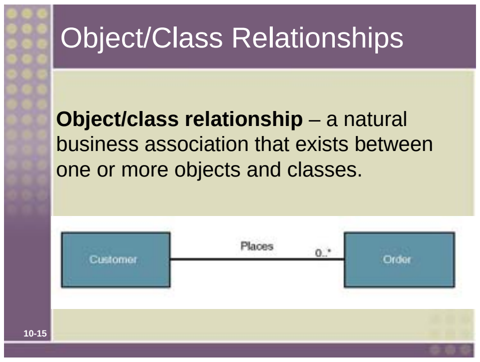


UML Representation of Generalization/Specialization

10-14

Teaching Notes
Have students identify what attributes and methods are inherited by the STUDENT and TEACHER classes.

Slide 15



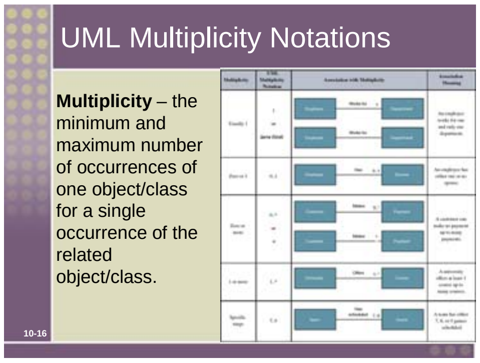
Object/Class Relationships

Object/class relationship – a natural business association that exists between one or more objects and classes.

10-15

Teaching Notes
Objects and classes do not exist in isolation. The things they represent interact with and impact one another to support the business mission.
An object/class relationship is graphically illustrated in UML as a connecting line between two classes. This relationship is commonly referred to as an association. The line is labeled with a verb phrase that describes the association. All associations are implicitly bidirectional, meaning that they can interpreted in both directions.
The figure above shows the complexity or degree of each association. For example, in the above business assertions, we must also answer the following questions:
Must there exist an instance of CUSTOMER for each instance of ORDER? Yes!
Must there exist an instance of ORDER for each instance of CUSTOMER? No!
How many instances of ORDER can exist for each instance of CUSTOMER? Many!
How many instances of CUSTOMER can exist for each instance of ORDER? One!

Slide 16



UML Multiplicity Notations

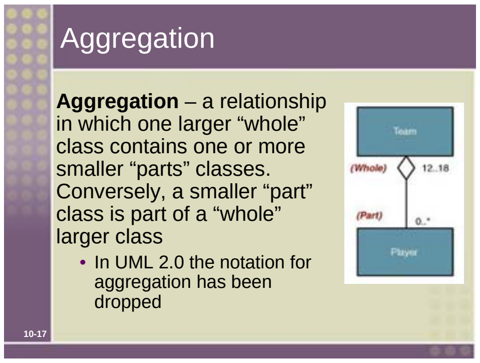
Multiplicity – the minimum and maximum number of occurrences of one object/class for a single occurrence of the related object/class.

Multiplicity	UML Multiplicity Notation	Association with Multiplicity	Association Meaning
Exactly 1	1 or leave blank	Employee Works for 1 Department; Employee Works for Department	An employee works for one and only one department.
Zero or 1	0..1	Employee Has 0..1 Spouse	An employee has either one or no spouse.
Zero or more	0..* or *	Customer Makes 0..* Payment; Customer Makes * Payment	A customer can make no payment up to many payments.
1 or more	1..*	University Offers 1..* Course	A university offers at least 1 course up to many courses.
Specific range	7..9	Team Has scheduled 7..9 Game	A team has either 7, 8, or 9 games scheduled.

10-16

Teaching Notes
Because all associations are implicitly bidirectional, multiplicity must be defined in both directions for every association.

Slide 17



Aggregation

Aggregation – a relationship in which one larger “whole” class contains one or more smaller “parts” classes. Conversely, a smaller “part” class is part of a “whole” larger class

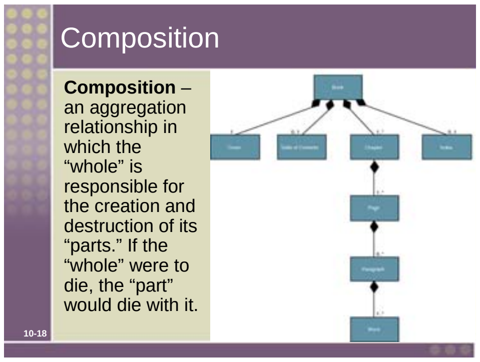
- In UML 2.0 the notation for aggregation has been dropped

Team
(Whole)
12..18
(Part)
0..*
Player

10-17

Teaching Notes
Aggregation relationships do not support inheritance. Their benefit lies in the ability to send a message to the parent class and that message is automatically applied to all the child classes.
Have students provide other examples of objects where aggregation relationships are appropriate (car – or any bill of material,, order-line item, etc.).

Slide 18



Composition

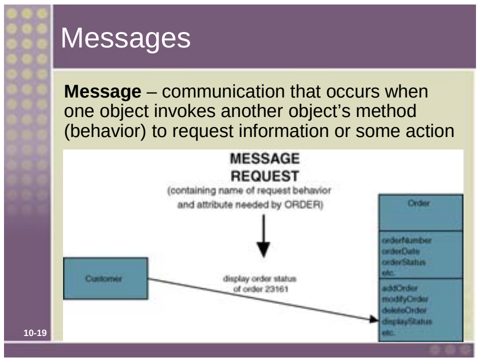
Composition – an aggregation relationship in which the “whole” is responsible for the creation and destruction of its “parts.” If the “whole” were to die, the “part” would die with it.

Book
Cover
Table of Contents
Chapter
Index
Page
Paragraph
Word

10-18

Teaching Notes
All composition relationships are aggregation relationships. But not all aggregation relationships are composition relationships.
Ask students why composition is appropriate for the Book and Chapter classes but not for the Team class.

Slide 19



Messages

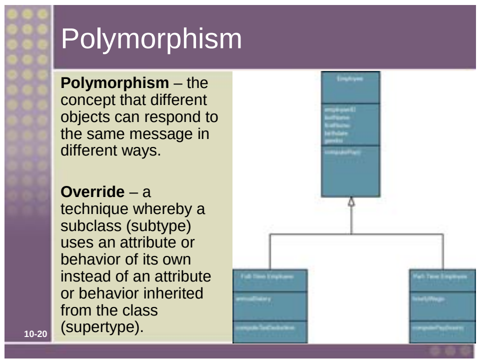
Message – communication that occurs when one object invokes another object’s method (behavior) to request information or some action

MESSAGE REQUEST
(containing name of request behavior and attribute needed by ORDER)
Customer
display order status of order 23161
Order
orderNumber
orderDate
orderStatus
etc.
addOrder
modifyOrder
deleteOrder
displayStatus
etc.

10-19

Teaching Notes
The object sending a message does not need to know how the receiving object is organized internally or how the behavior is to be accomplished, only that it responds to the request in a well-defined way.
A message can be sent only between two objects that have an association between them.

Slide 20


Employee
employeeID
lastName
firstName
birthDate
gender
computePay()
Full-Time Employee
annualSalary
computeTaxDeduction
Part-Time Employee
hourlyWage
computePay(hours)

Polymorphism

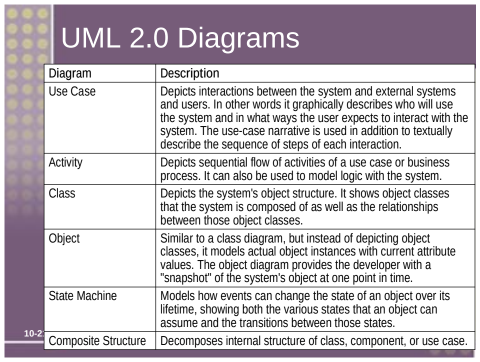
Polymorphism – the concept that different objects can respond to the same message in different ways.

