Web Application Design

Chapter 17

Randy Connolly and Ricardo Hoar

Chapter 17

1. Real-World Web Software Design
2. Principle of Layering
3. Software Design Patterns
4. Data and Domain Patterns
5. Presentation Patterns
6. Testing
7. Summary

Chapter 17

1. Real-World Web Software Design
2. Principle of Layering
3. Software Design Patterns
4. Data and Domain Patterns
5. Presentation Patterns
6. Testing
7. Summary

Real-World Web Software Design

Challenges in Designing Web Applications

It is quite possible to create complex web applications with little to no class design. The **page-oriented development approach** is such that each page contains most of the programming code it needs to perform its operations.

- rapidly thought-out systems are rarely able to handle unforeseen changes in an elegant way.
- a well-designed application infrastructure up front can make your web application easier to modify and maintain, easier to grow and expand in functionality, less prone to bugs, and thus, ultimately, in the long run easier to create

Chapter 17

1 Real-World Web Software Design

2 Principle of Layering

3 Software Design Patterns

4 Data and Domain Patterns

5 Presentation Patterns

6 Testing

7 Summary

Principle of Layering

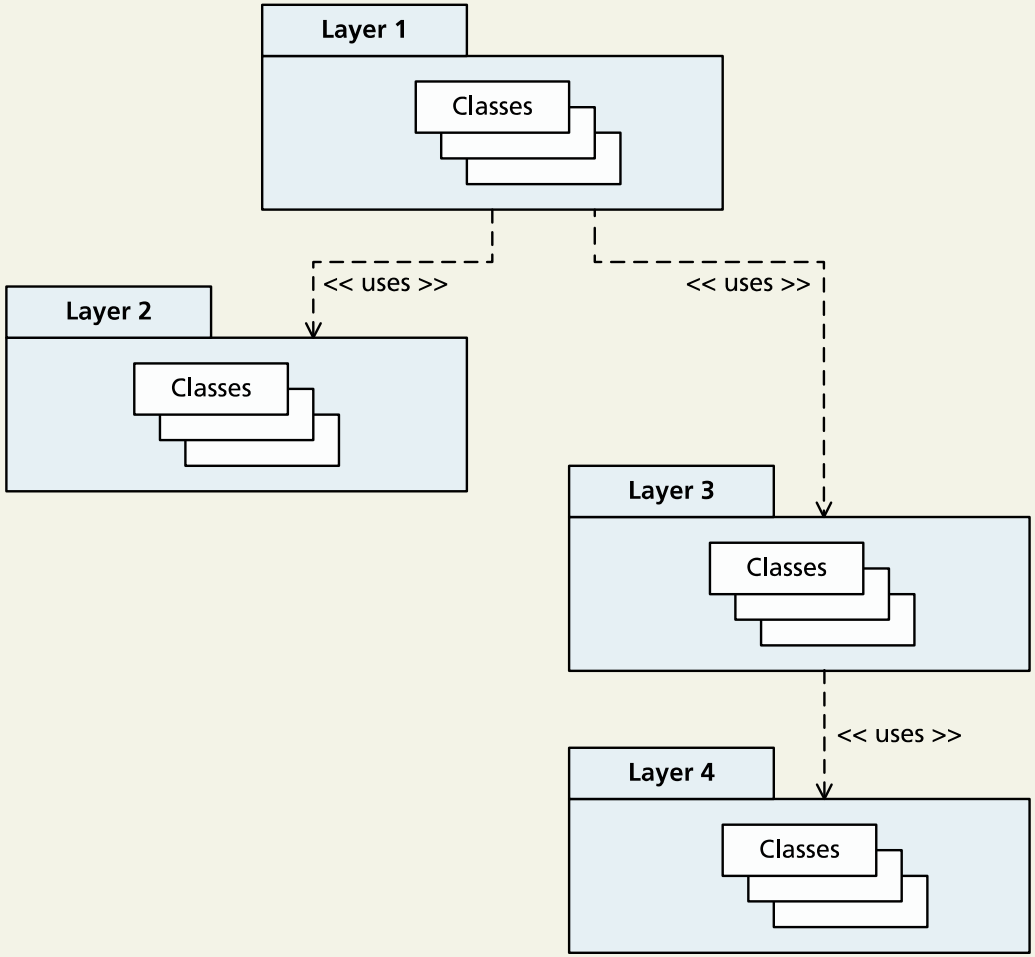
What Is a Layer?

A **layer** , in the context of application development, is simply a group of classes that are functionally or logically related;

- each layer in an application should demonstrate **cohesion**
- distribute the functionality of your software among classes so that **coupling** is minimized.
- A dependency is a relationship between two elements where a change in one affects the other.

