CPET 499/ITC 250 Web Systems

Chapter 14
Web Application Design

Text Book:

*** Fundamentals of Web Development, 2015, by Randy Connolly and Ricardo Hoar, published by Pearson**

Paul I-Hai Lin, Professor of Electrical and Computer Engr. Tech
http://www.etcs.ipfw.edu/~lin

Topics

- About **software design principles** specific to web applications
- How **design patterns** provide modular solutions to common problems
- Key web application design patterns

Topics

- 14.1 Real-World Web Software Design
  - Challenges in Designing Web Applications
- 14.2 Principle of Layering
  - Data layer, Application Layer, and Presentation layer
  - Common Layer Schemes
- 14.3 Software Design Patterns in the Web Context
  - Adapter Pattern, Simple Factory Pattern, Template Method Pattern
- 14.4 Data and Domain Patterns
  - Table Data Gateway Pattern, Domain Model Pattern, Active Record Pattern
- 14.5 Presentation Patterns
  - Model-View-Controller (MVC) Pattern, Front Controller Pattern

14.1 Real-World Web Software Design

- Software Requirements, Software Requirement Specifications, Software Design
- IEEE Recommended Practice for Software Requirements, 2010, http://www.cse.msu.edu/~cse870/IEEEXplore-SRS-template.pdf
- IEEE Draft Standard for Software Design Descriptions, 2005, http://www.san.uri.br/~pbetencourt/engsoftII/IEEE-P1016-d50.PDF
- Software Design Document (SSD) template (summarized from STD 1016), http://www.zeynepaltan.info/SDD-Template.pdf

Principle of Layering

- **Patterns Enterprise Application Architecture, 2003, by Martin Fowler**
  - "Layering is one of the most common techniques that software designers use to break apart a complicated software system."
- What is a Layer?
  - A layer is simply a group of classes that are functionally related
  - A conceptual grouping of classes
  - Each layer in an application should demonstrate "COHESION"
  - Cohesive layers and classes are generally easier to Understand, Reuse, and Maintain

Principle of Layering

- What is a Layer?
  - **Cohesive** layers and classes are generally easier to Understand, Reuse, and Maintain
  - The **goal** of layering is to distribute the functionality of your software among classes so that the Coupling of a given class to other classes is minimized
    - **Coupling** refers to the way in which one class is connected, or coupled, to other classes.
  - In the layered design approach, each class within the layer has a limited number of dependencies
    - A **Dependency** (uses relationship in UML) is a relationship between two elements where a change in one affects the other

Fig. 14.1 Visualizing Layers

UML - uses Relationship, Dependency

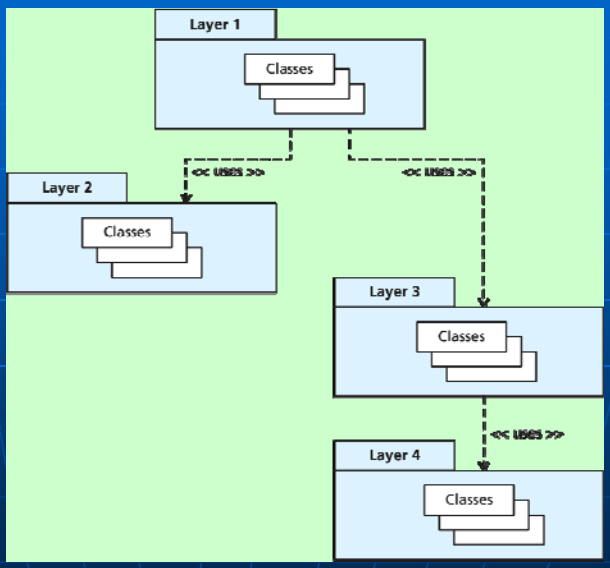


Fig. 14.2 Visualizing Tiers

- Tier – refers to a processing boundary
- Three-Tier Applications Architecture:
  - Presentation tier, Application tier, and Data tier; or
  - Presentation, Domain/Business, Data Access

