

**Module 4:
Object-Oriented Design
for Visual Basic .NET**

Overview

- Designing Classes
- Object-Oriented Programming Concepts
- Advanced Object-Oriented Programming Concepts
- Using Microsoft Visio

◆ Designing Classes

- Use Case Diagrams
- Use Case Diagram Example
- Use Case Descriptions
- Extending Use Cases
- Converting Use Cases into Classes

Use Case Diagrams

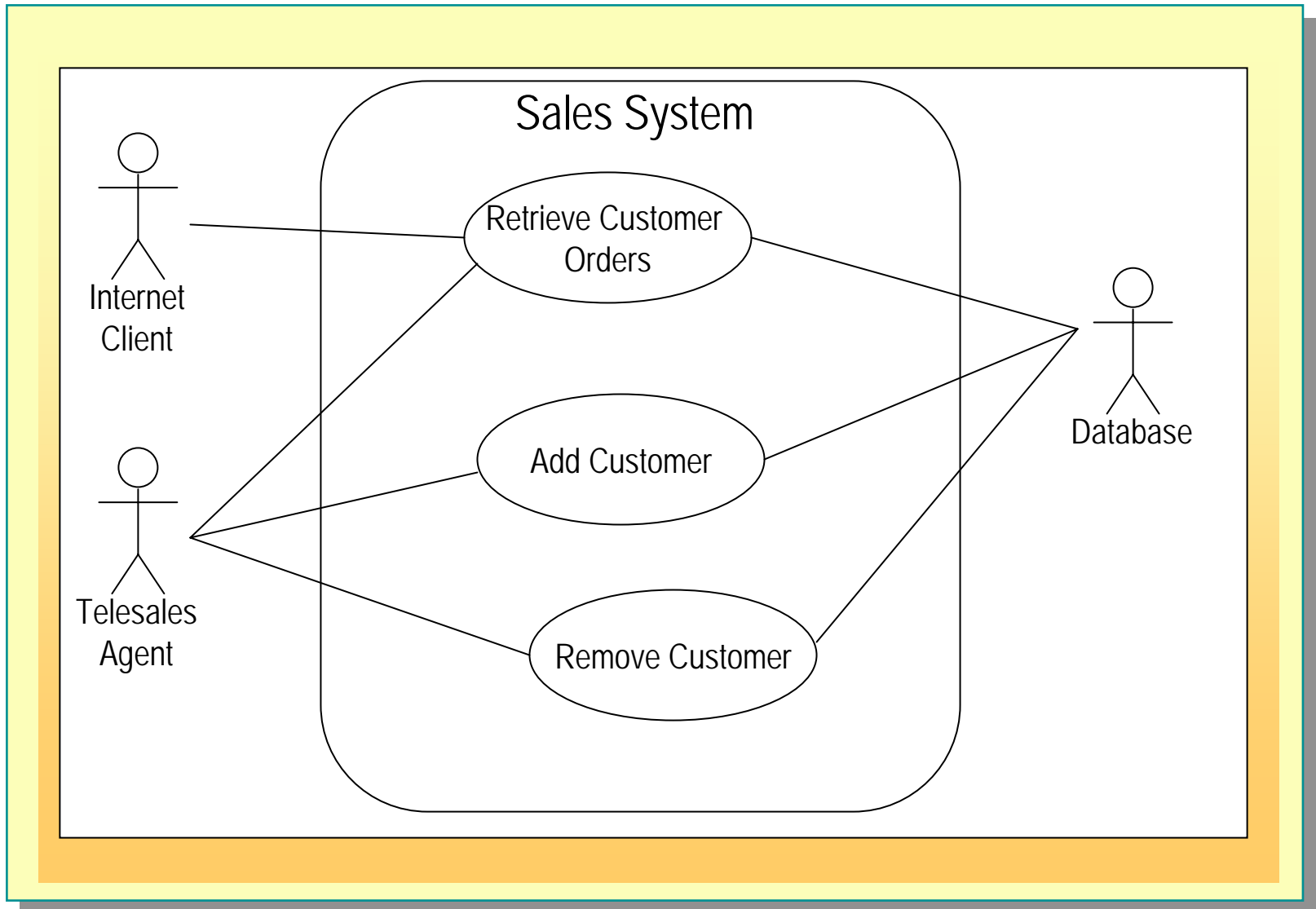
■ Use cases

- Provide a functional description of major processes
- Use a non-technical language to describe the process
- Show a boundary around the problem to be solved

