# Design Defects and Restructuring

LECTURE 14

SAT, DEC 07, 2019

# Enterprise Application Architecture Prerequisites

Layering

Organizing Domain Logic

Mapping to Relational Databases

Web Presentation

Concurrency

Session State

Distribution Strategies

# Layering

Benefits

- Understandability – Coherent, without knowing much about other layers
- Substitutability – With alternative implementation
- Minimal Dependency – Adaptation and Abstraction
- Standardization – Create your own standards for a layer
- High-level Service Use – Provide barrier to a layer below

Downsides

- Encapsulation
  - Not all elements are encapsulated
  - UI to Database?
- Too many Layers
  - Data transformation affects performance

# Layering

3 Principal Layers

- Presentation
  - Provision of services, display of information (e.g., in Windows or HTML, handling of user request (mouse clicks, keyboard hits), HTTP requests, command-line invocations, batch API)
- Domain
  - Logic that is the real point of the system
- Data Source
  - Communication with databases, messaging systems, transaction managers, other packages

# Organizing Domain Logic

Transaction Script
- Simple Logic

Domain Model
- Complex Logic

Table Module
- Moderate Logic and good API Tools around

Service Layer
- Provides API

# Mapping to Relational Databases

Architecture

- Mappings and Gateways

Behavioral Issues

- Data Reading
- Data Manipulation

Using Metadata

- Code Generation
- Reflective Programming

Database Connections

# Domain Logic Patterns

Transaction Script

Domain Model

Table Module

Service Layer

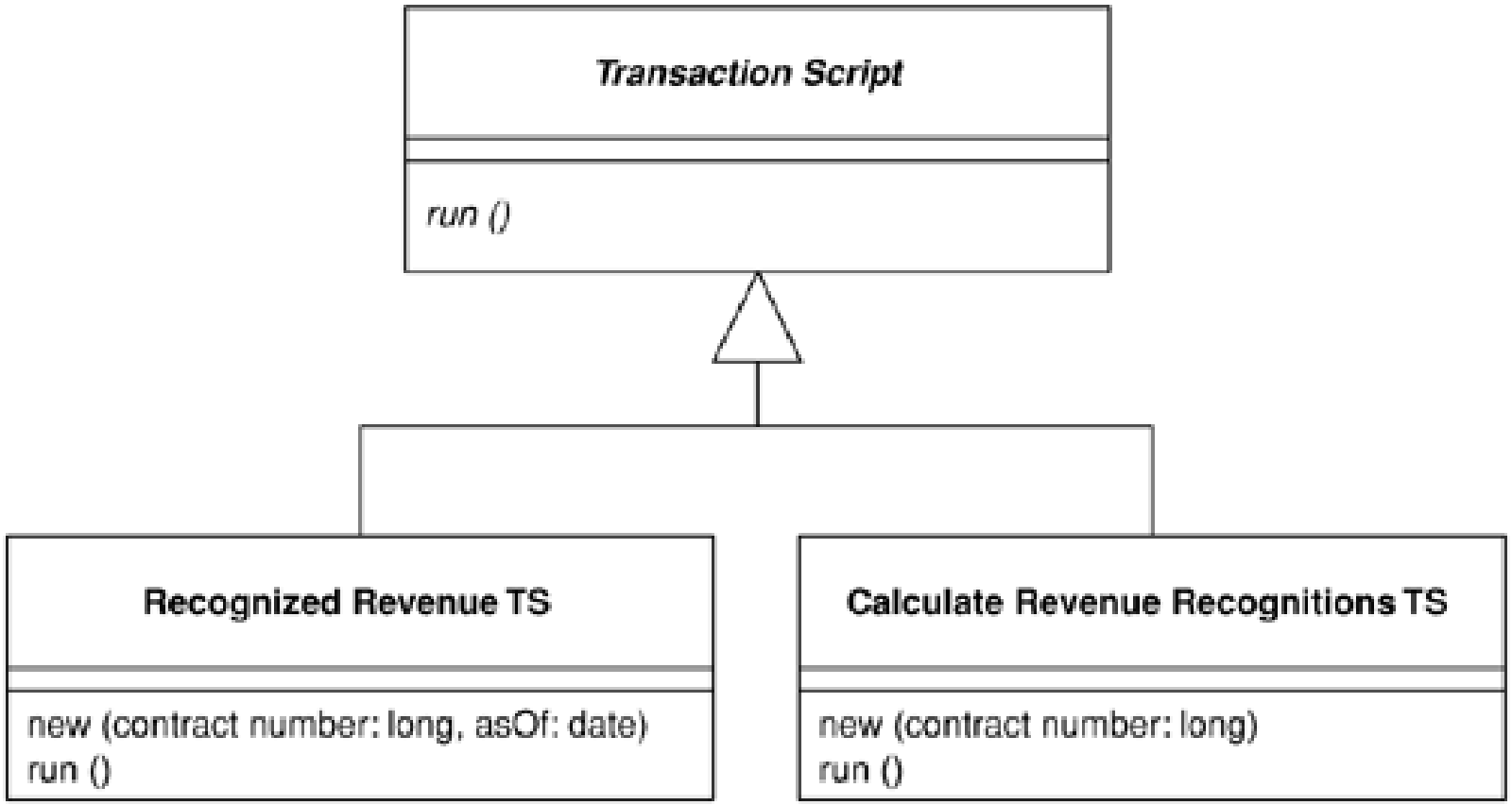
# Transaction Script

Organizes business logic by procedures where each procedure handles a single request from the presentation

