Lecture L13
Organizing Domain Logic

7	L13-29.09 **Domain Layer Design**	30.09 **Skilaverkefni 1 – Hlutbundin forritun og pattern 10%** kl. 22:00 í verkefnaskilakerfið	D07-01.10 **Vinnslulagið** Service Layer verkefni	L14-03.10 **GRASP** Barbin 2, 4
8	L15-06.10 **Database Design** Skilaverkefni 3 og 4 kynnt		D09-08.10 **Gangagrunnsforritun**	L16-10.10 **ORM**
9	L17-13.10 **Web Layer Design**		D12-15.10 **Veflagið**	L18-17.10 **REST API Design**
10	L19-20.10 **Application Design**	21.10 **Skilaverkefni 3 10%** kl. 22:00 í verkefnaskilakerfið	D14-22.10 **Heildarkerfi** **REST þjónustur og JavaScript**	L20-24.10 **Agile Principles** Barbin 6
11	L21-27.10 **Scalablity**	28.10 **Skilaverkefni 4 20%** kl. 22:00 í verkefnaskilakerfið	D14-29.10 **Skilaverkefni 4 sýnt í tíma**	L22-31.10 **Architecture and Agile** Brown 62-67
12	L23-03.11 **Summary and conclusion**		D15-05.11 **Viðtalstími**	L24-08.11 **Discussions**

Agenda

- Layering
- The Three Principal Layers
- Domain layer Patterns
  - Transaction Script
  - Domain Model
  - Table Module
  - Service Layer
- From Problem to Pattern
- Building the domain

Reading

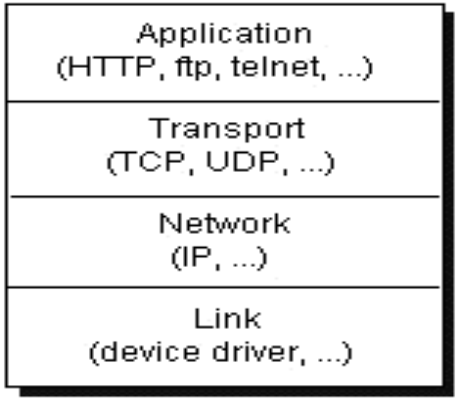
- Domain Model
- Multilayered Architecture

- P of EAA:
  - Table Module
  - Domain Model
  - Transaction Script

Layering

Layering

- Software systems can get complicated
  - Abstractions are needed
- Layering provides abstraction by separating computer systems in layers
  - Higher layers use services from lower layers
  - Each layer has dedicated tasks and hides complexity from upper layers



Benefits of Layering

- You can **understand** a single layer as a coherent whole without knowing much about other layers
- You can **substitute** layers with alternative implementation of the same basic service
- You **minimize dependencies** between layers
- Layers make good places for **standardization**
- Once you have a layer built, you can use it for many **higher-level** services

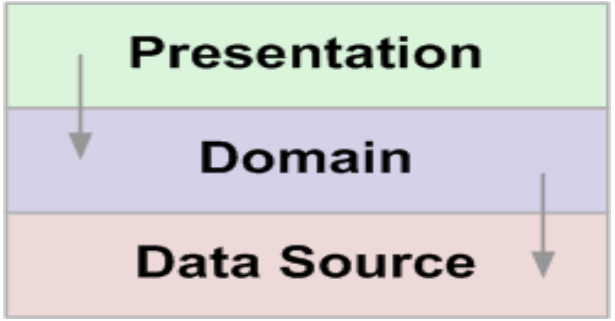
Downsides

- Layers encapsulate some, but not all, things well
  - Cascading changes
  - For example adding a field in the UI requires changes on each layer
- Extra layers can harm performance
  - At every layer things typically need to be transformed from one presentation to another
- Organizational problems
  - If difference teams work on different layers, code duplications and other problems emerge

The Three Principal Layers

The Three Layers

- **Presentation**
  - User's interface to the system
  - User can be another system
  - Accepts input, displays views
- **Domain**
  - The Application of the system
  - The "Business logic"
  - Tends to creep into presentation and data source
- **Data Source**
  - Connection to the database
  - Also Persistence



Where is the Business Logic?

- “Business Logic”
  - or “Complex business illogic”?
  - Business Rules including all special cases
  - It is easy to think of business logic but difficult to spot
    - many corner cases
  - Also referred to as **Domain Logic**
- Enterprise software has a lot of “noise”
  - “Plumbing” code around the domain logic
  - Is *getCustomer* domain logic?
  - Application Logic – or workflow

How to define Domain Logic?

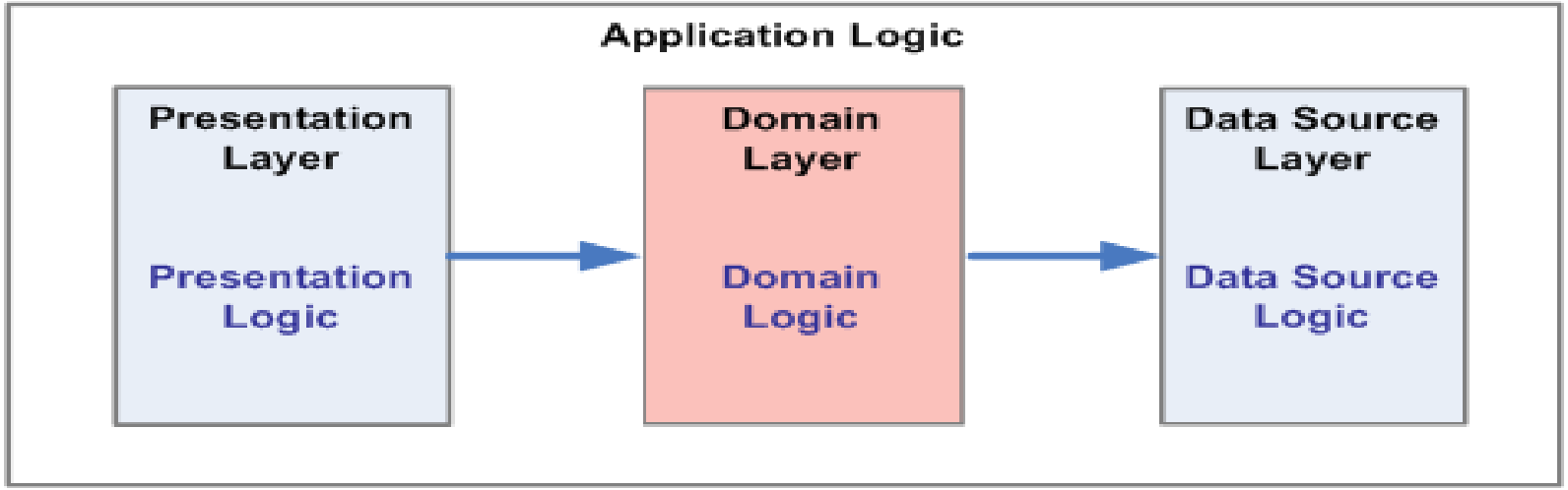
- Definition
  - If all environment (noise) is taken way, what is left should be **domain logic**
  - Environment is any middleware (web, data, ejb)
- Domain Logic should be
  - Simple classes and interfaces – POJOs
  - Easy to test!
  - Easy to move around
  - Easy to access
- Limited dependencies
  - Dependencies can be reduced by injection