■ Actors

- Graphically describe who or what will use the processes

Use Case Diagram Example



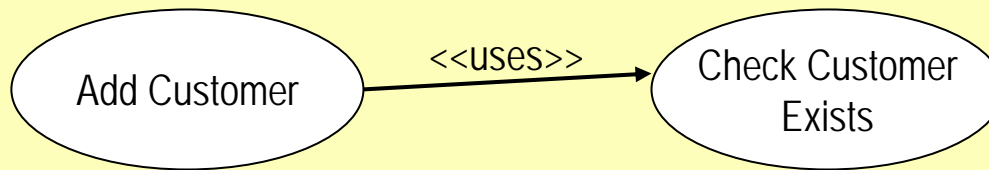
Use Case Descriptions

- “Retrieve Customer Orders” use case description

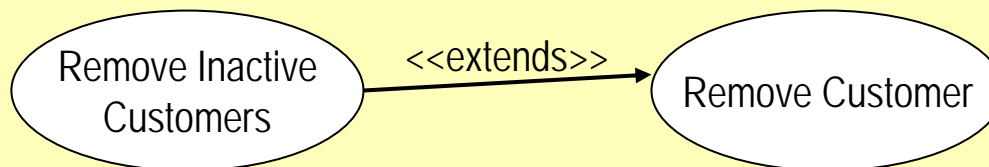
A user requests the orders for a customer by using a particular customer ID. The ID is validated by the database, and an error message is displayed if the customer does not exist. If the ID matches a customer, the customer’s name, address, and date of birth are retrieved, in addition to any outstanding orders for the customer. Details about each order are retrieved, including the ID, the date, and the individual order items that make up an order.

Extending Use Cases

- uses - reuses an existing use case



- extends - enhances an existing use case



Converting Use Cases into Classes

- Use case descriptions provide the basis for initial class design

- Nouns = classes or attributes

A **user** requests the **orders** for a **customer** by using a particular **customer ID**. The ID is validated by the database, and an error message is displayed if the customer does not exist. If the ID matches a customer, the customer's **name**, **address**, and **date of birth** are retrieved, in addition to any outstanding orders for the customer. Details about each **order** are retrieved, including the **ID**, the **date**, and the individual **order items** that make up an order.

- Verbs = operations (methods)

Example: ValidateCustomer, RetrieveOrders,
RetrieveOrderItems

Practice: Deriving Classes from Use Cases



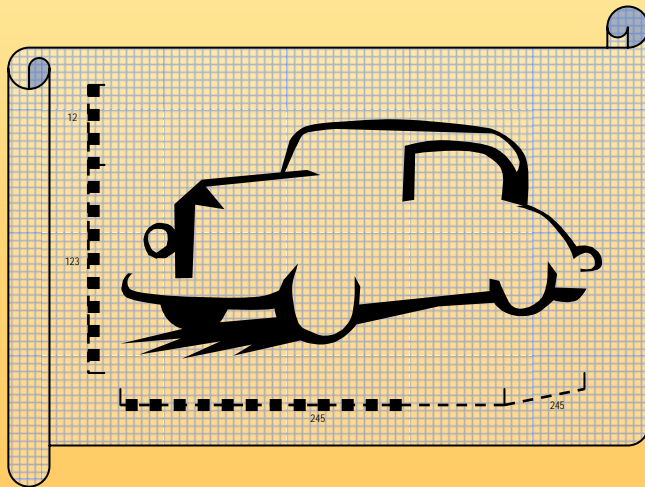
◆ Object-Oriented Programming Concepts

- Comparing Classes to Objects
- Encapsulation
- Abstraction
- Association
- Aggregation
- Attributes and Operations

Comparing Classes to Objects

Class

- A class is a template or blueprint that defines an object's attributes and operations and that is created at design time



Object

- An object is a running instance of a class that consumes memory and has a finite lifespan



