**Distributed Applications & Other Types of Applications in Java**

**1. Applications that can be developed using Java**

Java is a versatile programming language. Using it, we can build many types of applications:

a. **CUI and GUI Based Applications** (Standalone Applications)  
b. **Web Applications**  
c. **Distributed Applications**

**2. CUI and GUI Based Applications (Standalone Applications)**

These are applications that run on a **single machine**. Each user who wants to use them has to install the application on their system.

**a. CUI Applications (Character User Interface)**

* These are **console-based applications**.
* The interaction with the user happens through **text/commands**.
* Example: A program where you enter numbers in the console, and it calculates their sum.
* Built using **J2SE (Java Standard Edition)**.

👉 **Key Point**: No graphical elements (like buttons, menus, or windows). The interface is only text-based.

**b. GUI Applications (Graphical User Interface)**

* These are **desktop applications** where the interaction happens through graphical components.
* Java provides **Swing, AWT (Abstract Window Toolkit), and Applets** to build GUIs.
* Examples: **Eclipse IDE, Notepad++, GoToMeeting**.
* Built using **J2SE** as well.

👉 **Key Point**: GUI applications give a better user experience than CUI because they use **windows, buttons, menus, and images**.

**Characteristics of Standalone Applications (CUI/GUI)**

1. **Machine-specific**: They are installed and run on a particular machine.
2. **Installation required**: Each user needs to install the app on their own system.
3. **No network dependency**: They don’t need the internet or a server to run.
4. **Examples in real life**: Media Player, MS Word, Calculator.

👉 This is why they are called **Standalone Applications**. They work independently on one computer.

**3. Web Applications (Quick Overview)**

* These applications run on a **web server** and can be accessed using a browser (Chrome, Firefox, etc.).
* Built using **Java EE / Jakarta EE** (Servlets, JSP, Spring MVC, etc.).
* Example: Amazon, Flipkart, Gmail.
* Advantage: **No installation needed**. Users just open the URL in a browser.

**4. Distributed Applications (Quick Intro)**

* These are applications where the **software is distributed across multiple machines** connected via a network.
* Different parts of the application (client, server, database) run on different systems but work together as one application.
* Built using **RMI (Remote Method Invocation), CORBA, EJB, Web Services, Microservices with Spring Boot**.
* Example: Banking systems, Online reservation systems.

✅ **Summary (Self-Explanatory)**:

* **CUI/GUI applications** are **Standalone apps**. They run on one machine and need installation on every user’s system.
* **Web applications** run on a server and are accessed via browsers, no installation needed.
* **Distributed applications** run across multiple machines, sharing tasks between clients and servers over a network.

**Web Applications in Java**

**1. What is a Web Application?**

* A **web application** is an application that runs on a **server** and can be accessed through a **web browser** (like Chrome, Firefox, Edge) over the internet.
* Unlike standalone applications, users don’t need to **download** or **install** anything.
* Examples: **Gmail, Facebook, IRCTC, Amazon, Flipkart**.

👉 **Key Idea**: Web applications make it easy for anyone, anywhere, to access the application just by using the internet and a browser.

**2. Technology Used in Java for Web Applications**

* To build web applications in Java, we usually use:
  + **Java EE (Jakarta EE) Module**: Servlets and JSP (Java Server Pages).
  + **Spring MVC** (part of the Spring Framework) for building modern web apps with cleaner architecture and less boilerplate code.

**3. Characteristics of Web Applications**

1. **No installation required**
   * Users don’t need to install software.
   * Just open the browser, enter the URL, and use the app.
2. **Accessible from anywhere**
   * As long as there is an internet connection, users can access the app from any location.
3. **Client-Server Architecture**
   * **Client**: The browser (Chrome, Firefox, etc.). The client sends requests to the server.
   * **Server**: A computer where the project (application code) is deployed. The server processes requests and sends responses back to the client.

👉 For example:

* + You open Gmail in your browser (client).
  + The request goes to Google’s servers (server).
  + The server processes your email data and sends the result back to your browser.

