JAVA Programming

Unit-I: Java Programming Language

🚀 1. Introduction to Java Programming

🧠 What is Java?

- Java is a **high-level, object-oriented, class-based, platform-independent** programming language.
- Developed by **Sun Microsystems (1995)**, now owned by **Oracle**.
- **"Write Once, Run Anywhere" (WORA)** – Java code runs on JVM (Java Virtual Machine).

🧱 Java Code Structure

```
public class Hello {
   public static void main(String[] args) {
      System.out.println("Hello, Java!");
   }
}
```

- class Hello: Class declaration.
- main: Entry point of Java program.
- System.out.println: Prints to console.

🔧 Compilation & Execution

```
javac Hello.java     # Compiles to Hello.class
java Hello           # Runs via JVM
```

🔣 2. Operators, Data Types, Variables

✳️ Primitive Data Types

Type	Size	Example
int	4 bytes	int x = 10;

Type	Size	Example
float	4 bytes	float y = 3.4f;
double	8 bytes	double d = 5.6;
char	2 bytes	char c = 'A';
boolean	1 bit	boolean flag = true;

🔧 Operators

- **Arithmetic**: + - * / %
- **Relational**: == != > < >= <=
- **Logical**: && || !
- **Assignment**: = += -= *=
- **Increment/Decrement**: ++ --
- **Bitwise**: & | ^ ~ << >>

🔁 3. Control Statements

📌 Conditional:

```
if (a > b) {
  // code
} else if (...) {
  // code
} else {
  // code
}
```

🔄 Looping:

```
for (int i = 0; i < 5; i++) {...}
while (condition) {...}
do {...} while (condition);
```

💥 Jump:

break; continue; return;

📦 4. Classes, Objects, Methods

🧱 Class and Object

```
class Car {
   String color = "Red";
   void drive() {
      System.out.println("Driving...");
   }
}

public class Main {
   public static void main(String[] args) {
      Car c1 = new Car();  // Object
      c1.drive();
   }
}
```

📞 Methods

```
int add(int a, int b) {
  return a + b;
}
```

🧯 5. Packages & Interfaces

📦 Packages

```
package mypack;
public class MyClass { ... }
```

- Import: import mypack.MyClass;

🔌 Interface

```
interface Animal {
    void makeSound();
}

class Dog implements Animal {
    public void makeSound() {
        System.out.println("Bark");
    }
}
```

🧬 6. Inheritance & Polymorphism

👨‍👦 Inheritance

```
class Animal {
    void sound() { System.out.println("Animal sound"); }
}

class Dog extends Animal {
    void sound() { System.out.println("Dog barks"); }
}
```

🔁 Polymorphism (Overloading & Overriding)

```
// Overloading (same method, different params)
int sum(int a, int b)
int sum(int a, int b, int c)

// Overriding (child class modifies parent method)
```

⚠️ 7. Exception Handling

```
try {
   int x = 5 / 0;
} catch (ArithmeticException e) {
   System.out.println("Cannot divide by zero");
} finally {
   System.out.println("Always executed");
}
```

- Types: try, catch, finally, throw, throws

🔀 8. Multithreading

```
class MyThread extends Thread {
   public void run() {
      System.out.println("Thread running");
   }
}

MyThread t1 = new MyThread();
t1.start();
```

- You can extend Thread or implement Runnable.
- Important methods: start(), run(), sleep(), join(), yield()

💾 9. Java IO (java.io)

File Reading Example:

```
BufferedReader br = new BufferedReader(new FileReader("file.txt"));
String line = br.readLine();
```

File Writing Example:

```
BufferedWriter bw = new BufferedWriter(new FileWriter("file.txt"));
bw.write("Hello File");
```

```
bw.close();
```

🌐 10. Java Applet

```
import java.applet.*;
import java.awt.*;

public class HelloApplet extends Applet {
   public void paint(Graphics g) {
      g.drawString("Hello Applet", 20, 20);
   }
}
```

⚠️ Applets are now outdated in modern development, used only for academic understanding.