Encapsulation

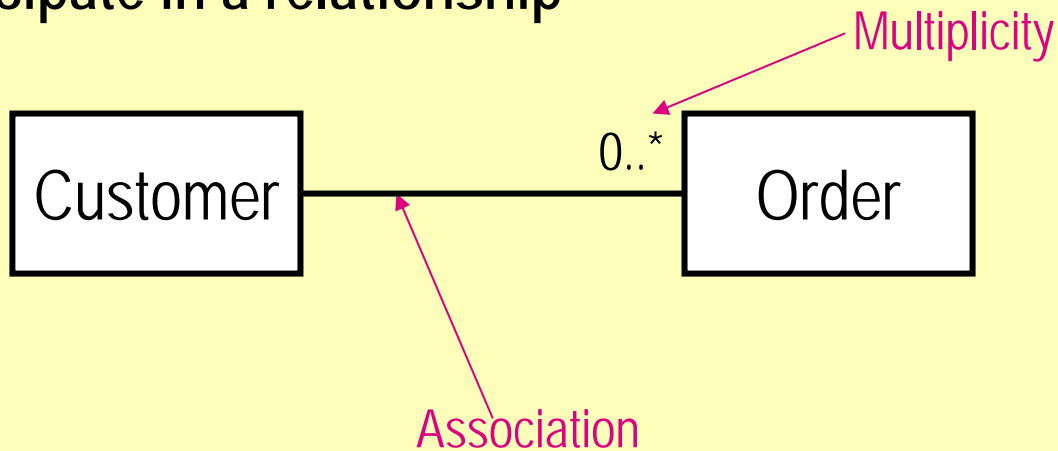
- How an object performs its duties is hidden from the outside world, simplifying client development
 - Clients can call a method of an object without understanding the inner workings or complexity
 - Any changes made to the inner workings are hidden from clients

Abstraction

- **Abstraction is selective ignorance**
 - Decide what is important and what is not
 - Focus on and depend on what is important
 - Ignore and do not depend on what is unimportant
 - Use encapsulation to enforce an abstraction

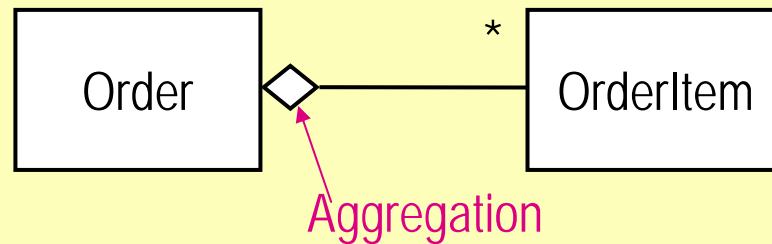
Association

- A class depends on another class to perform some functionality
- *Roles* are the direction of the association
- *Multiplicity* determines how many objects can participate in a relationship



