Total No. of Questions : 8]

SEAT No. :

PB3785

[Total No. of Pages : 2

[6262]-44

T.E. (Computer Engineering)

WEB TECHNOLOGY

(2019 Pattern) (Semester - II) (310252)

Time : 2½ Hours] *[Max. Marks : 70*

Instructions to the candidates:

1) *Answer Q.1 or Q.2, Q.3 or Q.4, Q.5 or Q.6, Q.7 or Q.8.*
2) *Figures to the right side indicate full marks.*

Q1) a) Explain doGet() & dopost() methods of servet. Differentiate do Get Vs do Post (Min 04). **[9]**

b) What is XML DTDs? Explain with example. Differentiate XML DTDs Vs XML schema (Min.04). **[9]**

OR

Q2) a) Explain the senlet lifecycle. Explain session management using cookies and URL Rewriting. **[12]**

b) Write note on AJAX. **[6]**

Q3) a) Explain the JSP support for MVC paradigm. **[8]**

b) Explain struts framework with respect to architecture, actions, interceptors & exception handling. **[9]**

OR

Q4) a) Explain JSP lifecycle. Differentiate JSP Vs Servlet. (Min.04). **[9]**

b) Explain the concept of web services. Explain in brief WSDL & SOAP. **[8]**

Q5) a) Explain various types of Arrays in PHP. Explain each with example code. **[9]**

b) Explain the following :

i) WAP & WML **[4]**

ii) C# Vs Java **[5]**

OR

P.T.O.

Q6) a) Explain the PHP with MySQL using example. **[6]**

b) Write note on : **[12]**

i) Session tracking in PHP.

ii) NET framework

iii) NodeJS

Q7) a) Explain Ruby with its advantages. Explain control statements in Ruby. **[10]**

b) Explain EJB concept & five basic example of using EJB. **[7]**

OR

Q8) a) Explain the arrays in Ruby. Explain Rails with AJAX. **[10]**

b) Explain Document Request in Rails. **[4]**

c) Explain advantages of Ruby and Rails. **[3]**

Q1) a) Explain doGet() & dopost() methods of servet. Differentiate do Get Vs do Post (Min 04).

Ans:

Servlet doGet() and doPost() Methods

Servlets use the `doGet()` and `doPost()` methods to handle HTTP GET and POST requests, respectively. These methods are part of the `HttpServlet` class, which is commonly extended when writing servlets.

- **`doGet()` Method:**
  - **Purpose:** This method is used to handle HTTP GET requests. It is typically used to request data from a source.
  - **Usage:** When a client sends a GET request (e.g., by typing a URL directly in the browser, clicking a hyperlink, or submitting an HTML form with `method="GET"`), the servlet container invokes the `doGet()` method.
  - **Protocol:** `protected void doGet(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException`
    - `HttpServletRequest request`: Contains the client's request information.
    - `HttpServletResponse response`: Contains the response that the servlet sends back to the client.
    - `ServletException, IOException`: Exceptions that can be thrown during servlet operation.
- **`doPost()` Method:**
  - **Purpose:** This method is used to handle HTTP POST requests. It is typically used to submit processed data to a source.
  - **Usage:** When a client sends a POST request (most commonly by submitting an HTML form with `method="POST"`), the servlet container invokes the `doPost()` method.
  - **Protocol:** `protected void doPost(HttpServletRequest request, HttpServletResponse response) throws ServletException,IOException`
    - The parameters are the same as `doGet()`: `HttpServletRequest request` and `HttpServletResponse response`.
    - `ServletException, IOException`: Exceptions that can be thrown.

Feature	`doGet()` Method	`doPost()` Method
Purpose	Requests data from the source.	Submits processed data to the source.
Data Transmission	Parameters are appended to the URL as a query string.	Parameters are sent in the body of the HTTP request.
Security	Less secure as parameters are visible in the URL and browser history.	More secure as parameters are not directly visible in the URL.
Efficiency	Generally more efficient than POST requests for simple data retrieval.	Can be less efficient due to sending data in the request body.
Bookmarks	Requests can be bookmarked and cached.	Requests cannot be bookmarked or easily cached.
Data Length	Limited by the maximum URL length (varies by browser/server).	No practical limit on data length.
Idempotence	Generally idempotent (multiple identical requests have the same effect as a single one).	Generally not idempotent (multiple identical requests might have different effects, e.g., creating multiple entries).

Q1) b) What is XML DTDs? Explain with example. Differentiate XML DTDs Vs XML schema (Min.04).

Ans:

XML DTDs (Document Type Definitions)

An XML DTD (Document Type Definition) is a set of rules used to define the legal building blocks of an XML document. It specifies the structure of an XML file, including the elements, attributes, and their relationships. Essentially, a DTD ensures that an XML document conforms to a predefined structure, making it "valid."

Example of an Internal DTD: An internal DTD is declared directly within the XML file.

XML

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE student [
  <!ELEMENT student (name,address,std,marks)>
  <!ELEMENT name (#PCDATA)>
  <!ELEMENT address (#PCDATA)>
  <!ELEMENT std (#PCDATA)>
  <!ELEMENT marks (#PCDATA)>
]>
<student>
  <name>Anand</name>
  <address>Pune</address>
  <std>Second</std>
  <marks>70 percent</marks>
</student>
```

In this example, the `<!DOCTYPE student [...]>` section defines the DTD internally. It declares that a `student` element must contain `name`, `address`, `std`, and `marks` elements in that specific sequence. Each of these child elements is defined to contain parsed character data (`#PCDATA`).

XML Schema

XML Schema is another way to define the structure and content of XML documents. Unlike DTDs, XML Schema itself is written in XML, making it more extensible and powerful. It provides a more robust and flexible way to specify data types, element relationships, and namespaces.

Example of XML Schema and an XML document referencing it:

`StudentSchema.xsd` (XML Schema file):

XML
```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Student">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="name" type="xs:string"/>
        <xs:element name="address" type="xs:string"/>
        <xs:element name="std" type="xs:string"/>
        <xs:element name="marks" type="xs:string"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

The `xs:schema` is the root element and indicates that the document follows XML Schema rules. It defines a `Student` element which is a complex type containing a sequence of `name`, `address`, `std`, and `marks` elements, all of type `xs:string`.