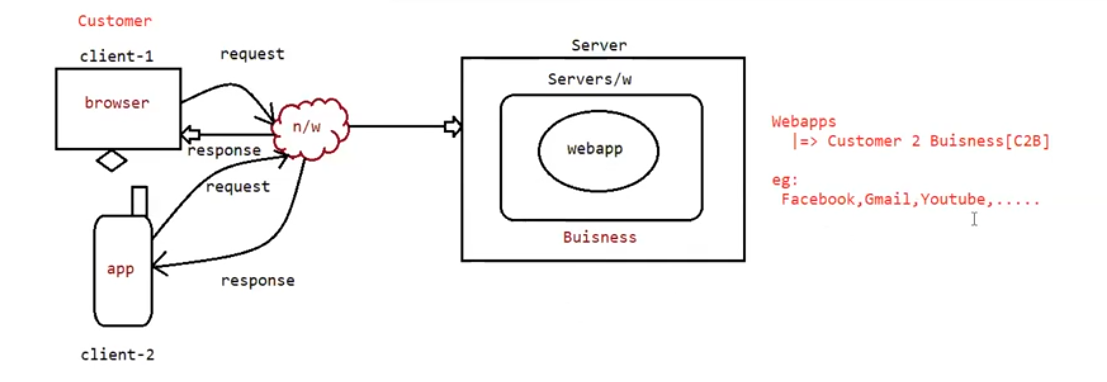
1. **Theme: Customer–Business Communication**
   * Web applications mainly help in connecting **customers** (users) with **businesses** (services or products).
   * Examples:
     + **IRCTC** connects customers with Indian Railways.
     + **Amazon/Flipkart** connect customers with sellers.
     + **Banking portals** connect customers with banks.

**4. Advantages of Web Applications**

* **Cross-platform**: Works on any device (Windows, Mac, Linux, Mobile) as long as a browser is available.
* **Easy updates**: Updates are done on the server, so users automatically get the new version without reinstalling.
* **Scalability**: Businesses can support thousands or even millions of users at the same time.
* **Centralized control**: Data and logic are stored on the server, making it easier to manage security and performance.

✅ **Summary (Self-Explanatory)**:

* A **web application** runs on a **server** and is accessed using a **browser** over the **internet**.
* Built using **Servlets, JSP, and Spring MVC** in Java.
* **No installation needed**, accessible from **anywhere**, and follows a **client-server model**.
* Purpose: To connect **customers** with **business services** (e-commerce, banking, ticket booking, social media, etc.).



**Web Applications Architecture (Explained with Diagram)**

**1. Participants in the Diagram**

1. **Customer (Users)**
   * Represented by **Client-1** and **Client-2** in the diagram.
   * They can use:
     + **Browser (Client-1)** → e.g., Chrome, Firefox, Safari.
     + **Mobile App (Client-2)** → e.g., Gmail app, YouTube app.
2. **Network (n/w)**
   * The communication medium (Internet).
   * It carries **requests** from the client to the server and **responses** back from the server to the client.
3. **Server**
   * The machine where the **business logic** and the **web application (webapp)** are deployed.
   * The server processes customer requests and sends back results.
   * Example: Gmail servers at Google Data Centers.

**2. Flow of Communication**

1. **Client sends a request**
   * Example: User enters **www.gmail.com** in browser or opens Gmail mobile app.
   * Request travels through the **network (internet)** to the server.
2. **Server processes the request**
   * Inside the server, the **web application** (built using Servlets, JSP, Spring MVC, etc.) takes the request.
   * The **business logic** (rules of the application, like authentication, fetching mails, etc.) runs and prepares a response.
3. **Response is sent back**
   * The processed result (like showing inbox, sending a mail, or displaying video) is sent back through the network.
   * The response reaches the **browser (Client-1)** or the **app (Client-2)**.

👉 This process repeats for every action like clicking a button, submitting a form, or watching a video.