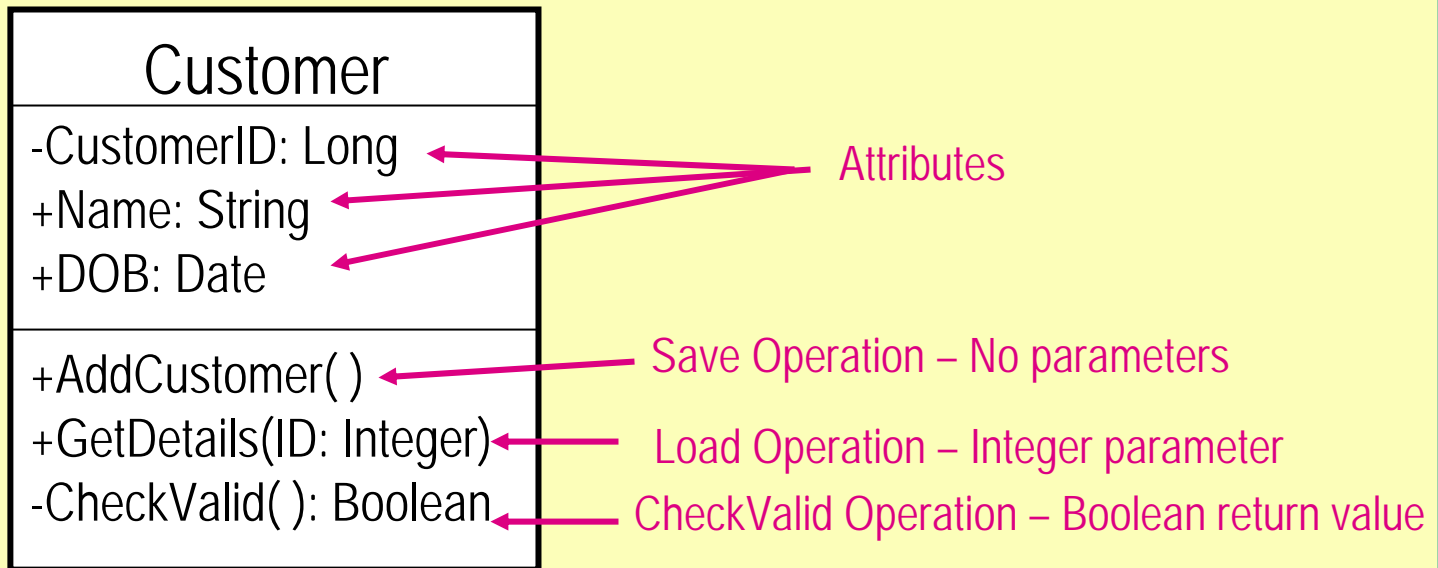
Aggregation

- A complex class containing other classes
- A “part-of” relationship
- Example:
 - An Order class contains an OrderItem class
 - An OrderItem class is a “part of” an Order class



Attributes and Operations

- Attributes are the data contained in a class
- Operations are the actions performed on that data
- Accessibility: Public (+), Private (-), Protected (#)

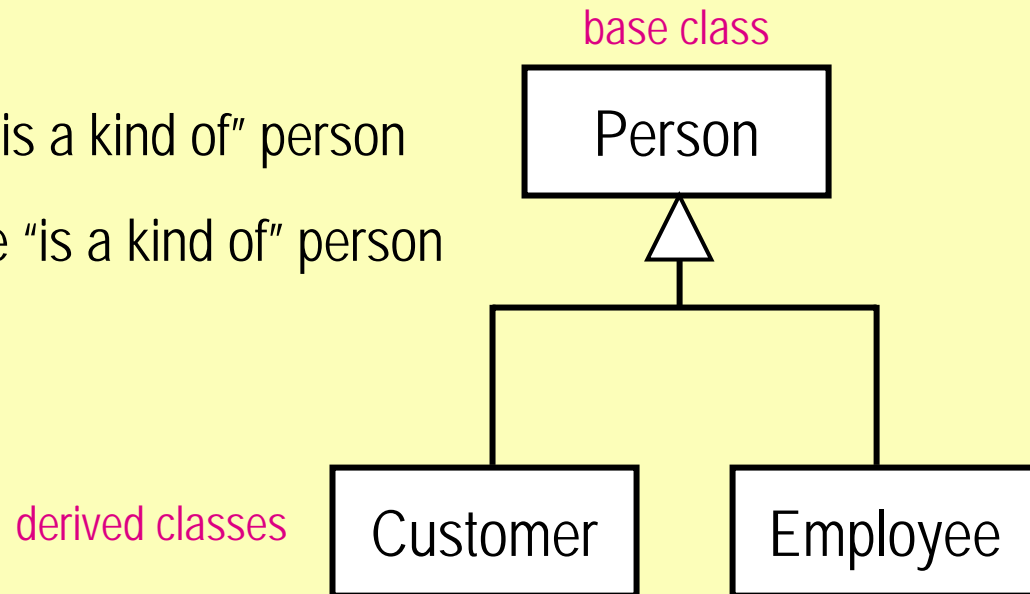


◆ Advanced Object-Oriented Programming Concepts

- Inheritance
- Interfaces
- Polymorphism

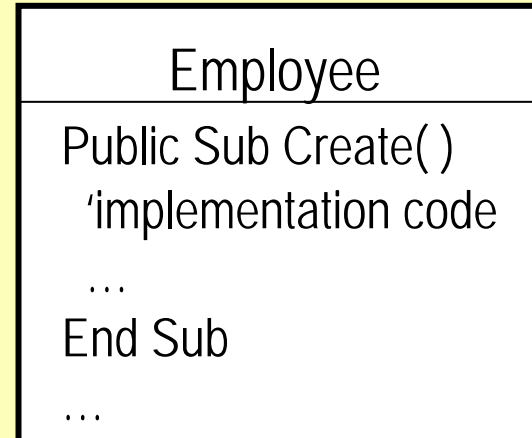
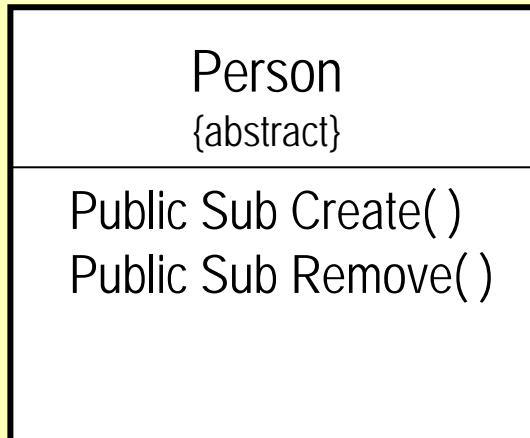
Inheritance

