[CS3704] Software Engineering

Dr. Chris Brown

Virginia Tech

10/18/2023

Announcements

- **HW3 due Friday at 11:59pm**

High-Level Design II

Package Diagrams
Database Design
Intro to Design Patterns

Learning Outcomes

By the end of the course, students should be able to:

- **Understand software engineering processes, methods, and tools used in the software development life cycle (SDLC)**
- **Use techniques and processes to create and analyze requirements for an application**
- **Use techniques and processes to design a software system**
- **Identify processes, methods, and tools related to phases of the SDLC**
- **Explain the differences between software engineering processes**
- **Discuss research questions and current topics related to software engineering**
- **Create and communicate about the requirements and design of a software application**

Warm-Up

TODO: Complete a stand-up meeting!

- **What I did.**
- **What I need to do next.**
- **What is blocking me.**

**** Share about progress since last standup meeting, standing is optional.***

Design

Goal: decide the structure of the software and the hardware configurations that support it.

  - The *how* of the project

- How individual classes and software components work together in the software system.
  - Programs can have 1000s of classes/methods
- *Software Artifacts:* design documents, class diagrams (i.e. UML)

Design Engineering

- The process of making decisions about HOW to implement software solutions to meet requirements.
- Encompasses the set of concepts, principles, and practices that lead to the development of high-quality systems.

High-Level Design

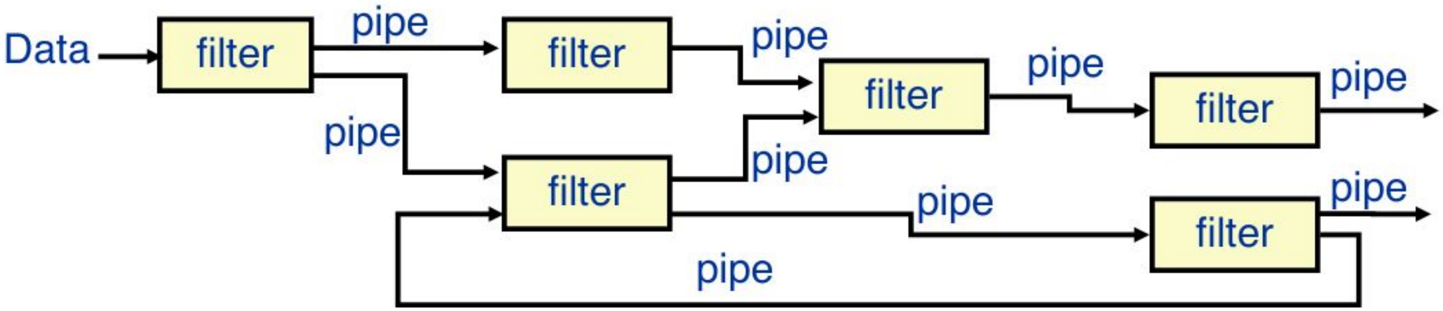
- Explains the architecture used to develop a system.
    - Also known as architectural design…
    - But there is debate on which term is most appropriate
- Provides a technical representation of functional (and some non-functional) requirements and the flow of information across assets or components in the system.

Architecture Patterns

- Common program structures:
  1. Pipe and Filter
  2. Event-based
  3. Layered

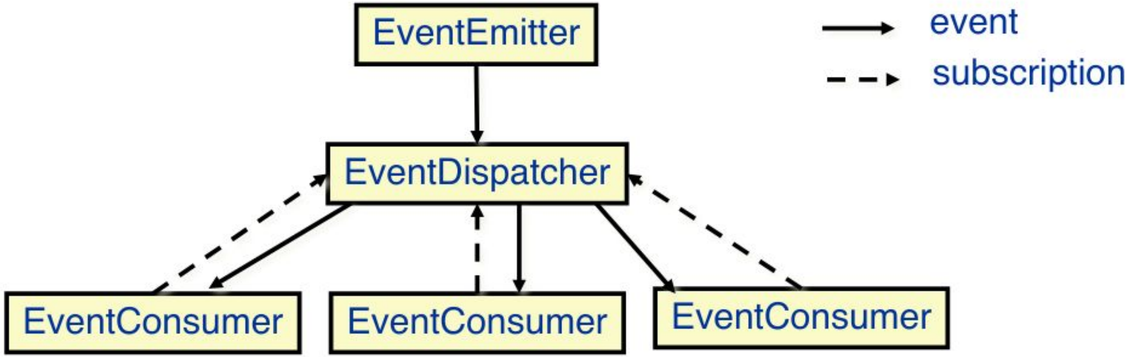
Pipe and Filter

- A pipeline contains a chain of data processing elements
  - –The output of each element is the input of the next element (usually with some buffering in between)



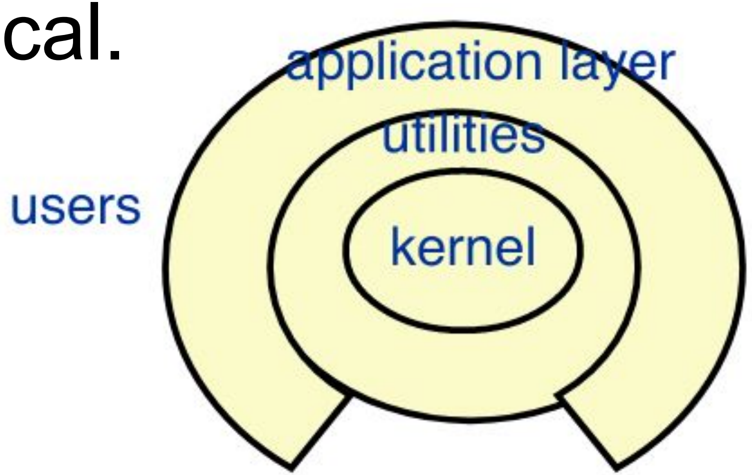
Event-Based Architecture

- Promotes the production, detection, consumption of, and reaction to events
- *Event-driven programming*



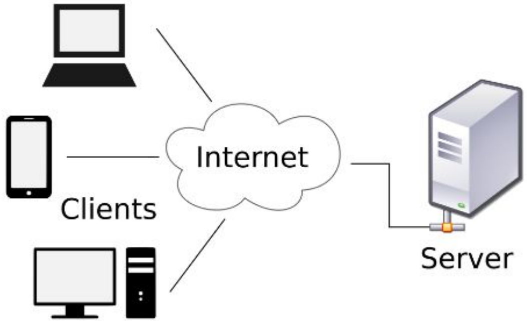
Layered/Tiered Architecture

- Multiple layers are defined to allocate responsibilities of a software product
- The communication between layers is hierarchical.