📝 11. String Handling

```
String s1 = "Hello";
String s2 = new String("World");
System.out.println(s1 + " " + s2);
System.out.println(s1.length());
```

- Useful methods: .length(), .charAt(), .substring(), .equals(), .compareTo(), .indexOf()

🌍 12. Networking in Java

```
Socket s = new Socket("localhost", 8080);
DataOutputStream dout = new DataOutputStream(s.getOutputStream());
dout.writeUTF("Hello Server");
```

- Classes: Socket, ServerSocket, DatagramSocket, InetAddress

🖱️ 13. Event Handling (AWT or Swing)

```
Button b = new Button("Click");
b.addActionListener(new ActionListener() {
   public void actionPerformed(ActionEvent e) {
      System.out.println("Clicked!");
   }
});
```

Unit-II: Introduction to AWT (Abstract Window Toolkit)

Syllabus Topics

- AWT Controls, Graphics, Layout Manager and Menus
- Images, Additional Packages
- Java Swing: Applet, Application
- Programming using Panes, Pluggable Look and Feel
- Labels, TextFields, Buttons, Toggle Buttons, Checkboxes, Radio Buttons
- Viewports, Scroll Panes, Scroll Bars, Lists, Combo Box, Progress Bar
- Menus, Toolbars, Layered Panes, Tabbed Panes, Split Panes, Layouts, Windows, Dialog Boxes

---

🖼️ 1. AWT Basics

🧠 What is AWT?

- **AWT (Abstract Window Toolkit)** is Java's **original GUI** (Graphical User Interface) toolkit.
- Part of the **java.awt** package.
- Platform-dependent (uses native OS components – heavy-weight).

✅ Common AWT Controls:

Component	Description
Label	Displays text

Component	Description
Button	Clickable button
TextField	Single-line input field
TextArea	Multi-line input field
Checkbox	Checkbox control
Choice	Drop-down
List	List of items

🔧 2. AWT Example

```
import java.awt.*;
public class MyAWT {
   public static void main(String[] args) {
      Frame f = new Frame("AWT Example");
      Label l = new Label("Name:");
      TextField tf = new TextField();
      Button b = new Button("Submit");

      l.setBounds(50, 50, 100, 30);
      tf.setBounds(150, 50, 150, 30);
      b.setBounds(100, 100, 80, 30);

      f.add(l); f.add(tf); f.add(b);
      f.setSize(400, 200);
      f.setLayout(null);
      f.setVisible(true);
   }
}
```

🧱 3. Layout Managers

Control how components are arranged.

Layout Manager	Description
FlowLayout	Left to right
BorderLayout	North, South, East, West, Center
GridLayout	Grid-like layout
CardLayout	Multiple panels, one visible at a time
GridBagLayout	Complex layout with grid constraints

🎨 4. AWT Graphics

```
public void paint(Graphics g) {
   g.drawString("Hello", 100, 100);
   g.drawLine(10, 10, 200, 200);
   g.drawRect(50, 50, 100, 50);
   g.setColor(Color.RED);
}
```

🍃 5. Java Swing (Better than AWT)

🧠 Why Swing?

- Built on AWT (extends functionality).
- **Lightweight** (doesn't depend on OS).
- More modern, customizable.

💡 Common Swing Classes:

Component	Description
JFrame	Top-level window

Component	Description
JPanel	Container for UI
JLabel	Display text/image
JButton	Button
JTextField	Single-line input
JTextArea	Multi-line input
JCheckBox	Checkbox
JRadioButton	Radio button

🧪 6. Swing Example

```
import javax.swing.*;

public class MySwing {
    public static void main(String[] args) {
        JFrame f = new JFrame("Swing Example");
        JLabel l = new JLabel("Name:");
        JTextField tf = new JTextField();
        JButton b = new JButton("Submit");

        l.setBounds(50, 50, 100, 30);
        tf.setBounds(150, 50, 150, 30);
        b.setBounds(100, 100, 80, 30);

        f.add(l); f.add(tf); f.add(b);
        f.setSize(400, 200);
        f.setLayout(null);
        f.setVisible(true);
```

```
        f.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    }
}
```