The Three Layers

- Dependencies
  - **Presentation** gets information from lower layers, preferable **Domain Layer**
  - **Domain** or **Data Source Layers** should not depend on the **Presentation Layer**
  - **Domain Layer** depends on **Data Source Layer**
  - **Data Source Layer** could use some objects from the **Domain Layer**

Logic that is not Domain Logic

- Logic in different places



Presentation Logic

- How to display items, how things look
  - Which colours to choose
  - Number and Date formats, zero padding
- The presentation should
  - Have easy and unified access to the data
  - Avoid having assumptions on the data
  - Avoid the need to calculate based on the data
- Example view models
  - HTML, Flash
  - JSP, ASP

Data Source Logic

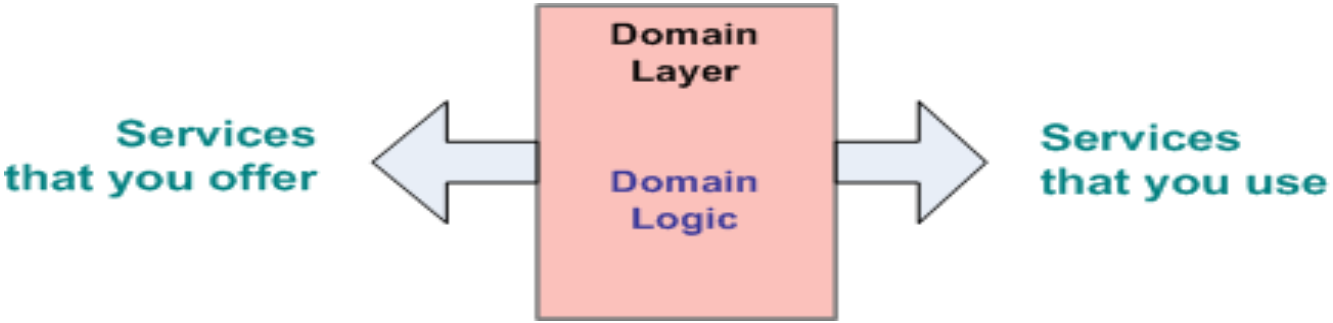
- How to query and store items
  - Optimization
  - Data manipulation for upper layer
- The **Data Source Layer** should
  - Provide simple interface to the domain layer
  - Maintain speed and integrity
  - Hide the database layout from the object model
- Example data model
  - DAO
  - Stored Procedures

Application Logic

- Having to do with application responsibilities
  - “Workflow” logic
  - Noise
- Example:
  - Notifying contract administrator
  - Checking HTTP parameters and selecting which controller to pass the responsibility to

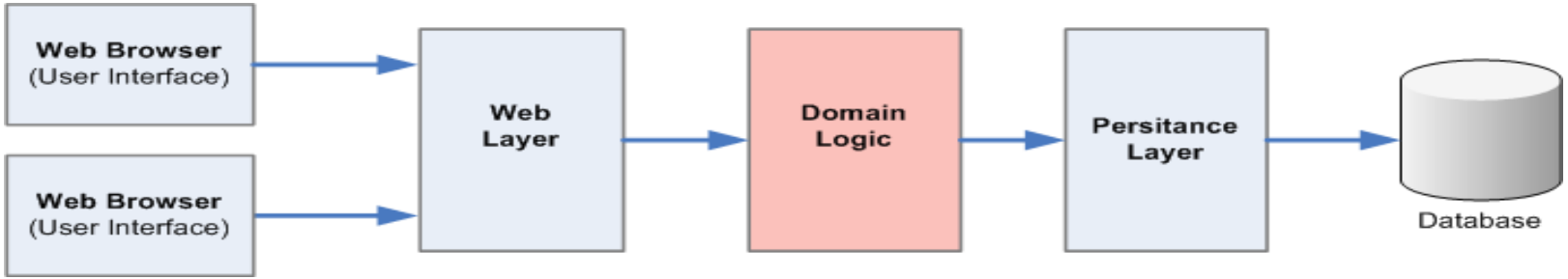
Interfaces

- Users can be other programs or services
  - Presentation is an external interface that you provide as a service to others
- Data can be other systems
  - Data source is an external interface of services that you use



Where is the Business Logic?

- Create a specific Layer for the Domain Logic
  - Well know patterns like **Domain Model**



  - An object model of the domain that incorporates both behaviour and data
  - Object-oriented programming!!!

Where is the Business Logic?

- Flooding of logic...

- If you don’t provide a place for the Domain Logic it will find a place – usually in the wrong!