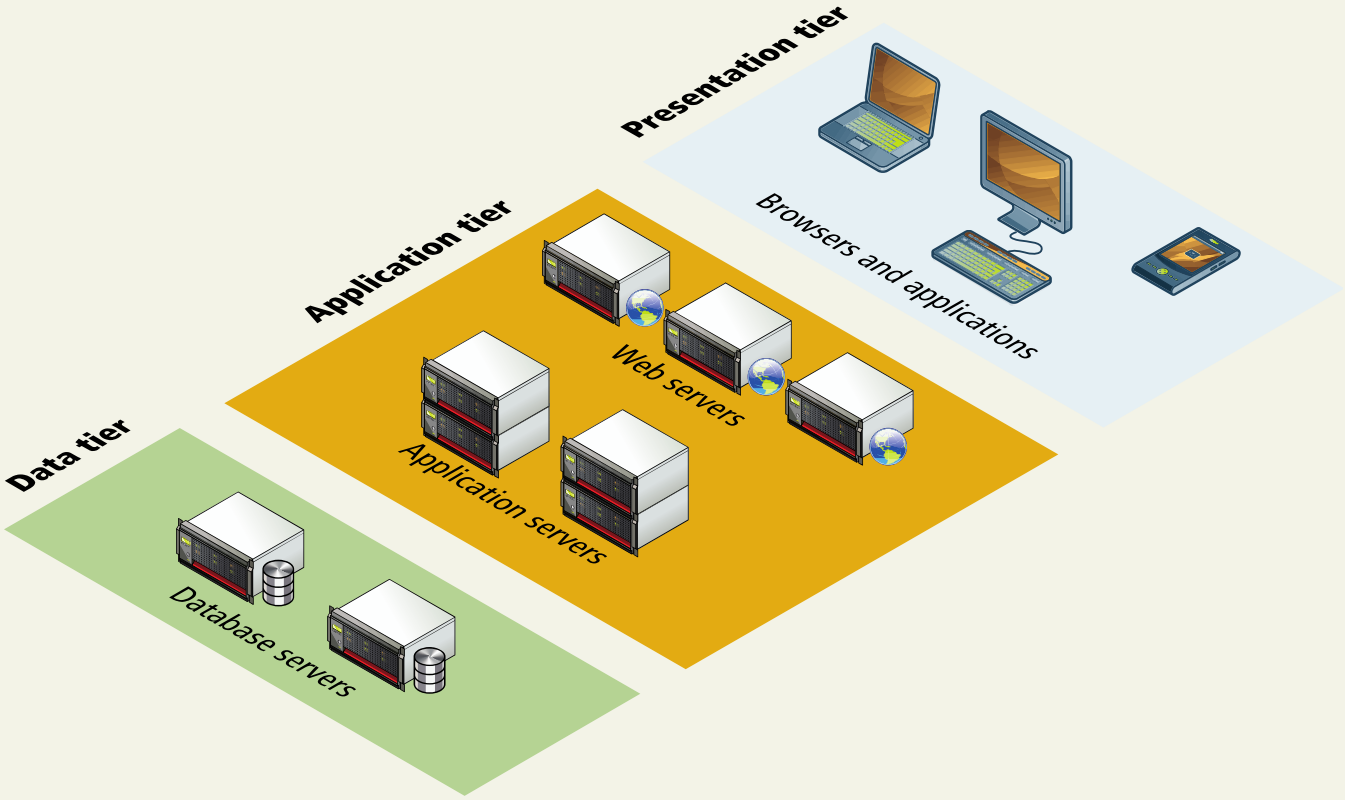
Principle of Layering

What Is a Layer?



Principle of Layering

Visualizing Tiers



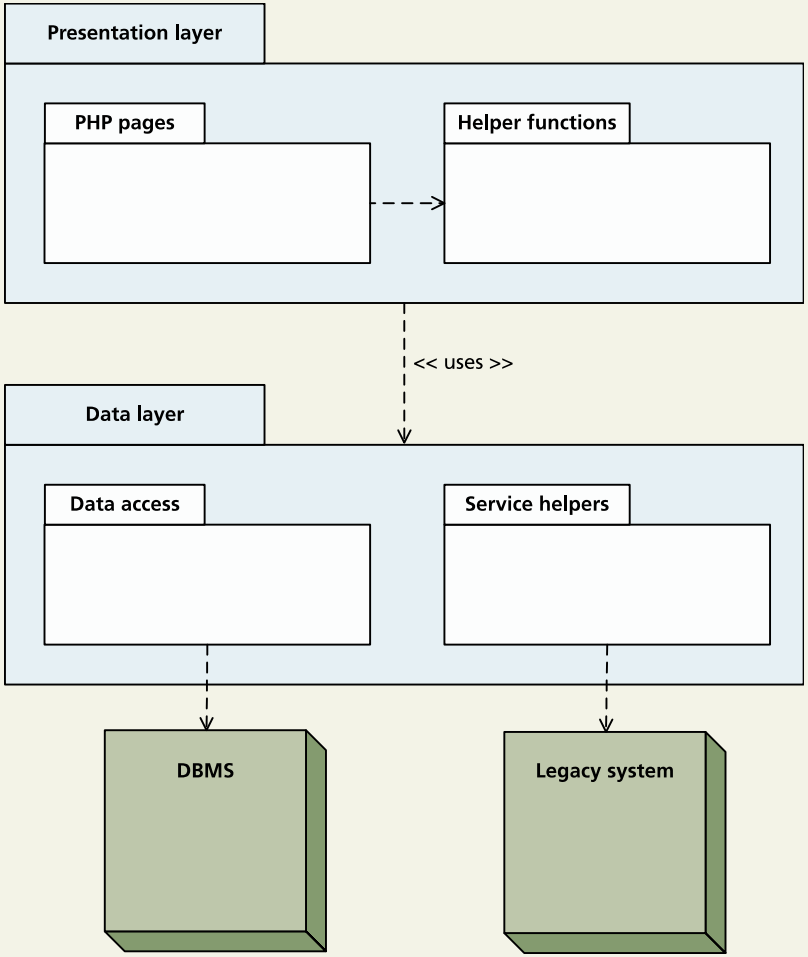
Principle of Layering

Common Layering Schemes

- **Presentation** Principally concerned with the display of information to the user, as well as interacting with the user.
- **Domain/Business** The main logic of the application. Some developers call this the business layer since it is modeling the rules and processes of the business for which the application is being written.
- **Data Access** Communicates with the data sources used by the application. Often a database, but could be web services, text files, or email systems.