🧩 7. Advanced Swing Components

Component	Description
JComboBox	Drop-down list
JList	List of items
JMenuBar	Menu bar
JToolBar	Tool buttons
JTabbedPane	Tabs (like browsers)
JSplitPane	Split area (resizable)
JProgressBar	Progress indicator
JDialog	Popup dialog window

💄 8. Pluggable Look and Feel (PLAF)

```
UIManager.setLookAndFeel(UIManager.getSystemLookAndFeelClassName());
```

Swing supports dynamic theme changes at runtime.

🖼️ 9. Java Viewport and Scroll Components

Component	Purpose
JScrollPane	Adds scroll to panel/component
JViewport	Manages visible area of component
JScrollBar	Scrollbars manually added

📚 10. Swing Layouts

Layout	Behavior
BorderLayout	North, South, East, West, Center
FlowLayout	Left to right
GridLayout	Grid format
BoxLayout	Vertical or horizontal box
GroupLayout	Used in NetBeans GUI builder

🪟 11. Dialog Boxes and Windows

```
JOptionPane.showMessageDialog(null, "Hello!");
String input = JOptionPane.showInputDialog("Enter name:");
```

- JOptionPane: Pre-built dialogs for input/output.
- JDialog: Custom dialog window.

🔁 AWT vs Swing

Feature	AWT	Swing
Lightweight	❌ Heavyweight	✅ Lightweight
OS Dependent	✅ Yes	❌ No
Customizable	❌ Limited	✅ Highly customizable
Package	java.awt.*	javax.swing.*

✅ Summary Cheat Sheet

Component	AWT	Swing Equivalent
Label	Label	JLabel
Text Input	TextField	JTextField
Button	Button	JButton

Component	AWT	Swing Equivalent
Checkbox	Checkbox	JCheckBox
Radio Button	Not native	JRadioButton
List	List	JList
Scroll	Scrollbar	JScrollPane
Dialog	Dialog	JOptionPane/JDialog

Unit-III: JDBC + Servlets

Syllabus Topics

- JDBC: Connectivity Model, JDBC/ODBC Bridge, MySQL Integration
- JDBC Driver Setup, Create Connection, Statements, Login/Logout, Insert/Update/Delete
- **Servlets**: Lifecycle, Containers, Configs, Parameters, Form Data, GET/POST
- HTML Forms, Servlet Handling

🧩 Part 1: JDBC (Java Database Connectivity)

🔌 What is JDBC?

JDBC is an API in Java used to connect **Java applications to relational databases** (like MySQL, PostgreSQL).

🚦 JDBC Architecture

- **DriverManager** → Manages DB drivers
- **Connection** → Connects to DB
- **Statement/PreparedStatement** → Executes SQL queries
- **ResultSet** → Handles SQL results

🔗 JDBC Connection Example (MySQL)

```
import java.sql.*;

public class JDBCExample {
    public static void main(String[] args) {
        try {
            // 1. Load driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2. Create connection
            Connection con = DriverManager.getConnection(
                "jdbc:mysql://localhost:3306/testdb", "root", "password");

            // 3. Create statement and execute query
            Statement stmt = con.createStatement();
            ResultSet rs = stmt.executeQuery("SELECT * FROM users");

            // 4. Process result
            while (rs.next()) {
                System.out.println(rs.getInt(1) + " " + rs.getString(2));
            }

            con.close();
        } catch (Exception e) {
            System.out.println(e);
        }
    }
}
```

🔑 JDBC CRUD Operations

Operation	Method Used
Create	INSERT INTO
Read	SELECT
Update	UPDATE
Delete	DELETE

🔐 Login/Logout System Using JDBC

You can validate login credentials using a SELECT query:

```
PreparedStatement ps = con.prepareStatement("SELECT * FROM users WHERE email=? AND password=?");

ps.setString(1, email);

ps.setString(2, password);

ResultSet rs = ps.executeQuery();

if(rs.next()) {

   // Login success

}
```

🌐 Part 2: Java Servlets

A **Servlet** is a Java class that handles **HTTP requests and responses**. Runs on a Java web server like Apache Tomcat.