`MySchema.xml` (XML document referencing the schema):

XML
```
<?xml version="1.0" encoding="UTF-8"?>
<Student      xmlns:xsi="http://www.w3.org/2001/XMLSchema-
instance"
xsi:noNamespaceSchemaLocation="StudentSchema.xsd">
  <name>Anand</name>
  <address>Pune</address>
  <std>Second</std>
  <marks>70 percent</marks>
</Student>
```

The `xmlns:xsi` attribute indicates that the XML document is an instance of an XML schema, and `xsi:noNamespaceSchemaLocation` links it to the `StudentSchema.xsd`

Feature	XML DTDs (Document Type Definitions)	XML Schema (XSD)
Syntax	Uses a non-XML syntax (SGML-like syntax).	Uses XML syntax itself.
Extensibility	Limited in terms of extensibility and reusability.	Highly extensible and designed for reusability.
Data Types	Very limited support for data types (e.g., PCDATA, CDATA).	Supports a rich set of built-in data types (e.g., string, integer, date, boolean) and allows custom data types.
Namespaces	Does not support XML Namespaces.	Fully supports XML Namespaces, preventing naming conflicts.
Readability	Can be harder to read and parse for complex structures.	Generally more readable for XML developers due to XML syntax.
Error Handling	Provides less precise error reporting.	Offers more precise error reporting due to detailed data typing.
Support for XML	Not XML-aware (defined outside the XML specification itself).	Fully integrated with XML, defined by W3C as part of the XML family of recommendations.

Q2) a) Explain the senlet lifecycle. Explain session management using cookies and URL Rewriting.

Ans:

Servlet Lifecycle

The **Servlet lifecycle** consists of four main stages, managed by the **Servlet container**:

1. **Loading & Instantiation**
   - The container loads the Servlet class into memory.
   - An instance of the Servlet is created using the no-argument constructor.
2. **Initialization**
   - The container calls the `init(ServletConfig config)` method.
   - This method is executed **only once** during the Servlet's lifecycle.
3. **Handling Requests**
   - The container creates `ServletRequest` and `ServletResponse` objects.
   - The `service(ServletRequest req, ServletResponse res)` method is invoked.
   - The request type (GET, POST, etc.) determines which method (`doGet()`, `doPost()`, etc.) is executed.
4. **Destruction**
   - The container calls the `destroy()` method before removing the Servlet.
   - Resources like database connections are released.
   - The Servlet instance is **garbage collected**.

For a detailed explanation, you can check this resource and this guide.

Session Management Using Cookies and URL Rewriting

1. Cookies-Based Session Management

- **Cookies** store session identifiers on the client's browser.
- The server sends a **Set-Cookie** header in the response.
- The browser includes the cookie in subsequent requests, maintaining session continuity.
- **Advantages:**
  - Persistent across multiple requests.
  - Works automatically without modifying URLs.
- **Disadvantages:**

- Users can disable cookies.
- Security risks if cookies are intercepted.

2. URL Rewriting-Based Session Management

- The **session ID** is appended to the URL as a parameter (`?sessionID=12345`).
- The server extracts the session ID from the request URL.
- **Advantages:**
  - Works even if cookies are disabled.
- **Disadvantages:**
  - Requires modifying every URL in the application.
  - Less secure, as session IDs are visible in URLs.

Q2) b) Write note on AJAX.

Ans:

A Note on AJAX

AJAX stands for Asynchronous JavaScript and XML. It is not a new programming language but rather a web development technique that uses a combination of existing technologies to create more interactive and dynamic web applications.

The core concept of AJAX is to allow web applications to exchange data with a server in the background and update parts of a web page without requiring a full page reload. This leads to a more responsive user experience, as the user's work is not disturbed by script execution.

How AJAX Works:

1. When a user initiates a request (e.g., clicks a button), the browser creates an `XMLHttpRequest` object.
2. A request is then made to the server over the internet without reloading the entire page.
3. The server processes this request and sends the required data back to the browser.
4. On the browser side, JavaScript processes the returned data.
5. Finally, the relevant part of the web document is updated accordingly, providing an appropriate response to the user.

AJAX utilizes JavaScript to perform the actual work and supports XML for data exchange, although it can also work with other data formats like JSON.

Q3) a) Explain the JSP support for MVC paradigm.

Ans:

JSP Support for MVC (Model-View-Controller) Paradigm

JSP (JavaServer Pages) technology, while initially used for both presentation and business logic, is most effectively utilized as the **View** component in the MVC architectural paradigm. The MVC pattern separates an application into three interconnected components to improve maintainability, scalability, and reusability.

- **Model:** Represents the application's business logic, data, and rules. In a Java-based web application, the Model is typically implemented using JavaBeans, POJOs (Plain Old Java Objects), or Enterprise JavaBeans (EJBs) that interact with databases or other data sources. JSP itself doesn't directly implement the Model; rather, it interacts with Model components.
- **View:** Responsible for displaying the data to the user and handling user interaction. This is where **JSP** plays its primary role. JSPs are ideal for generating dynamic HTML, XML, or other content. They are designed to present the data retrieved from the Model, allowing for a clean separation of presentation logic from business logic.
    - **How JSP acts as a View:** JSPs contain static HTML/CSS/JavaScript markup along with JSP tags (like `<%= ... %>`, `<c:forEach>`, `<jsp:include>`) to dynamically insert data from the Model and control the flow of presentation.
- **Controller:** Acts as an intermediary between the Model and the View. It receives user input (requests), processes them, interacts with the Model to retrieve or update data, and then selects the appropriate View to display the response. In Java web applications, **Servlets** typically serve as the Controller.
    - **How Servlets (Controllers) work with JSPs (Views):**
        1. A user request comes to a Servlet (Controller).
        2. The Servlet processes the request, potentially calls Model components to perform business operations or fetch data.
        3. After processing, the Servlet decides which JSP (View) should be displayed.
        4. The Servlet then forwards the request (including data from the Model, often set as attributes in the `HttpServletRequest` or `HttpSession` objects) to the chosen JSP.
        5. The JSP retrieves the data from the request scope, session scope, or directly from Model objects, and renders the dynamic content to the user's browser.