Override – a technique whereby a subclass (subtype) uses an attribute or behavior of its own instead of an attribute or behavior inherited from the class (supertype).

10-20

Teaching Notes
Polymorphism supports reusability in that the same message can be sent to different objects and be interpreted different ways. For example, let's say we have a method called "Compute Pay" and two objects named FULL-TIME EMPLOYEE and PART-TIME EMPLOYEE. The same "compute pay" message can be sent to both objects but how each object reacts/responds to the message is different. A full-time employee's pay may be composed of a weekly salary (minus deductions) whereas a part-time employee only gets paid for the hours worked (minus deductions). When the PART-TIME EMPLOYEE object receives a message to "compute pay," it will override the behavior of the EMPLOYEE supertype and use its own behavior. But because of polymorphism, the object sending the message never knows the difference.

Slide 21



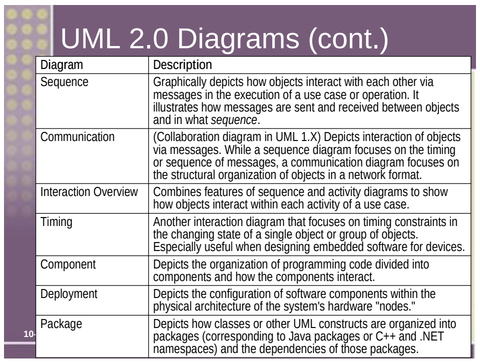
UML 2.0 Diagrams

Diagram	Description
Use Case	Depicts interactions between the system and external systems and users. In other words it graphically describes who will use the system and in what ways the user expects to interact with the system. The use-case narrative is used in addition to textually describe the sequence of steps of each interaction.
Activity	Depicts sequential flow of activities of a use case or business process. It can also be used to model logic with the system.
Class	Depicts the system's object structure. It shows object classes that the system is composed of as well as the relationships between those object classes.
Object	Similar to a class diagram, but instead of depicting object classes, it models actual object instances with current attribute values. The object diagram provides the developer with a "snapshot" of the system's object at one point in time.
State Machine	Models how events can change the state of an object over its lifetime, showing both the various states that an object can assume and the transitions between those states.
Composite Structure	Decomposes internal structure of class, component, or use case.

10-21

Conversion Notes
With the seventh edition we have switched to UML 2.0.
Teaching Notes
As we study an overview of the systems analysis life cycle, three chapters will delve into the core UML diagrams:
Chapter 7 – FAST Requirements Analysis Phase
 Use Case Diagrams
Chapter 10 – FAST Logical Design Phase
 Activity Diagrams
 System Sequence Diagrams (a high-level kind of Sequence Diagram)
 Class Diagrams
Chapter 18 – FAST Physical Design Phase
 Sequence Diagrams
 Class Diagrams (with more detail)
 State Machine Diagrams
 Communication Diagrams
 Component Diagrams
 Deployment Diagrams

Slide 22



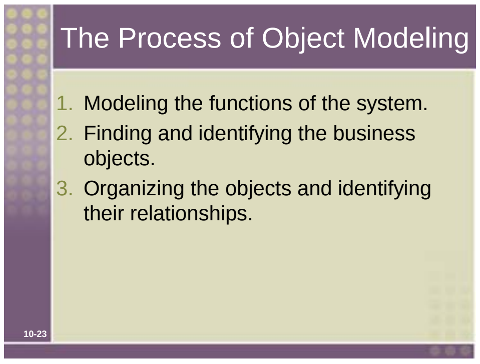
UML 2.0 Diagrams (cont.)

Diagram	Description
Sequence	Graphically depicts how objects interact with each other via messages in the execution of a use case or operation. It illustrates how messages are sent and received between objects and in what *sequence*.
Communication	(Collaboration diagram in UML 1.X) Depicts interaction of objects via messages. While a sequence diagram focuses on the timing or sequence of messages, a communication diagram focuses on the structural organization of objects in a network format.
Interaction Overview	Combines features of sequence and activity diagrams to show how objects interact within each activity of a use case.
Timing	Another interaction diagram that focuses on timing constraints in the changing state of a single object or group of objects. Especially useful when designing embedded software for devices.
Component	Depicts the organization of programming code divided into components and how the components interact.
Deployment	Depicts the configuration of software components within the physical architecture of the system's hardware "nodes."
Package	Depicts how classes or other UML constructs are organized into packages (corresponding to Java packages or C++ and .NET namespaces) and the dependencies of those packages.

10-

Conversion Notes
With the seventh edition we have switched to UML 2.0.
Teaching Notes
As we study an overview of the systems analysis life cycle, three chapters will delve into the core UML diagrams:
Chapter 7 – FAST Requirements Analysis Phase
Use Case Diagrams
Chapter 10 – FAST Logical Design Phase
Activity Diagrams
System Sequence Diagrams (a high-level kind of Sequence Diagram)
Class Diagrams
Chapter 18 – FAST Physical Design Phase
Sequence Diagrams
Class Diagrams (with more detail)
State Machine Diagrams
Communication Diagrams
Component Diagrams
Deployment Diagrams

Slide 23



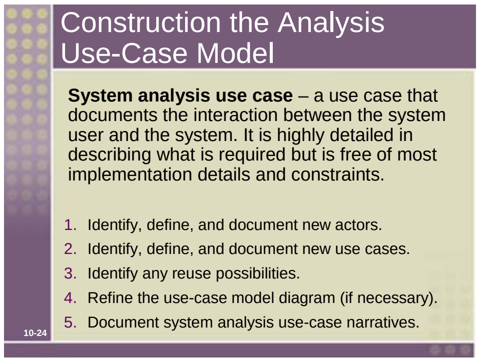
The Process of Object Modeling

1. Modeling the functions of the system.
2. Finding and identifying the business objects.
3. Organizing the objects and identifying their relationships.

10-23

Teaching Notes
These are object-oriented analysis general activities

Slide 24



Construction the Analysis Use-Case Model

System analysis use case – a use case that documents the interaction between the system user and the system. It is highly detailed in describing what is required but is free of most implementation details and constraints.

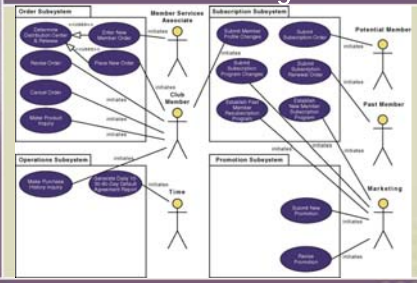
1. Identify, define, and document new actors.
2. Identify, define, and document new use cases.
3. Identify any reuse possibilities.
4. Refine the use-case model diagram (if necessary).
5. Document system analysis use-case narratives.

10-24

Teaching Notes
As the analyst continues to learn more about the system and its requirements, the analyst may discover new actors who interact with the system and new use cases.
When two use cases have the same business goal but different users or interface technology, both use cases may share common steps. We can extract these common steps into a separate use case called an abstract use case. Or we can extract complex steps of a single use case into an extension use case.