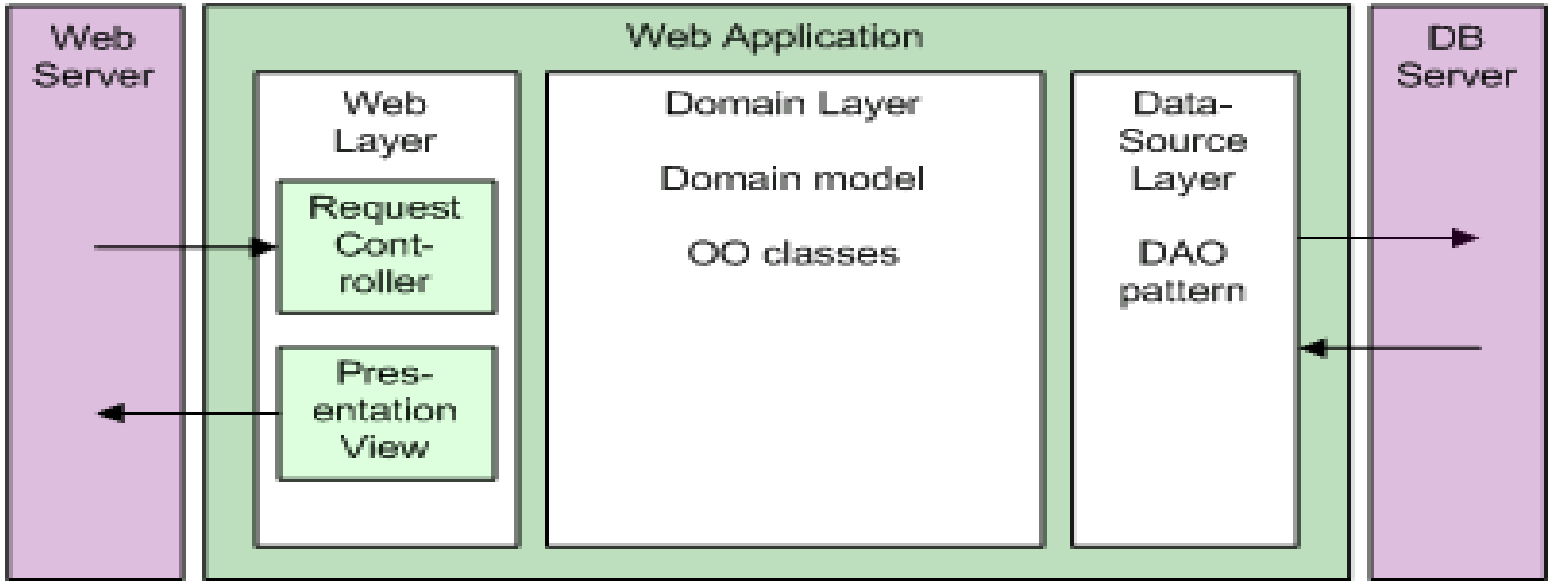
Enterprise Web Layers

- **Clients** usually run on the user's machine
  - Separate tier
  - Web Browser or Rich Internet Application (RIA)
  - Can also be processes or other applications
- **Web Components** run on a Web Server
- **Domain Components** run on Application Server
  - Web Server or EJB Server for EJBs
- **Data Source Components** run on the same Application Server (as the domain)
- Database is usually on a separate tier

Domain Layer Patterns

Domain Layer

- Provides the business logic of the application
  - Not part of the Presentation Layer nor the Data Source layer



Domain Layer Patterns

- **Transaction Script**
  - Organizes business logic by procedures where each procedure handles single request from the presentation
- **Domain Model**
  - An object model of the domain that incorporates both data and behaviour
- **Table Module**
  - One class that provides domain logic to table or view in database

Transaction Scripts

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

- Most business applications can be thought of as a series of transactions
  - A Transaction Script organizes all this logic primarily as a single procedure

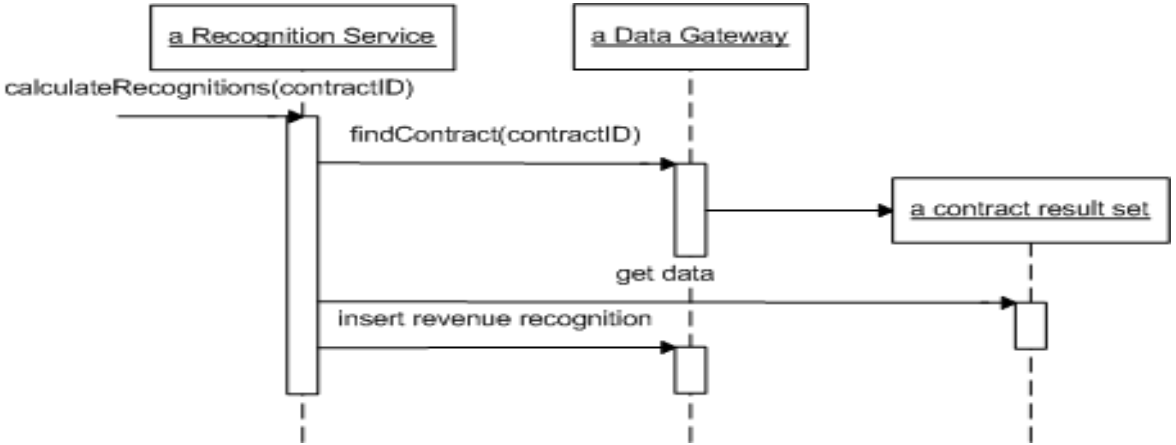
```
recognizedRevenue(contractNumber: long, asOf: Date) : Money
calculateRevenueRecognitions(contractNumber long) : void
```

Transaction Scripts

- Works well if model is simple
  - Small amount of logic
  - No state needed
  - Moving data between presentation and database
- Problems
  - Code duplication between transactions
  - Common code tends to be duplicated
  - Since no state is used, and each transaction is separate from any other, there might be too many calls to database layer

Transaction Scripts

- Revenue recognitions example
  - One script handles all the logic
  - Uses a data gateway to access data



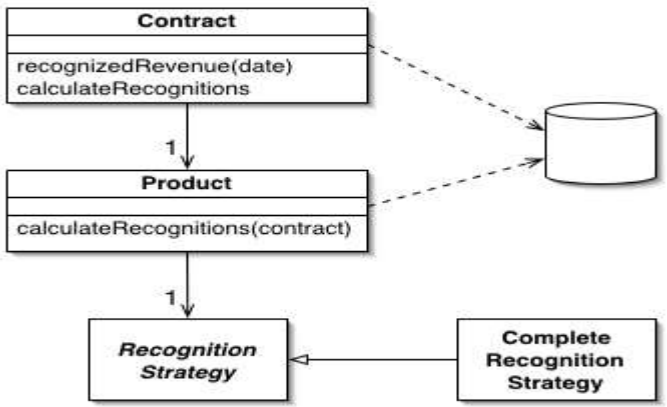
Domain Model

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

- Rules and logic describe many different cases and slants of behavior
- Web of interconnected objects
  - Where each object represents some meaningful entity
  - Dependencies between objects
- How it works
  - Object that represent data (value objects) and business rules (behavior)

Domain Model

- Simple model
  - Similar to the database design
  - Simple to map to the database
- Rich model
  - Different from the database design
  - Better for solving some complex logic and doing calculation



Domain Model

- Revenue recognitions example
  - Multiple classes each with different responsibility
  - Each class has data and logic to calculate