Benefits of using JSP in MVC:

- **Separation of Concerns:** Clearly separates presentation (JSP) from business logic (Model) and request handling (Controller), making the application easier to develop, debug, and maintain.
- **Modularity:** Changes to the UI (JSP) can be made without affecting the business logic (Model) or request handling (Controller), and vice-versa.
- **Team Collaboration:** Different team members (e.g., UI designers and backend developers) can work independently on their respective parts.
- **Reusability:** Model and Controller components can be reused across different views or even different applications.
- **Testability:** Each component can be tested independently.

Q3) b) Explain struts framework with respect to architecture, actions, interceptors & exception handling.

Ans:

The Struts framework is a popular open-source web application framework for developing Java EE web applications. It promotes the use of the Model-View-Controller (MVC) architectural pattern to organize web applications, separating concerns into distinct components for improved maintainability and scalability.

Here's an explanation of the Struts framework with respect to its architecture, actions, interceptors, and exception handling:

Architecture

Struts is built upon the MVC design pattern, aiming to cleanly separate the application's business logic (Model), presentation (View), and request handling (Controller).

- **Model:** This layer contains the application's business logic, data access, and validation rules. It's typically implemented using JavaBeans or Plain Old Java Objects (POJOs).
- **View:** The View is responsible for the presentation of data to the user and capturing user input. JSPs (JavaServer Pages) are commonly used for the View layer in Struts applications, though other technologies like XML/XSLT and Velocity are also supported.
- **Controller:** Struts provides the Controller component, which primarily consists of a Servlet (like `ActionServlet` in Struts 1.x or `FilterDispatcher` in Struts 2) and configuration files (e.g., `struts-config.xml` for Struts 1.x or `struts.xml` for Struts 2). The Controller receives client requests, maps them to appropriate Actions, and forwards the results to the correct View.

The flow typically involves:

1. A client sends a request (e.g., via a browser).
2. The Controller (`ActionServlet` or `FilterDispatcher`) intercepts the request.
3. Based on the configuration file (`struts.xml`), the Controller determines the appropriate `Action` to handle the request.
4. The `Action` processes the request, interacting with the Model if necessary.
5. The `Action` returns a result string, which the Controller uses to select the next View (e.g., a JSP page) to render the response.

Actions

In the Struts framework, an `Action` represents the logical unit of work that handles a client's request. `Action` classes encapsulate the business logic related to a specific user interaction.

- **Purpose:** An `Action` class receives input data from the presentation layer (often via a form bean), processes it, and then decides which result (View) should be displayed to the user.
- **Implementation:** In Struts 2, `Action` classes typically extend the `ActionSupport` class and override the `execute()` method, which contains the core logic for the action.
- **Mapping:** Actions are mapped to specific URLs in the Struts configuration file (`struts.xml`). This mapping defines which `Action` class and method should be executed for a given incoming request.
- **Return Value:** The `execute()` method of an `Action` usually returns a `String` (e.g., "success", "input", "error"), which corresponds to a defined result in the configuration, directing the flow to a specific View.

Interceptors

Interceptors are powerful components in Struts 2 that allow for cross-cutting concerns and customization of the framework's processing flow. They can execute code both before and after an `Action` is invoked.

- **Functionality:** Many core features of Struts 2, such as validation, exception handling, logging, file uploads, and protection against double submits, are implemented as interceptors.
- **Configuration:** Interceptors are configured in `struts.xml` and can be applied globally to all actions or specified on a per-action basis.
- **Interceptor Stacks:** Interceptors can be grouped into "interceptor stacks," which define the order in which they are executed. Struts provides default interceptor stacks, and developers can also create custom stacks or individual interceptors.
- **Flexibility:** Interceptors offer a flexible way to add or modify functionality without altering the core `Action` logic, promoting modularity and reusability.

Exception Handling

Struts provides a robust mechanism for handling exceptions, allowing developers to manage unexpected errors gracefully and declaratively.

- **Declarative Approach:** Struts supports declarative exception handling, meaning that exception mappings are defined in the configuration file (`struts.xml` for Struts 2) rather than being scattered throughout the Java code with `try-catch` blocks. This makes the application more maintainable and cleaner.
- **Global and Local Handling:**
  - **Global Exception Handling:** You can define exception mappings that apply to all actions within a Struts package or the entire application. This is typically done using the `<global-exception-mappings>` element in `struts.xml`.

- **Per-Action Exception Handling:** For specific actions that require unique exception handling, you can define an `<exception-mapping>` directly within the `<action>` configuration. Action-specific mappings take precedence over global ones.

- **Redirection:** When an uncaught exception occurs (one not handled by the application's code), the Struts framework intercepts it and redirects the user to a pre-configured error page (e.g., a JSP) based on the exception type and the defined mapping. This allows the application to fail gracefully and provide a user-friendly error message.

Q4) a) Explain JSP lifecycle. Differentiate JSP Vs Servlet. (Min.04)

Ans:

JSP Lifecycle

The JSP (JavaServer Pages) lifecycle describes the stages a JSP goes through from its creation to its destruction within a web container. It's similar to the Servlet lifecycle because, fundamentally, a JSP is compiled into a servlet.