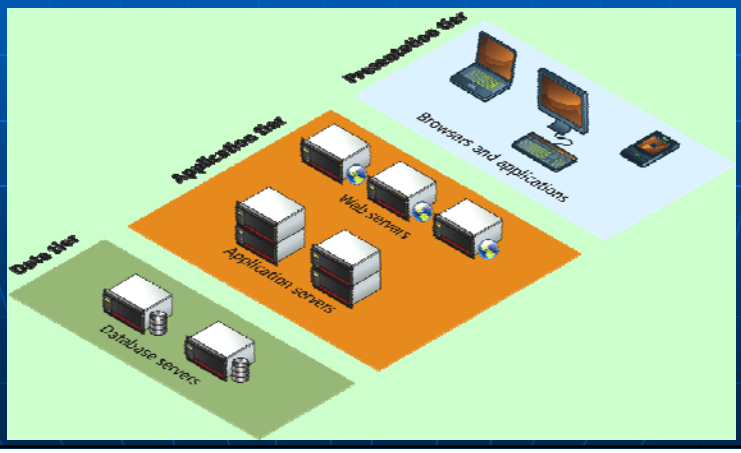
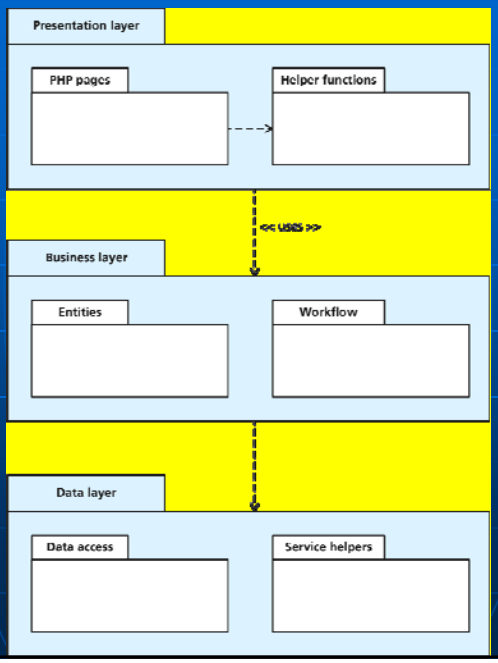


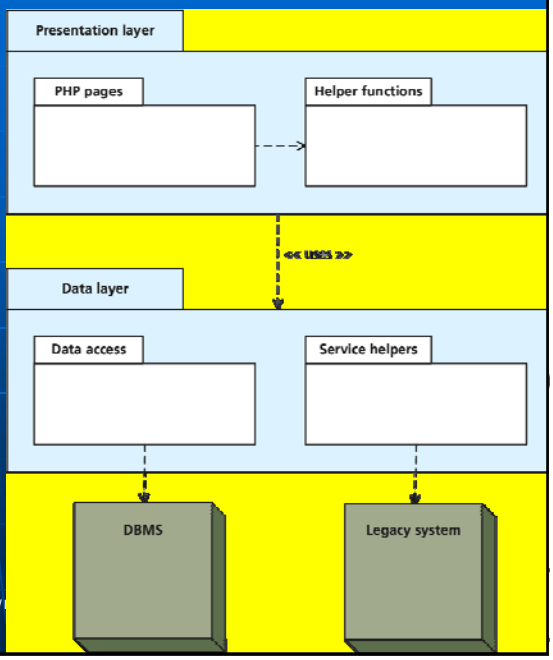
Fig. 14. 5 Three-Layer Model



9

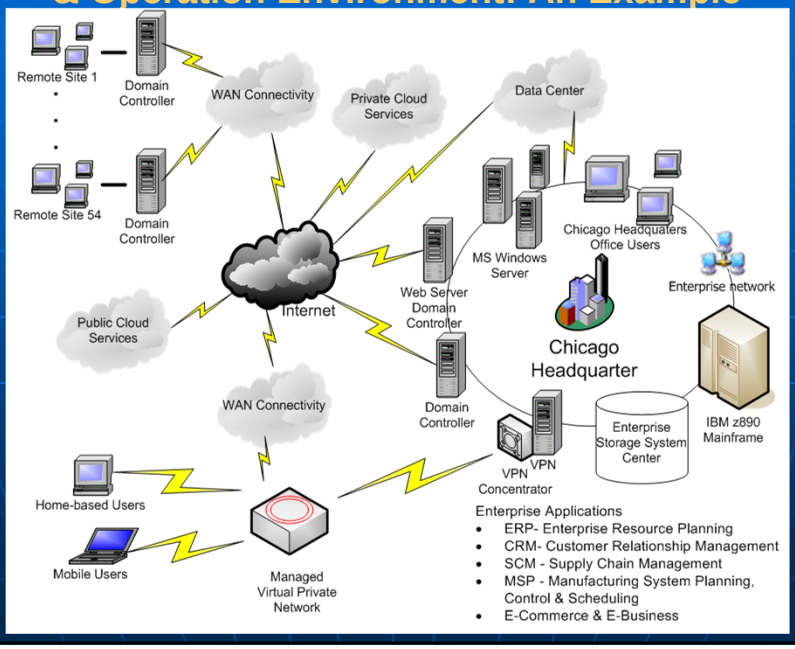
Fig. 14.3 Two-Layer Model

- Data access are contained in a set of Classes called a data Access layer
- Presentation layer interacts directly with the classes in this layer