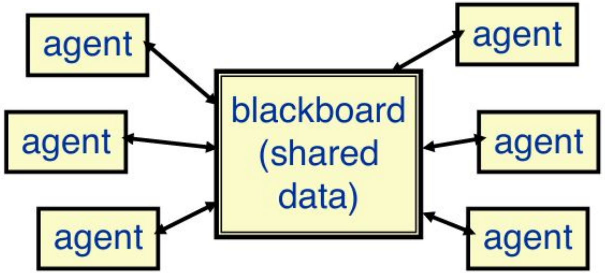
Client-Server Architecture

- Partition tasks or workloads between the providers and consumers of service or data (multiple hardware)
- Same system, different hardware, network communication



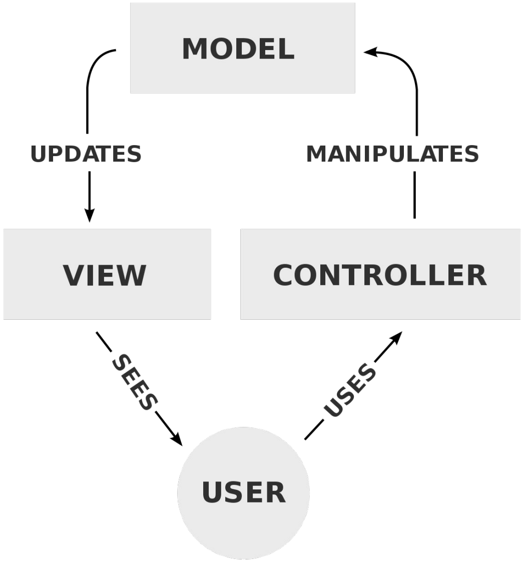
Data-Centric Architecture

- A data store resides at the center to be accessed frequently by agents
- Blackboard sends notification to subscribers when data of interest changes



Model-View-Controller Architecture

- Model-View-Controller
  - Includes UI (view) to interact with users
  - Store and retrieve information as needed



How to Do Architecture Design?

When decomposing a system into subsystems, take into consideration:

- how subsystems share data
  – data-centric or data-distributed
- how control flows between subsystems
  – as scheduled or event-driven
- how they interact with each other
  – via data or via method calls

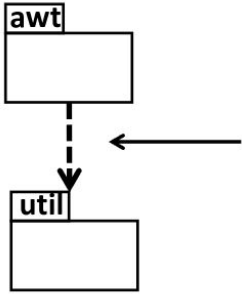
Architecture Modeling

- To organize architectural elements and diagrams into groups

UML Package Diagrams

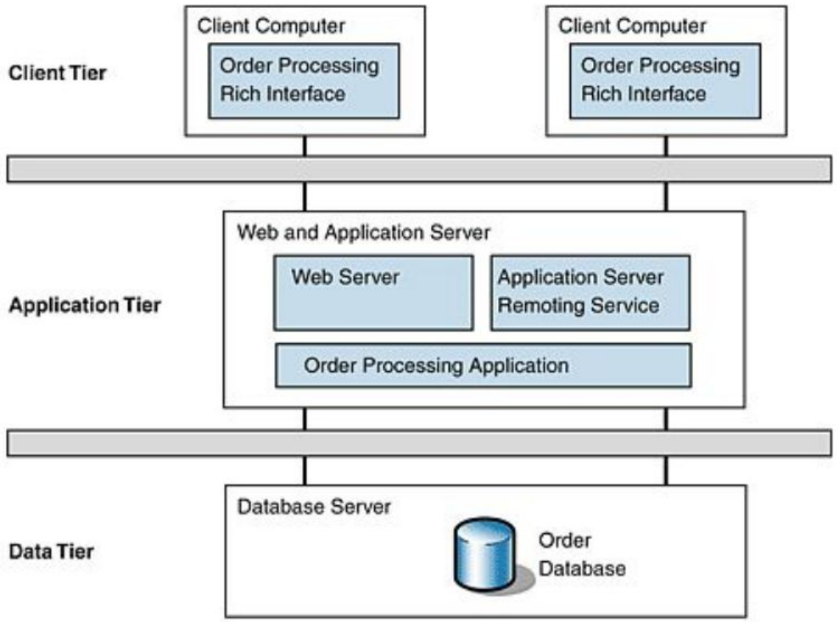
- To show packages and dependencies between the packages
- Can illustrate layered architecture
  – A layer, such as UI layer, can be modeled as a package named UI
  – Depicts relations between packages that make up a model