Slide 25

Revised System Use-Case Model Diagram


Order Subsystem
«includes»
Determine Distribution Center & Release
Enter New Member Order
«includes»
Revise Order
Place New Order
Cancel Order
Make Product Inquiry
Member Services Associate
Club Member
initiates
Operations Subsystem
Make Purchase History Inquiry
Generate Daily 10-30-60-Day Default Agreement Report
Time
Subscription Subsystem
Submit Member Profile Changes
Submit Subscription Order
Submit Subscription Program Changes
Submit Subscription Renewal Order
Establish Past Member Resubscription Program
Establish New Member Subscription Program
Potential Member
Past Member
Marketing
Promotion Subsystem
Submit New Promotion
Revise Promotion

10-25

Teaching Notes
A use case model diagram can be used to graphically depict the system scope and boundaries in terms of use cases and actors.
The use case model diagram for the Member Services System is shown in the above figure. It was created using Popkin Software's *System Architect* and represents the relationships between the actors and use cases defined for each business subsystem.
The subsystems (UML package symbol) represent logical functional areas of business processes.
The partitioning of system behavior into subsystems is very important in understanding the system architecture and is very key to defining your development strategy — which use cases will be developed first and by whom.

Slide 26

Use-Case Narrative

Member Services System

[illegible]

10-26

Teaching Notes
If Chapter 7 was covered, this will be review

Slide 27

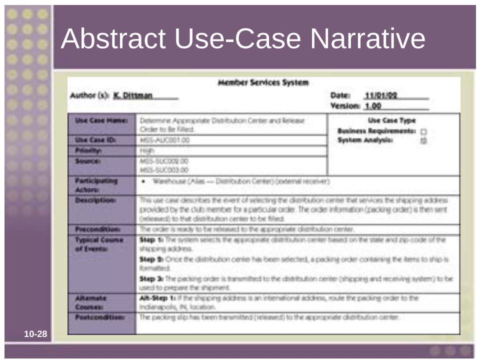
Use-Case Narrative (cont.)

[illegible]

10-27

Teaching Notes
If Chapter 7 was covered, this will be review

Slide 28



Abstract Use-Case Narrative

Member Services System

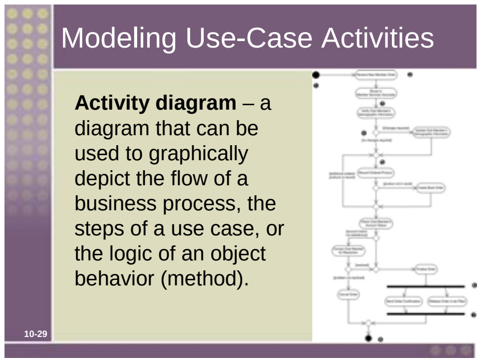
Author (s): K. Dittman **Date:** 11/01/02 **Version:** 1.00

Use Case Name:	Determine Appropriate Distribution Center and Release Order to Be Filled.	**Use Case Type** **Business Requirements:** ☐ **System Analysis:** ☑
Use Case ID:	MSS-AUC001.00	
Priority:	High	
Source:	MSS-SUC002.00 MSS-SUC003.00	
Participating Actors:	• Warehouse (Alias — Distribution Center) (external receiver)	
Description:	This use case describes the event of selecting the distribution center that services the shipping address provided by the club member for a particular order. The order information (packing order) is then sent (released) to that distribution center to be filled.	
Precondition:	The order is ready to be released to the appropriate distribution center.	
Typical Course of Events:	**Step 1:** The system selects the appropriate distribution center based on the state and zip-code of the shipping address. **Step 2:** Once the distribution center has been selected, a packing order containing the items to ship is formatted. **Step 3:** The packing order is transmitted to the distribution center (shipping and receiving system) to be used to prepare the shipment.	
Alternate Courses:	**Alt-Step 1:** If the shipping address is an international address, route the packing order to the Indianapolis, IN, location.	
Postcondition:	The packing slip has been transmitted (released) to the appropriate distribution center.	

10-28

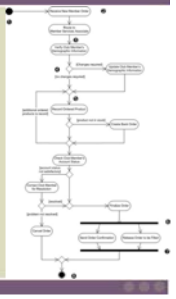
No additional notes.

Slide 29



Modeling Use-Case Activities

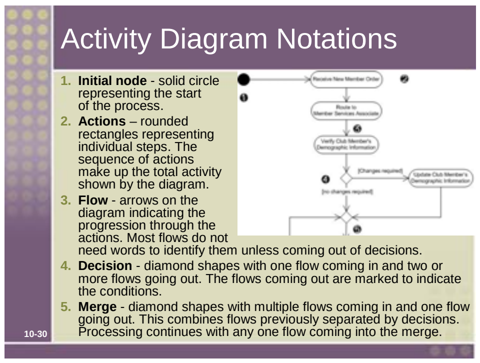
Activity diagram – a diagram that can be used to graphically depict the flow of a business process, the steps of a use case, or the logic of an object behavior (method).



10-29

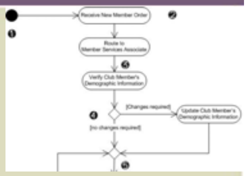
Conversion Notes
In UML 2.0 the activity diagram renames some symbols and uses them more formally.

Slide 30



Activity Diagram Notations

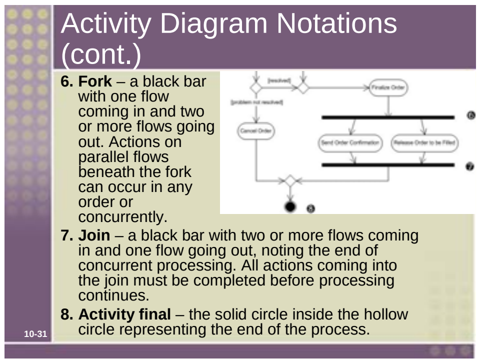
1. **Initial node** - solid circle representing the start of the process.
2. **Actions** – rounded rectangles representing individual steps. The sequence of actions make up the total activity shown by the diagram.
3. **Flow** - arrows on the diagram indicating the progression through the actions. Most flows do not need words to identify them unless coming out of decisions.
4. **Decision** - diamond shapes with one flow coming in and two or more flows going out. The flows coming out are marked to indicate the conditions.
5. **Merge** - diamond shapes with multiple flows coming in and one flow going out. This combines flows previously separated by decisions. Processing continues with any one flow coming into the merge.


Receive New Member Order
Route to Member Services Associate
Verify Club Member's Demographic Information
[Changes required]
Update Club Member's Demographic Information
[no changes required]

10-30

Conversion Notes
This slide has been extensively revised for the seventh edition to correspond with UML 2.0 notation.

Slide 31



Activity Diagram Notations (cont.)

6. Fork – a black bar with one flow coming in and two or more flows going out. Actions on parallel flows beneath the fork can occur in any order or concurrently.

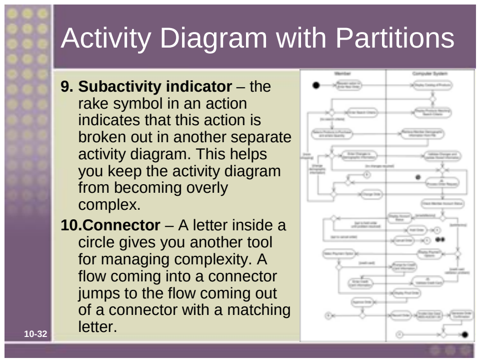
7. Join – a black bar with two or more flows coming in and one flow going out, noting the end of concurrent processing. All actions coming into the join must be completed before processing continues.