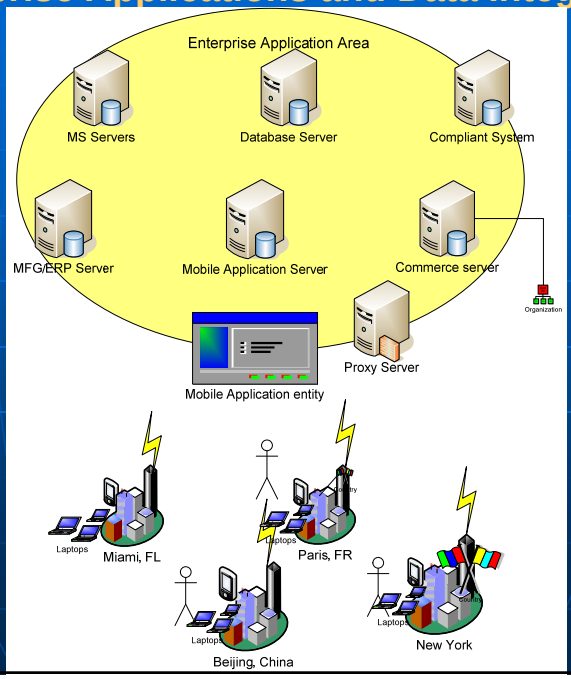
CPET 499/

Internet- and Web-enabled Enterprise IT Services & Operation Environment: An Example