Example with JDK Packages

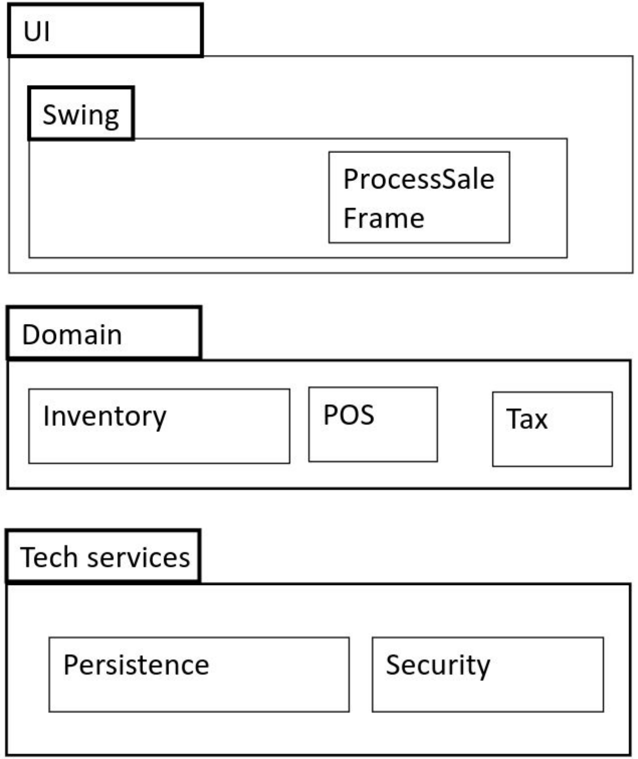


Case Study: Ordering System

- 3-layer architecture
  - User Interface
  - Application logic
    - Software objects representing domain-specific concepts (i.e. Sale)
  - Technical Services
    - General-purpose objects and subsystems that provide supporting services, such as interfacing with database or error logging
    - Usually application-independent and reusable across systems
    - Does ***not*** include the modeling of data!


Client Tier
Client Computer
Order Processing Rich Interface
Client Computer
Order Processing Rich Interface
Application Tier
Web and Application Server
Web Server
Application Server Remoting Service
Order Processing Application
Data Tier
Database Server
Order Database

Case Study: Ordering System (cont.)


UI

Swing

ProcessSale Frame

Domain

Inventory

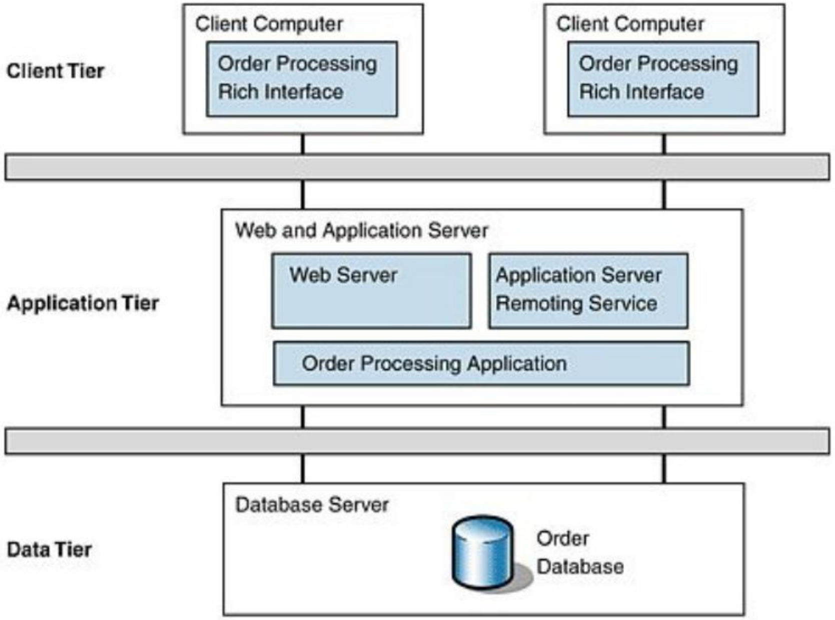
POS

Tax

Tech services

Persistence

Security


Client Tier

Client Computer

Order Processing Rich Interface

Client Computer

Order Processing Rich Interface

Application Tier

Web and Application Server

Web Server

Application Server Remoting Service

Order Processing Application

Data Tier

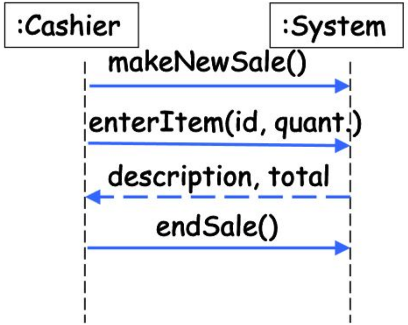
Database Server

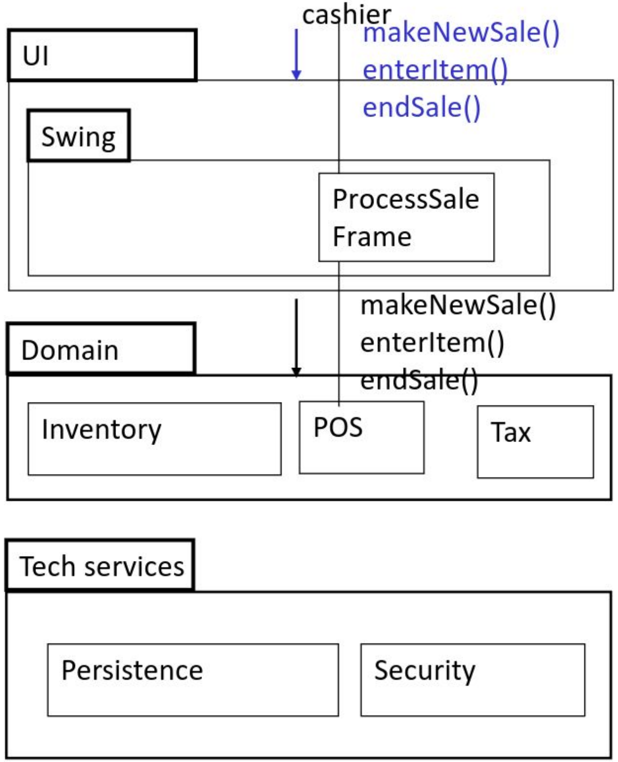
Order Database

What is the relationship with other diagrams?

Case Study: Ordering System (cont.)

Example: Messages illustrated in system sequence diagrams can correspond to messages sent from the UI layer to the domain layer.





Reminder: How is UML Really Used?