🔁 Servlet Lifecycle

1. **init()** – Initializes servlet (runs once)
2. **service()** – Handles requests (runs for every request)
3. **destroy()** – Destroys servlet (before removal)

📦 Servlet Packages

```
import javax.servlet.*;
import javax.servlet.http.*;
```

🛠️ Basic Servlet Example

```
public class HelloServlet extends HttpServlet {
    public void doGet(HttpServletRequest req, HttpServletResponse res) throws IOException {
        res.setContentType("text/html");
        PrintWriter out = res.getWriter();
        out.println("<h1>Hello from Servlet</h1>");
    }
}
```

🌍 Deploying Servlets

You deploy Servlets on a **Servlet Container** (e.g., Tomcat) and map them via web.xml or @WebServlet annotation.

```
@WebServlet("/hello")
public class HelloServlet extends HttpServlet { ... }
```

📝 HTML Form + Servlet Example (POST)

form.html

```
<form action="login" method="post">
    Email: <input type="text" name="email" />
    Password: <input type="password" name="password" />
    <input type="submit" />
</form>
```

LoginServlet.java

```
@WebServlet("/login")
public class LoginServlet extends HttpServlet {
```

```
    protected void doPost(HttpServletRequest req, HttpServletResponse res) throws
IOException {
        String email = req.getParameter("email");
        String pass = req.getParameter("password");

        PrintWriter out = res.getWriter();
        if (email.equals("admin") && pass.equals("1234")) {
            out.println("Login Success");
        } else {
            out.println("Login Failed");
        }
    }
}
```

🔧 Servlet Config vs Context

Feature	ServletConfig	ServletContext
Scope	Single Servlet	Whole App
Purpose	Init parameters	App-wide configuration
Access	getServletConfig()	getServletContext()

📜 Handling Form Data (GET & POST)

- Use request.getParameter("name") to get values.
- For GET: data appears in URL.
- For POST: secure, used for login, file uploads, etc.

📦 Summary Flow

HTML Form -> Servlet (HttpServlet) -> JDBC -> DB (MySQL)

- You can also send responses back to the browser (e.g., success messages, redirects, etc.)

📌 **Industry Practices**

Task	Recommendation
DB Connections	Use **Connection Pooling (HikariCP, DBCP)**
Servlet Mapping	Use @WebServlet instead of web.xml
Data Validation	Do both **frontend** and **backend**
Login Systems	Use **Hashing (bcrypt)** and session mgmt

Unit-IV: JavaServer Pages (JSP)

Syllabus Topics

- Introduction to JSP, JSP Life Cycle
- JSP Expressions, Declarations, Directives & Actions
- Implicit Objects
- JSP Tag Libraries (Standard + Custom)
- Expression Language (EL)

🧩 **What is JSP?**

JavaServer Pages (JSP) is a technology that helps you create **dynamic web pages** using **HTML + Java**.

- Runs on the **server**.
- Embedded Java code inside HTML (<% %>).
- Used as the **view** layer in MVC pattern.

🧬 **JSP Life Cycle**

1. **Translation** → JSP file is converted into a Servlet.

2. **Compilation** → Servlet is compiled into .class.
3. **Initialization** → jspInit() method is called.
4. **Request Processing** → jspService() handles each request.
5. **Destruction** → jspDestroy() called on shutdown.

🖍️ JSP Syntax Overview

➤ Expressions

Used to output data.

```
<%= "Hello " + request.getParameter("name") %>
```

➤ Scriptlets

Java code inside HTML.

```
<%
  int a = 10;
  out.println("A = " + a);
%>
```

➤ Declarations

Declare variables/methods.

```
<%! int counter = 0; %>
<%! public int add(int x, int y) { return x + y; } %>
```

📦 JSP Implicit Objects

Object	Description
request	Data from client
response	Send data to client
session	User session data
application	App-wide data

Object	Description
out	To print on browser
config	Servlet config info
pageContext	Context for the JSP page
exception	Exception object (only in error pages)
page	Current JSP page (this)

📄 **Directives**

➤ **Page Directive**

<%@ page language="java" contentType="text/html" %>

➤ **Include Directive**

<%@ include file="header.jsp" %> <!-- Static include -->

➤ **Taglib Directive**

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

⚙️ **JSP Actions**

➤ **jsp:include**

Dynamic include.