8. Activity final – the solid circle inside the hollow circle representing the end of the process.

10-31

Conversion Notes
This slide has been extensively revised for the seventh edition to correspond with UML 2.0 notation.

Slide 32



Activity Diagram with Partitions

9. Subactivity indicator – the rake symbol in an action indicates that this action is broken out in another separate activity diagram. This helps you keep the activity diagram from becoming overly complex.

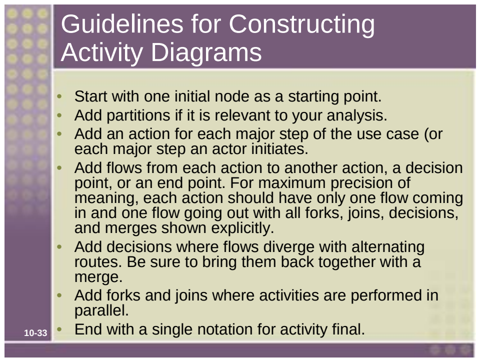
10. Connector – A letter inside a circle gives you another tool for managing complexity. A flow coming into a connector jumps to the flow coming out of a connector with a matching letter.

10-32

Conversion Notes
This slide has been extensively revised for the seventh edition to correspond with UML 2.0 notation.

Teaching Notes
In addition to the subactivity indicator and the connector, this activity diagram shows partitions (formerly swmlanes). Partitions are especially useful when including receiver actors.

Slide 33



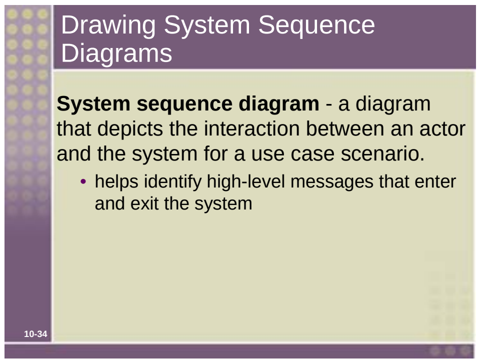
Guidelines for Constructing Activity Diagrams

- Start with one initial node as a starting point.
- Add partitions if it is relevant to your analysis.
- Add an action for each major step of the use case (or each major step an actor initiates.
- Add flows from each action to another action, a decision point, or an end point. For maximum precision of meaning, each action should have only one flow coming in and one flow going out with all forks, joins, decisions, and merges shown explicitly.
- Add decisions where flows diverge with alternating routes. Be sure to bring them back together with a merge.
- Add forks and joins where activities are performed in parallel.
- End with a single notation for activity final.

10-33

Conversion Notes
This is a new slide for the seventh edition.

Slide 34



Drawing System Sequence Diagrams

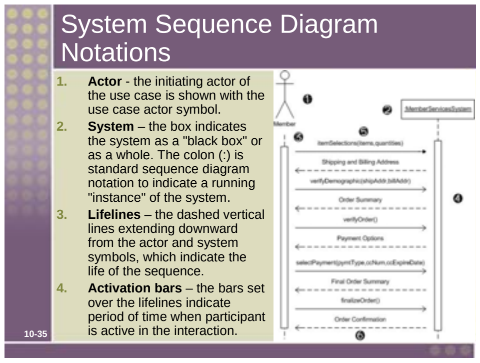
System sequence diagram - a diagram that depicts the interaction between an actor and the system for a use case scenario.

- helps identify high-level messages that enter and exit the system

10-34

Conversion Notes
This is a new topic in the seventh edition
Teaching Notes
Emphasize that unlike a context diagram, the system sequence diagram does not try to depict the entire system. It depicts only the interactions for a single scenario of a single use case.

Slide 35



System Sequence Diagram Notations

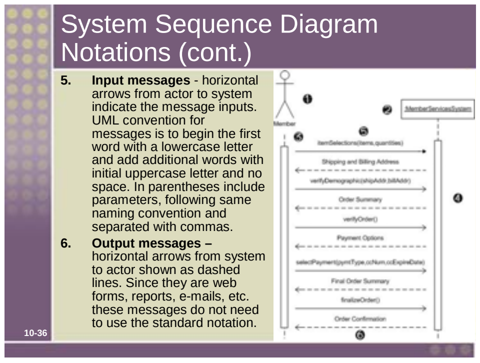
1. **Actor** - the initiating actor of the use case is shown with the use case actor symbol.
2. **System** – the box indicates the system as a "black box" or as a whole. The colon (:) is standard sequence diagram notation to indicate a running "instance" of the system.
3. **Lifelines** – the dashed vertical lines extending downward from the actor and system symbols, which indicate the life of the sequence.
4. **Activation bars** – the bars set over the lifelines indicate period of time when participant is active in the interaction.

Member
:MemberServicesSystem
1
2
3
4
5
6
itemSelections(items,quantities)
Shipping and Billing Address
verifyDemographic(shipAddr,billAddr)
Order Summary
verifyOrder()
Payment Options
selectPayment(pymtType,ccNum,ccExpireDate)
Final Order Summary
finalizeOrder()
Order Confirmation

10-35

Teaching Notes
The concepts are the same as with design-level sequence diagrams. But the system appears as a single entity.

Slide 36



System Sequence Diagram Notations (cont.)

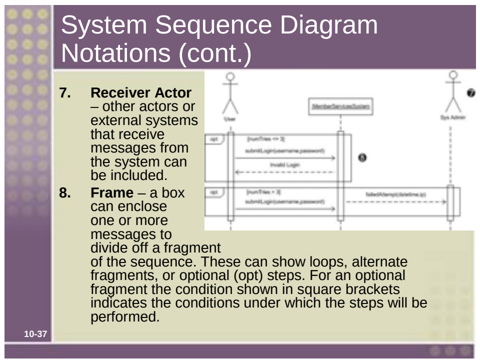
5. **Input messages** - horizontal arrows from actor to system indicate the message inputs. UML convention for messages is to begin the first word with a lowercase letter and add additional words with initial uppercase letter and no space. In parentheses include parameters, following same naming convention and separated with commas.
6. **Output messages –** horizontal arrows from system to actor shown as dashed lines. Since they are web forms, reports, e-mails, etc. these messages do not need to use the standard notation.

Member
:MemberServicesSystem
1
2
3
4
5
6
itemSelections(items,quantities)
Shipping and Billing Address
verifyDemographic(shipAddr,billAddr)
Order Summary
verifyOrder()
Payment Options
selectPayment(pymtType,ccNum,ccExpireDate)
Final Order Summary
finalizeOrder()
Order Confirmation

10-36

No additional notes.

Slide 37



System Sequence Diagram Notations (cont.)

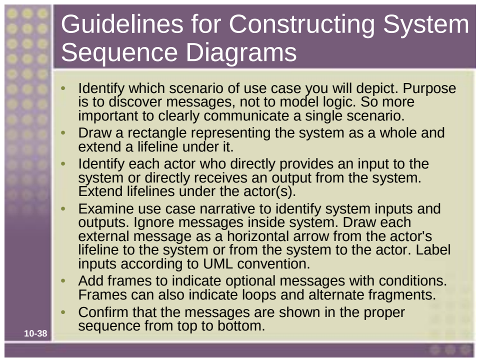
7. **Receiver Actor** – other actors or external systems that receive messages from the system can be included.
8. **Frame** – a box can enclose one or more messages to divide off a fragment of the sequence. These can show loops, alternate fragments, or optional (opt) steps. For an optional fragment the condition shown in square brackets indicates the conditions under which the steps will be performed.