- Inheritance specifies an “is-a-kind-of” relationship
- Multiple classes share the same attributes and operations, allowing efficient code reuse
- Examples:
 - A customer “is a kind of” person
 - An employee “is a kind of” person



Interfaces

- Interfaces only define the method signatures
- Classes define the implementation of the code for the Interface methods
- *Interface inheritance* means only the Interface is inherited, not the implementation code



Polymorphism

- The same operation behaves differently when applied to objects based on different classes
- Often based on Interface inheritance
 - Classes inherit from interface base class
 - Each derived class implements its own version of code
 - Clients can treat all objects as if they are instances of the base class, without knowledge of the derived classes

◆ Using Microsoft Visio

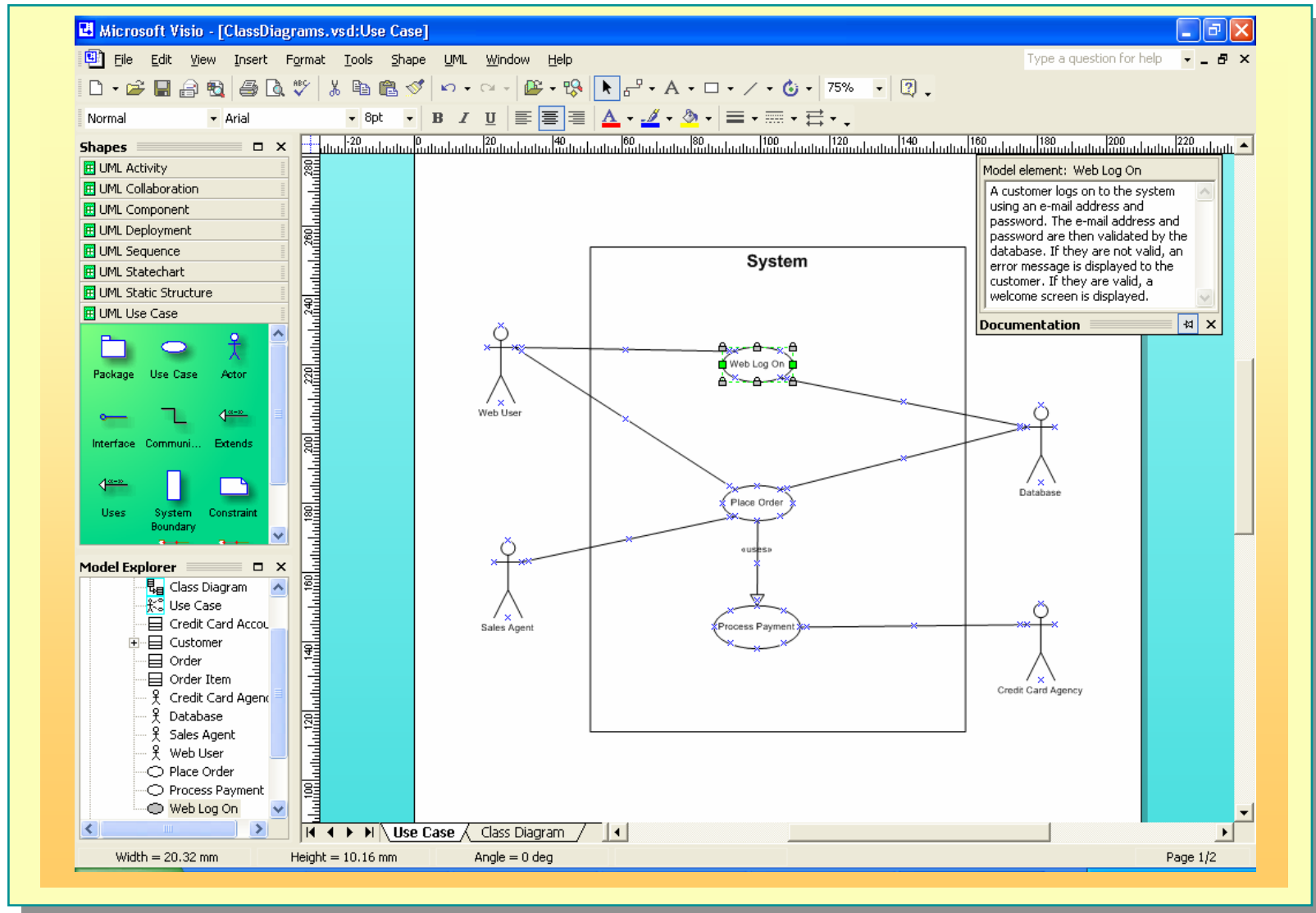
- Visio Overview
- Use Case Diagrams
- Class Diagrams
- Creating Class Diagrams