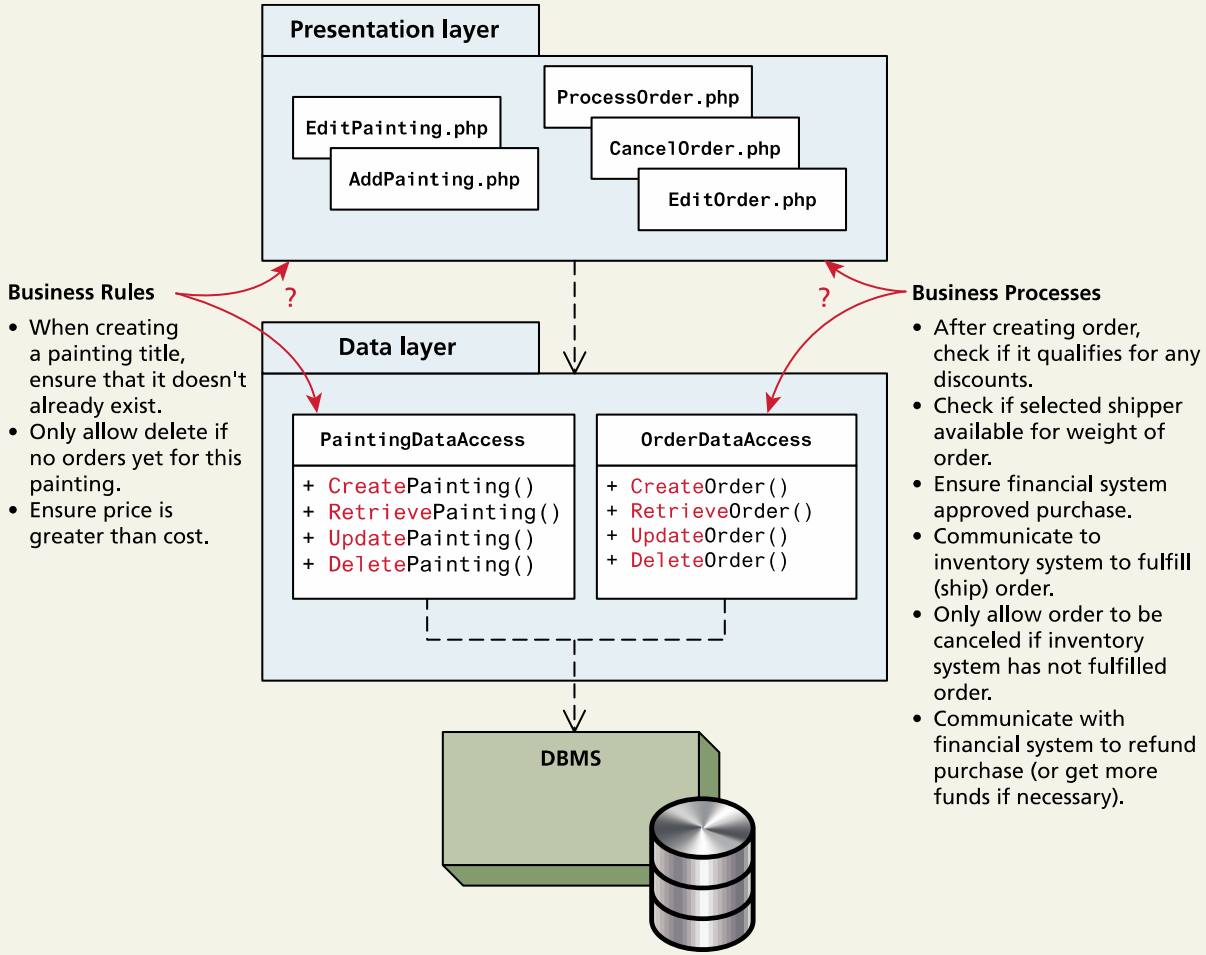
Principle of Layering

Two Layer Model



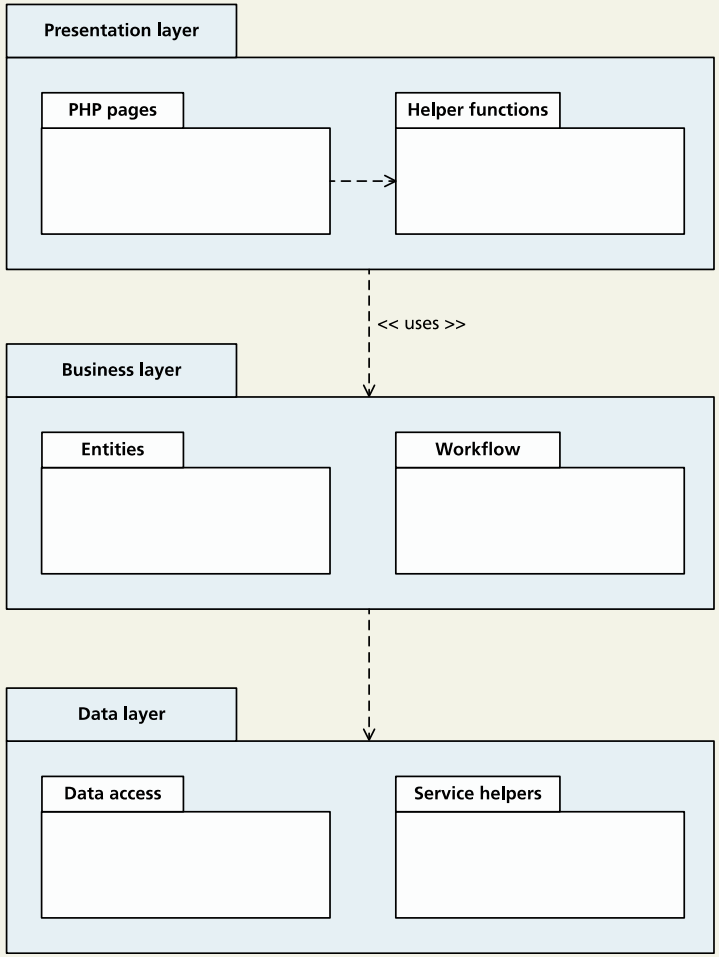
Principle of Layering

Business Rules and Processes



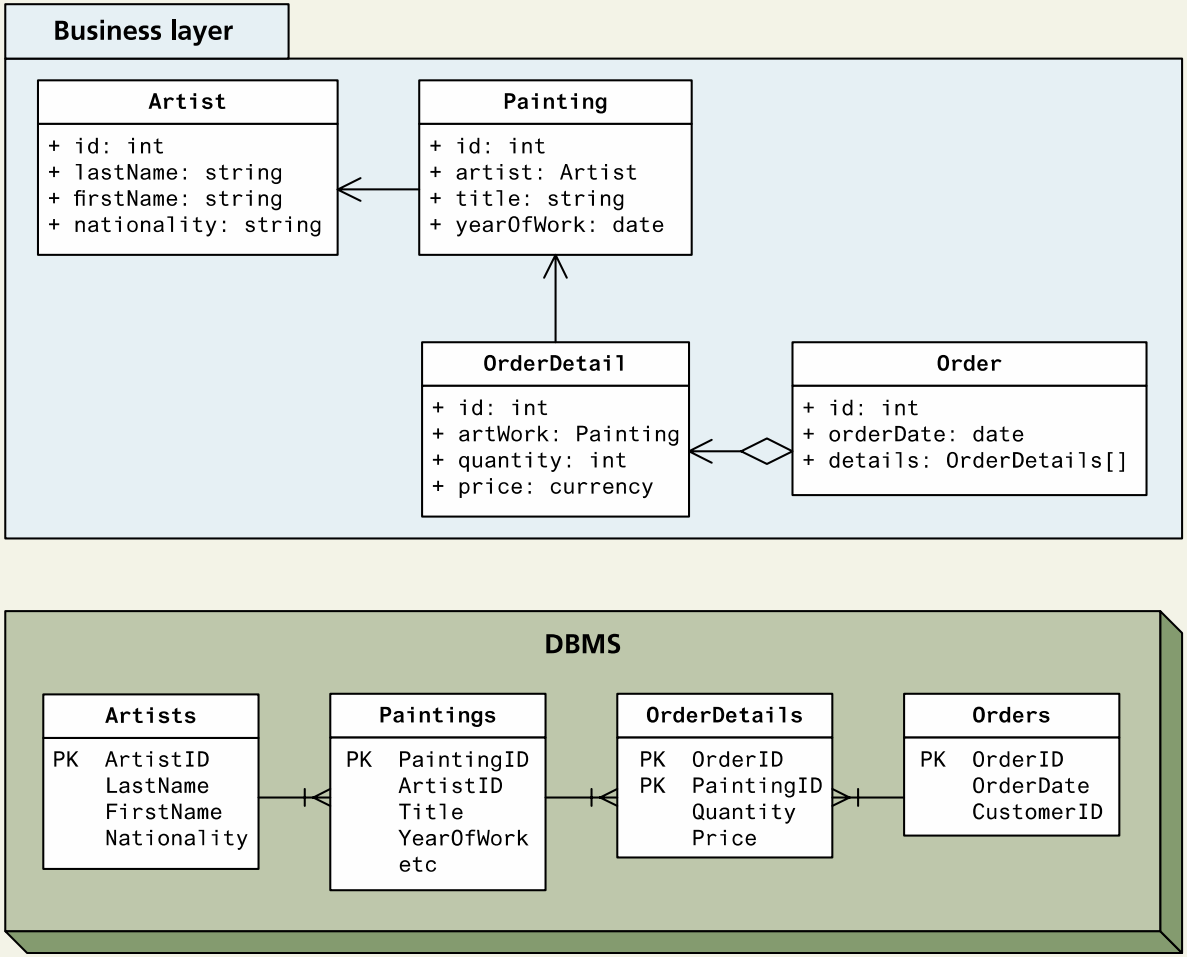
Principle of Layering

3-layer model



Principle of Layering

Simple mapping of tables to domain objects



Chapter 17

1 Real-World Web Software Design

2 Principle of Layering

3 Software Design Patterns

4 Data and Domain Patterns

5 Presentation Patterns

6 Testing

7 Summary

Software Design Patterns in the Web Context

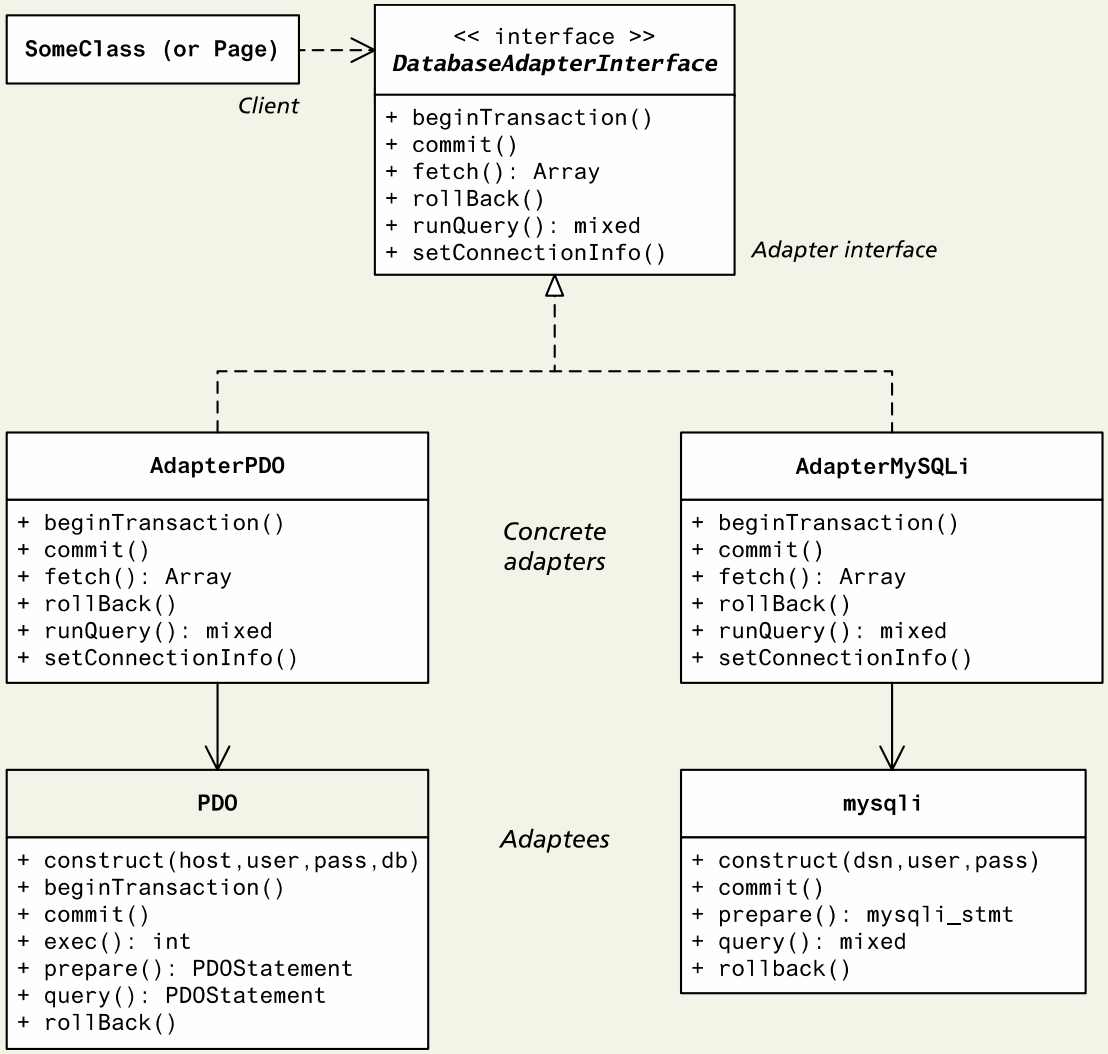
Adapter Pattern