**3. Key Points from the Diagram**

* **C2B (Customer to Business)**:
  + Web applications mainly focus on connecting customers with businesses.
  + Examples:
    - **Facebook, Gmail, YouTube, IRCTC, Amazon**.
* **Client Options**:
  + Customers may use **Browser (Web Client)** or **Mobile Apps (App Client)**.
  + Both interact with the same server-side web application.
* **Business Layer on Server**:
  + All main logic (e.g., user authentication, payments, bookings, searches) happens on the server.
  + This ensures **security, centralized control, and scalability**.

**4. Why Web Applications are Popular**

* **No installation required** on the client side (except sometimes a light mobile app).
* **Accessible worldwide** as long as internet is available.
* **Easier to maintain** because updates are done on the server, not on every user’s device.
* **Supports multiple devices** → PC, laptop, mobile, tablet.

✅ **Self-Explanatory Summary**:  
The diagram shows how **customers (clients)** use either a **browser** or a **mobile app** to send requests over the **internet (network)**. These requests reach the **server**, where the **web application and business logic** are hosted. The server processes the request and sends back a **response** (like inbox data, video content, or booking details). This model enables **Customer-to-Business (C2B)** communication, which is the foundation of web apps like **Facebook, Gmail, YouTube, IRCTC**.

**Distributed Applications in Java**

**1. What is a Distributed Application?**

* A **Distributed Application** is an application where **multiple systems (applications/servers)** talk to each other over a **network** to complete a single task.
* Here, instead of just **Customer → Business** communication (like in Web Applications), the communication extends to **Business → Business (B2B)**.

👉 In short:

* **Web Application** = Customer ↔ Business.
* **Distributed Application** = Business ↔ Business (applications communicating with each other).

**2. Technologies Used in Java**

To develop distributed applications in Java, we use:

* **Web Services** (SOAP-based services).
* **RESTful Services (REST APIs)** (lightweight, widely used today).

These services allow different applications (even written in different languages) to communicate with each other over the internet.

**3. Example Flow (Your Case: Flipkart Payment)**

Imagine a customer purchasing a product on **Flipkart**:

1. **Customer (Browser)** → Web Application (Flipkart Website).
   * User selects an item and proceeds to pay.
   * Flipkart is a **Web Application** that directly talks to the customer.
2. **Flipkart (Business App)** → **Payment Gateway (PayPal/PhonePe/RazorPay etc.)**.
   * Flipkart doesn’t handle payments directly.
   * It calls another **application** (distributed app) → PayPal.
3. **PayPal (Payment Gateway)** → **Card/UPI Networks (Visa, MasterCard, UPI)**.
   * PayPal internally calls other **applications** for card verification or UPI authentication.
4. **Card/UPI Networks** → **Banks (SBI, ICICI, HDFC)**.
   * Finally, the transaction request is sent to the customer’s bank for approval.
5. **Response travels back** → Bank → Visa/UPI → PayPal → Flipkart → Customer.

👉 **Key Observation**:

* The customer only sees Flipkart, but behind the scenes, **multiple business applications** (Flipkart, PayPal, Visa, SBI) talk to each other using distributed systems.
* This entire chain works within a few seconds because of **distributed application architecture**.