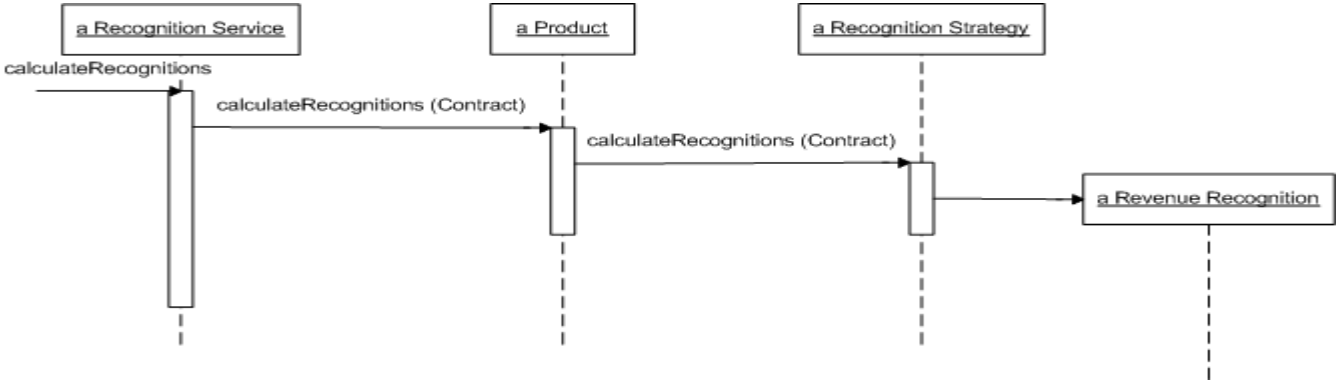


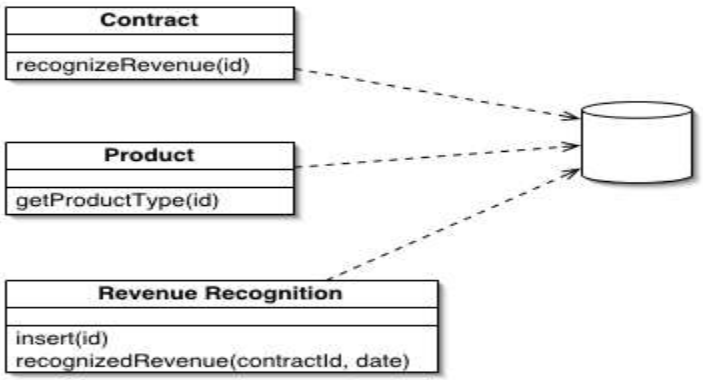
Table Module

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

- Organizes domain logic with one class per table in the database
  - Single instance of the class contains various procedures that will act on the data
  - One object per table handle
- How it works
  - One class that provides domain logic to table or view in database
  - A Domain Model will have one order object per order while a Table Module will have one object to handle all orders

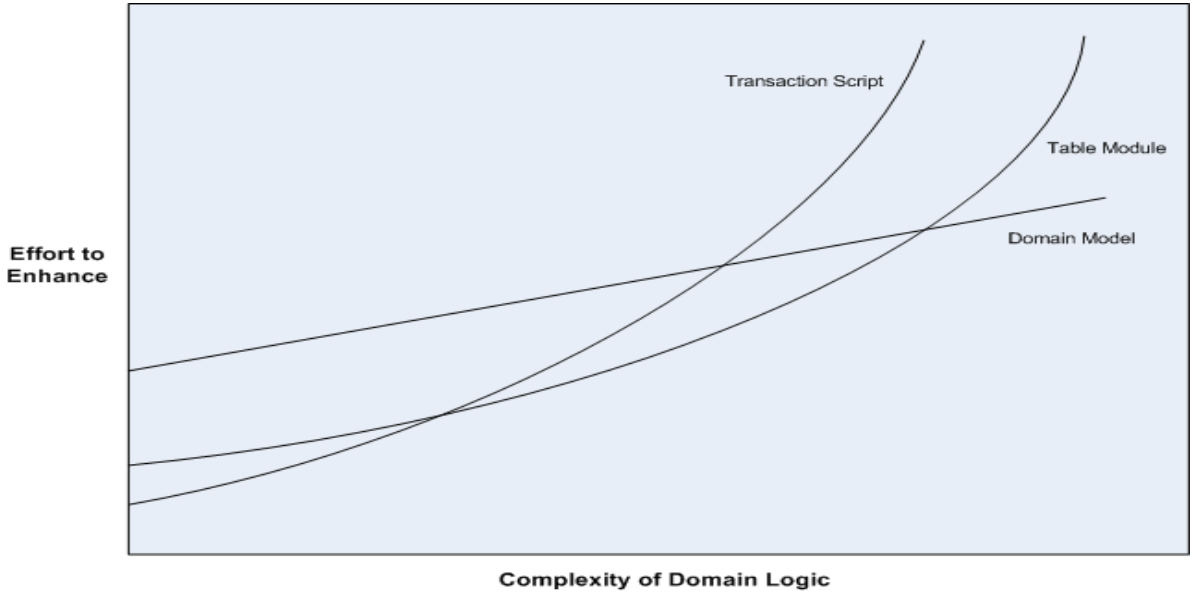
Table Module

- When to use it
  - Useful when application is centered on data
  - Objects in the model are similar to the database



Making a Choice

- Pattern depends on
  - Complexity of the domain logic and
  - How the domain maps to the database