“UML has been described by some as ‘the lingua franca of software engineering’. Evidence from industry does not necessarily support such endorsements. How exactly is UML being used in industry – if it is? This paper presents a corpus of interviews with 50 professional software engineers in 50 companies and identifies 5 patterns of UML use.” [Petre]

NONE!	**70%**
SELECTIVE	22%
AUTOMATIC CODE GEN	6%
RETROFIT	2%
WHOLE	0%

Of those that reported using it…

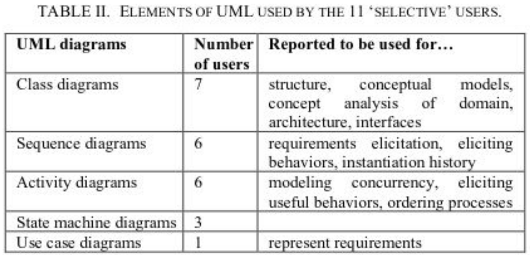


TABLE II. ELEMENTS OF UML USED BY THE 11 ‘SELECTIVE’ USERS.

UML diagrams	**Number of users**	**Reported to be used for…**
Class diagrams	7	structure, conceptual models, concept analysis of domain, architecture, interfaces
Sequence diagrams	6	requirements elicitation, eliciting behaviors, instantiation history
Activity diagrams	6	modeling concurrency, eliciting useful behaviors, ordering processes
State machine diagrams	3	
Use case diagrams	1	represent requirements

Database Design

Modern software is collecting and processing increasing amounts of data (data-centric).

- What is a database?
  - A system that stores data, and lets you create, read, update, and delete the data
    - Ex) files, spreadsheets, XML, relational, noSQL,...
- Why use databases?
  - Every non-trivial application uses databases to keep program states and to store manipulate, and retrieve data
  - Databases plays a critical role in applications

    – Corrupted data => execution failure

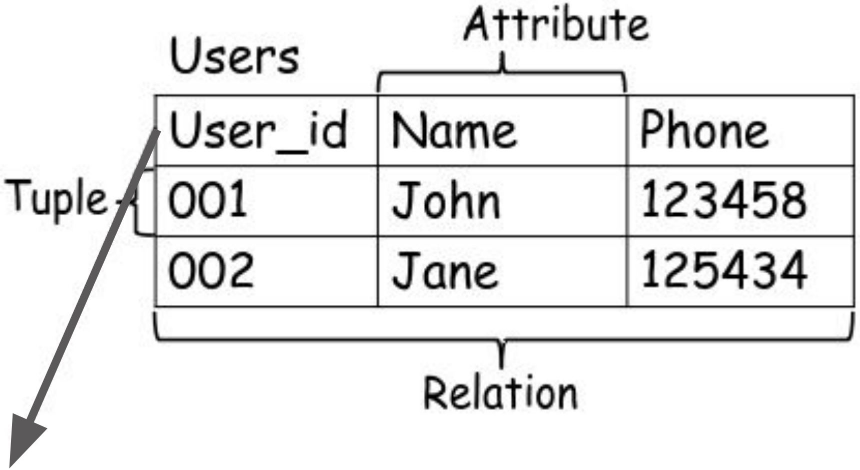
    – Poor data organization => poor performance
  - A poorly designed database allows developers and users to put in arbitrary data (i.e. “none” as a phone number) *or access data without authorization!*

Relational Databases

A digital database with a collection of tables.

- Each table contains rows and columns, with a unique key for each row
- Each entity type described in a database has its own table
  - E.g., “Employee”, “Item”, “Order”
- Each row represents an instance of the entity
  - E.g., “John Jenny”, “Soap”
- Each column represents an attribute
  - E.g., “phone number”, “price”

Relational Databases (cont.)



Users

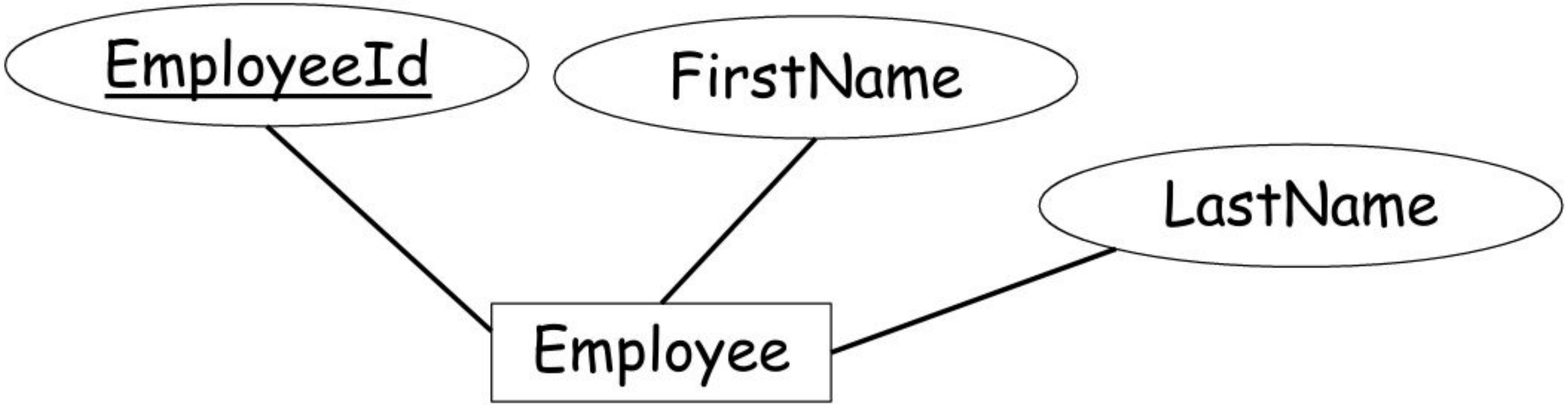
User_id	Name	Phone
001	John	123458
002	Jane	125434

Primary Key/Unique Key: to uniquely specify a tuple in a table

Foreign Key: an attribute in a relational table that matches the primary key column of another table. It can be used to cross-reference tables.

