Essay questions ONLY - #SEU_IT٢٣٠

Buffered Versus Unbuffered Queries

Buffered queries

- **Retrieve** the query results and store them in memory on the *client side*.
- **Subsequent** calls to get <u>rows</u> will simply spool through *local memory*.

<u>Advantage:</u>

You can move the “current row” pointer around in the result set freely; because it is all <u>in the client.</u>

<u>Disadvantage:</u>

<u>Extra memory is required</u> to store the result set, which could be **very large**, and that the PHP function used to run the query does not return until all the results have been retrieved.

Unbuffered queries

- **Limit you** to a strict sequential access of the results but do not require any extra memory for storing the entire result set.

- **You can start fetching** and processing or displaying rows as soon as the MySQL server starts returning them.

When using an unbuffered result set, you have to <u>retrieve</u> all rows with mysqli_*fetch*_row **or** <u>close</u> the result set with mysqli_*free*_result before sending any other command to the server.

ERROR HANDLING

TYPES OF ERRORS

Runtime Errors

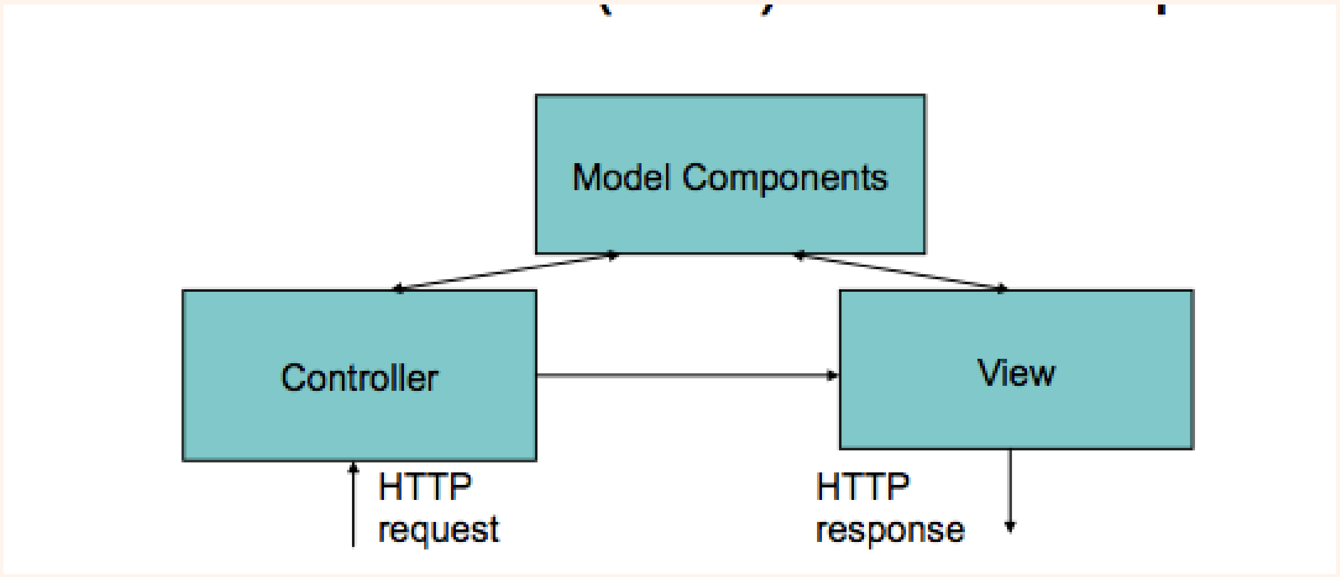
PHP Errors

E_ERROR	E_WARNING
This is a **fatal**, unrecoverable error. Ex: out-of-memory errors, uncaught exceptions, or class re-declarations.	**Most common** type of error. It normally signals that something you tried doing went wrong. Ex: missing function parameters, a database you could not connect to, or division by zero. (/0)

MVC

Wiki: Model–view–controller (MVC) is a software architectural pattern for implementing user interfaces.