When to use it

- Simplicity
- Smaller logic
- Beware of duplication

# Transaction Script



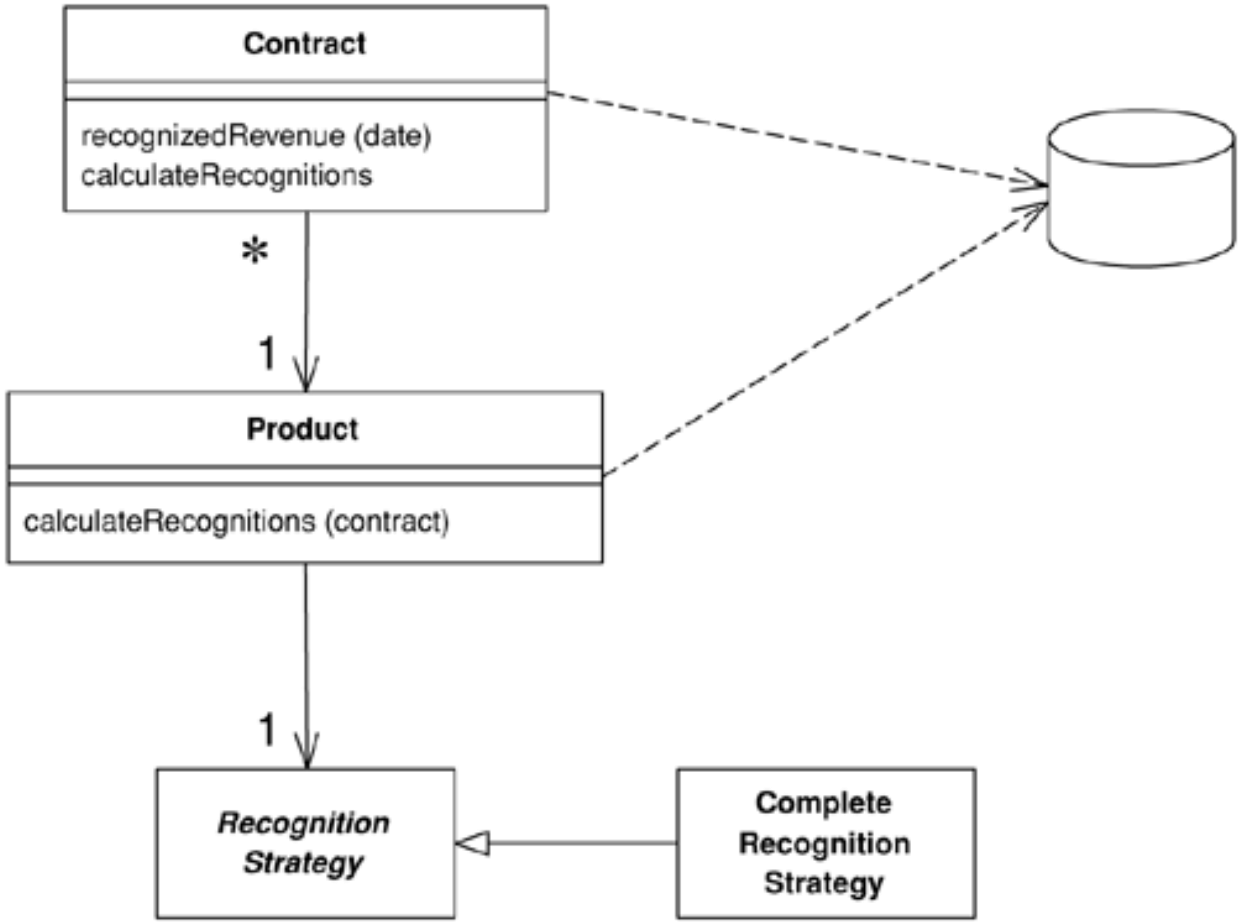
# Domain Model

An object model of the domain that incorporates both behavior and data

When to use it

- Complex logic
- Ever changing business rules
  - Involving validation, calculations, and derivations