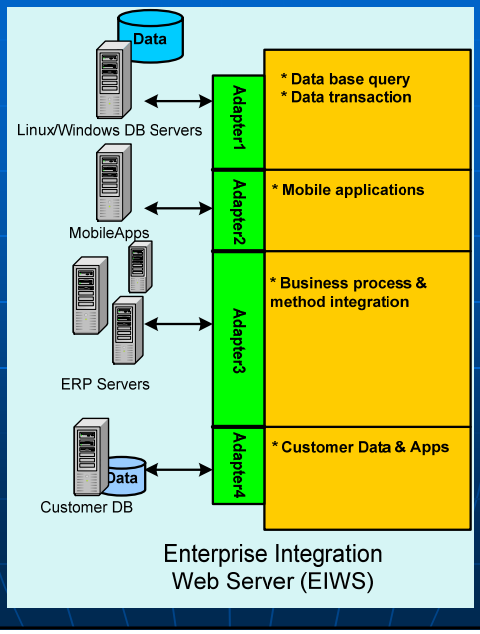
11

Enterprise Applications and Data Integration



12

Web Servers for Enterprise Applications and Data Integration



13

Business Rules and Processes

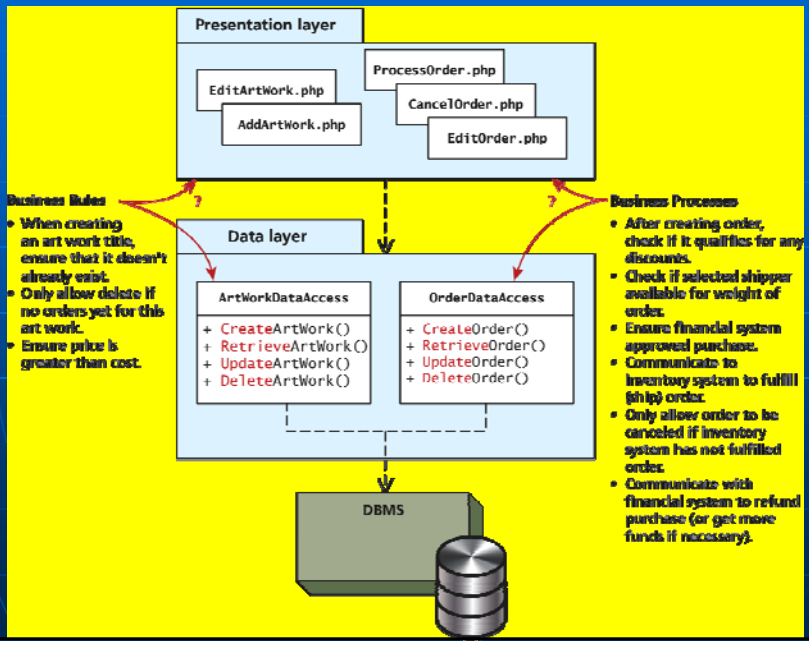


Fig. 14.6 Simple Mapping of Tables to Domain Objects

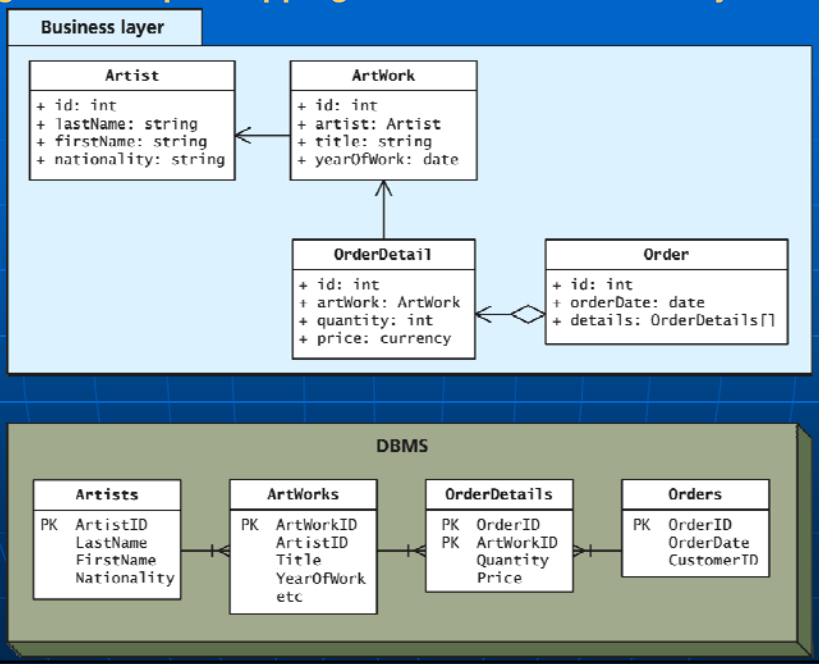
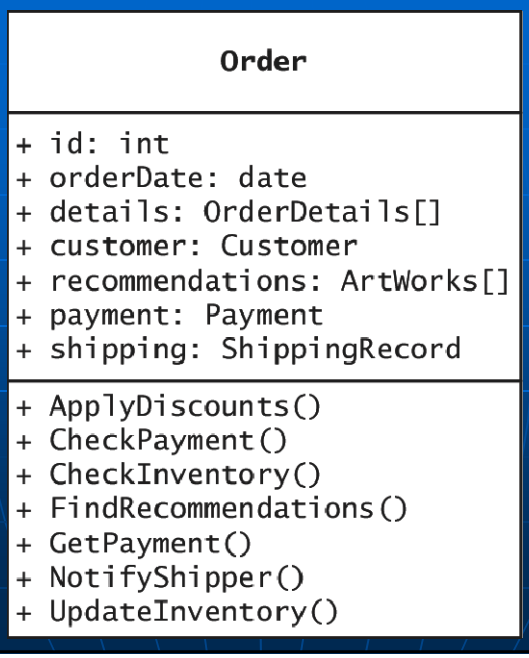


Fig. 14.7 Complex Domain Objects



16

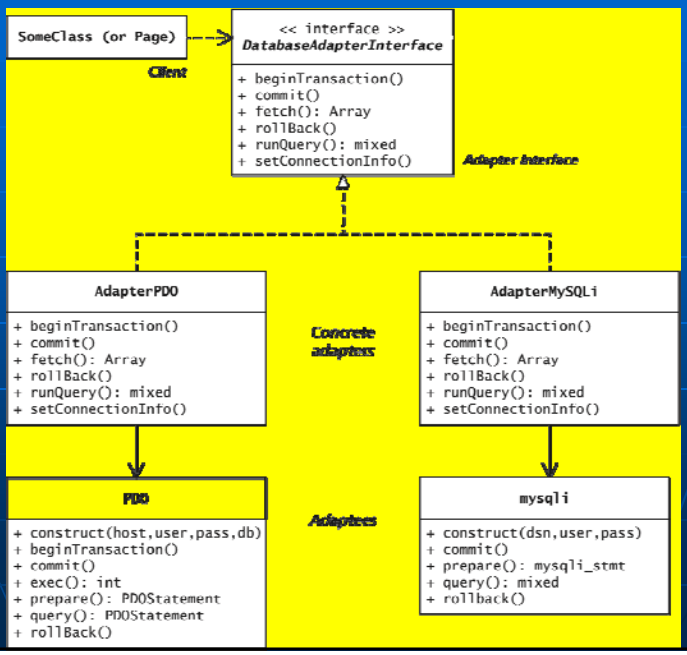
Software Design Patterns in the Web Context

- **Design Patterns**
  - Best practices that can be generalized into reusable solutions that cloud adapt to many different software project
  - 23 classic design patterns
  - Adapter Pattern – used to convert the interface of a set of classes to another different but preferred interface
  - Simple Factory Pattern: a special class that is responsible for the creation of subclasses (or concrete implementation of an interface), so that clients are not coupled to specific subclasses or implementations.
  - Template Method Pattern: One defines an algorithm in an abstract superclass and defers the algorithms steps that can very to the subclass.