- Many web apps are based on the Model-View-Controller (MVC) **architecture pattern**



Web:

The *Model-View-Controller (MVC)* pattern separates the modeling of the domain, the presentation, and the actions based on user input into three separate classes

- **Model**. The model manages the behavior and data of the application domain, responds to requests for information about its state (usually from the view), and responds to instructions to change state (usually from the controller).

- **View**. The view manages the display of information.

- **Controller**. The controller interprets the mouse and keyboard inputs from the user, informing the model and/or the view to change as appropriate.

According to wiki; Model View Controller

Interactions

In addition to dividing the application into three kinds of components, the model–view–controller design defines the interactions between them.

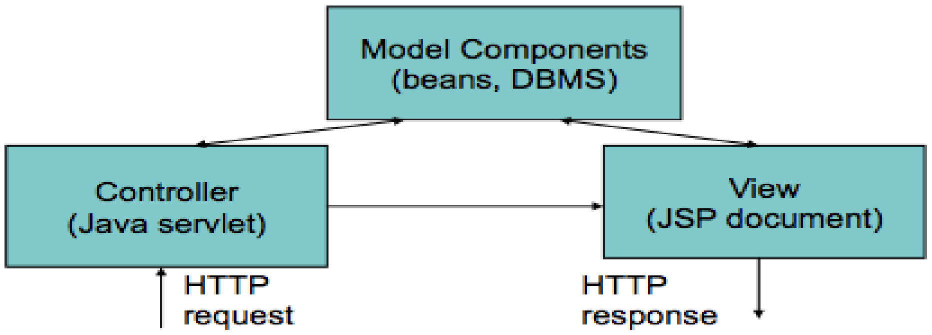
- A controller can send commands to the model to update the model's state (e.g., editing a document). It can also send commands to its associated view to change the view's presentation of the model (e.g., by scrolling through a document).

- A model stores data that is retrieved by the controller and displayed in the view. Whenever there is a change to the data it is updated by the controller.

- A view requests information from the model that it uses to generate an output representation to the user.