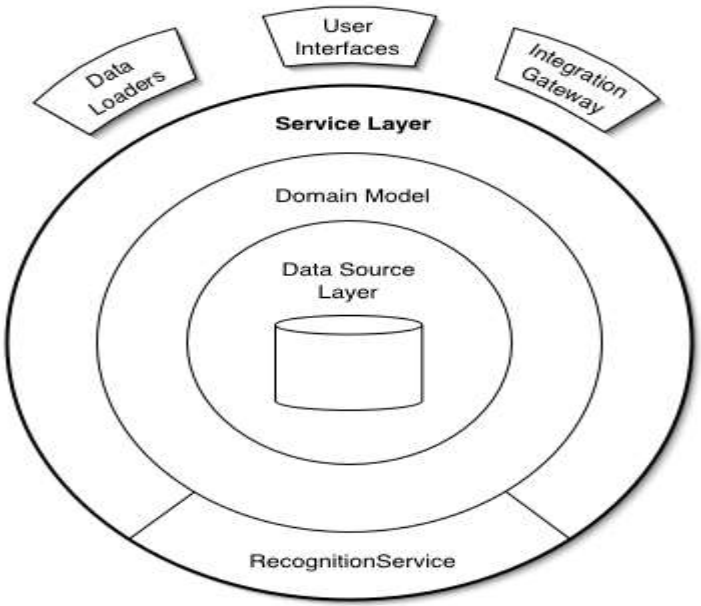
Service Layer

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

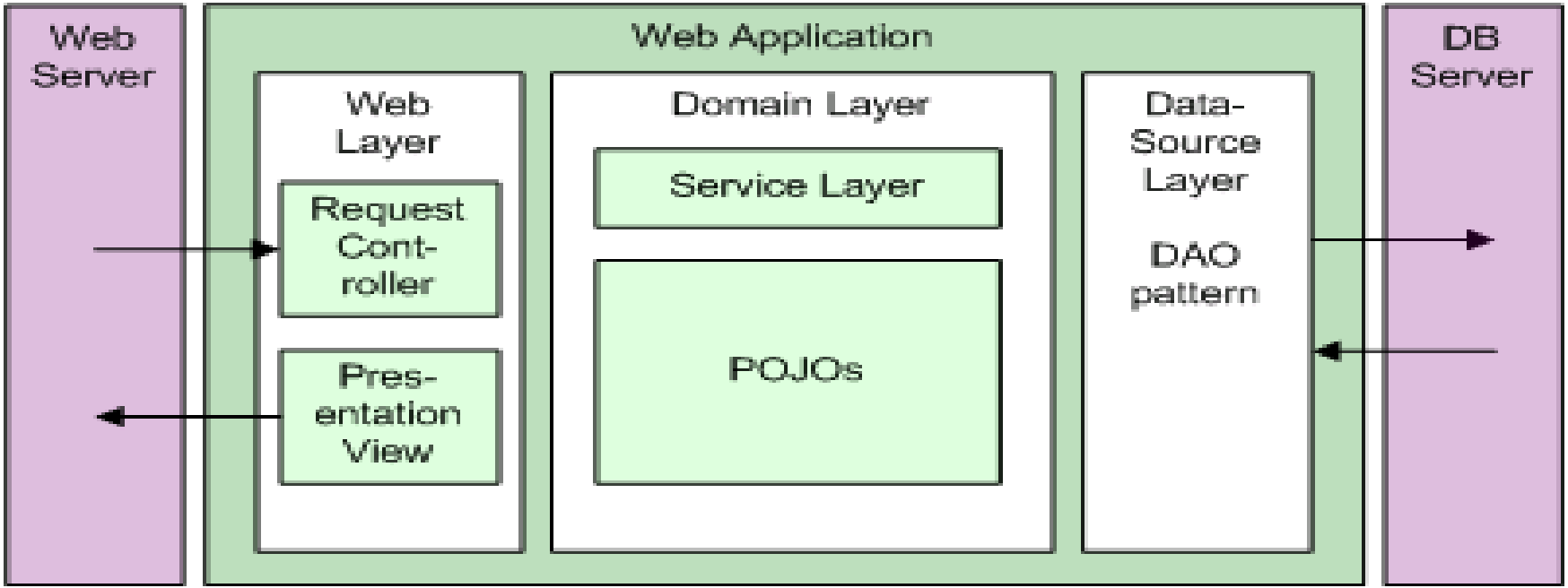
- Defines an application's boundary
  - Provides a set of available operations from the perspective of interfacing client layers
  - Encapsulates the application's business logic

Service Layer

- Domain façade
  - Provides thin interface to the Domain Model
  - Does not implement any business logic
- Operation Script
  - Implementation of application logic
  - Use the Domain Model for domain logic



Design with Service Layer



EXERCISE

I want to structure my domain layer and use a pattern that applies to each situation:

A) The logic is rather extensive

B) The logic is simple and procedure based

C) The logic is moderate and is centered around the database

From Problem to Pattern

How do I structure my domain layer?

A) The logic is rather extensive

Domain Model

Domain Model

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

- Rules and logic describe many different cases and slants of behaviour
- Web of interconnected objects
  - Where each object represents some meaningful entity
  - Dependencies between objects
- How it works
  - Object that represent data (value objects) and business rules (behaviour)

From Problem to Pattern

How do I structure my domain layer?
B) The logic is simple and procedure based

Transaction Script

Transaction Scripts

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

- Most business applications can be thought of as a series of transactions
  - A Transaction Script organizes all this logic primarily as a single procedure

```
recognizedRevenue(contractNumber: long, asOf: Date) : Money
calculateRevenueRecognitions(contractNumber long) : void
```

From Problem to Pattern

How do I structure my domain layer?

C) The logic is moderate and is centered around the database

Table Module

Table Module

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

- Organizes domain logic with one class per table in the database
  - Single instance of the class contains various procedures that will act on the data
  - One object per table handle
- How it works
  - One class that provides domain logic to table or view in database
  - A Domain Model will have one order object per order while a Table Module will have one object to handle all orders

From Problem to Pattern

How do I give my domain logic distinct API?

Service Layer

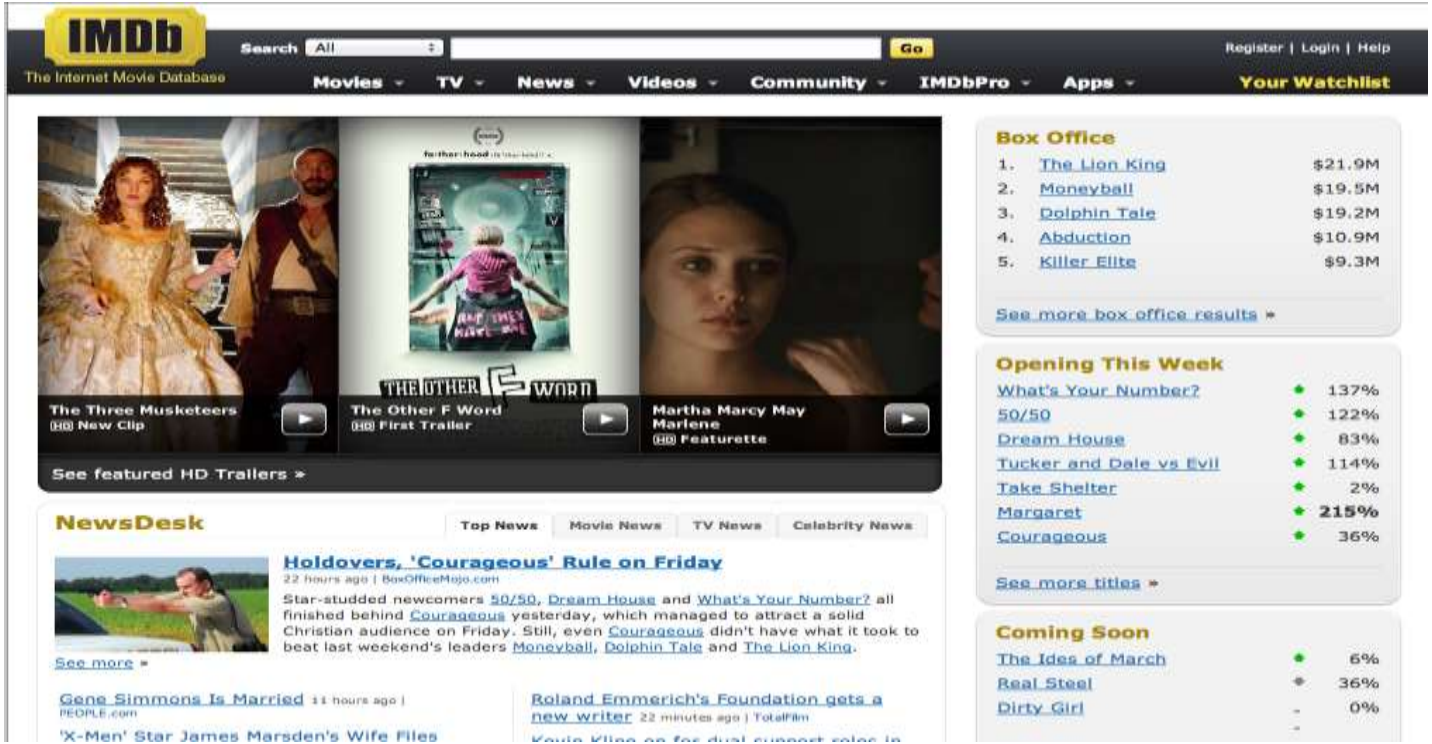
Service Layer

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

- Defines an application's boundary
  - Provides a set of available operations from the perspective of interfacing client layers
  - Encapsulates the application's business logic