User
MemberServicesSystem
Sys Admin
opt
[numTries <= 3]
submitLogin(username,password)
Invalid Login
opt
[numTries > 3]
submitLogin(username,password)
failedAttempt(datetime,ip)
7
8

10-37

No additional notes.

Slide 38



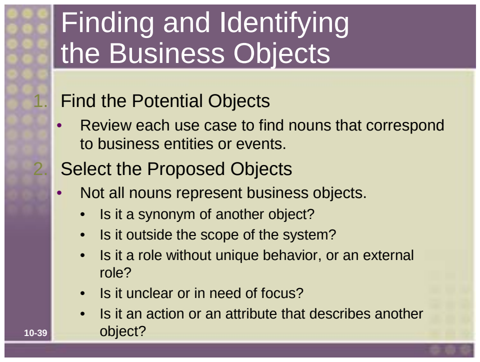
Guidelines for Constructing System Sequence Diagrams

- Identify which scenario of use case you will depict. Purpose is to discover messages, not to model logic. So more important to clearly communicate a single scenario.
- Draw a rectangle representing the system as a whole and extend a lifeline under it.
- Identify each actor who directly provides an input to the system or directly receives an output from the system. Extend lifelines under the actor(s).
- Examine use case narrative to identify system inputs and outputs. Ignore messages inside system. Draw each external message as a horizontal arrow from the actor's lifeline to the system or from the system to the actor. Label inputs according to UML convention.
- Add frames to indicate optional messages with conditions. Frames can also indicate loops and alternate fragments.
- Confirm that the messages are shown in the proper sequence from top to bottom.

10-38

Conversion Notes
This is a new slide for the seventh edition.

Slide 39



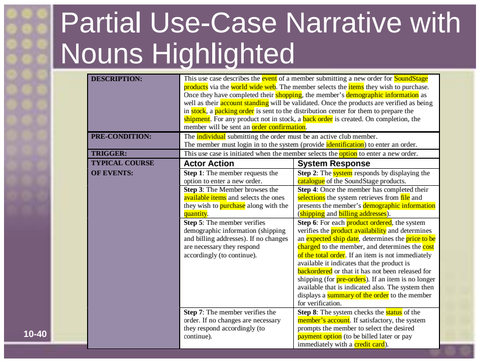
Finding and Identifying the Business Objects

1. Find the Potential Objects
   - Review each use case to find nouns that correspond to business entities or events.
2. Select the Proposed Objects
   - Not all nouns represent business objects.
     - Is it a synonym of another object?
     - Is it outside the scope of the system?
     - Is it a role without unique behavior, or an external role?
     - Is it unclear or in need of focus?
     - Is it an action or an attribute that describes another object?

10-39

Teaching Notes
Using use cases as a source for finding objects is a popular approach for object identification.

Slide 40



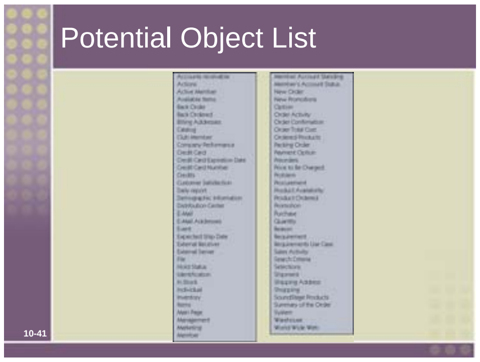
Partial Use-Case Narrative with Nouns Highlighted

DESCRIPTION:	This use case describes the event of a member submitting a new order for SoundStage products via the world wide web. The member selects the items they wish to purchase. Once they have completed their shopping, the member's demographic information as well as their account standing will be validated. Once the products are verified as being in stock, a packing order is sent to the distribution center for them to prepare the shipment. For any product not in stock, a back order is created. On completion, the member will be sent an order confirmation.	
PRE-CONDITION:	The individual submitting the order must be an active club member. The member must login in to the system (provide identification) to enter an order.	
TRIGGER:	This use case is initiated when the member selects the option to enter a new order.	
TYPICAL COURSE OF EVENTS:	**Actor Action**	**System Response**
	Step 1: The member requests the option to enter a new order.	**Step 2**: The system responds by displaying the catalogue of the SoundStage products.
	Step 3: The Member browses the available items and selects the ones they wish to purchase along with the quantity.	**Step 4**: Once the member has completed their selections the system retrieves from file and presents the member's demographic information (shipping and billing addresses).
	Step 5: The member verifies demographic information (shipping and billing addresses). If no changes are necessary they respond accordingly (to continue).	**Step 6**: For each product ordered, the system verifies the product availability and determines an expected ship date, determines the price to be charged to the member, and determines the cost of the total order. If an item is not immediately available it indicates that the product is backordered or that it has not been released for shipping (for pre-orders). If an item is no longer available that is indicated also. The system then displays a summary of the order to the member for verification.
	Step 7: The member verifies the order. If no changes are necessary they respond accordingly (to continue).	**Step 8**: The system checks the status of the member's account. If satisfactory, the system prompts the member to select the desired payment option (to be billed later or pay immediately with a credit card).

10-40

No additional notes.

Slide 41



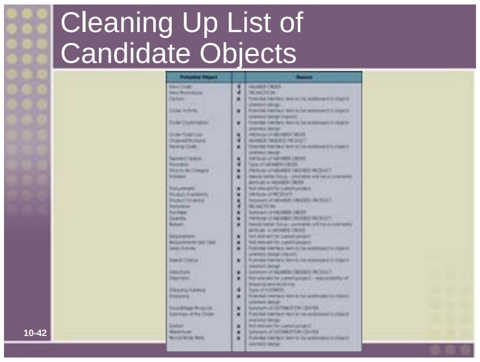
Potential Object List

Accounts Receivable	Member Account Standing
Actions	Member's Account Status
Active Member	New Order
Available Items	New Promotions
Back Order	Option
Back Ordered	Order Activity
Billing Addresses	Order Confirmation
Catalog	Order Total Cost
Club Member	Ordered Products
Company Performance	Packing Order
Credit Card	Payment Option
Credit Card Expiration Date	Preorders
Credit Card Number	Price to Be Charged
Credits	Problem
Customer Satisfaction	Procurement
Daily report	Product Availability
Demographic Information	Product Ordered
Distribution Center	Promotion
E-Mail	Purchase
E-Mail Addresses	Quantity
Event	Reason
Expected Ship Date	Requirement
External Receiver	Requirements Use Case
External Server	Sales Activity
File	Search Criteria
Hold Status	Selections
Identification	Shipment
In Stock	Shipping Address
Individual	Shopping
Inventory	SoundStage Products
Items	Summary of the Order
Main Page	System
Management	Warehouse
Marketing	World Wide Web
Member	

10-41

No additional notes.