<jsp:include page="footer.jsp" />

➤ **jsp:forward**

Forward request.

<jsp:forward page="nextPage.jsp" />

➤ **jsp:param**

Pass parameter to included/forwarded JSP.

<jsp:include page="info.jsp">

  <jsp:param name="username" value="ghanshyam"/>

</jsp:include>

✨ **Expression Language (EL)**

A cleaner way to access data without Java code.

➤ **EL Syntax**

${variableName}

${requestScope.name}

${param.username}

${sessionScope.user}

➤ **Example:**

Welcome ${param.name}

🧱 **JSP Tag Libraries**

➤ **Standard Tag Library (JSTL)**

Add logic using tags instead of Java.

Import JSTL Core:

<%@ taglib uri="http://java.sun.com/jsp/jstl/core" prefix="c" %>

Examples:

<c:if test="${user == 'admin'}">

  Welcome Admin

</c:if>

<c:forEach var="item" items="${items}">

  ${item}

</c:forEach>

🎯 **JSP vs Servlet**

Feature	Servlet	JSP
Code Style	Pure Java	Java in HTML
Best For	Logic (Controller)	View (UI rendering)
Life Cycle	More complex	Simpler, auto-managed
Compilation	Manually written Java file	Compiled automatically

---

👨‍💻 College Mini Example

login.jsp

```
<form action="LoginServlet" method="post">
  Email: <input type="text" name="email" />
  Password: <input type="password" name="password" />
  <input type="submit" />
</form>
```

afterLogin.jsp

```
<%@ page session="true" %>
<% String name = (String) session.getAttribute("name"); %>
<h2>Welcome <%= name %></h2>
```

🏭 Industry-Level Practices

Practice	Recommendation
Logic in JSP	Avoid – use MVC (Servlet → JSP)
Data in EL	Use EL and JSTL, not scriptlets
Session Management	Use HTTPS, tokens, and session validation
JSP Use Case	Frontend rendering only, logic in backend

Unit-V: Session Tracking and Enterprise JavaBeans (EJB)

Syllabus Topics

- **Session Tracking Techniques**: Cookies, URL Rewriting, Hidden Fields, HttpSession
- **Overview of EJB**
- **Types of EJB**: Session Bean, Entity Bean, Message-Driven Bean

---

🔐 SESSION TRACKING

💡 Why Session Tracking?

HTTP is **stateless**. Every request is treated as **new**. Session tracking lets us maintain **user state** across multiple requests (e.g., login sessions, shopping carts).

🌐 Techniques of Session Tracking

1. Cookies

Small pieces of data stored in the **browser** and sent back with every request.

Server Side (Java):

Cookie cookie = new Cookie("username", "ghanshyam");

response.addCookie(cookie);

Retrieve:

Cookie[] cookies = request.getCookies();

---

2. URL Rewriting

Appends session info to the **URL**.

Example:

Dashboard

- Works when cookies are disabled.
- Insecure for sensitive data.

---

3. Hidden Form Fields

Stores session info in hidden <input> fields inside forms.

```
<form action="nextPage.jsp">
  <input type="hidden" name="username" value="ghanshyam">
</form>
```

- Requires form submission.
- Not visible to user.

---

4. HttpSession (Recommended)

Server-side session management.