# Domain Model



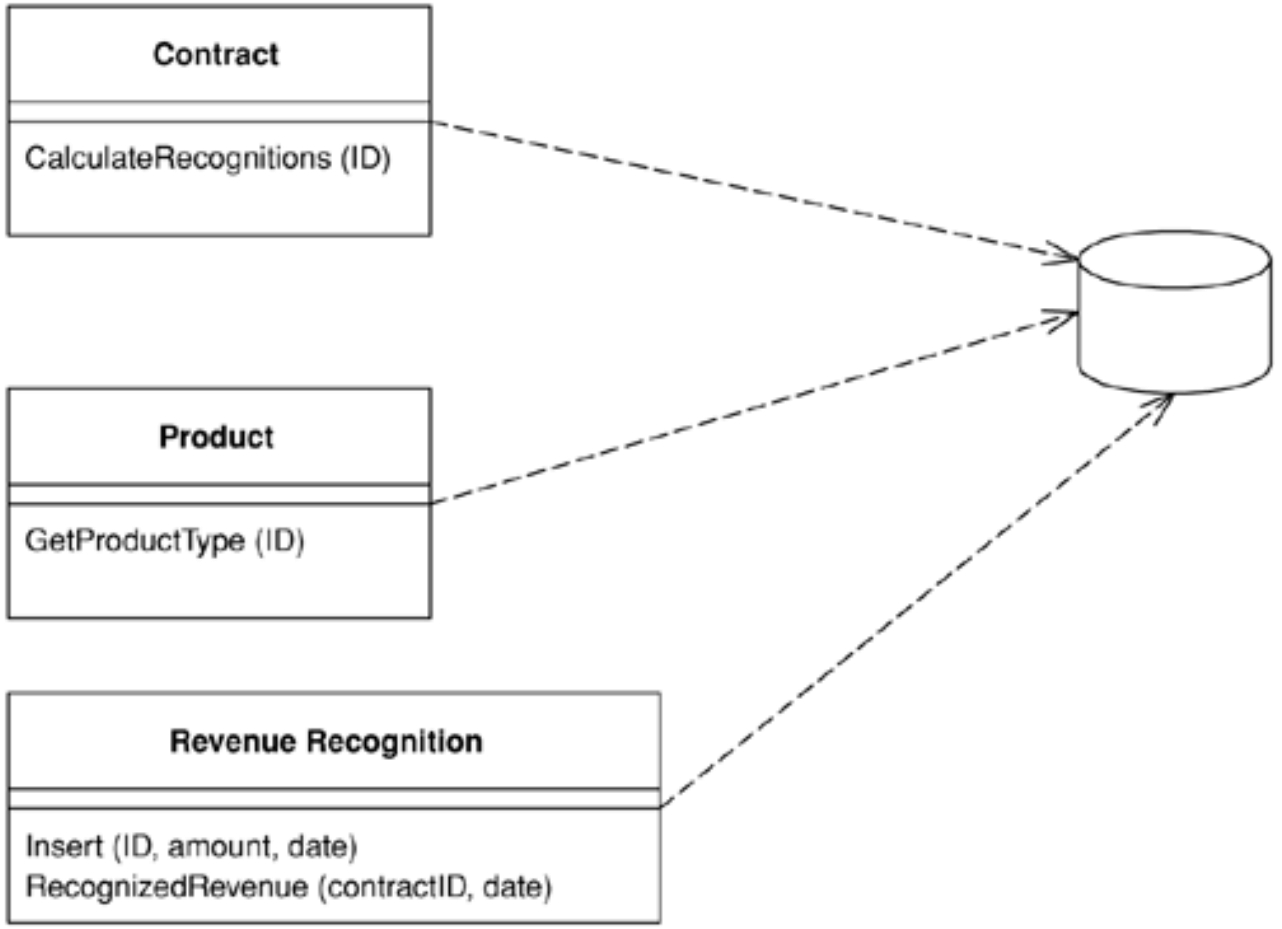
# Table Module

A single instance that handles the business logic for all rows in a database table or view

When to use it

- Based on table oriented data
  - Access using Record Set
- Data structure are fairly straightforward
- No direct instance-to-instance relationships

# Table Module



# Service Layer

Defines an application's boundary with a layer of services that establishes a set of available operations and coordinates the application's response in each operation

When to use it

- In an application with more than one kind of client of its business logic, and complex responses in its use cases involving multiple transactional resources, it makes a lot of sense to include a Service Layer with container-managed transactions

# Data Source Architectural Patterns

Table Data Gateway

Row Data Gateway

Active Record

Data Mapper

# Table Data Gateway

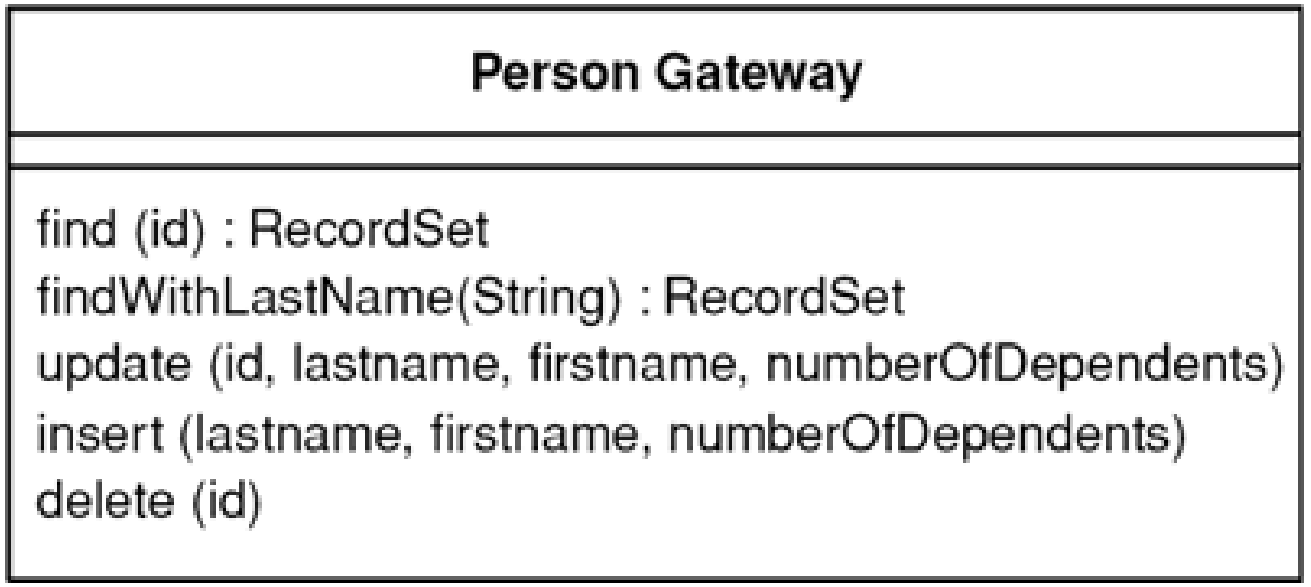
An object that acts as a Gateway to a database table