MVC

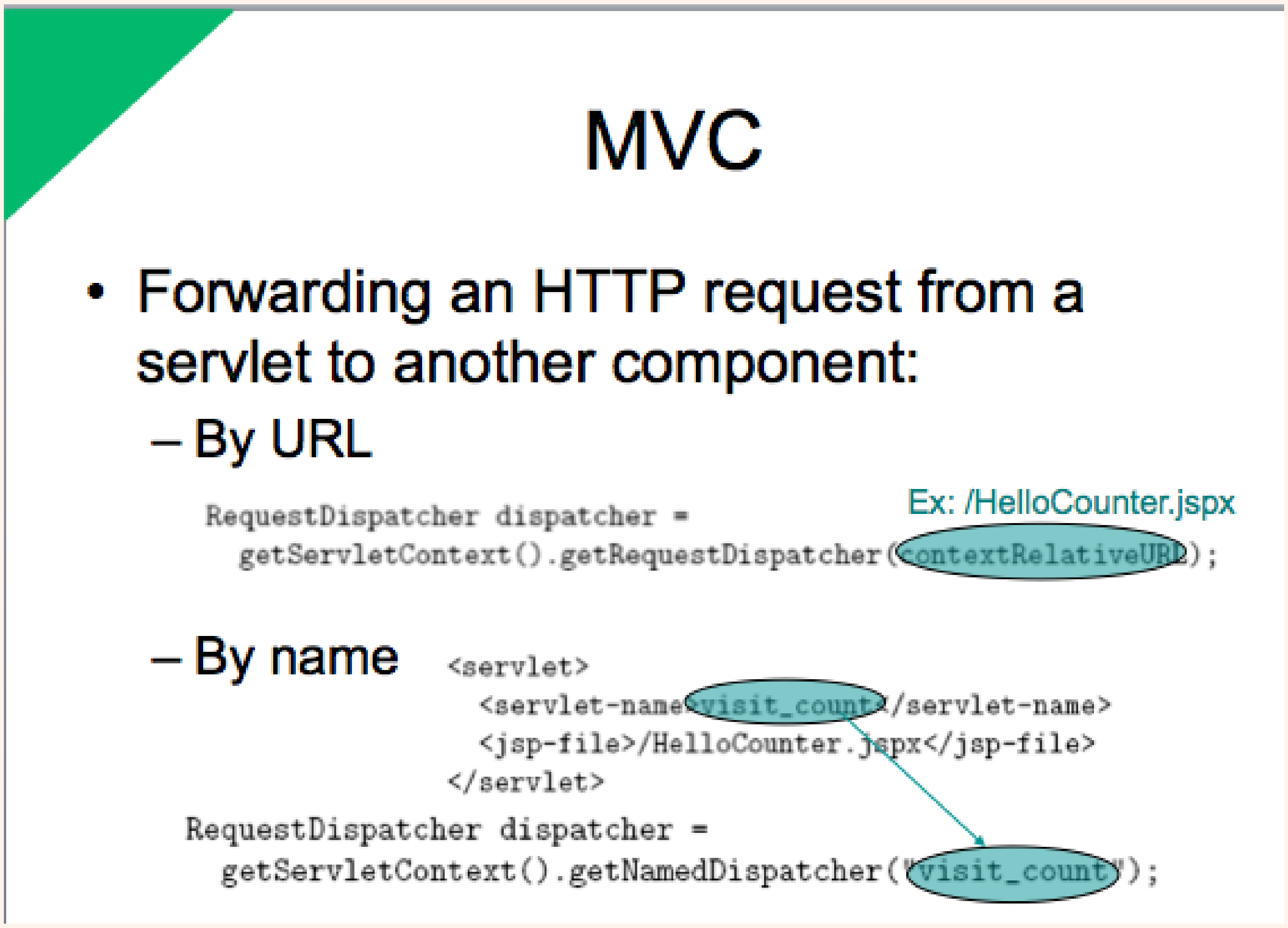
- Typical JSP implementation of MVC



Jackson, Web Technologies: A
Computer Science Perspective, ©
2007 Prentice-Hall, Inc. All rights

From web: How can I implement the MVC design pattern using JSP?

A common scenario might look like this, a user sends a request to a server. The request is handled by a Servlet (the controller) which will initialize any JavaBeans (the model) required to fulfill the user's request. The Servlet (the controller) will then forward the request, which contains the JavaBeans (the model), to a JSP (the view) page which contains only HTML and JSTL syntax. The JSTL will format the data into a human readable format before being sent back to the user. Thus completing the full MVC process.