Software Design Patterns in the Web Context

- **Design Patterns**
  - Data and Domain Patterns: enterprise patterns
  - Table Data Gateway Pattern
    - Data access object
    - Gateway – an object that encapsulates access to some external resources
  - Domain Model Pattern
    - A variety of related classes that represent objects in the problem domain of the application
  - Active Record Pattern
    - Populating the domain object from the database date
    - Writing the data within domain object back out to the database

Fig. 14.8 A Database API Adapter (Adapter Pattern)



Listing 14.1 Interface for adapter

```
<?php
/*Specifies the functionality of any database adapter*/
interface DatabaseAdapterInterface
{
function setConnectionInfo($values=array());
function closeConnection();
function runQuery($sql, $parameters=array());
function fetchField($sql, $parameters=array());
function fetchRow($sql, $parameters=array());
function fetchAsArray($sql, $parameters=array());
function insert($tableName, $parameters=array());
function getLastInsertId();
```

Listing 14.1 Interface for adapter

```
<?php
/*Specifies the functionality of any database adapter*/
interface DatabaseAdapterInterface
{
…
function update( $tableName, $updateParameters=array(),
$whereCondition='', $whereParameters=array());
function delete( $tableName, $whereCondition=null,
$whereParameters=array());
function getNumRowsAffected();
function beginTransaction();
function commit();
function rollBack();}
?>
```


Listing 14.2 Concrete implementation of an adapter interface

```
<?php
/*
Acts as an adapter for our database API so that all database API
specific code will reside here in this class. In this example, we
will use the PDO API.
*/
include_once("Listing14.01.php");
class DatabaseAdapterPDO implements DatabaseAdapterInterface
{
  private $pdo;
  private $lastStatement = null;
  public function __construct($values) {
    $this->setConnectionInfo($values);
  }
```

Listing 14.2 Concrete implementation of an adapter interface

```
/*Creates a connection using the passed connection information */
function setConnectionInfo($values=array())
{
$connString = $values[0];
$user = $values[1];
$password = $values[2];
$pdo = new PDO($connString,$user,$password);
$pdo->setAttribute(PDO::ATTR_ERRMODE,
PDO::ERRMODE_EXCEPTION);
$this->pdo = $pdo;
 }
```


Listing 14.2 Concrete implementation of an adapter interface

```
/*Executes a SQL query and returns the PDO statement object */
public function runQuery($sql, $parameters=array()) {
// Ensure parameters are in an array
 if (!is_array($parameters)) {
   $parameters = array($parameters);    }
   $this->lastStatement = null;
   if (count($parameters) > 0) {
   // Use a prepared statement if parameters
  $this->lastStatement = $this->pdo->prepare($sql);
  $executedOk = $this->lastStatement->execute($parameters);
   if (! $executedOk) {   throw new PDOException;     }   }
  else {     …
```

Listing 14.2 Concrete implementation of an adapter interface

```
// Execute a normal query
$this->lastStatement = $this->pdo->query($sql);
if (!$this->lastStatement) {
  throw new PDOException;     }   }
return $this->lastStatement;  }
// implementations of all the other methods defined in the interface  }
?>
```