- One instance handles all the rows in the table

When to use it

- Work well with Table Module and Transaction Script
- Encapsulation of database access

# Table Data Gateway



# Row Data Gateway

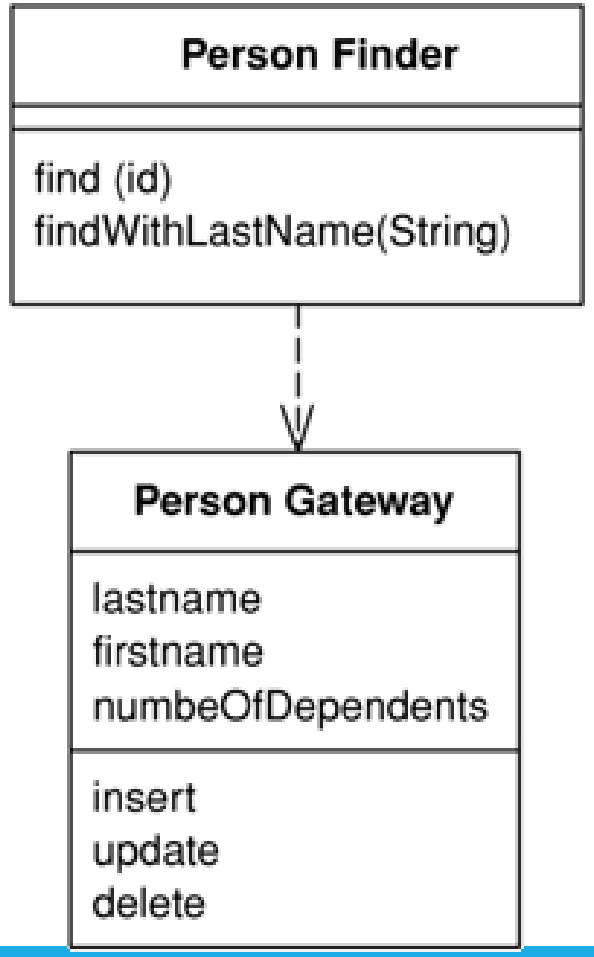
An object that acts as a Gateway to a single record in a data source