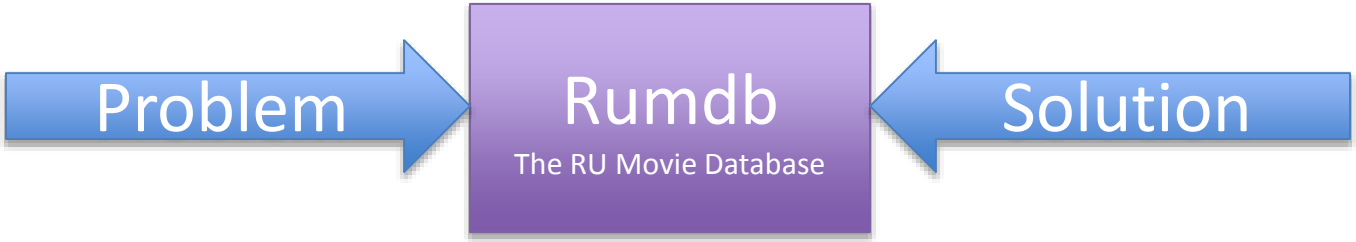
From Problem to Patterns

Problem – Ru Movie DB (rumdb)



From Problem to Pattern

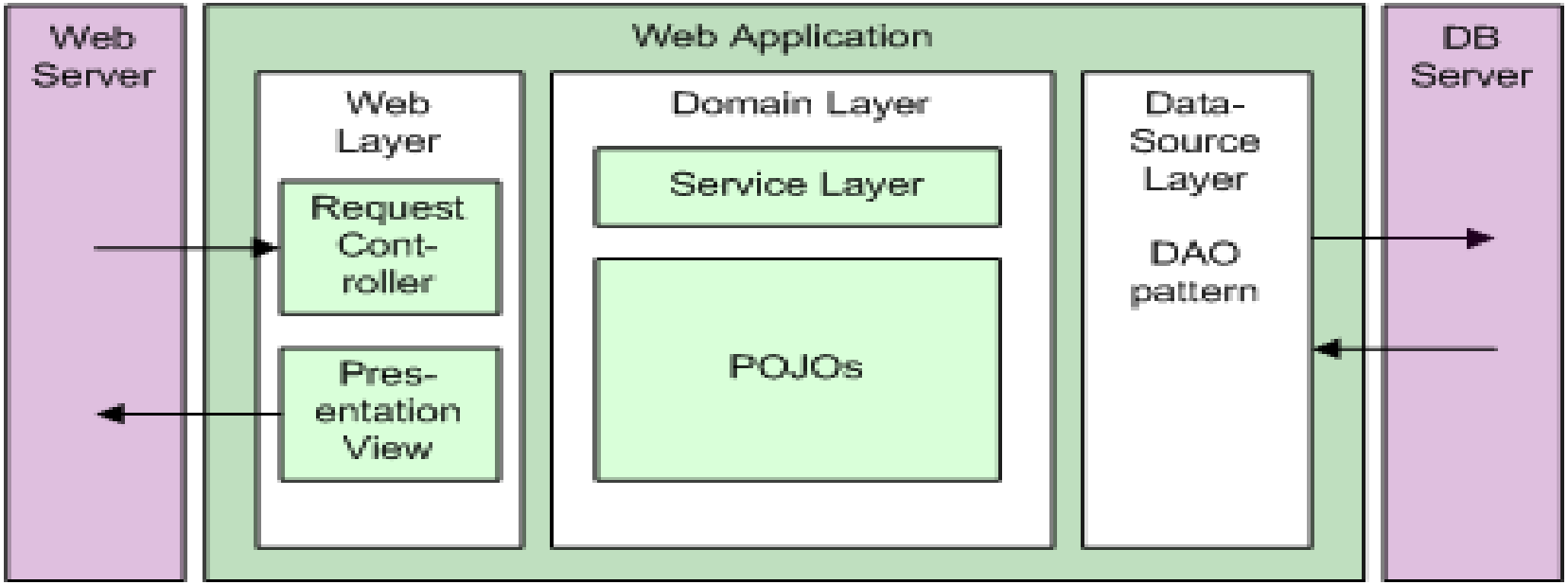
- We need to build a solution
  - We know the problem – sort of



- To get to the solution
  - What patterns to use?

Solution?

- How to we get to this?



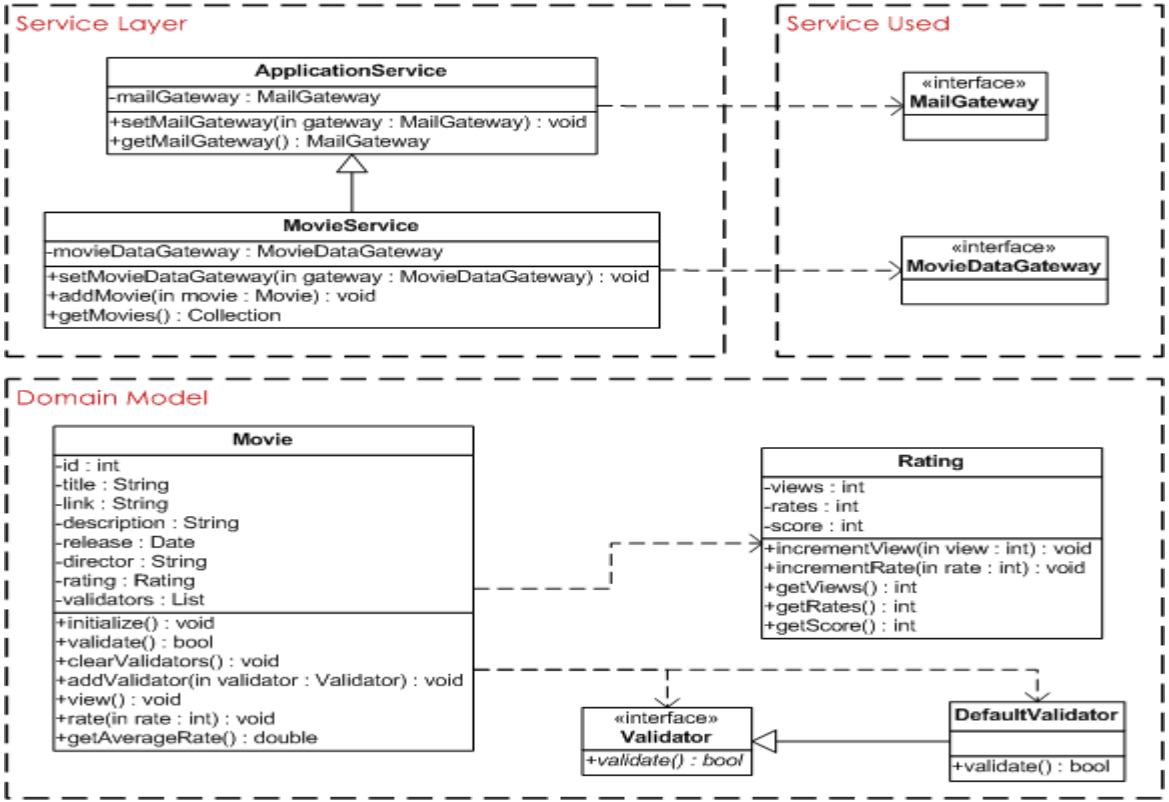
How to define Domain Logic?