1. **Translation Phase:**
   - When the browser requests a JSP for the first time, or if the JSP has been modified, the web container translates the JSP page into its equivalent Java source code (a servlet .java file). This servlet will contain all the static HTML/XML content as `println` statements and dynamic JSP elements (scriptlets, expressions, directives) converted into Java code.
2. **Compilation Phase:**
   - The Java source file generated in the translation phase is then compiled into a Java bytecode file (a `.class` file). This `.class` file is the actual servlet that the container will load and execute.
3. **Loading and Instantiation Phase:**
   - The servlet container loads the compiled `.class` file into memory.
   - An instance of the generated servlet class is created.
4. **Initialization Phase (`jspInit()`):**
   - After instantiation, the container calls the `jspInit()` method on the servlet instance. This method is called only once, during the servlet's initialization, similar to a Servlet's `init()` method. It's used for one-time initialization tasks for the JSP.
5. **Request Processing Phase (`_jspService()`):**
   - For every client request to the JSP, the container invokes the `_jspService()` method. This method is responsible for processing the request and generating the response.
   - The `_jspService()` method takes `HttpServletRequest` and `HttpServletResponse` objects as parameters.
   - Developers generally do not override this method directly; instead, they embed Java code within the JSP using scriptlets, expressions, and custom tags, which are then compiled into the `_jspService()` method.
6. **Destruction Phase (`jspDestroy()`):**
   - When the web application is undeployed, the server shuts down, or the container decides to remove the JSP (servlet) instance from memory, the `jspDestroy()` method is called.
   - Like `jspInit()`, it's called only once. It's used to release resources held by the JSP, such as database connections.

Feature	JSP (JavaServer Pages)	Servlet
Primary Role	Primarily for **presentation** (View in MVC).	Primarily for **request processing** (Controller in MVC).
Code Structure	HTML/XML centric with embedded Java code (scriptlets, expressions, custom tags).	Java-centric, with HTML/XML generated via println statements.
Ease of Development	Easier for web designers to create and modify layout/UI due to HTML-first approach.	More suitable for Java developers, as it's pure Java code.
Compilation	Automatically translated into a Servlet and compiled by the container at runtime.	Must be manually compiled by the developer before deployment.
Protocol Handling	Does not directly handle HTTP methods (doGet(), doPost()). The generated servlet's _jspService() method handles this.	Directly handles HTTP methods via doGet(), doPost(), etc.
Performance	Can be slightly slower on first request due to translation and compilation, but then similar to Servlets.	Generally faster on first request as it's already compiled.
Maintenance	Easier to modify presentation logic without recompiling Java code.	Changes to presentation (HTML) require recompiling the Java code.
Best Use Case	Generating dynamic web content, displaying data.	Handling form submissions, complex business logic, acting as controllers.

Q4) b) Explain the concept of web services. Explain in brief WSDL & SOAP.
Ans:

Web Services

Web services are a standardized way of integrating web-based applications using open standards over an internet protocol backbone. They allow different applications, developed in different programming languages and running on different platforms, to communicate and exchange data with each other. This inter-application communication is often called "Machine-to-Machine" interaction.

The core idea behind web services is to enable loosely coupled, reusable components that can be discovered and invoked programmatically over a network (typically the internet). They promote interoperability and facilitate the creation of distributed applications.

Characteristics of web services:

- **Platform Independent:** Can be consumed by applications running on any operating system or programming language.
- **Language Independent:** The service can be implemented in one language (e.g., Java) and consumed by an application written in another (e.g., Python).
- **Standardized Protocols:** They rely on standard internet protocols (HTTP, XML, SOAP, WSDL, UDDI) for communication and description.
- **Self-describing:** They provide a way for other applications to understand their functionality and how to interact with them.

WSDL (Web Services Description Language)

WSDL (Web Services Description Language) is an XML-based language used to describe the functionality offered by a web service. It acts like a "contract" or a "user manual" for a web service, providing all the necessary details for a client application to understand how to interact with it.

A WSDL document specifies:

- **What operations** the web service performs (e.g., `getCustomerInfo`, `placeOrder`).
- **What messages** (input and output parameters, data types) are required for each operation.
- **Where the service is located** (its network address/endpoint).

- **How to communicate** with the service (the communication protocols and message formats).

In essence, a client application reads the WSDL document to learn how to call a web service's operations, what data to send, and what data to expect in return.

SOAP (Simple Object Access Protocol)

SOAP (Simple Object Access Protocol) is an XML-based messaging protocol for exchanging structured information in the implementation of web services. It's an application of XML that defines a message format for web services communication.

Key aspects of SOAP:

- **XML-based:** SOAP messages are formatted in XML, making them human-readable and platform-independent.
- **Protocol Neutral:** While commonly used over HTTP/HTTPS, SOAP can be transported over other protocols like SMTP, FTP, or JMS. HTTP is preferred because it works well through firewalls.
- **Standardized Structure:** A SOAP message consists of:
  - **Envelope:** The root element that defines the start and end of the message.
  - **Header (optional):** Contains application-specific information like authentication or transaction IDs.
  - **Body:** Contains the actual message payload, including the request or response information.
  - **Fault (optional):** Used for reporting errors.
- **Message Exchange:** SOAP defines a way for a client to request a service from a server (by sending a SOAP message) and for the server to send back a response (another SOAP message).

Q5) a) Explain various types of Arrays in PHP. Explain each with example code.
Ans:

In PHP, there are primarily three types of arrays: Indexed Arrays, Associative Arrays, and Multidimensional Arrays.

1. Indexed Arrays (Numeric Arrays)

Indexed arrays store elements with a numeric index, starting from 0 by default. Each element is assigned an integer key automatically.

Example:

PHP
```
<?php
$fruits = ["Apple", "Banana", "Cherry"];
echo $fruits[0]; // Output: Apple
?>
```

2. Associative Arrays

Associative arrays store elements with named keys (strings) instead of numeric indices. This allows you to use meaningful labels to refer to elements, making the code more readable.

Example:

PHP
```
<?php
$age = ["Peter" => 35, "Ben" => 37];
echo $age["Peter"]; // Output: 35
?>
```

3. Multidimensional Arrays

Multidimensional arrays are arrays that contain one or more arrays within them. They are used to store data in a table-like format (rows and columns). A two-dimensional array is the most common type, representing a matrix or a table.