Fig. 14.9 Template Method Pattern

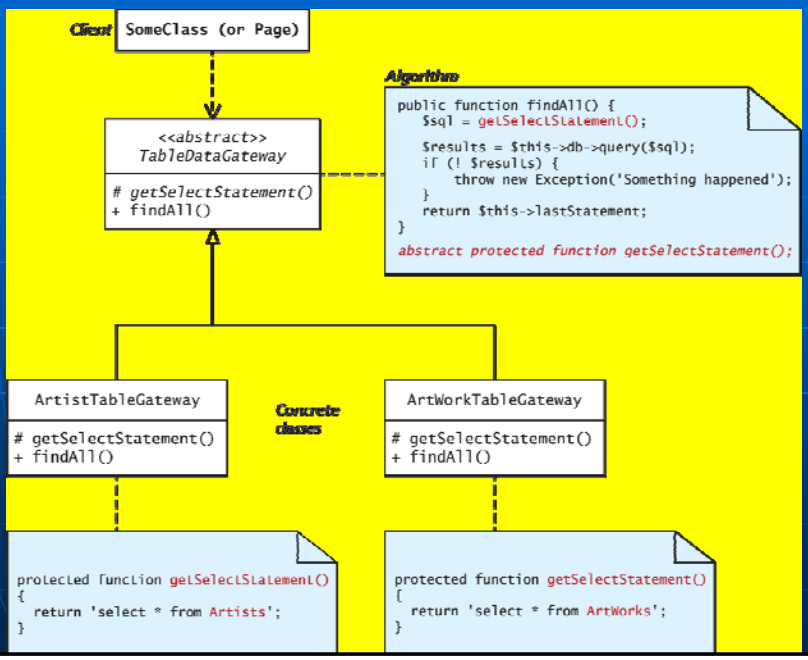


Fig. 14.10 Table Data Gateway Pattern

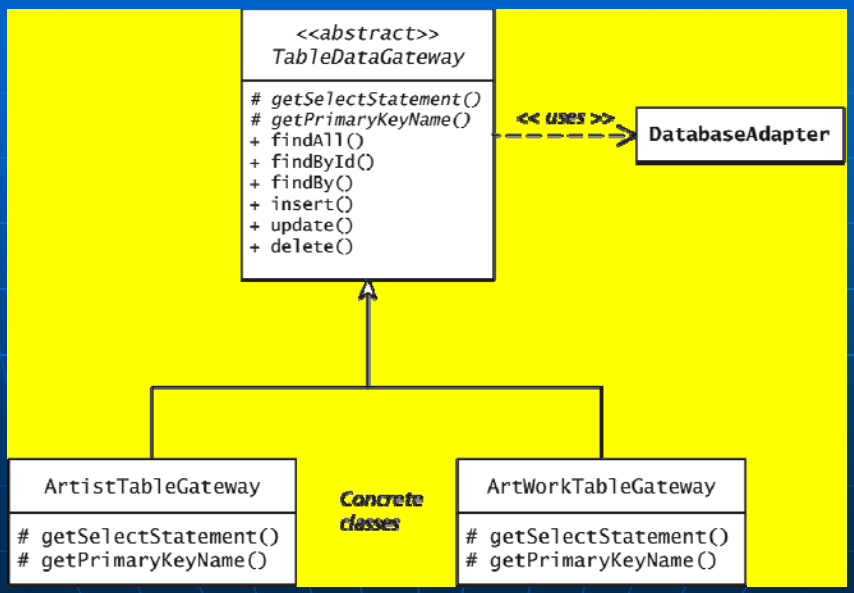


Fig. 14.11 Example Domain Model

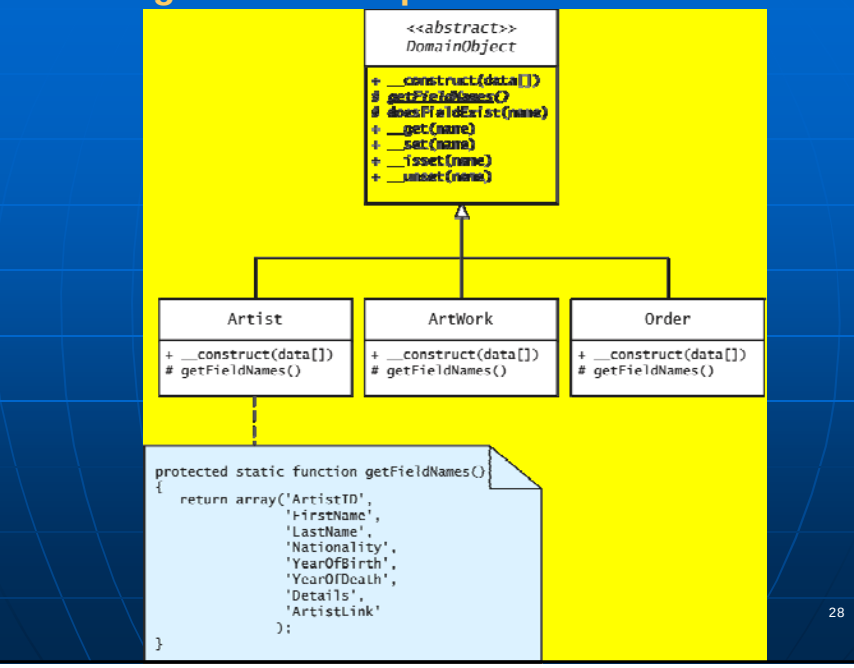
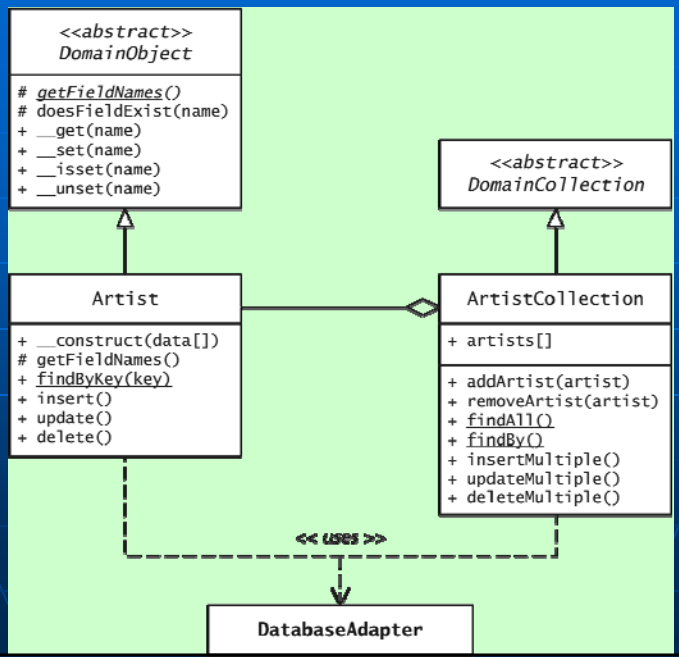


Fig. 14. 12 Active Record version of the Artist and ArtistCollection Classes



Presentation Patterns

- Model-View-Controller (MVC) Pattern divides an application into classes that fall into three different roles
  - **Model**: represents the data of the application
    - It cloud be
      - Domain model classes
      - Active record classes
      - Table gateway classes, or something else
  - **View**: represents the display aspects of the user interface
  - **Controller**: acts as brains of the application and coordinates activities between the view and model