- Definition
  - If all environment (noise) is taken way, what is left should be **domain logic**
  - Environment is any middleware (web, data, ejb)
- Domain Logic should be
  - Simple classes and interfaces – POJOs
  - Easy to test!
  - Easy to move around
  - Easy to access
- Limited dependencies
  - Dependencies can be reduced by injection

Building the Domain

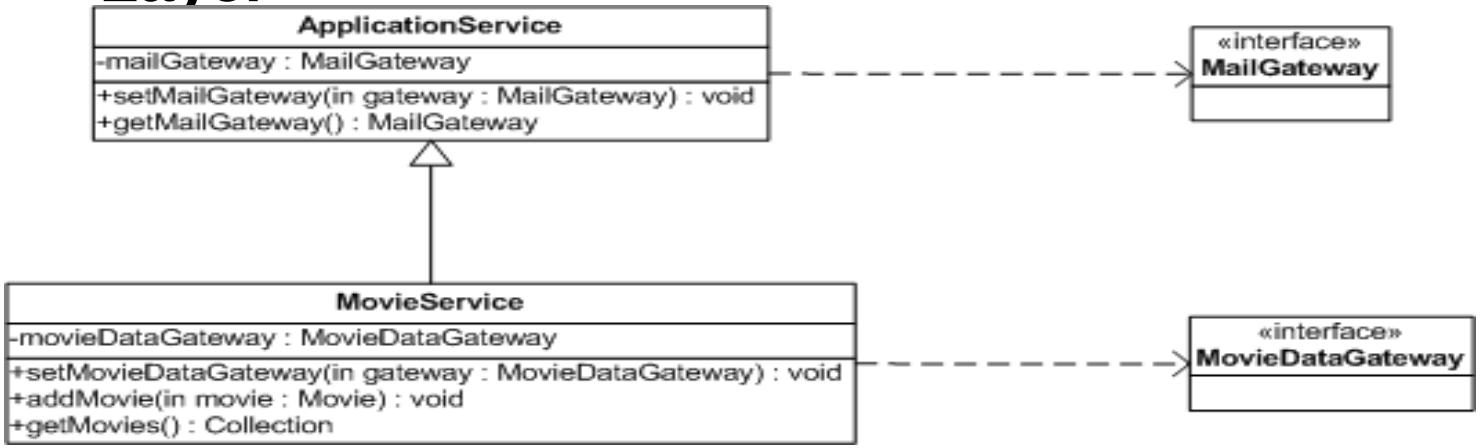
- Objects – the nouns
  - Movie, Ratings, Channel, …
- User Stories
  - What you do with the objects
- Example
  - As a Operator, I want to add new movie
  - As a User, I want to get all movies for a specific channel
  - As a User, I want to rate specific movie

Movie Database Example



Movie Database Example

- The Service Layer
  - **MovieService** Provides simple and clean interface for manipulation of **Movies**
  - External services can be *injected* into the **Service Layer**



Movie Database Example

- **MovieService**
  - Provides an interface to manipulate **Movies**

```
public class MovieService extends ApplicationService
{
  private MovieDataGateway movieDataGateway;

  public void addMovie(Movie movie) throws ServiceException
  {
    movie.initialize();
    if(!movie.validate())
    {
      throw new ServiceException("Movie is invalid");
    }
    movieDataGateway.addMovie(movie);
    getMailGateway().SendMessage("netfang@ru.is", "netfang@ru.is",
                                 "New Movie", "New Movie was added!");
  }
  ...
}
```

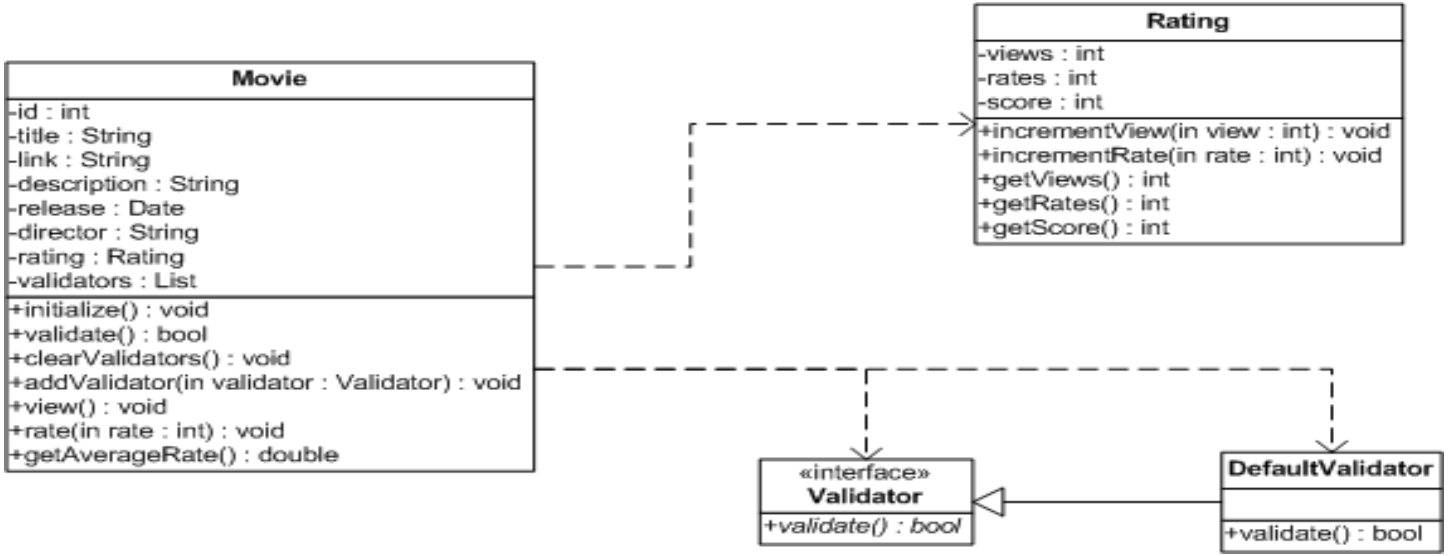
Movie Database Example

- **ApplicationService**
  - **Layered Supertype** for the Service Layer
  - Provides support for all service classes

```
public class ApplicationService
{
  private MailGateway mailGateway;

  public MailGateway getMailGateway()
  {
    return mailGateway;
  }

  public void setMailGateway(MailGateway mailGateway)
  {
    this.mailGateway = mailGateway;
  }
}
```