Entities and Attributes

- An entity is similar to a semantic object
- It includes attributes that describe the object

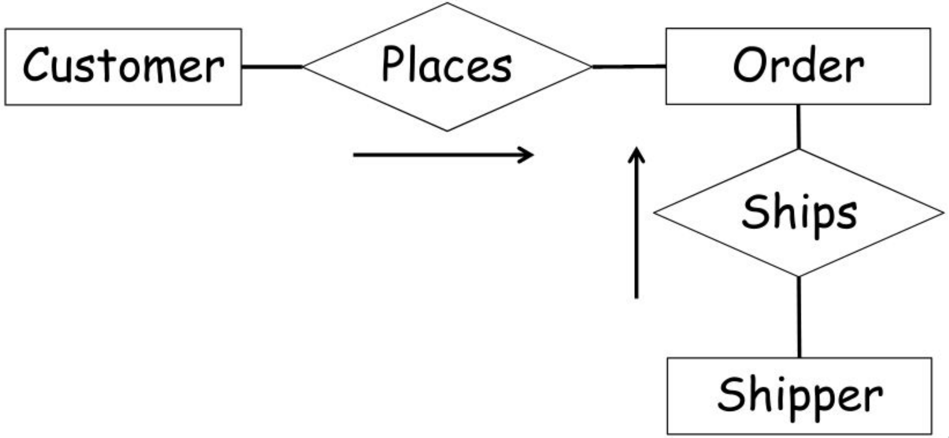


Entity-Relationship Models

- Entity-relationship (ER) diagrams are similar to semantic object modelings (i.e., class diagrams)
- They use different notations
- Focus is more on relations and less on class structure

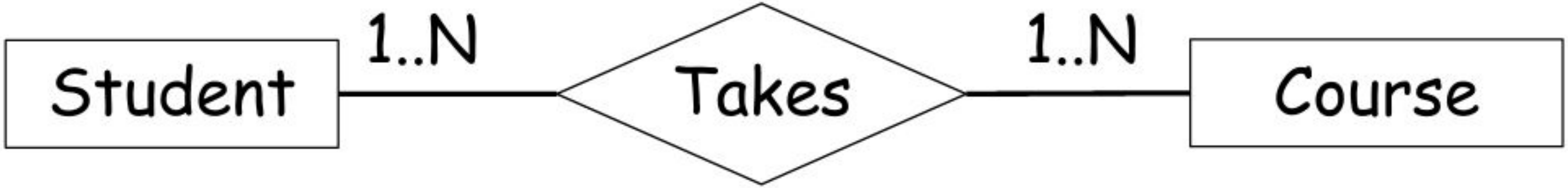
Relationships

- An ER diagram indicates a relationship between entities with a diamond
- Sometimes arrows are added to indicate direction of relationship



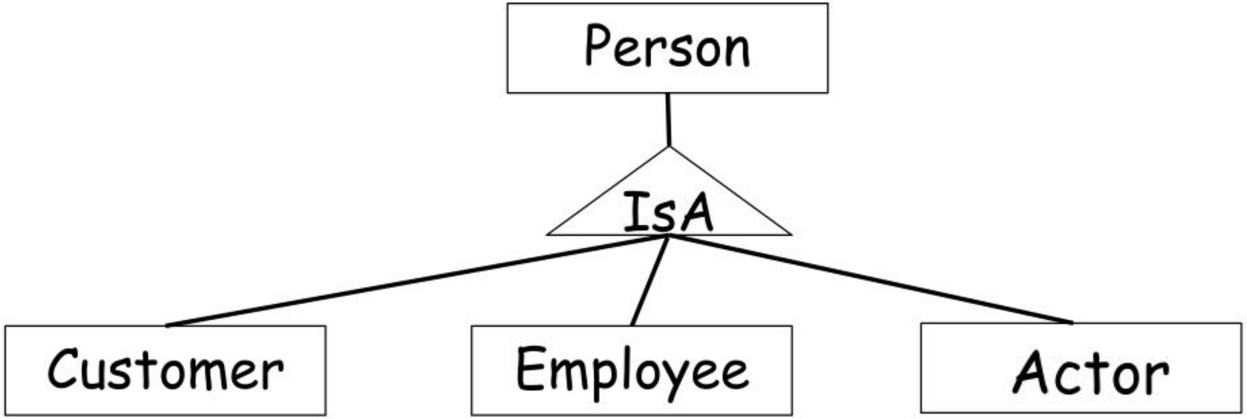
Cardinality

- Numbers used to describe relationship quantitatively.



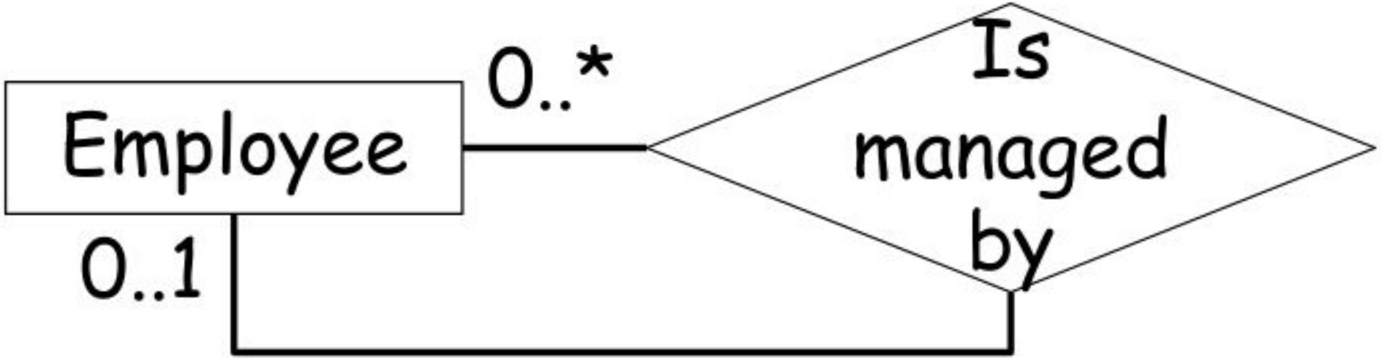
Inheritance

- A triangle named “IsA” represents the inheritance relationship.



Reflexive Associations

- An object refers to an object of the same class.