Fig. 14.13 Classic Model-View Control (MVC) Pattern

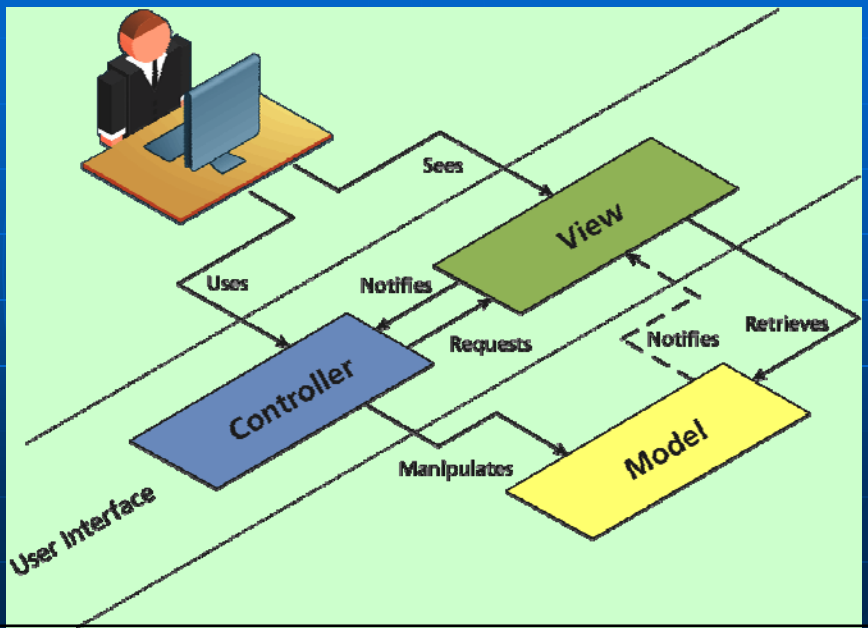


Fig. 14.14 MVC Split Between the Client and the Server

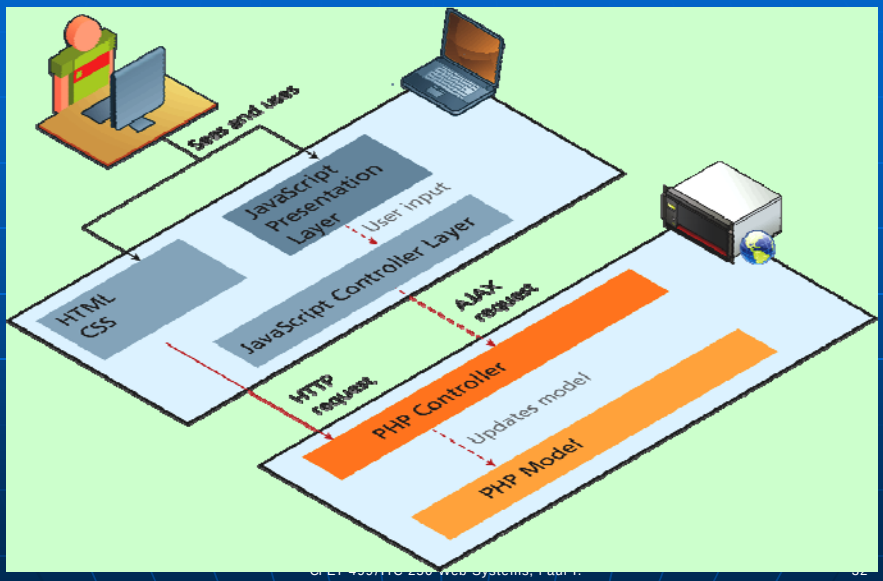


Fig. 14.15 Response in the MVC Between Client and Server