Movie Database Example

- The Domain Model
  - POJOs with attributes and operations



Movie Database Example

- **Movie**

```
public class Movie implements Comparable
{
  private int id;
  private String title;
  private String link;
  private String description;
  private Date release;
  private String director;
  private Rating rating = new Rating();
  private List<Validator> validators = new ArrayList<Validator>();
```

  - Object for describing and manipulating contents
  - Has a **Rating** object which keeps count of views and rates

Movie Database Example

- Methods in **Movie**

```
public void initialize()
{
  rating.reset();
  clearValidators();
  addValidator(new DefaultMovieValidator(this));
}

public boolean validate()
{
  for(Validator v : validators)
  {
    if(!v.validate())
      return false;
  }
  return true;
}
```

Movie Database Example

- Methods in **Movie**
  - Has *validators* – easily extendable according to the **Open-Close Principle**

```
public void clearValidators()
{
  validators.clear();
}

public void addValidator(Validator validator)
{
  validators.add(validator);
}
```

Movie Database Example

- Methods in **Movie**

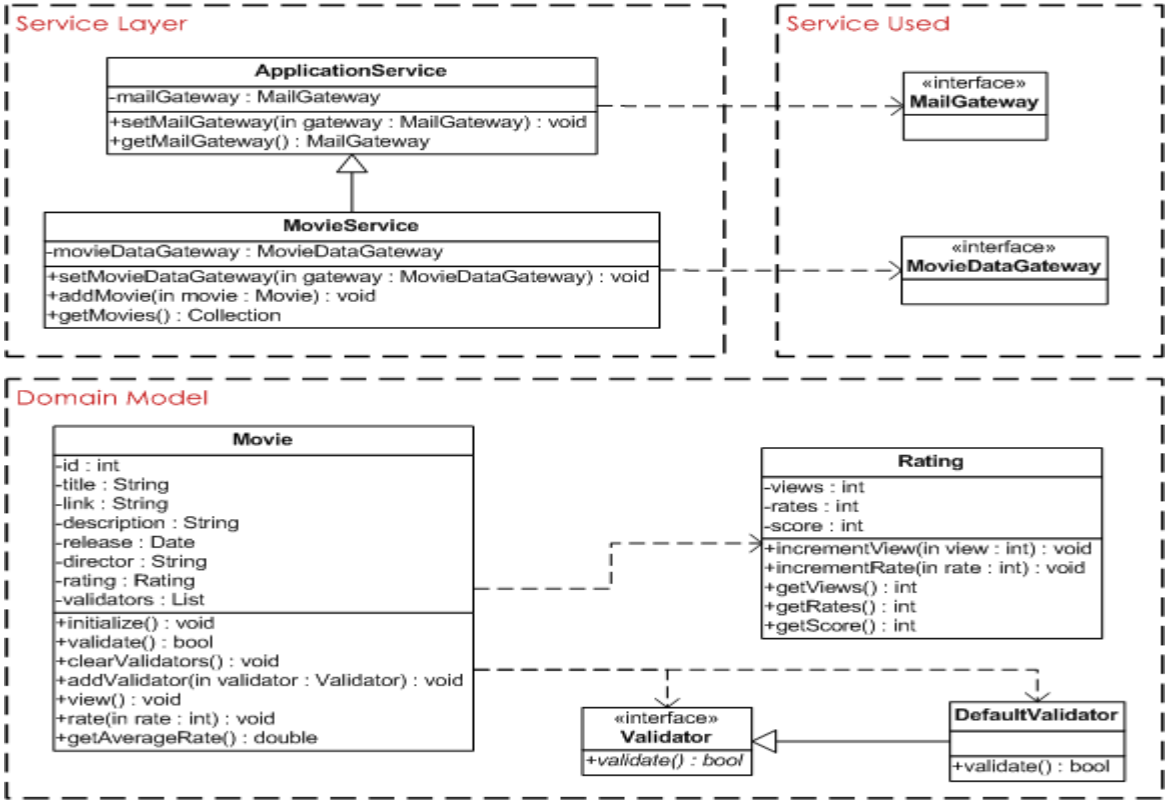
```
public void view()
{
  rating.incrementView();
}

public void rate(int rate)
{
  rating.incrementRate(rate);
}

public double getAverageRate()
{
  return rating.getAverageRate();
}
```

Movie Database Example

- **Rating**

```
public class Rating
{
  private int views;      // Number of requests
  private int rates;      // Number of rates
  private int score;      // Combined values of scores
  public void incrementView()
  {
    views++;
  }
  public void incrementRate(int rate)
  {
    rates++;
    score += rate;
  }
  public double getAverageRate()
  {
    return score/rates;
  }
```

Movie Database Example



Movie Database Example

- TestSimple

```
public TestSimple()
{
  MovieService movieService = new MovieService();
  movieService.setMovieDataGateway(new MovieDataGatewayStub());
  movieService.setMailGateway(new MailServerStub());
  try {
    movieService.addMovie(new Movie(1, "Movie 1”, "http1", "", new Date(), ""));
    movieService.addMovie(new Movie(1, "Movie 2", "http2", "", new Date(), ""));
    movieService.addMovie(new Movie(1, "Movie 3", "http3", "", new Date(), ""));
    movieService.addMovie(new Movie(1, "Movie 4", "http4", "", new Date(), ""));
  }
  catch (ServiceException sex) { ... }
  List<Movie> list = movieService.getMovies();
  for (Movie movie : list)
  {
    System.out.println(movie);
  }
```

Summary

- Layering
- Three Principal Layers
- Domain layer Patterns to choose from
  - **Transaction Script** for simple business logic
  - **Domain Model** for complex business logic
  - **Table Module** for moderate business logic when you have good tools around Record Set
  - **Service Layer** to provide API
- Building the domain
- From Problem to Pattern