Example (Two-dimensional array):

PHP

```php
<?php
$students = [
    [10, "AAA"],
    [20, "BBB"]
];
echo $students[0][1]; // Output: AAA
?>
```

Q5) b) Explain the following:

i) WAP & WML

ii) C# Vs Java

Ans:

i) WAP (Wireless Application Protocol) & WML (Wireless Markup Language)

WAP (Wireless Application Protocol) is a set of standards designed to extend Internet services to mobile phones and Personal Digital Assistants (PDAs). A WAP browser is a web browser specifically for mobile devices that uses this protocol. The mobile network connects to the Internet through a WAP Gateway, which converts WAP requests to web requests and vice-versa. This allows mobile phones to act as clients when connected to the internet, displaying demanded web pages if they are WAP enabled. Since standard HTML pages load slowly on mobile phones, web pages for these devices are written in Wireless Markup Language (WML). The WAP model follows the OSI model and includes an Application Layer called the Wireless Application Environment (WAE), which supports WAP application development using WML.

WML (Wireless Markup Language) is the language used to write web pages specifically for mobile phones and devices enabled with WAP. Unlike HTML, WML sites consist of "cards" instead of pages, where each card is displayed on the screen at one time. More than one card can be inserted into a single WML document.

Example of WML Structure (Cards and Decks): WML documents are structured with `<wml>` as the root element and contain one or more `<card>` elements. Each card is like a screen of content on the mobile device.

XML

```
<?xml version="1.0"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.2//EN"
"http://www.wapforum.org/DTD/wml12.dtd">
<wml>
    <card id="one" title="Welcome Card">
        <p>Hello, This is first card.</p>
    </card>
    <card id="two" title="GoodBye Card">
        <p>Good bye, this is second card.</p>
    </card>
</wml>
```

In WML, it is important to close every opened tag, unlike HTML, otherwise the WML page will not execute. Comments in WML are placed inside ``, similar to HTML, and WAP browsers ignore all comments. The <p> tag defines a paragraph of text, which is always displayed on a new line in WML.

ii) C# Vs Java

C# and Java are both powerful, object-oriented programming languages widely used for developing a variety of applications. While they share many similarities in their syntax and concepts, they also have fundamental differences, primarily stemming from their ecosystems and design philosophies.

Feature	C# (C Sharp)	Java
Origin/Platform	Developed by Microsoft as part of the .NET framework. Primarily runs on the .NET platform (Windows, Linux, macOS via .NET Core).	Developed by Sun Microsystems (now Oracle). Runs on the Java Virtual Machine (JVM).
Execution	Code compiles into Common Intermediate Language (CIL) and is executed by the Common Language Runtime (CLR).	Code compiles into bytecode and is executed by the Java Virtual Machine (JVM).
Key Frameworks	.NET (ASP.NET, WinForms, WPF, Xamarin, Unity).	Java SE, Java EE (Jakarta EE), Android, Spring, Hibernate.
Memory Mgmt.	Automatic garbage collection (handled by CLR).	Automatic garbage collection (handled by JVM).
Syntax	Similar to C++, Java, and JavaScript.	Similar to C++ and C.
Primary OS Focus	Historically Windows-centric, now cross-platform with .NET.	Designed for "Write Once, Run Anywhere" (WORA), inherently cross-platform.
Tooling/IDEs	Visual Studio (Microsoft), Visual Studio Code.	Eclipse, IntelliJ IDEA, NetBeans.
Performance	Generally comparable, C# can sometimes have a slight edge in specific Windows-native scenarios.	Generally comparable, optimized for various platforms.
Market Share	Strong in enterprise applications, game development (Unity), and Windows desktop apps.	Dominant in Android app development, large-scale enterprise systems, and big data.

Q6) a) Explain the PHP with MySQL using example.

Ans:

Connecting PHP with MySQL

Connecting PHP to a MySQL database allows you to dynamically manage and retrieve data for your web applications. The process generally involves establishing a connection, selecting a database, and then performing various database operations like inserting, updating, deleting, or querying data.

Steps Involved in Connecting to MySQL with PHP

1. **Connecting to Database:** You establish a connection to the MySQL server using a PHP function. This function typically requires the server address (e.g., `localhost`), username, and password. For example, the `mysql_connect()` function was traditionally used for this purpose.

   PHP

   ```
   <?php
   // Make a MySQL Connection
   $conn=mysql_connect("localhost:3306/mydb","root","mypassw
   ord"); // [cite: 20]
   if(!$conn) {
       die('error in connection'.mysql_error()); // [cite: 20]
   } else {
       // Connection successful
   }
   ?>
   ```

   Note: `mysql_connect()` and other `mysql_*` functions are deprecated in newer PHP versions. `mysqli` or PDO are recommended for modern PHP development.

2. **Selecting Database:** After establishing a connection to the MySQL server, you need to select the specific database you want to work with.

   PHP

   ```
   <?php
   // Assuming $conn is an established connection
   mysql_select_db("mydb", $conn); // [cite: 21]
   ?>
   ```

Performing Database Operations

Once connected and a database is selected, you can execute SQL queries to manipulate data.

- **Inserting Data:** To add new records to a table, you use the `INSERT` SQL query.

  PHP

```
<?php
// Make a MySQL Connection (as above)
// mysql_select_db("mydb", $conn); //select the database
[cite: 21]

$query="INSERT INTO my_table VALUES(1,'SHILPA')"; // [cite:
7]
mysql_query($query, $conn); //Execution of Query [cite: 7]
mysql_close($conn); //closing the database [cite: 21]
?>
```

  This example inserts a record with `id=1` and `name='SHILPA'` into a table named `my_table`.

- **Deleting Data/Database:** You can delete records from a table using the `DELETE` query or even an entire database using the `DROP DATABASE` query.

  PHP

```
<?php
// Make a MySQL Connection (as above)
// mysql_select_db("mydb", $conn); //select the database
[cite: 21]

// Delete a record
$query="DELETE FROM my_table WHERE id=1"; // [cite: 21]
mysql_query($query, $conn); //Execution of Query [cite: 21]

// Delete an entire database
// $query="DROP DATABASE mydb"; // [cite: 21]
// mysql_query($query, $conn); //Execution of Query [cite:
21]

mysql_close($conn); //closing the database [cite: 21]
?>
```

Q6) b) Write note on:

i) Session tracking in PHP

ii) NET framework

iii) NodeJS

Ans:

i) Session Tracking in PHP

Session tracking in PHP is a mechanism to maintain state information about a user across multiple page requests. Since HTTP is a stateless protocol, each request is treated independently. Sessions allow web applications to "remember" a user as they navigate through different pages.