Slide 42



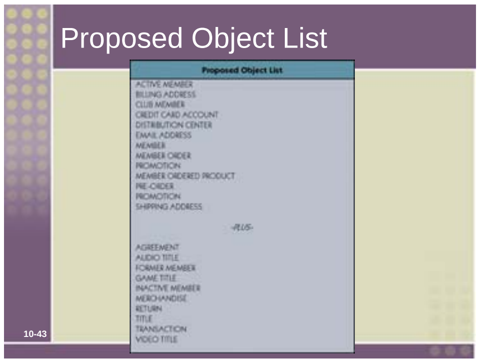
Cleaning Up List of Candidate Objects

Potential Object		Reason
New Order	✓	MEMBER ORDER
New Promotions	✓	PROMOTION
Option	✗	Potential interface item to be addressed in object-oriented design
Order Activity	✗	Potential interface item to be addressed in object-oriented design (report)
Order Confirmation	✗	Potential interface item to be addressed in object-oriented design
Order Total Cost	✗	Attribute of MEMBER ORDER
Ordered Products	✓	MEMBER ORDERED PRODUCT
Packing Order	✗	Potential interface item to be addressed in object-oriented design
Payment Option	✗	Attribute of MEMBER ORDER
Preorders	✓	Type of MEMBER ORDER
Price to Be Charged	✗	Attribute of MEMBER ORDERED PRODUCT
Problem	✗	Needs better focus – probably will be a comments attribute in MEMBER ORDER
Procurement	✗	Not relevant for current project
Product Availability	✗	Attribute of PRODUCT
Product Ordered	✗	Synonym of MEMBER ORDERED PRODUCT
Promotion	✓	PROMOTION
Purchase	✗	Synonym of MEMBER ORDER
Quantity	✗	Attribute of MEMBER ORDERED PRODUCT
Reason	✗	Needs better focus – probably will be a comments attribute in MEMBER ORDER
Requirement	✗	Not relevant for current project
Requirements Use Case	✗	Not relevant for current project
Sales Activity	✗	Potential interface item to be addressed in object-oriented design (report)
Search Criteria	✗	Potential interface item to be addressed in object-oriented design
Selections	✗	Synonym of MEMBER ORDERED PRODUCT
Shipment	✗	Not relevant for current project – responsibility of shipping and receiving
Shipping Address	✓	Type of ADDRESS
Shopping	✗	Potential interface item to be addressed in object-oriented design
SoundStage Products	✗	Synonym of DISTRIBUTION CENTER
Summary of the Order	✗	Potential interface item to be addressed in object-oriented design
System	✗	Not relevant for current project
Warehouse	✗	Synonym of DISTRIBUTION CENTER
World Wide Web	✗	Potential interface item to be addressed in object-oriented design

10-42

Teaching Notes
Additional objects were included that are part of the case study but were not identified in the use case. These additional objects are introduced here because they appear in later diagrams.

Slide 43



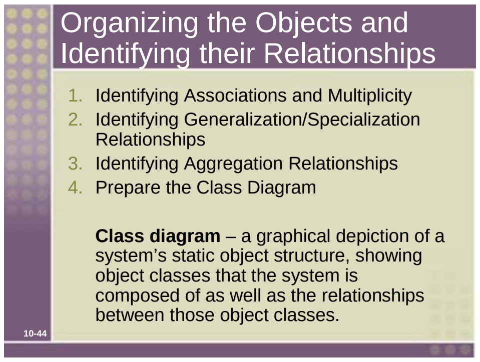
Proposed Object List

Proposed Object List
ACTIVE MEMBER
BILLING ADDRESS
CLUB MEMBER
CREDIT CARD ACCOUNT
DISTRIBUTION CENTER
EMAIL ADDRESS
MEMBER
MEMBER ORDER
PROMOTION
MEMBER ORDERED PRODUCT
PRE-ORDER
PROMOTION
SHIPPING ADDRESS
-PLUS-
AGREEMENT
AUDIO TITLE
FORMER MEMBER
GAME TITLE
INACTIVE MEMBER
MERCHANDISE
RETURN
TITLE
TRANSACTION
VIDEO TITLE

10-43

No additional notes.

Slide 44



Organizing the Objects and Identifying their Relationships

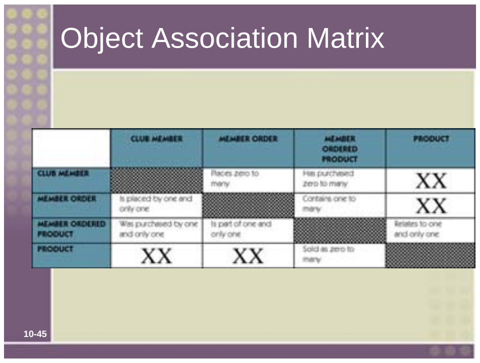
1. Identifying Associations and Multiplicity
2. Identifying Generalization/Specialization Relationships
3. Identifying Aggregation Relationships
4. Prepare the Class Diagram

Class diagram – a graphical depiction of a system's static object structure, showing object classes that the system is composed of as well as the relationships between those object classes.

10-44

Teaching Notes
It is very important that the analyst not only identify relationships that are obvious or recognized by the users. On way to help ensure that possible relationships are identified is to use a object/class matrix. This matrix lists the objects/class as column headings as well as row headings. The matrix can then be used as a check list to ensure that each object/class appearing on a row is checked against *each* object/class appearing in a column for possible relationships. The name of the relationship and the multiplicity can be recorded directly in the intersection cell of the matrix.
Generalization/Specialization relationships may be discovered by looking at the class diagram. Do any associations exist between two objects that have a one-to-one multiplicity? If so, can you say the sentence "object X *is a* object Y" and it be true? If it is true, you may have a generalization/specialization relationship. Also look for objects which have common attributes and behaviors. It may be possible to combine the common attributes and behaviors into a new super-object. Why do we want generalization/specialization relationships? It allows us to take advantage of inheritance which facilitates the reuse of objects and programming code.
Aggregation relationships are asymmetric, in that object B is part of Object A but, object A is not part of object B. Aggregation relationships do not imply inheritance, in that object B does not inherit behavior or attributes from object A. Aggregation relationships propagate behavior in that behavior applied to the whole is automatically applied to the parts.

Slide 45



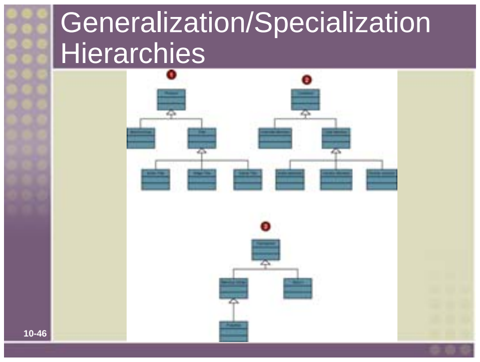
Object Association Matrix

	CLUB MEMBER	MEMBER ORDER	MEMBER ORDERED PRODUCT	PRODUCT
CLUB MEMBER	▨	Places zero to many	Has purchased zero to many	XX
MEMBER ORDER	Is placed by one and only one	▨	Contains one to many	XX
MEMBER ORDERED PRODUCT	Was purchased by one and only one	Is part of one and only one	▨	Relates to one and only one
PRODUCT	XX	XX	Sold as zero to many	▨

10-45

Teaching Notes
This tool can be used to record the possible association between any two objects.
To interpret the contents of the cells, start with the object on the left of the row, read the contents of the cell, and then finish with the object at the top of the column. For example:
A CLUB MEMBER places zero to many orders.
A CLUB MEMBER and PRODUCT have no association between them.

Slide 46

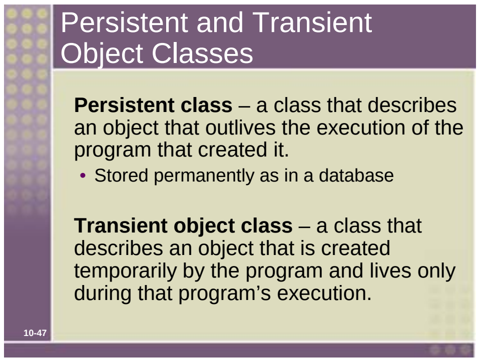


Generalization/Specialization Hierarchies

10-46

No additional notes.