MVC

- Forwarding an HTTP request from a servlet to another component:
  - By URL

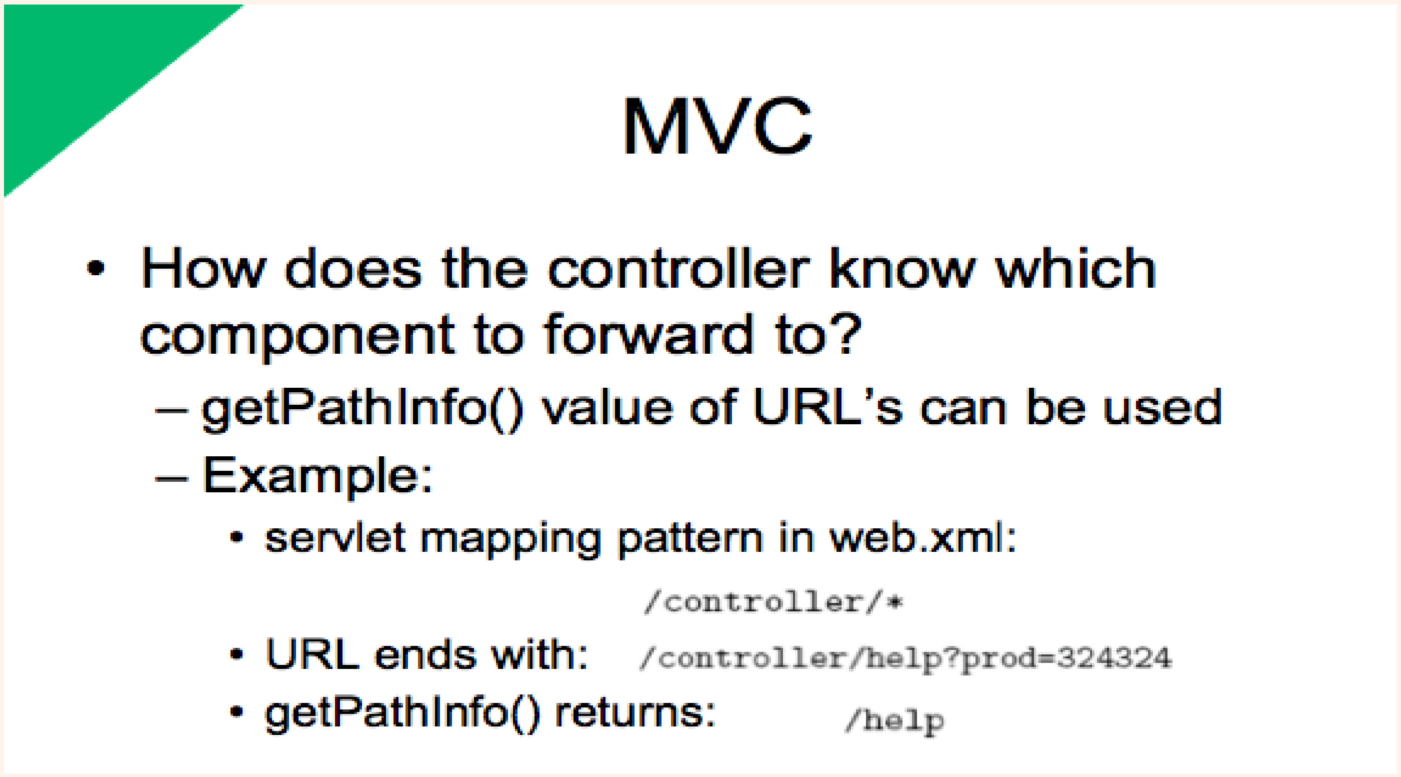
Ex: /HelloCounter.jspx

```
RequestDispatcher dispatcher =
  getServletContext().getRequestDispatcher("contextRelativeURL");
```

  - By name

```
<servlet>
  <servlet-name>visit_count</servlet-name>
  <jsp-file>/HelloCounter.jspx</jsp-file>
</servlet>
```

```
RequestDispatcher dispatcher =
  getServletContext().getNamedDispatcher("visit_count");
```



MVC

- How does the controller know which component to forward to?
  - getPathInfo() value of URL's can be used
  - Example:
    - servlet mapping pattern in web.xml:

      `/controller/*`
    - URL ends with: `/controller/help?prod=324324`
    - getPathInfo() returns: `/help`

MVC

- JSP include action (not the same as the include directive!)

```
<table style="width:100%" border="0">
  <tbody>
    <tr>
      <td style="width:20%"
         ><jsp:include page="/navbar.jspx" /></td>
      <td style="width:80%"
         ><jsp:include page="/mainContent.jspx" /></td>
    </tr>
  </tbody>
</table>
```

Execute specified component and include its output in place of the include element

MVC

- Adding parameters to the request object seen by an included component:

```
<jsp:include page="/navbar.jspx">
 <jsp:param name="currentPage" value="home" />
</jsp:include>
```

request object seen by navbar.jspx will include parameter named currentPage with value home

JDBC

❖ Seven Steps for Database Access (Concept & Code) Essay

1. Load the JDBC driver
2. Define the connection URL
3. Establish the connection
4. Create a Statement object
5. Execute a query or update
6. Process the result set
7. Close the statement and connection

1- Load the JDBC driver.