- There is one instance per row

When to use it

- Use it with Transaction Script

# Row Data Gateway



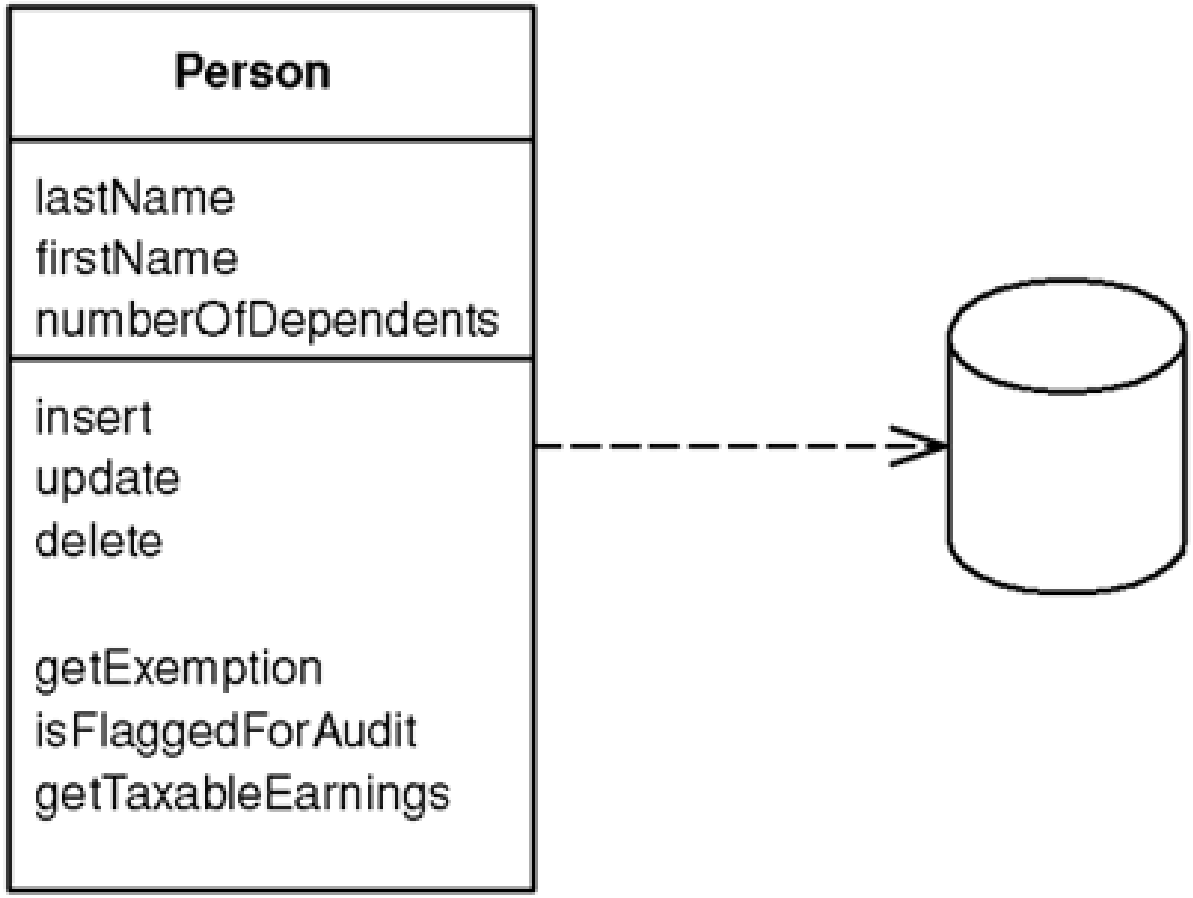
# Active Record

An object that wraps a row in a database table or view, encapsulates the database access, and adds domain logic on that data

When to use it

- Logic is not complex

# Active Record



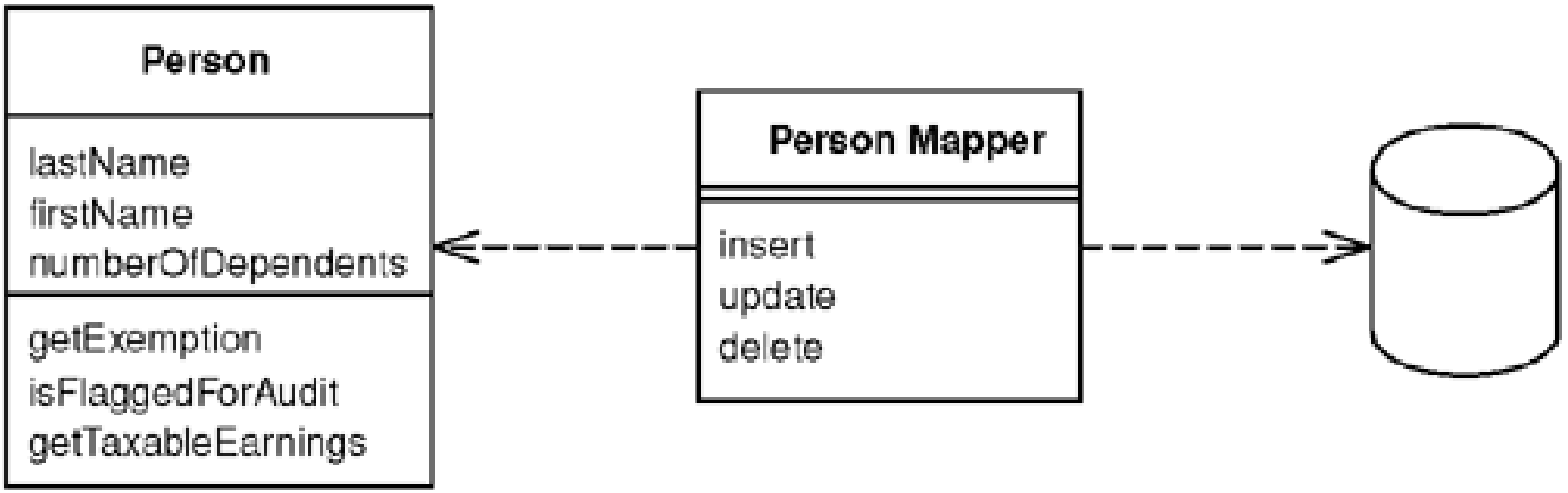
# Data Mapper

A layer of Mappers that moves data between objects and a database while keeping them independent of each other and the mapper itself

When to use it

- When you want the database schema and the object model to evolve independently
- When you are using Domain Model

# Data Mapper


Person
lastName
firstName
numberOfDependents
getExemption
isFlaggedForAudit
getTaxableEarnings
Person Mapper
insert
update
delete

# Object – Relational Behavioral Patterns

Unit of Work

Identity Map

Lazy Load

# Unit of Work

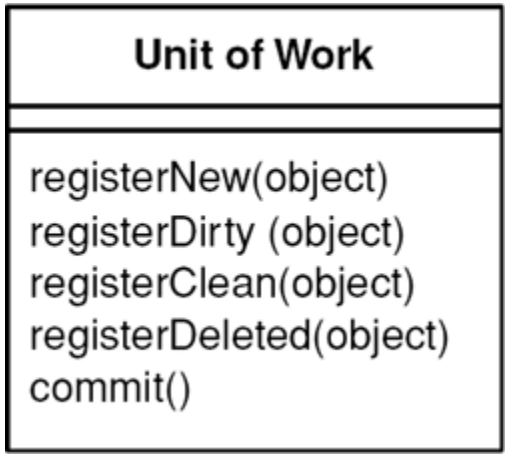
Maintains a list of objects affected by a business transaction and coordinates the writing out of changes and the resolution of concurrency problems