How PHP Sessions Work:

1. **Session Start:** When a user visits a PHP page that calls `session_start()`, PHP generates a unique session ID.
2. **Session ID Transmission:** This session ID is then typically sent to the client's browser as a **cookie** (usually named `PHPSESSID`). If cookies are disabled, PHP can fall back to **URL rewriting**, where the session ID is appended to all URLs.
3. **Session Data Storage:** On the server, PHP creates a session file (or uses another storage mechanism like a database) associated with this session ID. This file stores session variables (e.g., `$_SESSION['username'] = 'JohnDoe'`).
4. **Subsequent Requests:** For every subsequent request from the same user, the browser sends back the session ID (via cookie or URL). PHP retrieves the corresponding session file, making the session variables available to the script.
5. **Session End:** Sessions typically expire after a certain period of inactivity or when explicitly destroyed using `session_destroy()`.

Key PHP Functions:

- `session_start()`: Initiates a session or resumes an existing one.
- `$_SESSION`: A superglobal array used to store and access session variables.
- `session_unset()`: Frees all session variables.
- `session_destroy()`: Destroys all data registered to a session.

ii) .NET Framework

The .NET Framework is a software development platform developed by Microsoft. It's a comprehensive and consistent programming model for building applications that run on Windows. It consists of a large class library called the Framework Class Library (FCL) and a runtime environment called the Common Language Runtime (CLR).

Components and Concepts:

- **Common Language Runtime (CLR):** This is the execution engine of the .NET Framework. It provides services such as garbage collection (automatic memory management), security, type safety, and exception handling. Code written for the .NET Framework is compiled into an intermediate language called **Common Intermediate Language (CIL)** (formerly MSIL), which the CLR then compiles into native machine code at runtime (Just-In-Time compilation).
- **Framework Class Library (FCL):** A vast library of reusable classes, interfaces, and value types that provide access to system functionality. It includes classes for basic data types, file I/O, database access (ADO.NET), web application development (ASP.NET), Windows desktop application development (WinForms, WPF), and more.
- **Language Interoperability:** A key feature of .NET is its language interoperability. Developers can write code in various .NET-compatible languages (like C#, VB.NET, F#) and have them interact seamlessly because they all compile down to CIL.

iii) Node.js

Node.js is an open-source, cross-platform JavaScript runtime environment that allows developers to execute JavaScript code outside a web browser. It's built on Chrome's V8 JavaScript engine and is widely used for building fast, scalable network applications, especially backend services (APIs).

Key Concepts and Features:

- **JavaScript Everywhere:** Node.js enables full-stack JavaScript development, meaning both frontend and backend can be written in JavaScript, leading to code sharing and simplified development.
- **Asynchronous, Event-Driven Architecture:** Node.js operates on a single-threaded, non-blocking I/O model. This means that instead of blocking the execution for I/O operations (like database queries or file reads), it registers callbacks and continues processing other requests. Once the I/O operation completes, the callback is executed. This makes Node.js highly efficient for applications with many concurrent connections.
- **npm (Node Package Manager):** Node.js comes with npm, the world's largest ecosystem of open-source libraries. npm simplifies package management, allowing developers to easily install, share, and manage project dependencies.
- **Scalability:** Its non-blocking nature makes Node.js very efficient for handling high concurrency, making it suitable for real-time applications (e.g., chat apps, streaming services), APIs, and microservices.
- **Common Use Cases:** Building RESTful APIs, microservices, real-time applications (chat, gaming), data streaming, server-side rendering of single-page applications, and command-line tools.

Q7) a) Explain Ruby with its advantages. Explain control statements in Ruby.
Ans:

Ruby

Ruby is an open-source, dynamic, object-oriented programming language known for its simplicity and productivity. It was designed to be a "programmer's best friend," focusing on developer happiness and ease of use. It emphasizes convention over configuration, which can speed up development, especially when combined with frameworks like Ruby on Rails.

Features and Philosophy:

- **Pure Object-Oriented:** In Ruby, everything is an object, including basic data types like numbers and booleans. This provides a consistent and unified programming model.
- **Dynamic and Interpreted:** Ruby is dynamically typed (you don't declare variable types) and interpreted, which allows for rapid prototyping and flexibility.
- **Metaprogramming:** Ruby has powerful metaprogramming capabilities, allowing code to write code at runtime, which is a key enabler for frameworks like Rails.
- **Readable Syntax:** Ruby's syntax is often described as natural and easy to read, resembling spoken English.

Advantages of Ruby:

1. **Developer Productivity:** Ruby's concise syntax, object-oriented nature, and rich standard library contribute to faster development cycles. The "convention over configuration" principle, especially in frameworks like Ruby on Rails, means less boilerplate code.
2. **Rich Ecosystem:** Ruby boasts a vast collection of open-source libraries called "Gems" (managed by RubyGems), which provide solutions for almost any development need, from web development to data processing.
3. **Active Community:** It has a large, vibrant, and supportive community of developers, leading to extensive documentation, tutorials, and quick problem-solving.
4. **Flexibility:** Ruby is highly flexible, allowing developers to modify language features or add new ones. This power comes with responsibility but enables creative solutions.
5. **High Readability:** The language's syntax is designed to be expressive and natural, making Ruby code generally easier to read and understand, which aids in maintenance.
6. **Full-Stack Capabilities:** While famously known for web development with Ruby on Rails, Ruby can also be used for scripting, data analysis, automation, and desktop applications.

Control Statements in Ruby

Control statements (or control flow statements) in Ruby allow you to control the order in which instructions are executed based on certain conditions or for repetitive tasks.

1. Conditional Statements (Selection)

These execute a block of code if a specified condition is true.

- **`if` / `elsif` / `else` / `end`:** Executes code conditionally.