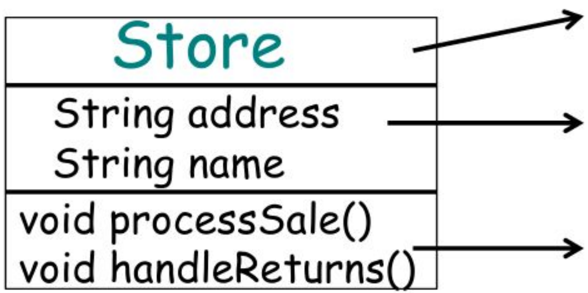


Mapping Class Diagrams to Tables

Can often map content of class diagrams to ER diagrams to show relationships between data.

- Does not work for other classes
- Sometimes you need to explicitly add a primary key to distinguish data in tables
- Database management systems (DBMSs) usually provides functionality to automatically increment primary key

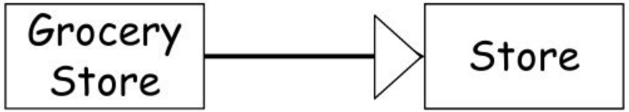
Reminder: Class Diagram Syntax



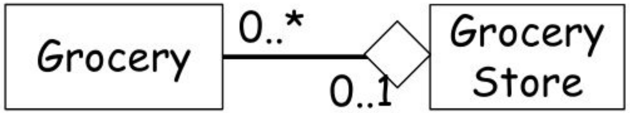
Class name: abstract concept

Attributes: properties relevant to the use case

Operation (Method signatures): behaviors of the class

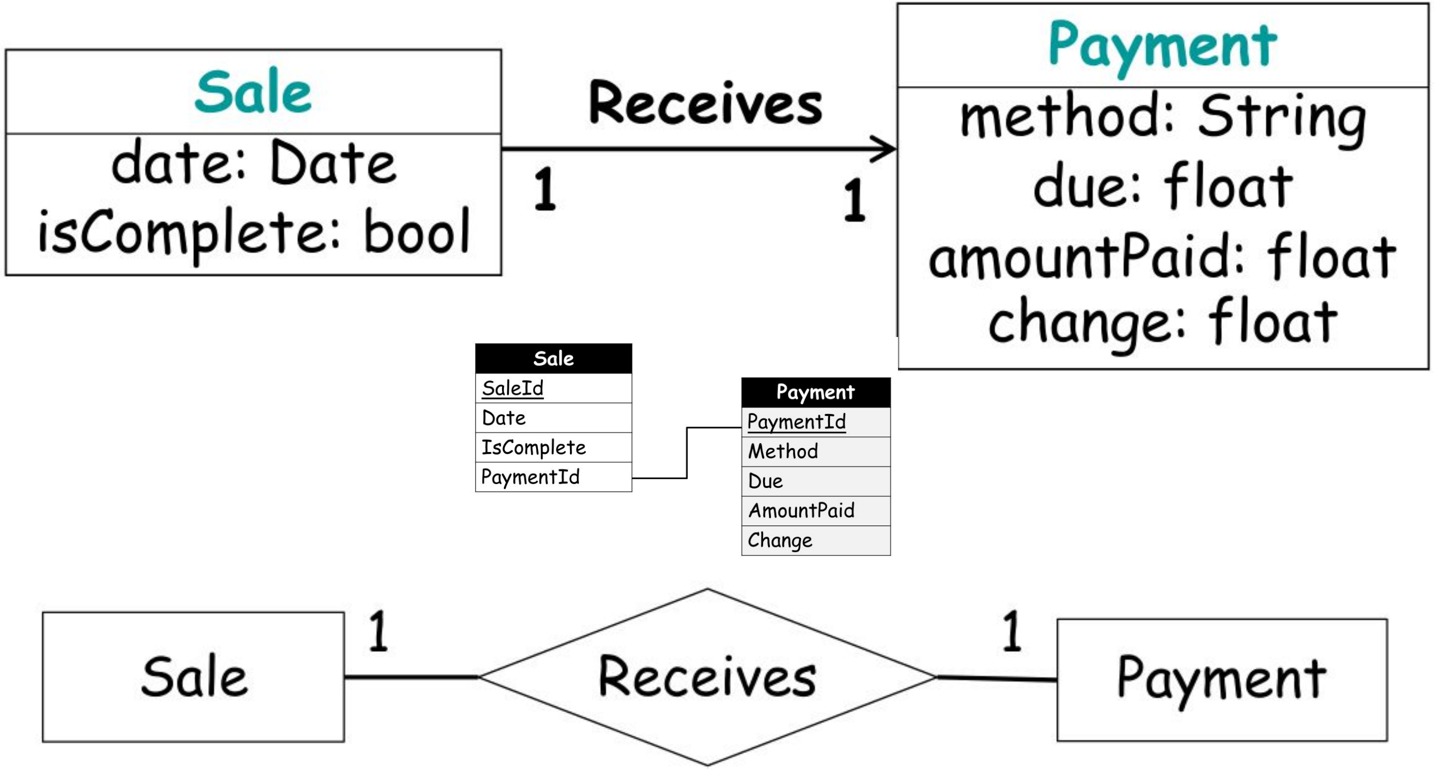


Generalization: “is-a” relationship. A sub-class inherits all attributes and operations of its super class.



Aggregation: “has-a” relationship. The container and elements can exist independently from each other