The **Adapter pattern** is used to convert the interface of a set of classes that we need to use to a different but preferred interface.

- frequently used in web projects as a way to make use of a database API (such as PDO or mysqli) without coupling the pages over and over to that database API.

Software Design Patterns in the Web Context

A database API adaptor



Software Design Patterns in the Web Context

Simple Factory Pattern

A **factory** is a special class that is responsible for the creation of subclasses (or concrete implementations of an interface), so that clients are not coupled to specific subclasses or implementations.

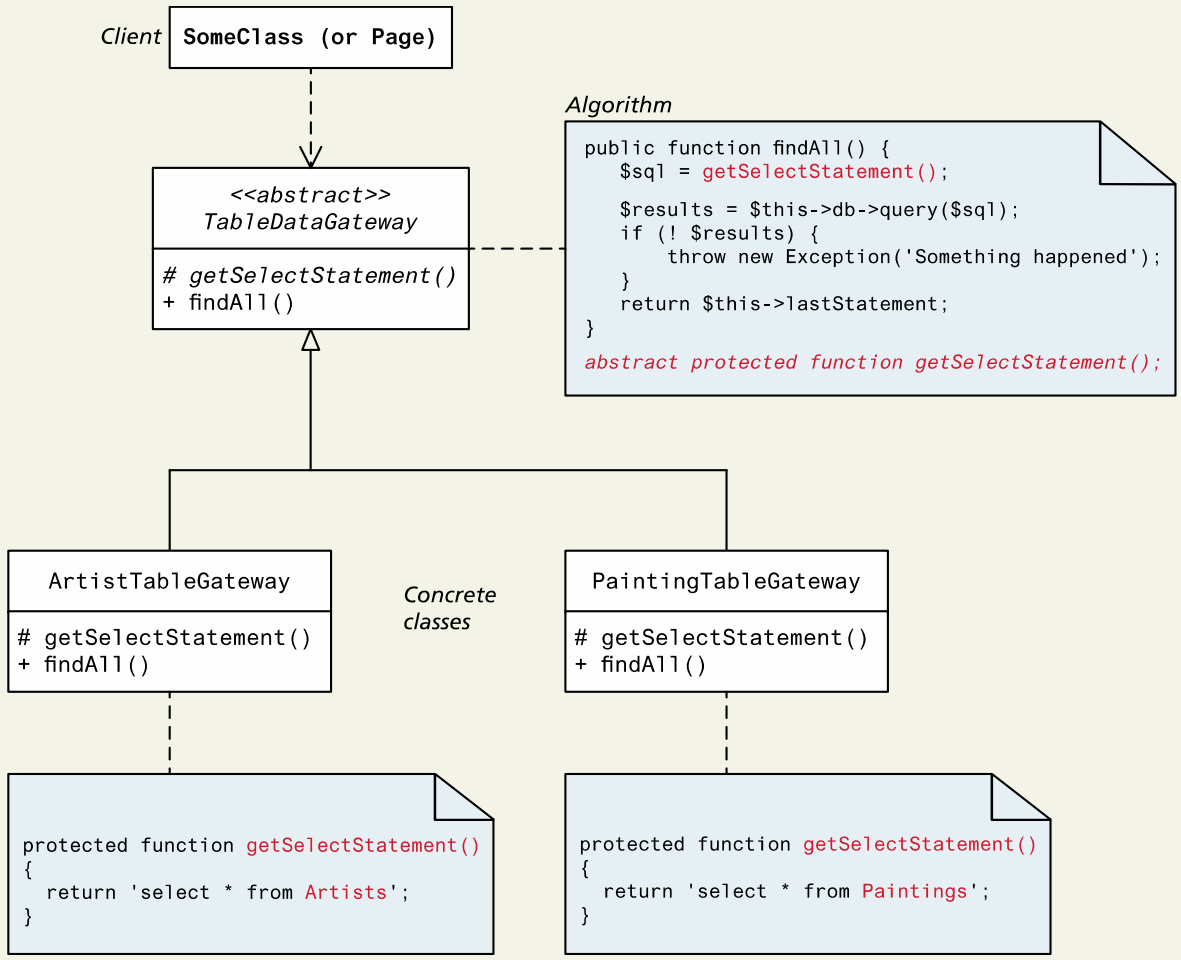
Software Design Patterns in the Web Context