**4. Characteristics of Distributed Applications**

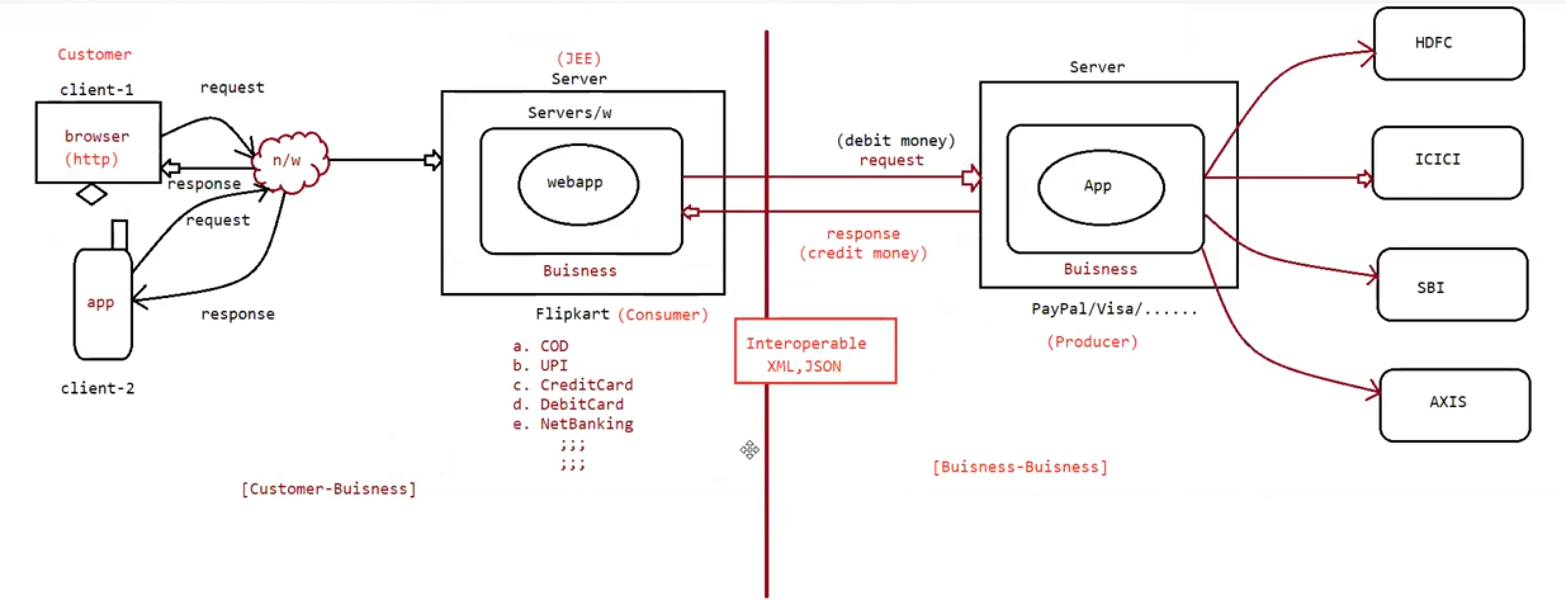
1. **Multiple systems working together**: Not just client-server, but server-to-server communication.
2. **Business-to-Business communication**: Businesses integrate their applications with other businesses.
3. **Platform-independent**: Services (REST/SOAP) allow apps built in Java, .NET, Python, etc., to talk to each other.
4. **Scalable and flexible**: Businesses can plug in new services (like new payment providers) without affecting the customer experience.
5. **Secure communication**: Usually uses HTTPS, authentication tokens, and encryption since sensitive data like payments are shared.

**5. Real-Life Examples**

* **E-commerce Payments**: Flipkart → PayPal → Visa/UPI → Bank.
* **Travel Booking**: MakeMyTrip → Indigo Airlines → Payment Gateway → Bank.
* **Food Delivery**: Zomato → Restaurant system → Delivery partner system → Payment gateway → Bank.
* **Banking**: One bank’s app (SBI) talks to another bank (ICICI) for fund transfers (NEFT/RTGS/UPI).

✅ **Summary (Self-Explanatory)**

* **Distributed Applications** are applications where one business application communicates with other business applications.
* Built using **Web Services (SOAP)** and **RESTful Services (REST APIs)**.
* Focus is **B2B communication** (Business ↔ Business).
* Example: Flipkart (web app) connects with PayPal (distributed app), which connects with Visa/UPI (distributed app), which finally connects with SBI/ICICI (distributed app).
* This chain of applications working together makes distributed systems powerful and essential in modern software.