- Caller Registration
- Object Registration
- Unit of Work Controller

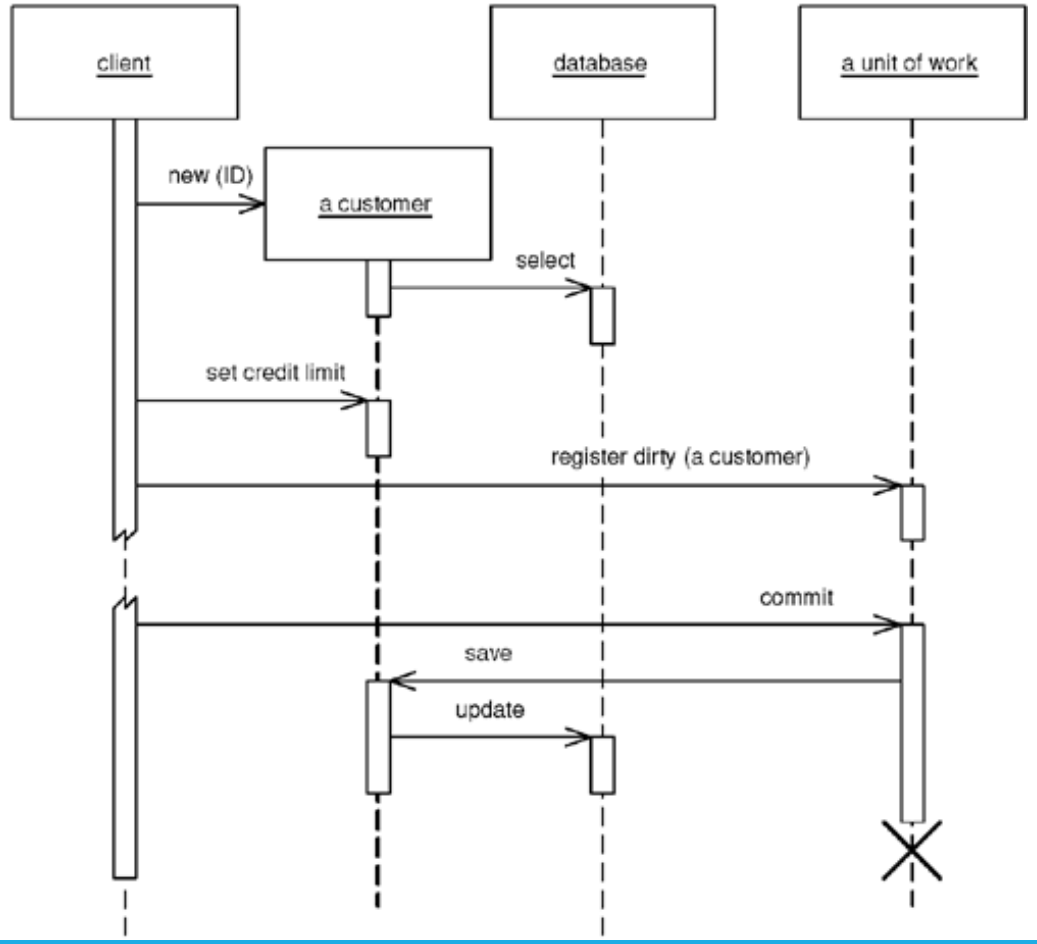
When to use it

- Keeping track of various objects
- Reducing database access

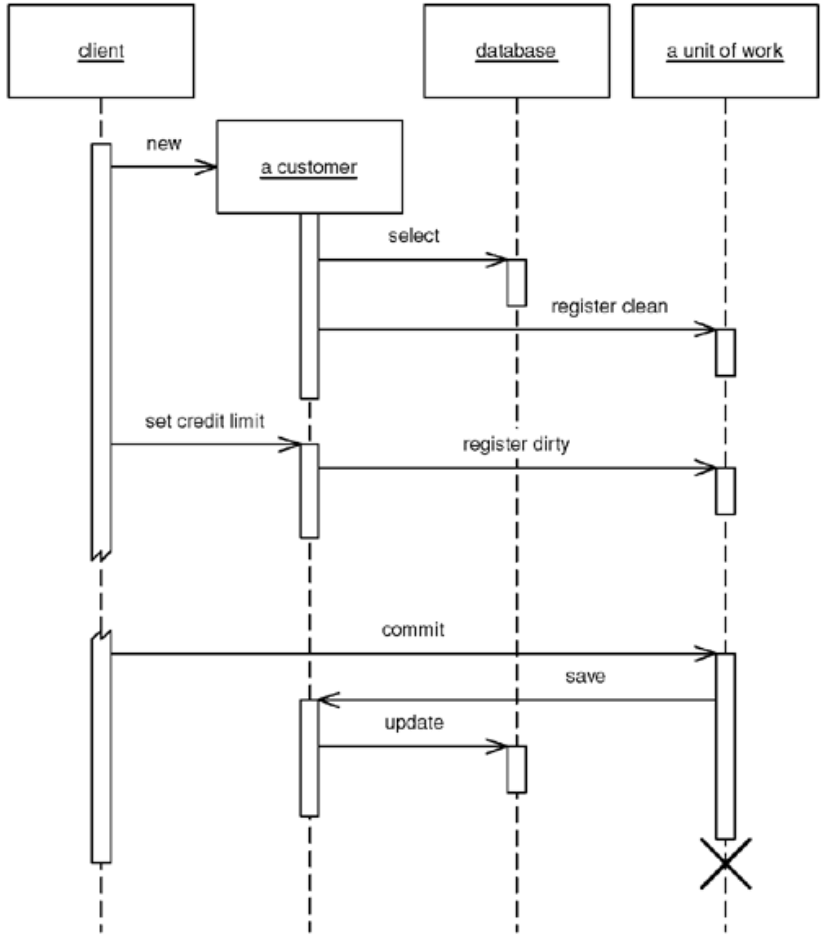
# Unit of Work


Unit of Work
registerNew(object)
registerDirty (object)
registerClean(object)
registerDeleted(object)
commit()