  Ruby

  ```
  age = 20
  if age >= 18
    puts "Eligible to vote"
  elsif age >= 16
    puts "Can get a learner's permit"
  else
    puts "Too young"
  end
  ```

- **`unless` / `else` / `end`:** Executes code if a condition is *false* (opposite of `if`).

  Ruby

  ```
  is_sunny = false
  unless is_sunny
    puts "Bring an umbrella"
  else
    puts "Enjoy the sun!"
  end
  ```

- **`case` / `when` / `else` / `end`:** Used for multiple conditions, providing a cleaner alternative to nested `if/elsif`.

grade = 'B' case grade when 'A' puts "Excellent!" when 'B' puts "Good job!" when 'C' puts "Needs improvement." else puts "Invalid grade" end ```

2. Looping Statements (Iteration)

These execute a block of code repeatedly.

- **`while` / `end`:** Executes a block of code as long as a condition is true.

  Ruby

  ```
  count = 0
  while count < 3
  ```

```
  puts "Count: #{count}"
  count += 1
end
# Output:
# Count: 0
# Count: 1
# Count: 2
```

- **`until` / `end`:** Executes a block of code as long as a condition is *false*.

  Ruby

```
num = 5
until num == 0
  puts "Num: #{num}"
  num -= 1
end
# Output:
# Num: 5
# Num: 4
# Num: 3
# Num: 2
# Num: 1
```

- **`for` / `in` / `end`:** Iterates over a range or collection.

  Ruby

```
for i in 1..3
  puts "Iteration #{i}"
end
# Output:
# Iteration 1
# Iteration 2
# Iteration 3
```

- **`each` (Iterator):** A common and idiomatic way to iterate over collections (arrays, hashes, ranges).

  Ruby

```
[10, 20, 30].each do |number|
  puts "Number: #{number}"
end
```

```
# Output:
# Number: 10
# Number: 20
# Number: 30
```

- **`loop` / `do` / `end`:** Creates an infinite loop that must be explicitly broken out of (e.g., with `break`).

  Ruby

```
i = 0
loop do
  puts "Looping... #{i}"
  i += 1
  break if i > 2
end
# Output:
# Looping... 0
# Looping... 1
# Looping... 2
```

3. Loop Control Keywords

These alter the flow of loops.

- **`break`:** Terminates the loop immediately.
- **`next`:** Skips the rest of the current iteration and moves to the next iteration.
- **`redo`:** Restarts the current loop iteration without re-evaluating the loop's condition.

Q7) b) Explain EJB concept & five basic example of using EJB.

Ans:

EJB (Enterprise JavaBeans) Concept (Brief)

EJB (Enterprise JavaBeans) is a server-side component architecture for building modular, scalable, and secure enterprise Java applications. It allows developers to focus on business logic while the EJB container handles complex services like transaction management, security, concurrency, and persistence. This simplifies the development of distributed applications.

Benefits: Scalability, transaction management, security, and reusability are key advantages.

Types of EJB (Brief)

1. **Session Beans:** Represent business processes or tasks.
   - **Stateless:** Don't maintain client-specific state across calls (e.g., a calculator service).
   - **Stateful:** Maintain client-specific state across multiple calls (e.g., a shopping cart).
2. **Entity Beans:** Represent persistent data in a database (e.g., a Customer record).
   - **BMP (Bean-Managed Persistence):** Developer handles database code.
   - **CMP (Container-Managed Persistence):** Container handles database code automatically.
3. **Message-Driven Beans (MDBs):** Handle asynchronous messages (e.g., processing orders from a queue).

Basic Examples of Using EJB (Brief)

Here are five concise examples illustrating EJB usage:

1. **Stateless Session Bean (Service Example):**
   - **Concept:** A single instance can serve multiple clients without remembering prior interactions. Good for general services.
   - **Use Case:** A "GreetingService" that returns "Hello, [name]!".
2. **Stateful Session Bean (Account Example):**
   - **Concept:** Maintains a unique state for each client, remembering conversational data.
   - **Use Case:** A "BankAccount" bean where each client has their own balance that changes with `deposit()` and `withdraw()` calls.

3. **Client Accessing EJB (from a Servlet):**
   - **Concept:** A web component (like a Servlet) gets a reference to an EJB instance and calls its methods.
   - **Example:** A servlet getting an `Account` EJB and calling `deposit()` or `getBalance()`.
4. **Remote Interface Usage:**
   - **Concept:** Used when the client application is running on a different server or JVM than the EJB. Communication happens over a network.
   - **Example:** A separate desktop application calling a remote EJB deployed on a distant application server.
5. **Local Interface Usage:**
   - **Concept:** Used when the client and the EJB are deployed within the same application server/JVM. Communication is direct and faster.
   - **Example:** A JSP or another Servlet within the same web application calling a local EJB.

Q8) a) Explain the arrays in Ruby. Explain Rails with AJAX.

Ans:

Arrays in Ruby (Concise)

In Ruby, an **Array** is an ordered, zero-indexed collection of items. Arrays can hold different types of data and automatically resize.

Example:

Ruby

```
# Creating an array
colors = ["red", "green", "blue"]

# Accessing elements
puts colors[0]  # Output: red
puts colors[-1] # Output: blue (last element)

# Adding elements
colors << "yellow" # Adds "yellow" to the end
puts colors.inspect # Output: ["red", "green", "blue",
"yellow"]

# Iterating
colors.each { |color| puts color.upcase }
# Output:
# RED
# GREEN
# BLUE
# YELLOW
```

Rails with AJAX (Concise)

Ruby on Rails simplifies creating dynamic web applications using **AJAX** (Asynchronous JavaScript and XML). Rails uses Unobtrusive JavaScript (UJS) to send background requests and update parts of a page without a full reload.

How it works (Simplified):

1. **HTML Setup:** Add `data-remote="true"` to a form or link in your HTML.

HTML

```
<%= form_with url: '/products', remote: true do
|form| %>
  <%= form.text_field :name %>
  <%= form.submit "Create Product" %>
<% end %>
```

2. **Controller Action:** In your Rails controller, handle the request and respond with a specific format (e.g., JavaScript).

app/controllers/products_controller.rb class ProductsController < ApplicationController def create @product = Product.new(params.require(:product).permit(:name)) @product.save respond_to do |format| format.html { redirect_to products_path } # For non-AJAX format.js # For AJAX, renders create.js.erb end end end ```

3. **JavaScript Response:** Rails executes a `.js.erb` file if `format.js` is rendered. This JavaScript updates the page.