One-to-One Associations



Sale

date: Date

isComplete: bool

Receives

1

1

Payment

method: String

due: float

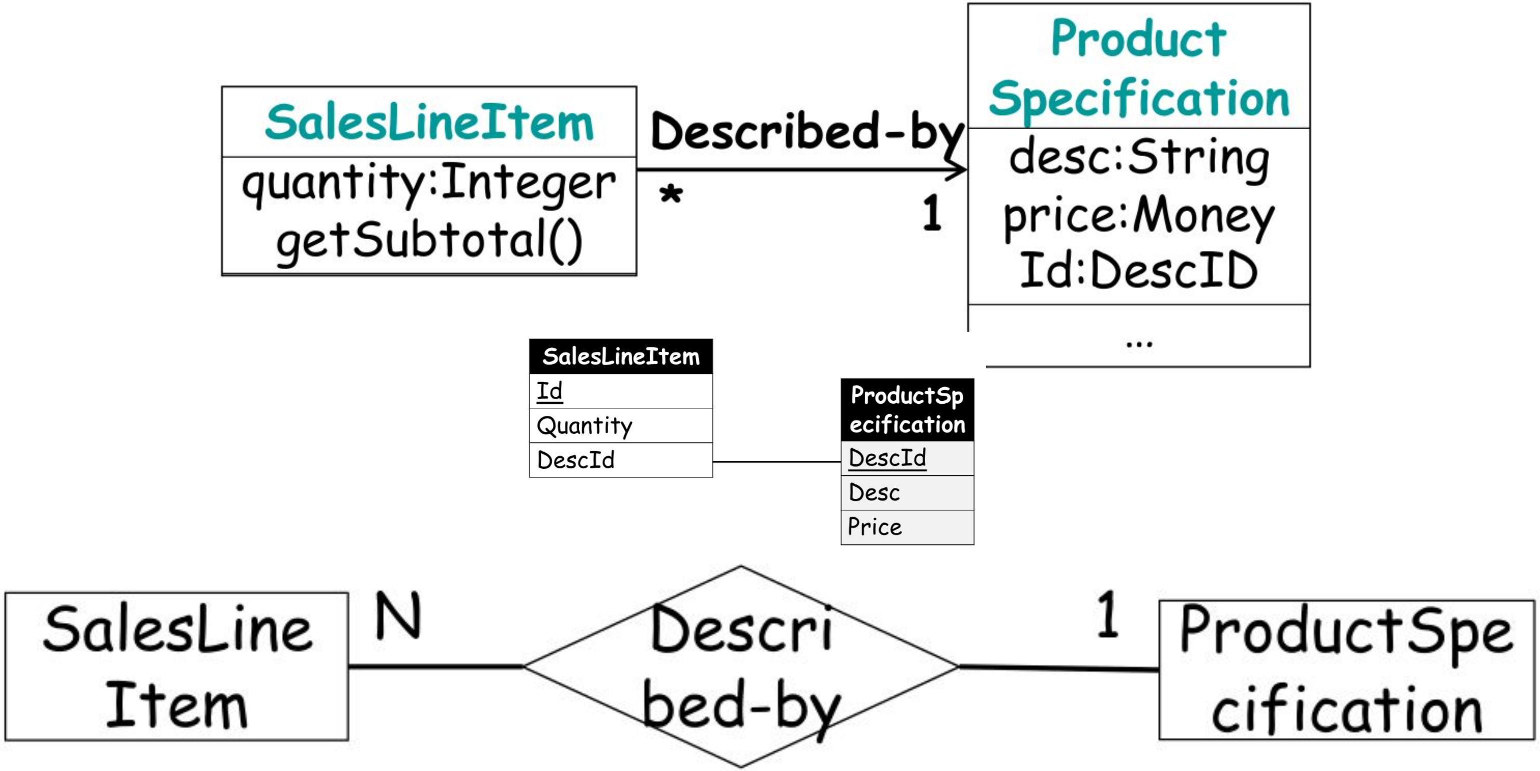
amountPaid: float

change: float

Sale
SaleId
Date
IsComplete
PaymentId

Payment
PaymentId
Method
Due
AmountPaid
Change

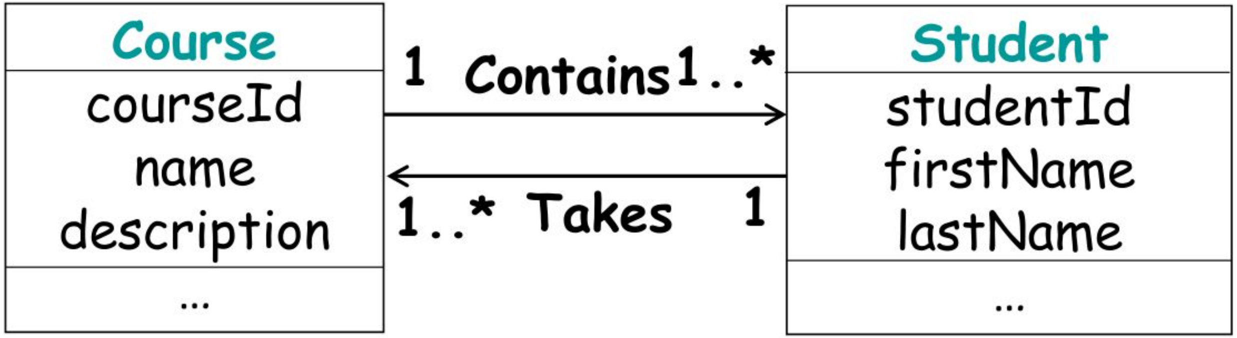
One-to-Many Associations



SalesLineItem
Id
Quantity
DescId

ProductSpecification
DescId
Desc
Price

Many-to-Many Associations



Course

courseId

name

description

...

1 Contains 1..*

1..* Takes 1

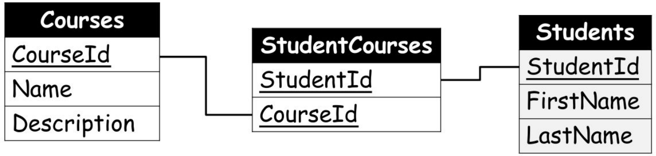
Student

studentId

firstName

lastName

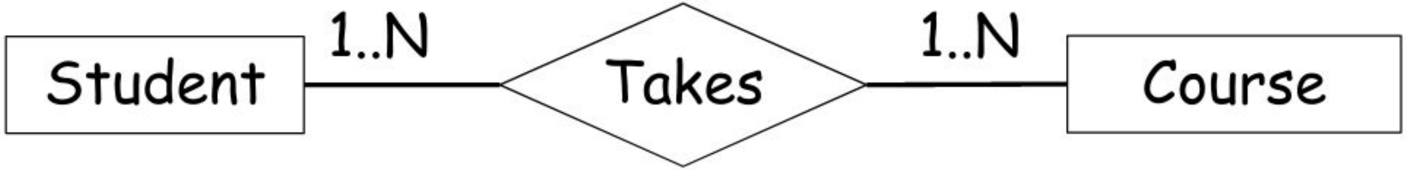
...