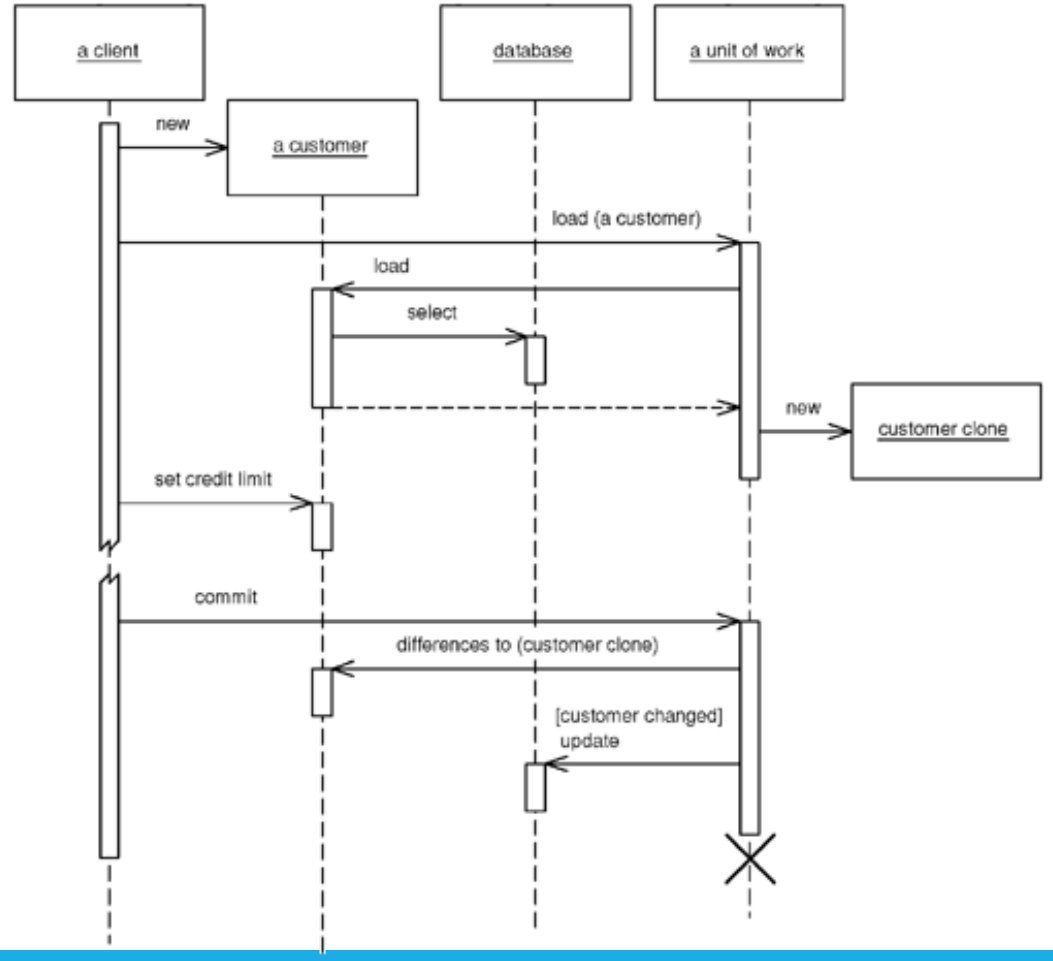
# Unit of Work – Caller Registration



# Unit of Work – Object Registration



# Unit of Work – Controller



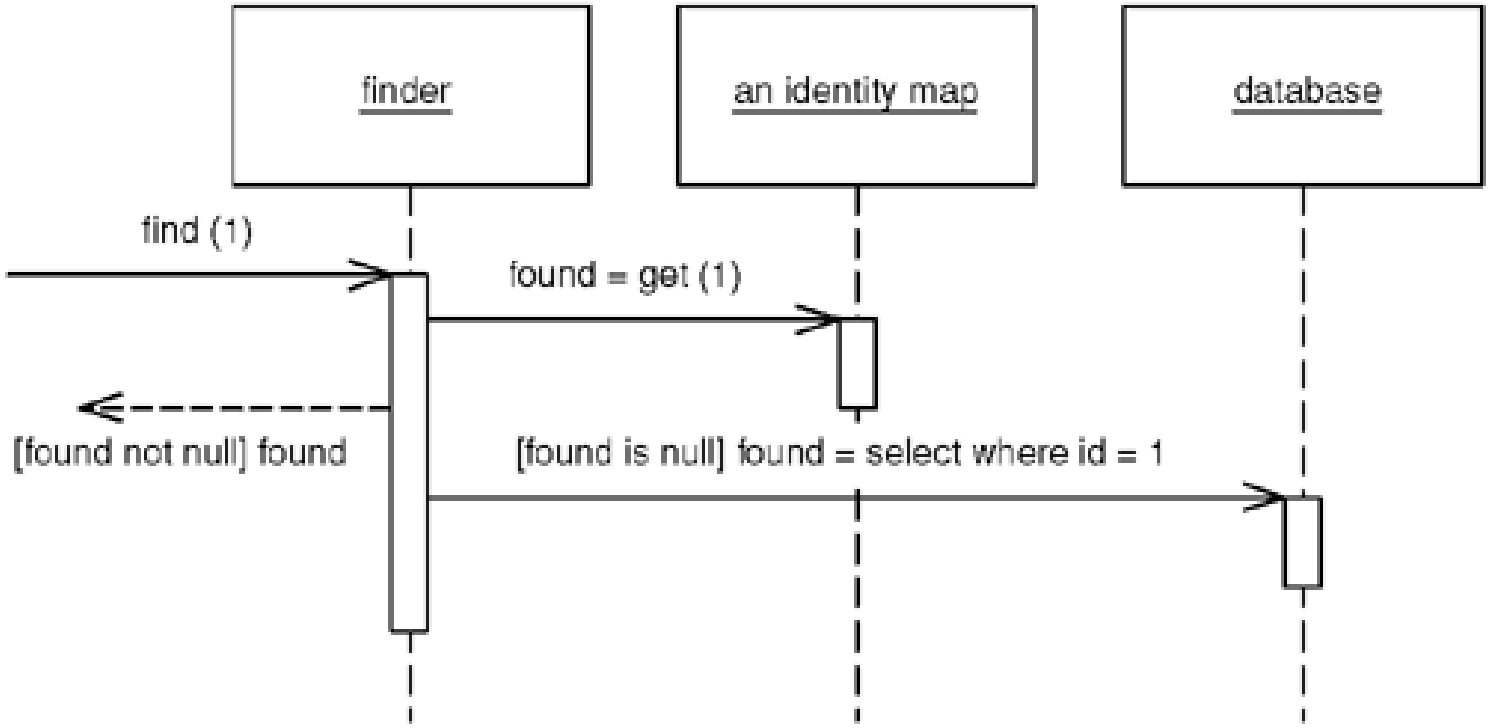
# Identity Map

Ensures that each object gets loaded only once by keeping every loaded object in a map. Looks up objects using the map when referring to them

When to use it

- Treat it as an in memory cache
- To remove in consistency
- Key field – Surrogate Key
- Explicit or Generic
- Mapper per Class or Mapper per Session