**Distributed Applications Architecture (Explained with Diagram)**

**1. Participants in the Diagram**

**Customer Side**

* **Client-1 (Browser)**: Customer uses a browser (HTTP-based) like Chrome, Firefox to place an order.
* **Client-2 (App)**: Customer can also use a mobile app (Flipkart app, Amazon app, etc.).
* Both clients send a **request** over the **network (n/w)**.

**Business 1: Flipkart (Consumer)**

* **Flipkart’s Web Application** runs on a **JEE Server**.
* This is the **Customer-facing Web Application** (Customer ↔ Business).
* Flipkart provides multiple **payment options**:  
  a. Cash on Delivery (COD)  
  b. UPI  
  c. Credit Card  
  d. Debit Card  
  e. NetBanking

👉 Flipkart **doesn’t handle payments directly**. Instead, it forwards the payment request to another application (Producer).

**Business 2: Payment Gateway (Producer)**

* Examples: **PayPal, Visa, MasterCard, Razorpay, UPI Server**.
* Acts as a **middle business application** that connects Flipkart with banks.
* Flipkart (Consumer) sends: **Debit money request** → PayPal (Producer).
* PayPal processes the request and returns: **Credit money response** → Flipkart.

**Business 3: Banks (Final Systems)**

* Payment gateways further communicate with banks like **HDFC, ICICI, SBI, AXIS**.
* Example:
  + If you pay with an SBI debit card → request goes to SBI servers.
  + If you pay via ICICI net banking → request goes to ICICI servers.

👉 This is **Business-to-Business (B2B)** communication.

**2. Interoperability (How Apps Talk to Each Other)**

* Since Flipkart, PayPal, and Banks may be built using **different technologies** (Java, .NET, Python, PHP), there must be a **common language** for communication.
* This is achieved using:
  + **XML** (used in SOAP-based web services).
  + **JSON** (lightweight, widely used in REST APIs).

👉 This ensures interoperability → applications can talk to each other regardless of language/platform.

**3. Flow of Communication (Step by Step)**

1. **Customer places order** (Browser/App → Flipkart WebApp).
2. Flipkart confirms the product and forwards **payment details** to Payment Gateway (PayPal).
3. PayPal processes the payment by communicating with the correct bank (SBI/ICICI/etc.).
4. **Bank approves/rejects** the transaction and sends a response to PayPal.
5. PayPal sends the **final response** (success/failure) back to Flipkart.
6. Flipkart shows the **order status** (Payment Successful/Failed) to the customer.

**4. Key Points from Diagram**

* **Customer ↔ Business** = Customer to Flipkart (WebApp).
* **Business ↔ Business** = Flipkart ↔ PayPal ↔ Banks.
* **Flipkart = Consumer** (requests services from PayPal).
* **PayPal = Producer** (provides payment services).
* **Communication** = Interoperable using XML/JSON.

**5. Summary (Self-Explanatory)**

* **Distributed Applications** allow **multiple business applications** to work together over the internet.
* Customers only interact with **one application (Flipkart)**, but behind the scenes, multiple apps (Payment Gateways, Banks) communicate.
* Built in Java using **Web Services** and **RESTful Services**.
* **Example**: Flipkart (Consumer) → PayPal/Visa (Producer) → Banks (SBI/ICICI/HDFC).
* This enables smooth **Business-to-Business (B2B) communication** while keeping customer experience simple.

**Why Do We Need Distributed Applications?**

**1. Reusing Business Processes**

* Businesses often provide **core processes** that other businesses can reuse instead of reinventing them.
* Example:
  + **Payment Processing**: PayPal, Visa, UPI gateways already have secure payment logic.
  + Flipkart doesn’t build its own banking/payment system → it simply **reuses** PayPal’s service.
* This way, businesses can focus on their **main domain** (e-commerce, travel, food delivery) without rebuilding common processes like payments, authentication, shipping, etc.

👉 This **reuse of business processes** is the foundation of distributed applications.

**2. Business-to-Business Integration (B2B)**

* Distributed applications make it possible for businesses to **integrate with each other’s systems**.
* Example:
  + Flipkart integrates with **PayPal** (payments).
  + PayPal integrates with **Visa/UPI** (card/UPI networks).
  + Visa/UPI integrate with **Banks** (SBI, ICICI, HDFC).