Courses
CourseId
Name
Description

StudentCourses
StudentId
CourseId

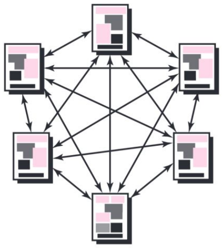
Students
StudentId
FirstName
LastName

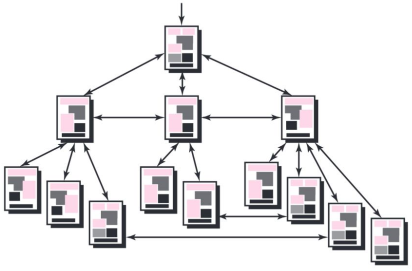


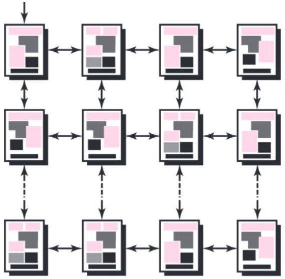
Student 1..N Takes 1..N Course

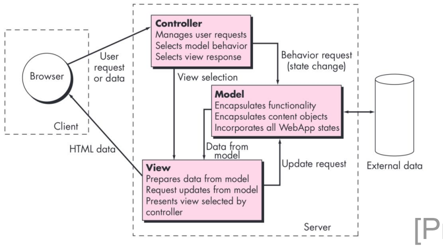
A brief digression on web app design

- What is a web app?
  - A program that uses a web browser to perform specific functions.
- *“There are essentially two basic approaches to [web] design: the artistic ideal of expressing yourself and the engineering ideal of solving a problem for a customer”* [Nielsen]
- Aesthetics, layout, graphic design, content, navigation,...









[Pressman]

…mobile app design

- What is a mobile app?
  - A program that uses a mobile device to perform specific functions.
- Still concerned with aesthetics, layout, graphic design, content, navigation,...
- And multiple hardware and software platforms!
  - Smartphones, tablets, wearable devices, etc.
  - Android, iOS, Blackberry, Windows, etc.
  - App stores have different rules
  - More complex interactions
  - Power and space/storage management
  - Security and privacy

Design Patterns (i.e. Low-Level Design)

Design patterns are descriptions of *communicating objects and classes* that are <u>customized</u> to solve a general design problem in a particular context.

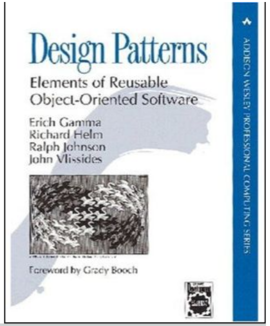
The design pattern identifies the participating classes and instances, their roles and collaborations, and the distribution of responsibilities.

Design Patterns (cont.)

Why design patterns?

- Appy working solutions to approaches
- Based on the implementations of many systems
- Capture and pass on the knowledge of experienced designers
    - Useful for inexperienced
    - Communicating about design

But do software engineers actually use them?


Design Patterns
Elements of Reusable Object-Oriented Software
Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides
Foreword by Grady Booch
ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES

Design Pattern Families

Creational

Concerned with the process of object creation

- Increases flexibility and reuse of code

Structural

Deal with the composition of classes or objects

- Organizing different classes and modules to form larger structures or add new functionality

Behavioral

Characterize the ways in which classes or objects interact and distribute responsibility

- Algorithms and assignment of responsibilities between objects

Creation Patterns

- **Abstract Factory**: Creates an instance of several families of classes
- **Builder**: Separates object construction from its representation
- **Factory Method**: Creates an instance of several derived classes
- **Object Pool**: Avoid expensive acquisition and release of resources by recycling objects that are no longer in use
- **Prototype**: A fully initialized instance to be copied or cloned
- **Singleton** A class of which only a single instance can exist

More details later…

Structural Patterns

- **Adapter**: Match interfaces of different classes
- **Bridge**: Separates an object's interface from its implementation
- **Composite**: A tree structure of simple and composite objects
- **Decorator**: Add responsibilities to objects dynamically
- **Facade**: A single class that represents an entire subsystem
- **Flyweight**: A fine-grained instance used for efficient sharing
- **Private Class Data**: Restricts accessor/mutator access
- **Proxy**: An object representing another object

Behavioral Patterns

- **Chain of responsibility**: A way of passing a request between a chain of objects
- **Command**: Encapsulate a command request as an object
- **Interpreter**: A way to include language elements in a program
- **Iterator**: Sequentially access the elements of a collection
- **Mediator**: Defines simplified communication between classes
- **Memento**: Capture and restore an object's internal state
- **Null Object**: Designed to act as a default value of an object
- **Observer**: A way of notifying change to a number of classes
- **State**: Alter an object's behavior when its state changes
- **Strategy**: Encapsulates an algorithm inside a class
- **Template method**: Defer the exact steps of an algorithm to a subclass
- **Visitor**: Defines a new operation to a class without change

Design Disclaimer

- No silver bullet for choosing high-level or low-level design patterns.
- Design will change as requirements and code change.
  - First Law!

High-level design processes, patterns, and issues will differ based on the domain of the product you are implementing!

Next Time…

- Design Pattern Workshop on Friday (10/20)
    - Led by GTA Xiaoxiao Gan

- **HW3 due Friday (10/20 at 11:59pm)**

References

- RS Pressman. *“Software engineering: a practitioner's approach”.*
- Cast Software *“What is Software Architecture?”*. <https://www.castsoftware.com/glossary/what-is-software-architecture-tools-design-definition-explanation-best>
- K.D. Cooper, L. Torczon, *“Engineering a Compiler”.*Theo Mandel. *“Golden Rules of User Interface Design”.* <https://theomandel.com/resources/golden-rules-of-user-interface-design/>
- <https://medium.com/swlh/ordering-food-and-the-mvc-architecture-d5cbf3859d60>
- Na Meng and Barbara Ryder
- Chris Parnin
- Sarah Heckman