To load a driver, you specify the classname of the database driver in the Class.forName method. By doing so, you automatically create a driver instance and register it with the JDBC driver manager.

```
try {

Class.forName("connect.microsoft.MicrosoftDriver");

Class.forName("oracle.jdbc.driver.OracleDriver");

Class.forName("com.sybase.jdbc.SybDriver");

} catch(ClassNotFoundException cnfe) {

System.err.println("Error loading driver: " + cnfe);

}
```

2. Define the connection URL.

In JDBC, a connection URL specifies the server host, port, and database name with which to establish a connection.

```
String host = "dbhost.yourcompany.com";

String dbName = "someName";

int port = 1234;
```

```
String oracleURL = "jdbc:oracle:thin:@" + host +

":" + port + ":" + dbName;

String sybaseURL = "jdbc:sybase:Tds:" + host +

":" + port + ":" + "?SERVICENAME=" + dbName;

String msAccessURL = "jdbc:odbc:" + dbName;
```

3. Establish the connection.

With the connection URL, username, and password, a network connection to the database can be established.

Once the connection is established, database queries can be performed until the connection is closed.

try/catch :

```
String username = "jay_debesee";

String password = "secret";

Connection connection =

DriverManager.getConnection(oracleURL, username, password);
```

(getDriverName, getDriverVersion):

```
DatabaseMetaData dbMetaData = connection.getMetaData();

String productName =

dbMetaData.getDatabaseProductName();

System.out.println("Database: " + productName);

String productVersion =

dbMetaData.getDatabaseProductVersion();

System.out.println("Version: " + productVersion);
```

4. Create a Statement object.

Creating a Statement object enables you to send queries and commands to the database.

```
Statement statement = connection.createStatement();
```

5. Execute a query or update.

Given a Statement object, you can send SQL statements to the database by using the execute, executeQuery, executeUpdate, or executeBatch methods.

```
String query = "SELECT col1, col2, col3 FROM sometable";

ResultSet resultSet = statement.executeQuery(query);
```

6. Process the results.

When a database query is executed, a ResultSet is returned. The ResultSet represents a set of rows and columns that you can process by calls to next and various getXxx methods.

```
while(resultSet.next()) {

System.out.println(resultSet.getString(1) + " " +

resultSet.getString(2) + " " +

resultSet.getString("firstname") + " "

resultSet.getString("lastname"));

}
```

7. Close the connection.

When you are finished performing queries and processing results, you should close the connection, releasing resources to the database.

```
connection.close();
```

Design patterns (Strategy pattern, singleton pattern, factory pattern, observer pattern) (Essay & concept)