👉 Each business focuses on its **specialized process**, and others can reuse it.

**3. Advantages of Reusing Business Processes**

1. **Saves Time & Cost**: No need to build payment, logistics, or communication systems from scratch.
2. **Security & Reliability**: Businesses like PayPal/UPI already ensure compliance, encryption, fraud detection, etc.
3. **Interoperability**: Services built in different technologies can communicate using **XML/JSON**.
4. **Scalability**: Easy to plug in new services (e.g., adding RazorPay along with PayPal).
5. **Customer Satisfaction**: Customer experiences a smooth, integrated workflow without knowing multiple systems are working in the background.

**4. Real-World Example**

* **Travel Booking**: MakeMyTrip → reuses **Airline APIs** (Indigo, Air India) and **Banking APIs** (for payments).
* **Food Delivery**: Zomato → reuses **Restaurant apps** for menus + **Payment Gateways** for payments + **Logistics services** for delivery tracking.
* **Bank Transfers**: SBI app reuses **NPCI UPI service** to transfer funds to ICICI bank instantly.

✅ **Summary (Self-Explanatory)**  
We need **Distributed Applications** because they allow **reusing existing business processes** instead of rebuilding them. This leads to faster development, cost savings, better security, and smooth **Business-to-Business (B2B) communication**. Example: Flipkart reuses PayPal’s payment process, PayPal reuses Visa/UPI process, and Visa/UPI reuses banking processes.

**Technologies to Build Distributed Applications in Java**

**a. CORBA (Common Object Request Broker Architecture)**

* An **old technology** used for distributed applications.
* Allows applications written in **different programming languages** to communicate with each other.
* Works by using an **Object Request Broker (ORB)** which manages requests between client and server objects.
* Example: A C++ application can talk to a Java application using CORBA.  
  👉 Not widely used today, mostly replaced by Web Services.

**b. RMI (Remote Method Invocation)**

* A **Java-only technology** for distributed applications.
* Allows one Java program to **call methods** of another Java program running on a different machine (over a network).
* Example: A Java client can call a method of a Java server as if it were a local method.  
  👉 Limitation: Only works with Java-to-Java communication (not cross-language).

**c. EJB (Enterprise Java Beans)**

* Part of **Java EE (Enterprise Edition)**.
* Used to build large-scale, secure, transactional distributed applications.
* EJBs provide built-in support for:
  + Transactions
  + Security
  + Remote access
  + Scalability
* Example: A banking application where multiple clients interact with a central server using secure distributed components.  
  👉 Heavyweight, less popular today, but still used in legacy enterprise systems.

**d. Web Services (SOAP-based)**

* **SOAP (Simple Object Access Protocol)** based services.
* Communication happens using **XML** over HTTP.
* Allows interoperability between different platforms (Java, .NET, Python, etc.).
* Example: SBI bank’s SOAP service for balance check can be consumed by any Java or .NET client.  
  👉 Reliable, but heavier compared to REST (more complex XML structures).

**e. RESTful Services (Representational State Transfer Services)**

* A **lightweight and modern approach** to build distributed applications.
* Communication happens using **HTTP methods** (GET, POST, PUT, DELETE).
* Data is exchanged in **JSON** (sometimes XML).
* Advantages: Faster, simpler, less bandwidth than SOAP.
* Example: Flipkart uses REST APIs to connect with PayPal, RazorPay, or UPI servers.  
  👉 Most widely used today for distributed applications.

**✅ Summary (Self-Explanatory)**

* **CORBA** → Old, cross-language using Object Request Broker.
* **RMI** → Java-to-Java communication using remote methods.
* **EJB** → Enterprise Java Beans for secure, transactional distributed apps.
* **Web Services (SOAP)** → XML-based, cross-platform, but heavy.
* **RESTful Services** → JSON-based, lightweight, widely used today for modern B2B applications.