Create or retrieve session:

```
HttpSession session = request.getSession();
session.setAttribute("user", "ghanshyam");
```

Get value later:

```
String user = (String) session.getAttribute("user");
```

🧪 **Comparison Table**

Technique	Data Location	Visibility	Security	Recommended
Cookies	Client	Visible	Medium	✅
URL Rewriting	Client (URL)	Visible	Low	❌
Hidden Fields	Client (form)	Hidden	Medium	❌
HttpSession	Server	Hidden	High	✅ ✅ ✅

🧱 **EJB (Enterprise JavaBeans) — College Level + Intro to Industry**

EJB is a **server-side component** architecture in Java EE used to build **scalable, secure, transactional** enterprise apps.

🧩 **Types of EJB**

Type	Purpose
Session Bean	Business logic
Entity Bean	Represents data in DB (now deprecated)
Message-Driven Bean	Handles JMS messages

---

1. Session Beans

Handles business logic.

- **Stateless**: No client data stored (e.g., calculator).
- **Stateful**: Remembers client data across calls (e.g., shopping cart).

```
@Stateless
public class MyBean {
  public String sayHello(String name) {
    return "Hello, " + name;
  }
}
```

---

2. Entity Beans (Old & Deprecated)

- Mapped to a DB row.
- Replaced by **JPA (Java Persistence API)** in modern systems.

---

3. Message-Driven Beans (MDB)

- Consumes messages from **JMS (Java Message Service)**.
- Used in async processing like notifications, emails, queues.

```
@MessageDriven
public class EmailListener implements MessageListener {
  public void onMessage(Message msg) {
    // process email request
```

```
  }
}
```

🏭 Industry View: EJB is Rare Today

Instead of EJB	Use
EJB (business logic)	Spring Boot Services
EJB Entity Beans	JPA + Hibernate
EJB Session Beans	Spring Services + @Component
MDB	Apache Kafka / RabbitMQ

💡 Real Use of Sessions in Industry:

- JWT tokens (stateless)
- Spring Session
- OAuth tokens for auth flows

🎓 Mini College Example: HttpSession

login.jsp

```
<%
  String user = request.getParameter("username");
  HttpSession session = request.getSession();
  session.setAttribute("username", user);
%>
```

home.jsp

```
<%
  HttpSession session = request.getSession(false);
  String name = (String) session.getAttribute("username");
%>
<h1>Welcome <%= name %></h1>
```

✅ **Summary**

Topic	College Focus	Industry Focus
Session Tracking	Cookies, URL, Hidden, Session	Token-based Auth (JWT/OAuth), APIs
EJB	Learn all 3 types	Only for legacy, now use Spring Boot

Let me know if you want:

- A **practice assignment** using session tracking (login/logout),
- Or to **start revision** or **another subject** like *Computer Networks*.

What's next?

Awesome! We've completed **Java Programming** according to both your **college syllabus** and **industry standards**. Here's a final checklist to confirm:

✅ **Java Programming – Final Recap**

✅ **College Topics Covered:**

Unit	Topics
I	Core Java basics (JDK, JVM, classes, OOP, packages)
II	Exception handling, threads, multithreading
III	Applets, event handling, AWT
IV	JDBC, Servlets
V	Session Tracking, Enterprise Java Beans (EJB)

✅ **Industry-Level Topics Covered:**

Domain	Covered
OOP principles	✔ With real-world mapping
Multithreading	✔ With examples (Runnable, Thread)

Domain	Covered
Modern alternatives	✔ (Spring Boot vs EJB, JDBC vs ORM)
Real web app structure	✔ (Servlet lifecycle, session, authentication)
Java for backend	✔ Industry style session handling, APIs, cookies

---

🧠 **You are now ready to:**

- Build small Java apps (CLI & GUI)
- Connect Java apps with databases
- Create dynamic web apps using Servlets + JSP
- Understand modern frameworks like Spring Boot (recommended as next step!)

🔄 **Optional Next Steps (After This Syllabus):**

- ✅ Build a small **Login + CRUD project** using Java + JDBC + JSP/Servlet
- ✅ Learn **Spring Boot** (used widely in backend industry)
- ✅ Learn **JPA + Hibernate** (instead of raw JDBC)
- ✅ Use **Postman** + **HTTP APIs** from Java