# Identity Map



# Lazy Load

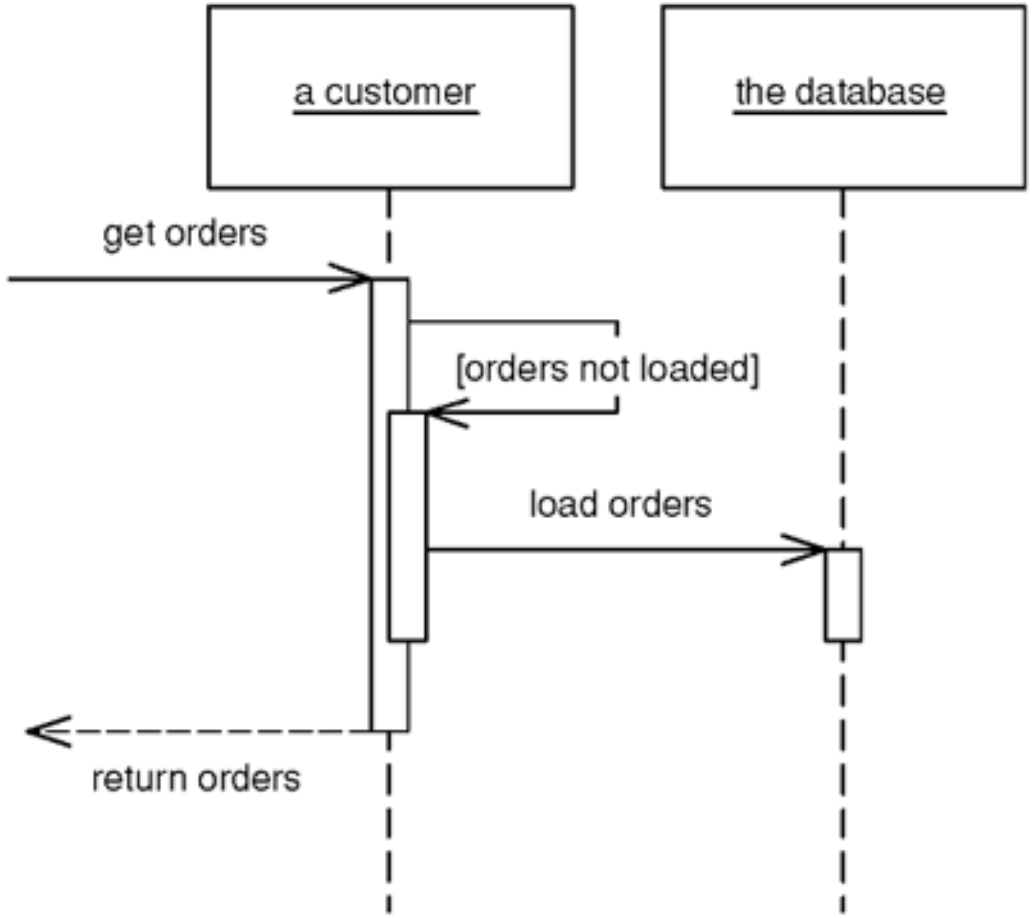
An object that does not contain all of the data you need but knows how to get it

- Lazy initialization
  - Gateways
  - Getter Method
- Virtual proxy
  - Mappers
  - Proxy pattern
- Value holder
  - Wraparound object
  - Single loading
- Ghost
  - Real object in partial state

When to use it

- Performance in reducing in memory objects

# Lazy Load


a customer
the database
get orders
[orders not loaded]
load orders
return orders