Simple Factory Pattern

```
<?php
class DatabaseAdapterFactory {
        /*
        Notice that this creation method is static. The $type parameter
        Is used to specify which adapter to instantiate
        */
        public static function create($type, $connectionValues) {
                $adapter = "DatabaseAdapter" . $type;
                if ( class_exists($adapter) ) {
                        return new $adapter($connectionValues);
                }
                else {
                        throw new Exception("Data Adapter type does not
exist");
                }
        }
}
?>
```

Software Design Patterns in the Web Context

Template Method Pattern



Software Design Patterns in the Web Context

Dependency Injection

```
abstract class TableDataGateway
{
        protected $dbAdapter;
        public function __construct($dbAdapter){
                if (is_null($dbAdapter) )
                        throw new Exception("Database
adapter is null");
                $this->dbAdapter = $dbAdapter;
        }
...
}
```

Chapter 17

1 Real-World Web Software Design

2 Principle of Layering

3 Software Design Patterns

4 Data and Domain Patterns

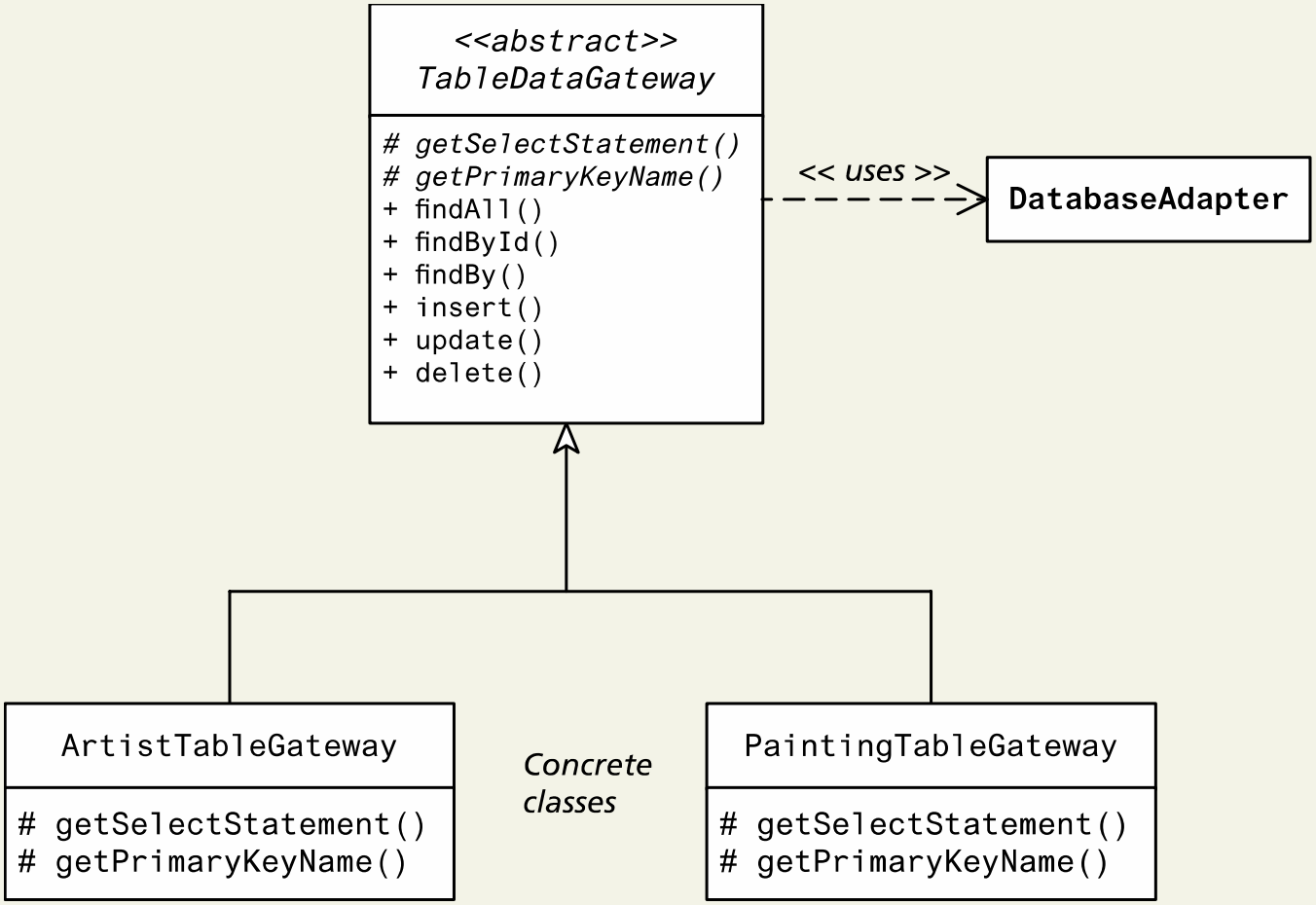
5 Presentation Patterns

6 Testing

7 Summary

Data and Domain Patterns

Table Data Gateway Pattern



Data and Domain Patterns

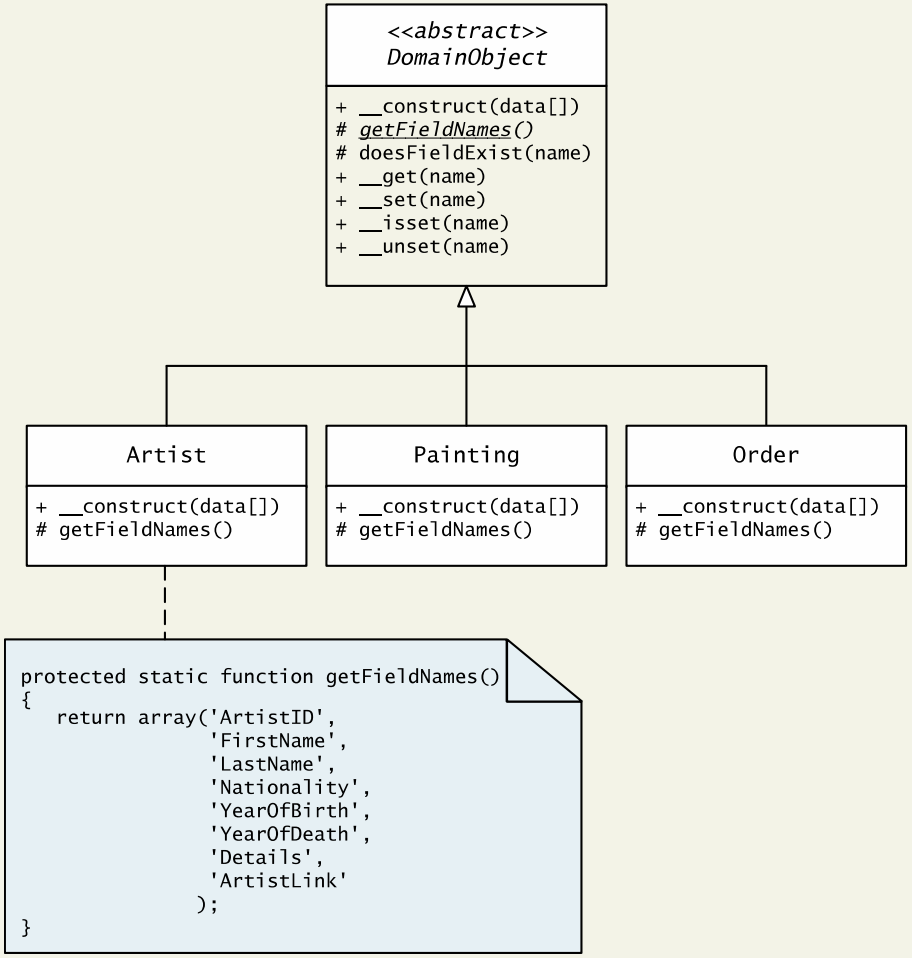
Domain Model Pattern

For programmers who are familiar with object-oriented design, the **Domain Model** pattern is a natural one. In it, the developer implements an object model : that is, a variety of related classes that represent objects in the problem domain of the application.

- Often the domain model will be similar to the database schema

Data and Domain Patterns

Example Domain Model



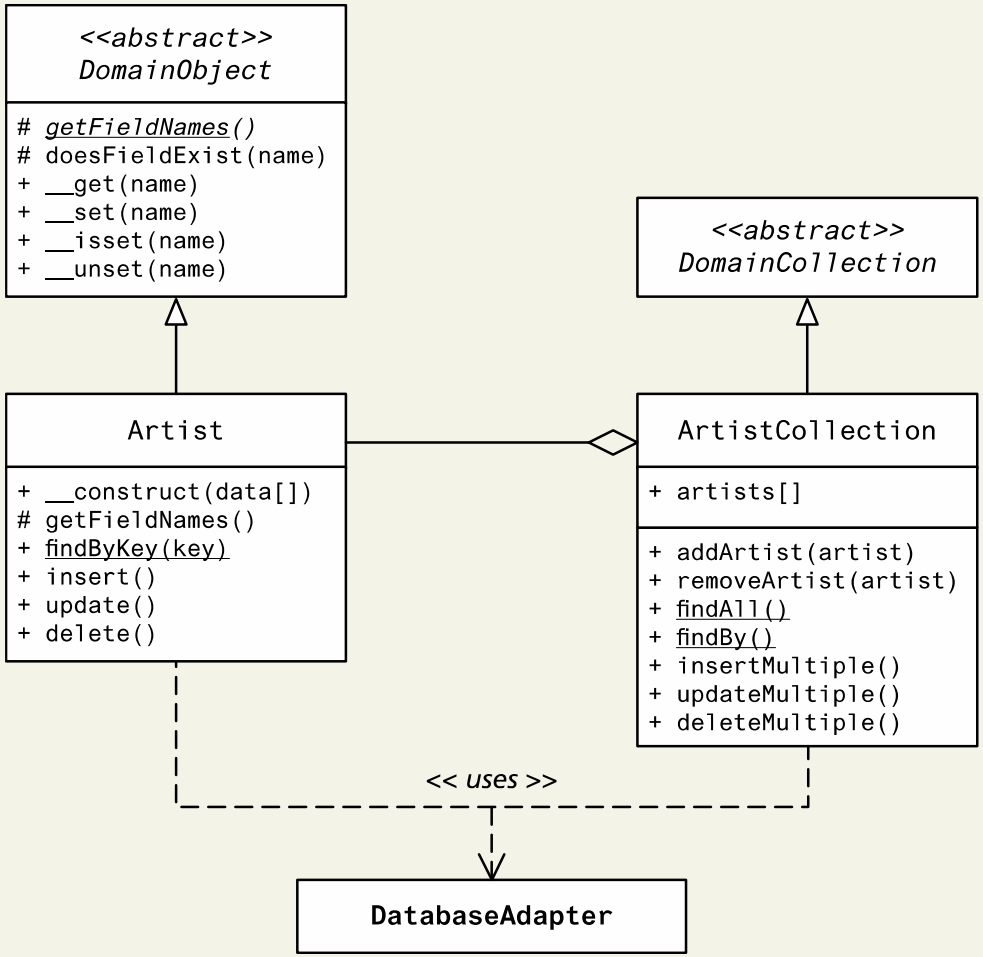
Data and Domain Patterns

Active Record Pattern

- the domain objects have the responsibility for
  - retrieving themselves from the database,
  - updating or inserting the data into the underlying database
- the properties of each class must mirror quite closely the underlying table structure