   Code snippet

```
// app/views/products/create.js.erb
// This JavaScript runs after the form is submitted
via AJAX
$("#products_list").append("<li><%=
escape_javascript(@product.name) %></li>");
$("#product_name").val(''); // Clear the input field
```

Q8) b) Explain Document Request in Rails.

Ans:

Document Request in Rails

In a Ruby on Rails application, a "document request" (which essentially means a request for a web page or a specific resource) follows a well-defined flow, primarily leveraging the Model-View-Controller (MVC) architectural pattern. When a user's browser makes a request for a URL in a Rails application, the request goes through several layers to eventually render the desired HTML document (or other formats like JSON, XML).

1. **Web Server (e.g., Puma, Passenger, Nginx/Apache with a Rails adapter):**
   - The client's browser sends an HTTP request (e.g., `GET /products/1`).
   - The web server receives this request and forwards it to the Rails application server.
2. **Rack:**
   - Rails applications run on Rack, a minimal interface between web servers and Ruby frameworks. Rack translates the raw HTTP request into a Ruby hash (containing request parameters, headers, etc.) and expects a three-element array (status code, headers, body) as a response.
3. **Router (`config/routes.rb`):**
   - The Rails router is the first stop within the Rails application. It inspects the incoming URL and HTTP method (GET, POST, PUT, DELETE).
   - It then matches the request against the defined routes in `config/routes.rb` and determines which **controller action** should handle the request.
   - For example, a request to `/products/1` (GET) might be routed to `ProductsController#show` (meaning the `show` method within the `ProductsController`).
4. **Controller (e.g., `ProductsController`):**
   - The chosen controller action is invoked.
   - The controller's responsibility is to:
     - **Process Request Parameters:** Extract data sent from the browser (e.g., `params[:id]` for `/products/1`).
     - **Interact with the Model:** Perform business logic. For example, it might fetch data from the database using an ActiveRecord model (`@product = Product.find(params[:id])`).
     - **Prepare Data for the View:** Assign instance variables (e.g., `@product`) that will be accessible in the view.

- **Choose the View:** Implicitly (by default, Rails renders a view with the same name as the action) or explicitly, the controller determines which view template should be rendered.

5. **View (e.g., `app/views/products/show.html.erb`):**
   - The view template (typically an `.html.erb` file) receives the data prepared by the controller (via instance variables).
   - It contains the HTML structure and uses ERB (Embedded Ruby) tags (`<%= %>` for output, `<% %>` for logic) to embed dynamic data and control presentation.
   - The view's primary job is to generate the final HTML (or other format) that will be sent back to the client.
6. **Response:**
   - Once the view has rendered the HTML, the Rails application packages this HTML (along with HTTP headers like Content-Type, status code 200 OK) into an HTTP response.
   - This response is sent back through Rack to the web server.
7. **Web Server to Client:**
   - The web server sends the HTTP response back to the client's browser.
   - The browser then renders the HTML document, displaying the web page to the user.

Q8) c) Explain advantages of Ruby and Rails.

Ans:

Advantages of Ruby and Ruby on Rails

Ruby and its popular web framework, Ruby on Rails, offer several significant advantages for developers and businesses:

Advantages of Ruby (the language):

1. **Developer Productivity and Happiness:**
   - **Concise and Expressive Syntax:** Ruby's syntax is often described as natural and human-friendly, reducing the amount of code needed to achieve a task. This leads to faster development.
   - **Object-Oriented Everything:** Its pure object-oriented nature provides consistency and makes it easy to model real-world problems.
   - **Dynamic Nature:** Ruby's dynamic typing and interpreted nature allow for rapid prototyping and flexibility during development.
2. **Rich Ecosystem:**
   - **RubyGems:** A vast collection of open-source libraries (gems) is available, offering solutions for almost any programming need, which saves development time.
3. **Strong Community Support:**
   - Ruby has a large, active, and supportive community, leading to extensive documentation, tutorials, and readily available help.
4. **Flexibility and Metaprogramming:**
   - Ruby is highly flexible, allowing developers to modify or extend its core functionalities. Its metaprogramming capabilities enable powerful abstractions and code generation at runtime.

Advantages of Ruby on Rails (the framework):

1. **Rapid Application Development (RAD):**
   - **Convention Over Configuration (CoC):** Rails minimizes the need for explicit configuration by providing smart defaults and conventions, allowing developers to get started quickly and build features rapidly.
   - **Don't Repeat Yourself (DRY):** Rails encourages writing code only once, which reduces redundancy, improves maintainability, and makes the codebase cleaner.
   - **Generators:** Built-in generators automatically create boilerplate code (e.g., models, controllers, views), significantly speeding up development.

2. **MVC Architecture:**
    - Rails enforces the Model-View-Controller (MVC) architectural pattern, which promotes a clear separation of concerns, making applications modular, scalable, and easier to maintain.
3. **Full-Stack Capabilities:**
    - Rails is a full-stack framework, providing everything needed to build both frontend (using ERB, JavaScript, CSS) and backend (database interactions, routing, API creation) components of a web application.
4. **Integrated Tooling:**
    - Rails comes with a powerful command-line interface (CLI) that simplifies tasks like database migrations, testing, and application deployment.
5. **Active Record ORM:**
    - Its built-in Object-Relational Mapping (ORM) library, Active Record, makes database interactions intuitive and object-oriented, abstracting away complex SQL queries.
6. **Strong Security Features:**
    - Rails includes built-in security features to guard against common web vulnerabilities like Cross-Site Request Forgery (CSRF) and SQL injection.
7. **Scalability:**
    - While sometimes questioned for raw performance compared to other frameworks in specific benchmarks, Rails applications are highly scalable, especially when deployed with proper architecture (e.g., using caching, background jobs, horizontal scaling).