Slide 47



Persistent and Transient Object Classes

Persistent class – a class that describes an object that outlives the execution of the program that created it.

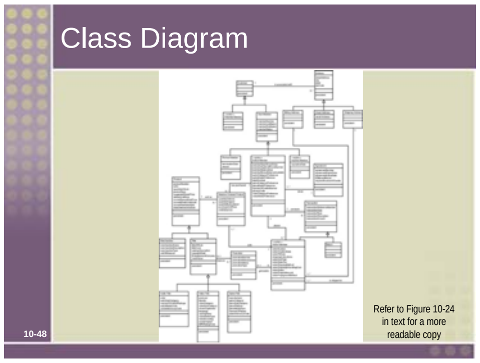
- Stored permanently as in a database

Transient object class – a class that describes an object that is created temporarily by the program and lives only during that program's execution.

10-47

No additional notes.

Slide 48



Class Diagram

Refer to Figure 10-24 in text for a more readable copy

10-48

No additional notes.

Answers to End of Chapter Questions and Exercises

Review Questions

1. The Unified Modeling Language (UML). It is not a method for developing system, but rather a descriptive and analytical set of tools that are used to perform object-oriented analysis and modeling

2. Something refers to the different types of objects in our environment, which may include a person, place, thing, or event. Some common examples are teacher, student, building, office, home, book, automobile, order, payment, and invoice.

   Data refers to the attributes of an object, such as employee name, employee number, classification, job title, SSN, etc.

   Behavior refers to the things that can be done by that object and only that object, such as the behavior of the object "Broom" would include "sweep." In an object-oriented approach, an object's behavior is also known as methods, operations, or services.

3. Encapsulation means that the attributes and behaviors of an object are part of the object. Only through the behavior of the object can its attributes be accessed or changed.

4. Class: Vehicle
   Objects: Automobile, pickup truck, minivan, motorcycle

4. The concept of inheritance means that the attributes and/or behaviors belonging to any one object class can be inherited or reused by another object class. Supertype is an object class – also known as a parent class – that has behaviors and/or attributes which are common to one or more class subtypes, also known as a child class. Subtype is an object class that inherits attributes and behaviors from the supertype class, but which may also contain other attributes and behaviors that are unique to the subtype itself.

5. In OOA and modeling, objects and classes represent entities that must interact with each other in order to sustain the business mission. By definition, each object and class has at least one business relationship with another object or class.

6. In UML, a connecting line between the two related classes is used. UML refers to this line as an association. A verb phrase is shown above the connecting line which describes the association, e.g., "a traffic officer **writes** a

ticket. The degree of association, or multiplicity, is also shown in UML graphical notation

8. A hollow diamond depicts a basic aggregation relationship, in which a larger "whole" class contains zero or more smaller "part" classes, such as the relationship of a baseball team to the individual players.

   A solid diamond depicts a composition aggregation relationship. This is a stronger type of aggregation relationship in which the "whole" class is responsible for the creation and destruction of its "parts," such as the relationship of the human body to its organs.

9. Polymorphism refers to when two different objects have a common behavior to perform, but carry out their behaviors differently. Polymorphism may be used when a specific behavior in the subtype needs to override the behavior in the supertype.

10.
    - Use-case Model Diagrams
    - Static Structure Diagrams
    - Interaction Diagrams
    - State Diagrams
    - Implementation Diagrams

11. Sequence diagrams show the sequence of interaction between different objects through message sending when a use case or an operation executes. .

    Collaboration diagrams illustrate the interaction between objects in a network format without taking into account sequence or timing of the interactions.

12. 1) Modeling the functions of the system
    2) Finding and identifying the business objects
    3) Organizing the objects and identifying their relationships

13. An activity diagram is used to portray the sequential flow of business process activities, the steps of a use case, or the logic of an object behavior.

14. A candidate object should be discarded if there is an affirmative answer to any of the following questions:
    - Is the candidate a synonym of another object?
    - Is the candidate outside the scope of the system?
    - Is the candidate a role without unique behavior, or is it an external role?
    - Is the candidate unclear and in need or focus?
    - Is the candidate an action or attribute that describes another object?

15. • Identify associations and multiplicity
    • Identify generalization/ specialization relationships
    • Identify aggregation relationships
    • Prepare the class diagram

Problems and Exercises

1. a. UML does provide a standard notation method for object modeling. UML does not provide a systems development methodology.

   b. The intent in developing UML was to replace the myriad of object oriented development methods with one single standard process.

   c. We would probably have an even greater of confusion of different object-oriented methodologies than in the early 1990's. This would tend to increase development costs and time, and reduce product portability and quality.

2. a. An object is something that a person can see, touch or sense, and which has data and behavior associated with it. Object-oriented analysis studies objects with the object of combining and using them to design and build an information system.

   b. Object modeling identifies the objects and the relationships between them in an information systems environment.

   c. Traditional development approaches focus on defining and using data and process models, while OOA is focused on using objects to define and build static structure and dynamic behavior models.
      Because traditional development concepts and OOA concepts are so different, many experienced designers have more problems learning OOA than students who have never had any experience with traditional design methods. This may be because the experienced designers must first "unlearn" the old way of viewing system design before they can grasp the new concepts.

3. a. Using the textbook terminology, a movie DVD is a 'thing' object.

   b. Some of the attributes for an object called "Movie DVD" would include:
      a. Title
      b. ISBN
      c. Studio
      d. Year Released

e. Description
f. Format

Note: These are only a few of the attributes for a movie DVD; there are many others!

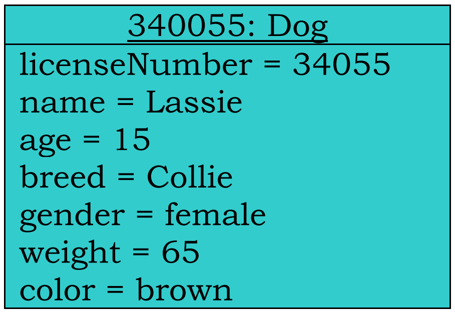
c. An example of an object instance for a movie DVD is the movie Fools Rush In, ISBN 0-7678-0421-X, Columbia TriStar Home Video, 1997.

d. The object class of Movie DVD could be considered a subtype of the object class of DVD, which is a supertype, since there are other classes of DVDs which contain attributes and behaviors unique to them.

4. a. A dog would be considered a “thing” object.

b. Attributes might include:
   a. License #
   b. Name
   c. Age
   d. Breed
   e. Gender
   f. Weight
   g. Color

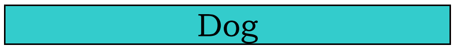
c. Example of a “Dog” object instance:



340055: Dog
licenseNumber = 34055
name = Lassie
age = 15
breed = Collie
gender = female
weight = 65
color = brown

d. Behaviors could include awake or asleep; walking, running, jumping, sitting or lying down; barking, silent, howling or crying.

e.



Dog

-licenseNumber -name -age -breed -gender -weight -color

+awake()
+asleep()
+running()
+walking()
+jumping()
+sitting()
+lying down()
+barking()
+crying()
+silent()
+howling()

5. a. Some examples include:
    A dog is owned by zero or more persons
    A person owns zero or more dogs
    A dog is walked by one and only one person
    A person walks zero or more dogs.
    A dog is petted by zero or more persons
    A person pets zero, one or two dogs

6. a. "Dog" will send "pet me" message by wagging tail and looking at "Person." Return behavior of "Person" will be to pet "Dog."

    b. The object sending the message doesn't need to know the internal organization or the processes that the receiving object goes through to accomplish the request. The only that matters to the sending object is knowing that the request will be responded to in a specific fashion.

    c. As association must exist between the two objects in order for the message to

7. a. Polymorphism means that the same request made of different objects doesn't necessarily result in the same responses; different objects can respond different ways.

    b. Like message sending, polymorphism. The object making the request knows which behavior to request.

c. Overriding behaviors refers to situations where a subclass uses its own attributes or behaviors rather than those inherited from the class (super-type).