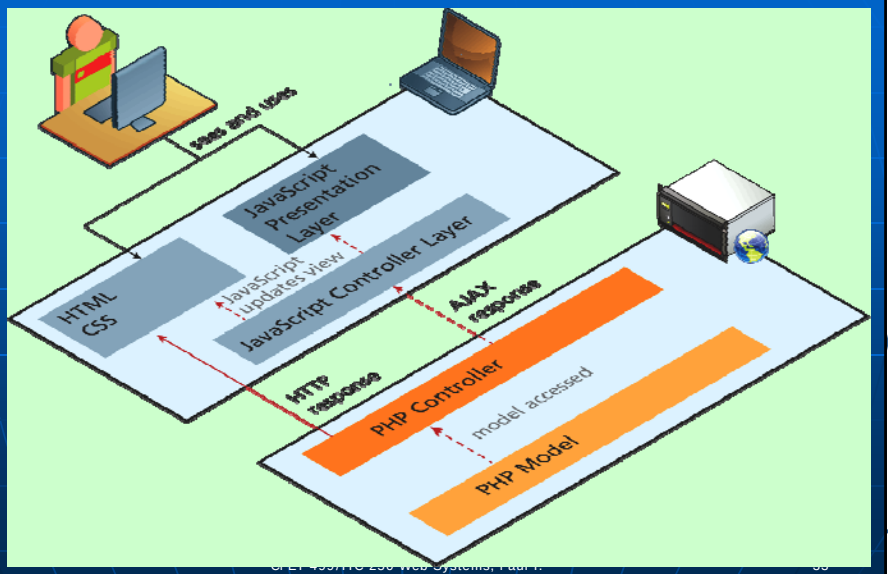
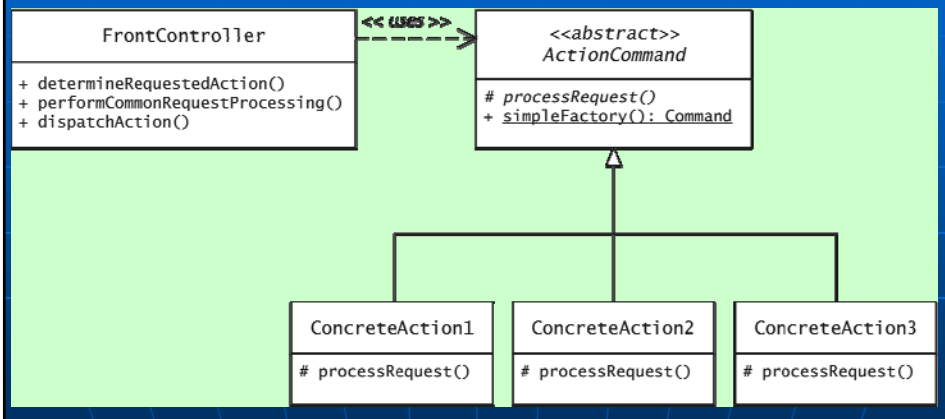
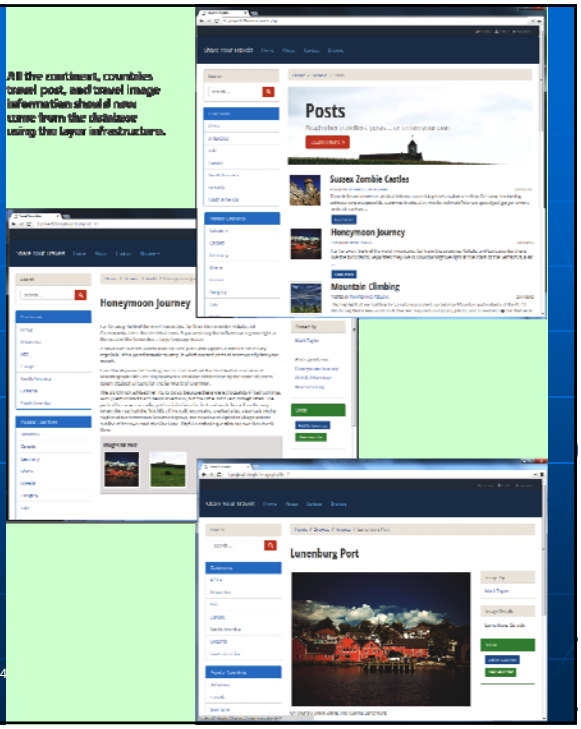


Fig. 14.16 Front Controller Pattern

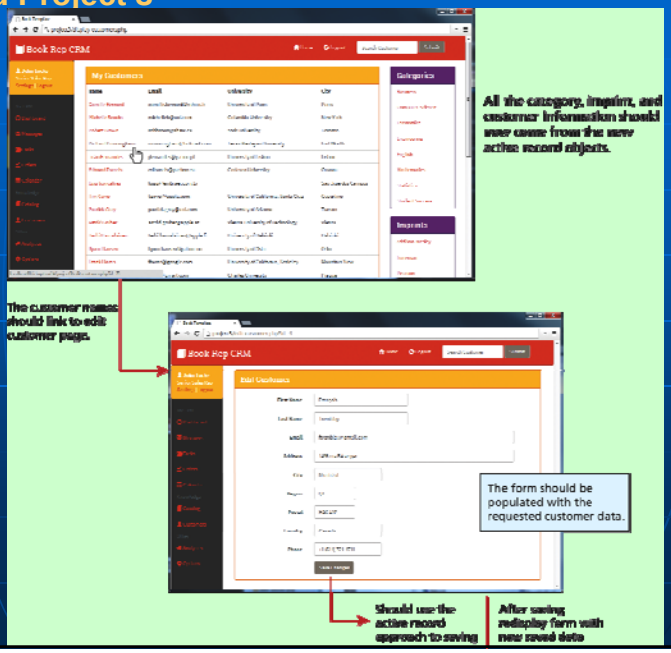


Completed Project 2



CPET 4

Completed Project 3



36

Summary and Conclusion

Q/A ?