Visio Overview

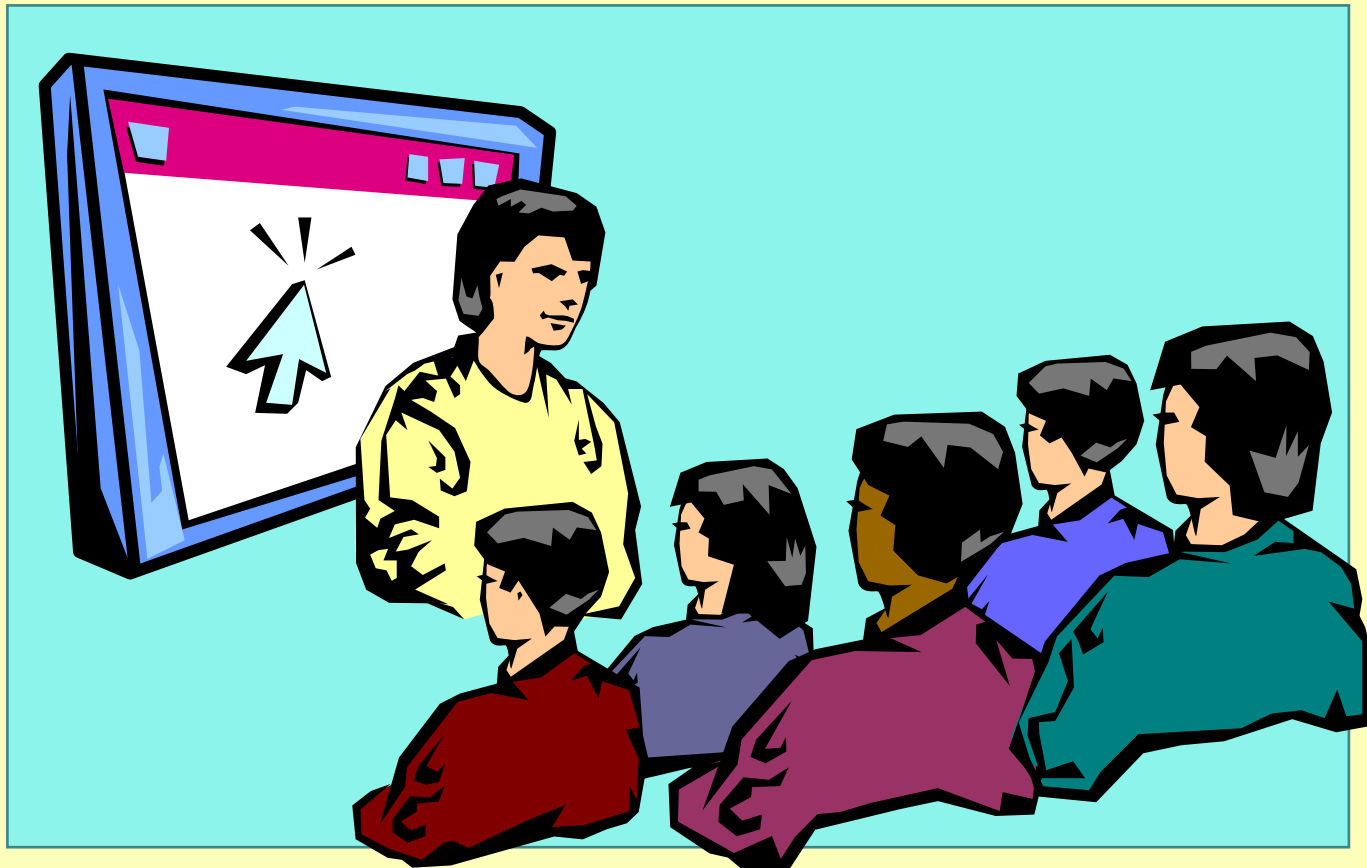
■ Supports:

- Use case diagrams
- Class or static structure diagrams
- Activity diagrams
- Component diagrams
- Deployment diagrams
- Freeform modeling

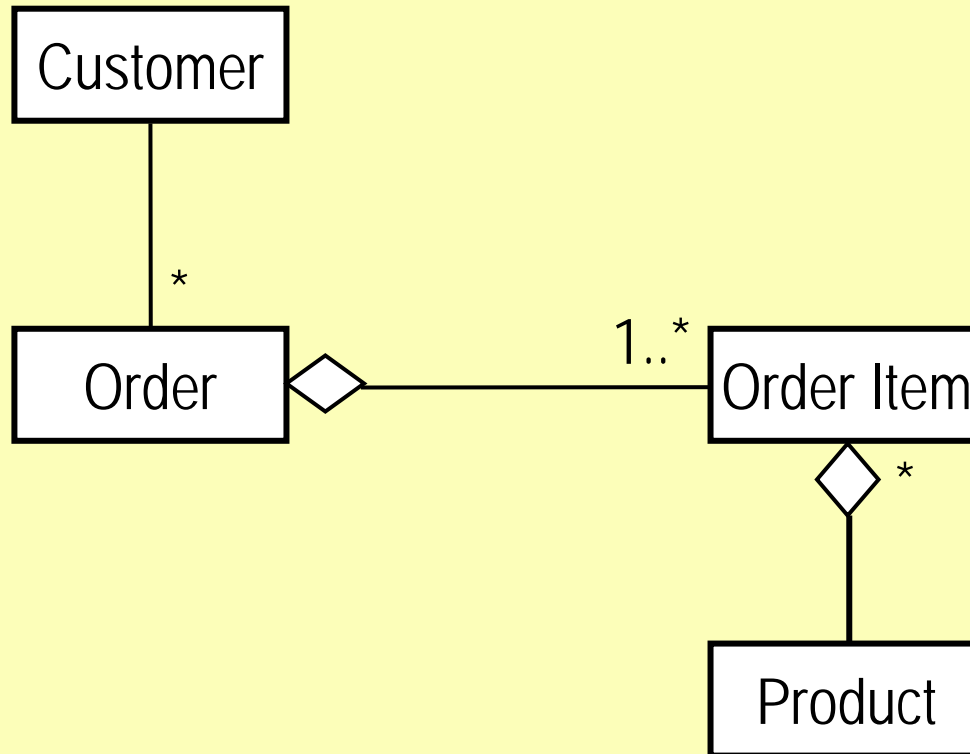
Use Case Diagrams



Demonstration: Creating Use Case Diagrams



Class Diagrams



Creating Class Diagrams

The screenshot displays the Microsoft Visio interface for creating a UML Class Diagram. The main workspace shows a class diagram with the following classes and relationships:

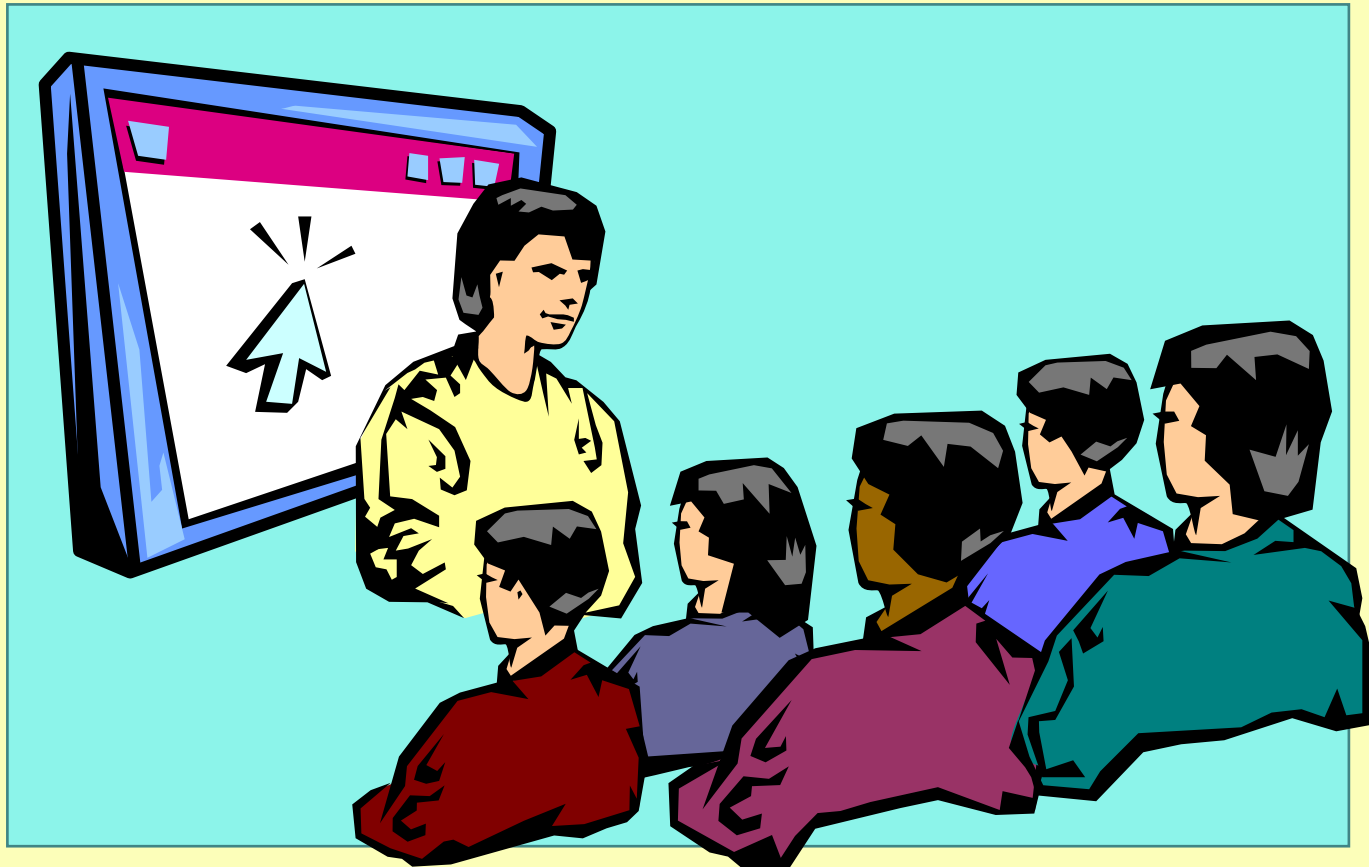
- Customer**: Attributes include CustomerID (Integer), Address (String), FullName (String), Email (String), Password (String), and LogOn (in Email: String, in Password: String) (Boolean). It has a 1-to-0..* association with Credit Card Account.
- Credit Card Account**: Attributes include ExpiryDate (Date), BillingAddress (String), CardNumber (String), and CustomerName (String). It has a 1-to-0..* association with Customer.
- Order**: Attributes include OrderID (Integer) and PlaceOrder() (operation). It has a 1-to-1..* association with Order Item.
- Order Item**: Attributes include ProductID (Integer) and Quantity (Integer), and ConfirmItem() (operation). It has a 1-to-0..* association with Product.
- Product**: Attributes include Price (Double), ProductID (Integer), Category (String), Description (String), Image (String), Manufacturer (String), ProductName (String), and RetrieveDetails() (operation).

The **UML Class Properties** dialog box is open for the **Customer** class. It shows the following details:

- Categories:** Class
- Name:** Customer
- Full path:** UML System 1::Static Model::Top Package::Customer
- Stereotype:** (empty)
- Visibility:** public
- Documentation:** The Customer class stores information about a particular customer and allows logging on to the system.

The **Model Explorer** on the left shows the project structure, including the **Customer** class and its associated attributes and operations.

Demonstration: Creating Class Diagrams



Lab 4.1: Creating Class Diagrams from Use Cases



Review

- Designing Classes
- Object-Oriented Programming Concepts
- Advanced Object-Oriented Programming Concepts
- Using Microsoft Visio