Data and Domain Patterns

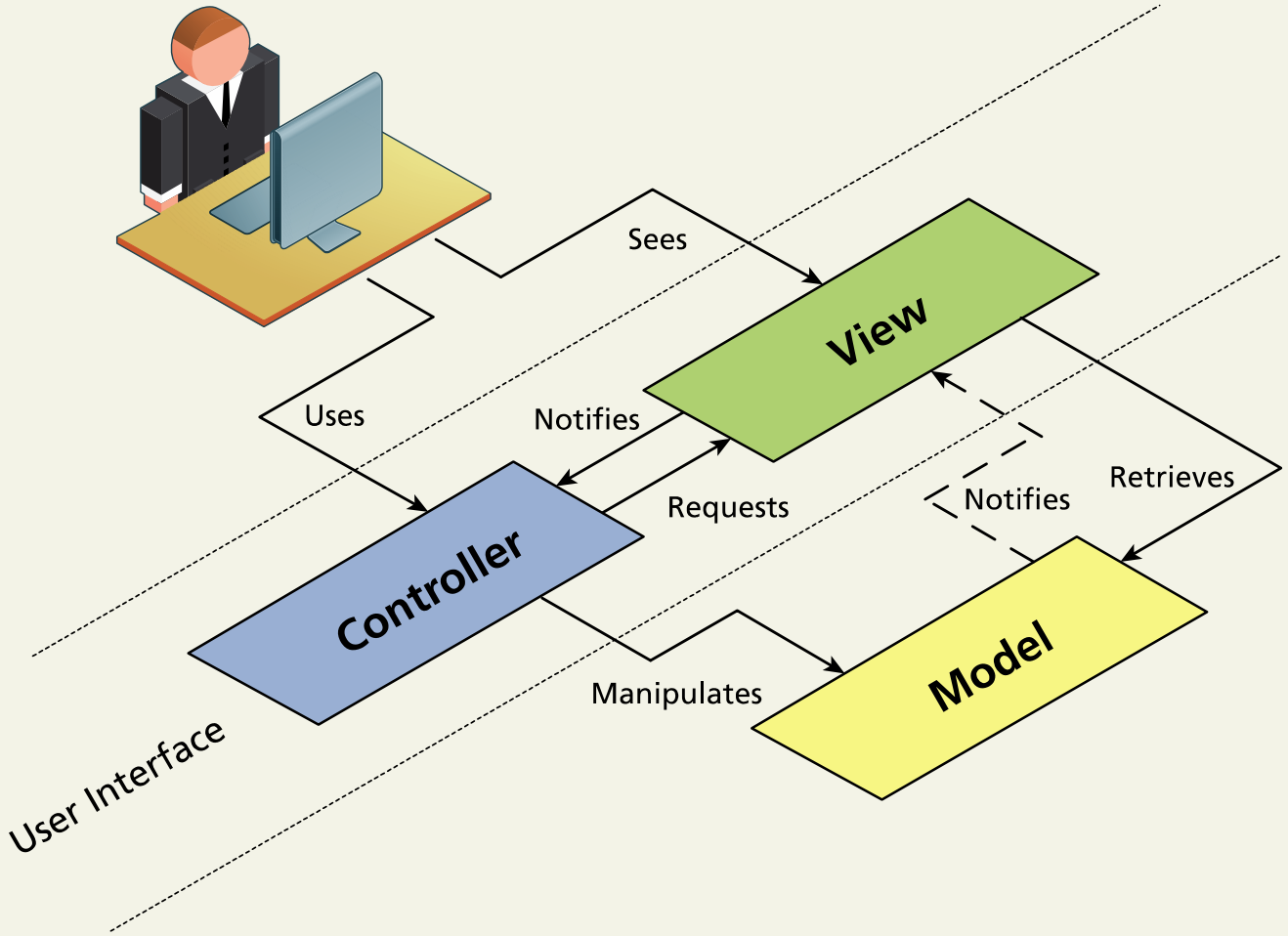
Active Record Pattern



Chapter 17

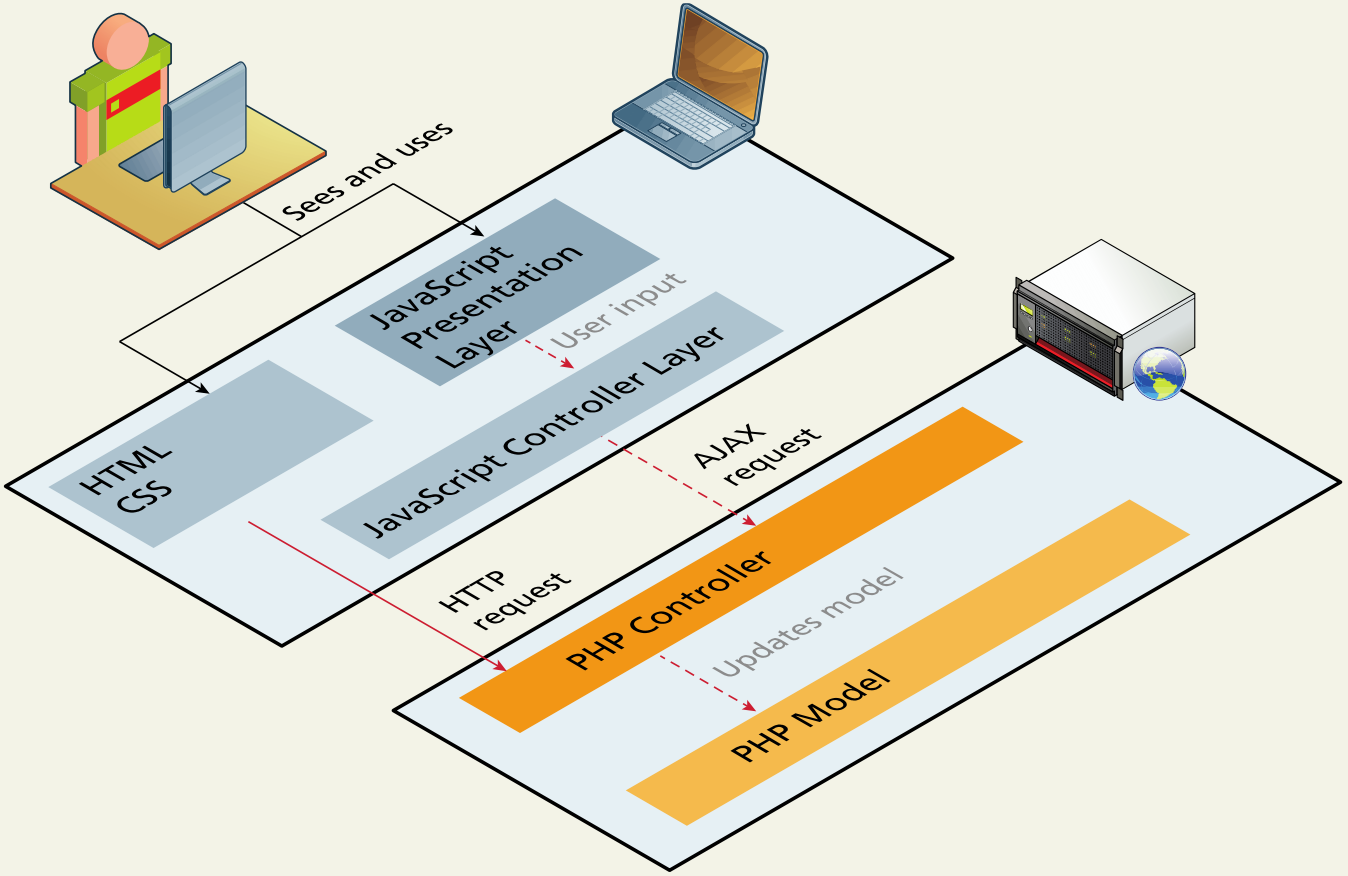
Presentation Patterns

Classic Model-View-Controller (MVC) Pattern



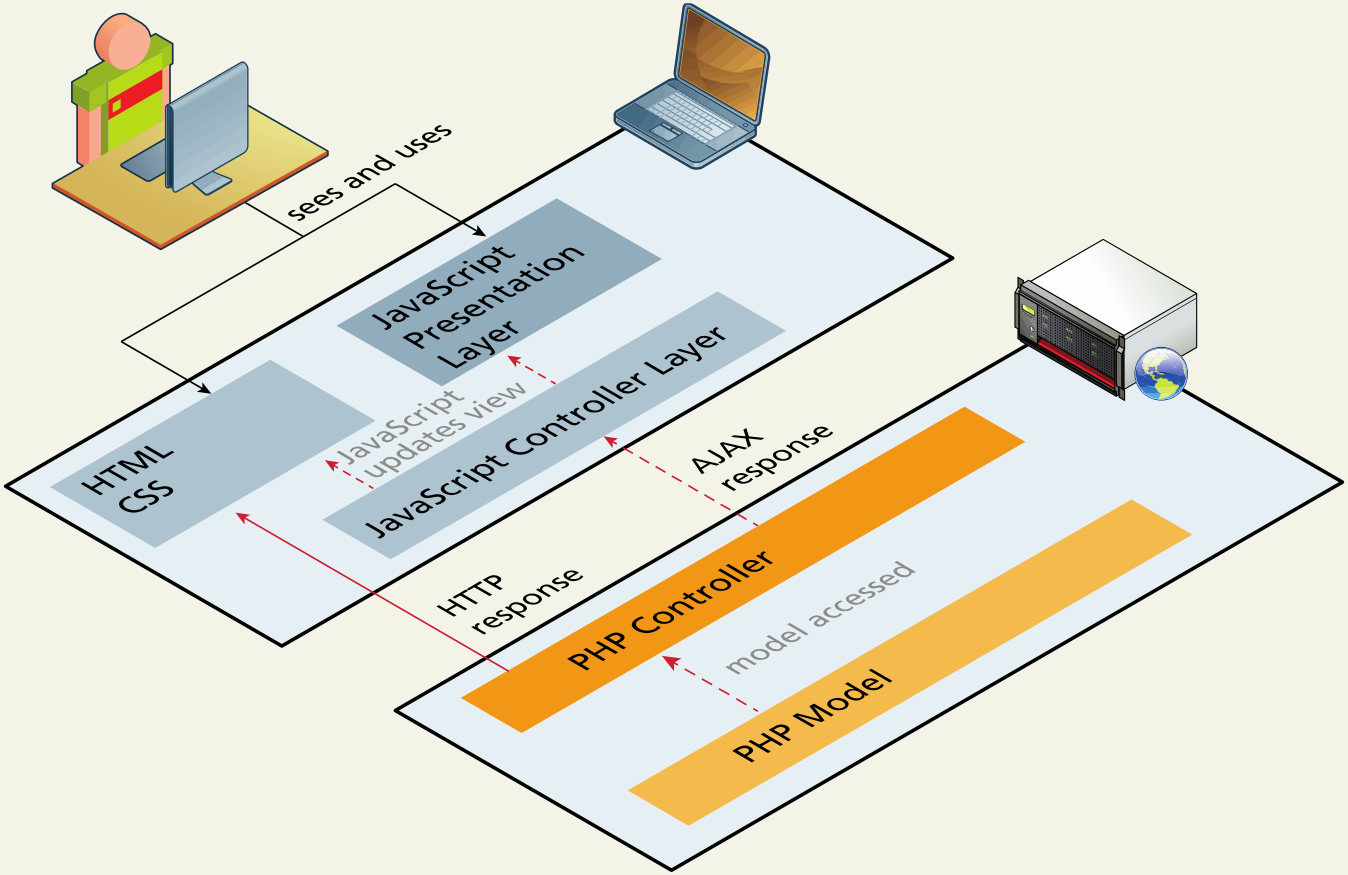
Presentation Patterns

Model-View-Controller (MVC) Pattern Split across Client/server



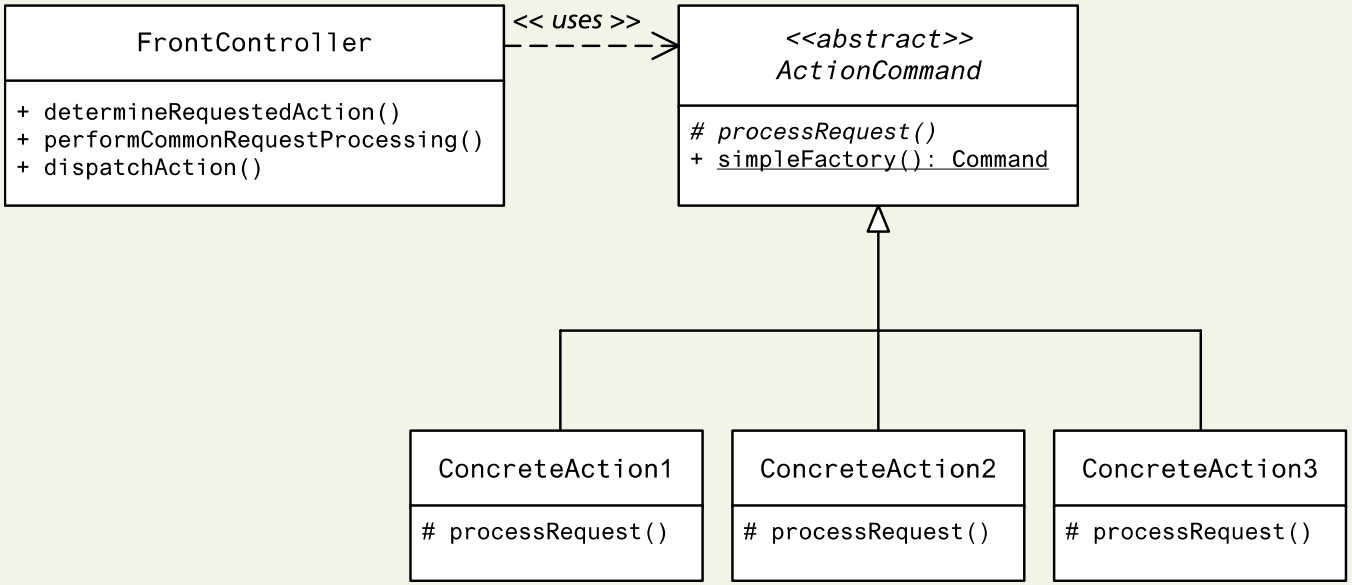
Presentation Patterns

Model-View-Controller (MVC) Pattern Reponses across Client/server



Presentation Patterns

Front Controller Pattern



Chapter 17

1 Real-World Web Software Design

2 Principle of Layering

3 Software Design Patterns

4 Data and Domain Patterns

5 Presentation Patterns

6 Testing

7 Summary

Testing

- **Functional testing** is testing the system's functional requirements.
- **Non-functional testing** refers to a broad category of tests that do not cover the functionality of the application, but instead evaluate quality characteristics such as
  - Usability
  - Security
  - Performance

Summary

Key Terms

Adapter pattern
business layer
business objects
business process
business rule
cohesion
controller
coupling
CRUD
data access objects
Dependency
Dependency Injection pattern
design patterns
domain layer
Domain Model pattern
domain objects
entities
enterprise patterns
functional testing
gateway
layer
model
Model-View-Controller (MVC) pattern
non-functional testing
object model
ORM
page-oriented development approach
Simple Factory pattern
software design
Table Data Gateway pattern
table gateways
Template Method pattern
tier
two-layer model
use cases
variable variables
view

Summary

Questions?