8. a. The different groups of UML diagrams and what they depict and/or model are:
    1. Use Case Model Diagrams show the system, its users and their interactions.
    2. Static Structure Diagrams model the system’s static structure by showing its object classes, the relationships between them, and by modeling actual object instances in order to provide developers with a “snapshot” at a specific point in time.
    3. Interaction Diagrams model the system’s dynamic behavior by showing interactions, relationships and messages for a set of objects.
    4. State Diagrams also model the dynamic behavior of the system by depicting its objects in each of their various states, and by depicting the sequential flow of activities.
    5. Implementation Diagrams, which model the structure of the information system by depicting its software components’ organization and dependencies, and its physical architecture.

b. The purpose of use case modeling is to identify the functionality required of the information system.

c. The three major tasks in conducting object-oriented analysis are
    1. Modeling system functions
    2. Finding and identifying the business objects
    3. Organizing the objects and identifying their relationships

d. Business requirements use-case models are refined and changed into analysis use-case models in order to provide the level of detail that will be needed to actually design and build the system. This refinement process is generally a five-step process:
    1. The first step is to identify, define and document any actors who are new or who were not identified when the business requirements use-case model was first developed during the analysis phase.
    2. Next is identifying, defining and documenting any new use cases, since new interactions are created whenever there is a new actor.
    3. Third is looking for any reuse possibilities, where two or more use cases have the same business goals and have a number of steps in common, which can then be separated and placed in abstract use case.
    4. Update the use-case model diagram if necessary because of new actors and/or use case diagrams.

5. Document the system analysis use-case narratives, which provide more detail than the business analysis use-case narratives, but which do not provide implementation details except at a high level.

9. a. Different types of narratives are used because, unlike regular use case narratives, abstract and extension use cases are not invoked by actors. Rather, they are invoked by other use cases.

   b. Abstract and extension use cases do not include as many data fields as regular use cases and as a result, they are usually much shorter.

   c. No, by definition, an abstract use case is invoked by two or more use cases.

   d. Yes, an extension use case is reusable because it can invoke other abstractor and/or extension use cases.

   e. Yes, and only by a single use case.

   f. No, but it can invoke other abstract and/or extension use cases.

10. a. UML activity diagrams are different from flowcharts because unlike flowcharts, they have the capability to show parallel activities that occur concurrently. This means an activity diagram can show both the actions and the results of those actions.

    b. UML activity diagrams and flowcharts are similar in that they both show activities that occur in sequential order.
    It is a synchronization bar to show parallel activities. It is used both at the beginning and at the end of the parallel activities; these parallel activities can occur in any order, but the process can't terminate until all parallel activities are completed.

11. a. There are just too many nouns in the typical requirements documentation for this to be practical method!

    b. Use case modeling simplifies identifying potential objects because the entire scope of the system is represented in the use cases that are created. This dramatically reduces the amount of searching required.

    c. Some candidate objects, upon examination, may be redundant, may not be useful as a business object, may be outside the project domain, or may not even be an object.

    d. Candidate objects should be eliminated if they meet any of the following criteria:

a. The same as another object
b. Outside the scope of the system
c. Behavior is not unique
d. Role is an external one
e. Ambiguous and unfocused
f. Not an object, but an action or an attribute of another object

Candidate objects that were determined to actually be attributes of another object should be documented, because they will be used at a later point in the class diagram

12. a. Static

b. Because multiplicity is association-specific and cannot be defined until the association is identified.

c. An object/class can help to identify associations that may not be obvious or which have not yet been recognized, by ensuring that each possible combination is checked for an association.

d. If you have 72 objects and classes, there will be 72 null cells, since an object or class can't have an association with itself.

13. a. The remaining steps are:
    1. Identifying the generalization/specialization relationships
    2. Identifying the aggregation relationships
    3. Creating the class diagram

b. Because they show inheritance, which promotes code and object reusability.

c. Find associations between two classes that have a one-to-one multiplicity, and find classes that have common attributes and behaviors, which may allow the creation of a supertype class.

d. The subtype class in a generalization/specialization relationship inherits the supertype class's attributes and behaviors, but also has its own unique attributes and behaviors. Aggregation relationships are asymmetric and do not imply relationship; further, behavior of the whole is also behavior of each of the parts.

e. Yes, it is possible, but it would be highly unusual, since the inherent nature of business domain classes is persistent.

Project and Research

1. a. The newest version is UML 2.0, which as of mid-2005, was in the process of being released.
   b. The Object Management Group is a not-for-profit computer industry specifications consortium whose membership consists of most of the large computer companies in this country.
   c. Response can be open-ended, but should be thoughtful, logical and should reference some of the available articles.
   d. Same as for Question 1c, and should reference the specific specification(s).
   e. Same as for Question 1c
   f. Students should find this is a competitive and proprietary field. For example, as of mid-2005, IBM is planning to continue supporting the UML, but it also plans to release a "domain-specific modeling language" as part of Visual Studio.net in the near future.

2. Responses are open-ended, but should indicate that student adequately explored the subject with the designer, and provided a thoughtful, logical, and cohesive response.

3. Responses are open-ended as to requirements, so there will be a wide latitude of objects, classes, attributes, relationships and associations that are identified and described. The tables and diagrams do not need to be all-inclusive nor exhaustively detailed, but they should not be cursory either. The associations and relationships should be accurately diagrammed and consistent with the methods described in the textbook, as well as with the sample templates, in order to demonstrate that the student understands and can apply basic UML tools and techniques.

4. As with the preceding question, a wide latitude of responses should be expected. The use-case narratives should be well thought out and viable. The activity diagrams should be accurate and reasonably comprehensive in their use of UML notations

5. What is important in this exercise are not the results per se, but that students understand and are able to go through the process of finding and selecting objects, and identifying the associations and multiplicity that exist. Also, most if not all students should find objects and classes that they had not previously identified.

6. This may be a difficult exercise to do completely, since the preceding exercises did not require all objects and classes in the system to be identified and analyzed. Nonetheless, based upon the information the students do have, they should be able to create a reasonably students should be able to

piece together enough information to depict many of the generalization/specialization hierarchies and classes, and their relationships.

Mini-Cases

1. Note to professor: problems 1 and 2 are meant to be combined into a full paper. Problem #1 is a scope and needs background, which is meant to be the introduction (of sorts). Students have a tendency to get skimpy with this. However, this introduction and background sets the stage and the understanding for later work, so it is imperative that they are complete in their work. Have them focus the paper to the government department's head (manager).

2. Note to professor: This is the second part, the bulk of the paper, started in problem #1. After the students have completed the work, have them join the two into a full paper, bind it, and present it to their government liaison.

3. Note to professor: It cannot be stressed enough that clarity and completeness are essential. Students many times think that their work is "obvious" and that it is "easy" to understand. Help them with a reality check by having another student team code the Use Cases that they submit. The other team may not confer, and may only look at the diagrams and descriptions presented to them.

4. Note to professor: Look for interesting use of technology, appropriate attire, and good presentation capabilities (such as speaking clearly and facing the class).

Team and Individual Exercises

There are no answers to this section.