DESIGN PATTERNS

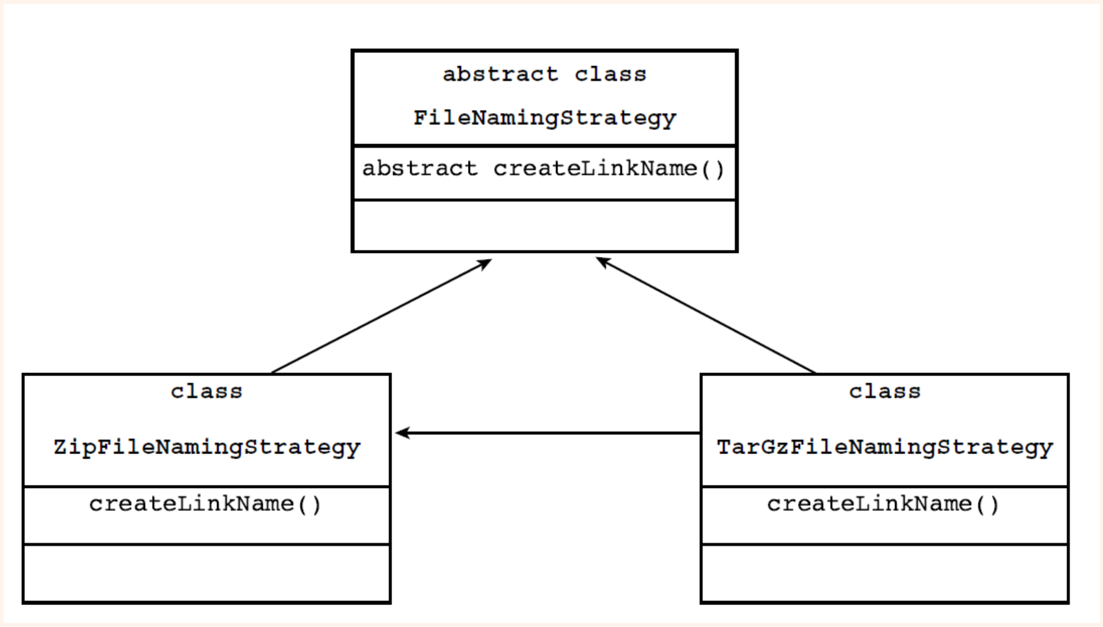
When designing software, certain programming patterns repeat themselves. Some of these have been addressed by the software design community and have been given accepted general solutions. These repeating problems are called **design patterns.**

Advanced OOP and Design Patterns (4 patterns)

❖ Strategy Pattern

The strategy pattern is typically used when your programmer's algorithm should be **interchangeable** with different variations of the algorithm.
For example, if you have code that creates an image, under certain circumstances, you might want to create JPEGs and under other circumstances, you might want to create GIF files.

The strategy pattern is usually implemented by declaring an abstract base class with an algorithm method, which is then implemented **by inheriting concrete classes**. At some point in the code, it is decided what concrete strategy is relevant; it would then be instantiated and used wherever relevant.



❖ Singleton Pattern

The **singleton pattern** is probably one of the best-known design patterns. You have probably encountered(face) many situations where you have an object that handles some centralized operation in your application, such as a logger object.

In such cases, it is usually preferred for only one such application-wide instance to exist and for all application code to have the ability to access it.

Specifically, in a logger object, you would want every place in the application that wants to print something to the log to have access to it, and let the centralized logging mechanism handle the filtering of log messages according to log level settings. For this kind of situation, the singleton pattern exists.

❖ Factory Pattern

- **Polymorphism** and the use of base class is really the center of OOP.

At some stage, a concrete instance of the base class's subclasses must be created.

This is usually done using the **factory pattern**. A Factory class has a static method that receives some input and, according to that input, it decides what class instance to create (usually a subclass).

Say that on your web site, different kinds of users can log in.

Some are guests, some are regular customers, and others are administrators.

In a common scenario, you would have a base class User and have three subclasses: GuestUser, CustomerUser, and AdminUser.

Likely User and its subclasses would contain methods to retrieve information about the user

For example, **permissions on what they can access on the web site and their personal preferences**

The best way for you to write your web application is to use the base class User as much as possible, **so** that the code would be **generic** and that it would be easy to add additional kinds of users when the need arises.

❖ Observer Pattern

PHP applications, usually manipulate data.

In many cases, *changes to one piece of data* ***can affect many*** *different parts of your application's code*.

For example, the price of each product item displayed on an e -commerce site in the customer's local currency is affected by the current exchange rate.

Now, assume that each product item is represented by a PHP object that most likely originates from a database; the exchange rate itself is most probably being taken from a different source and is not part of the item's database entry.

Let's also assume that each such object has a ***display()*** *method that outputs the HTML relevant to this product.*

The **observer pattern** allows for objects to **register** on certain events and/or data, and when such an event or **change in data occurs,** it is **automatically notified.**

In this way, you could develop the product item to be an observer on the currency exchange rate, and before printing out the list of items, you could trigger an event that updates all the registered objects with the correct rate.

Doing so gives the objects a chance to update themselves and take the

new data into account in their display() method.