

1. Write a simple "Hello World" program in two different programming languages of your choice. Compare the structure and syntax.

Python Version

```
print("Hello, World!")
```

Java Version

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

Comparison

Feature	Python	Java
Simplicity	Very concise	More verbose
Compilation	Interpreted directly	Needs to be compiled
Entry Point	No specific entry point needed	Requires a main method
Syntax	Minimalist, no semicolon needed	Requires class, method, braces, and semicolon

Feature	Python	Java
Typing	Dynamically typed	Statically typed

Summary

- **Python** is more beginner-friendly and minimalistic.
- **Java** is more structured, emphasizing explicit declarations and object orientation.

2. Research and create a diagram of how data is transmitted from a client to a server over the internet.

Client to a server:

Client to Server Data Flow

Client → Router → ISP → Internet → Server

Steps

1. **Client** sends a request (e.g., open a website).
 2. **Router** forwards it to the **ISP**.
 3. **DNS** resolves the domain name to an IP address.
 4. **ISP/Internet** routes the data to the **Server**.
 5. **Server** processes and sends back a response.
-

Key Protocols

- **DNS** – Finds the server's IP.
- **TCP/IP** – Sends data reliably.
- **HTTP/HTTPS** – Handles web communication.

3. Design a simple HTTP client-server communication in any language.

Here's a **simple HTTP client-server example** using **Python** with its built-in `http.server` (for the server) and `requests` library (for the client).

Server (Python)

```
from http.server import BaseHTTPRequestHandler, HTTPServer
```

```
class SimpleHandler(BaseHTTPRequestHandler):
```

```
    def do_GET(self):
```

```
        self.send_response(200)
```

```
        self.send_header("Content-type", "text/plain")
```

```
        self.end_headers()
```

```
        self.wfile.write(b"Hello from the server!")
```

```
# Start the server
```

```
if __name__ == "__main__":
```

```
server_address = ("localhost", 8000)

httpd = HTTPServer(server_address, SimpleHandler)

print("Server running at http://localhost:8000")

httpd.serve_forever()
```

Client (Python)

```
import requests
```

```
response = requests.get("http://localhost:8000")


print("Server says:", response.text)
```

How it Works

1. **Start the server** script first.
2. The **client** sends a GET request to `http://localhost:8000`.
3. The server responds with "Hello from the server!".

4. Research different types of internet connections (e.g., broadband, fiber, satellite) and list their pros and cons.

1. Broadband (DSL/Cable)

- **Speed:** 5–500 Mbps
-  Widely available, affordable

- ❌ Slower than fiber, peak-time slowdowns
-

2. Fiber-Optic

- **Speed:** 100 Mbps–10+ Gbps
 - ✅ Very fast, low latency
 - ❌ Limited in rural areas, higher cost
-

3. Satellite

- **Speed:** 25–100 Mbps
 - ✅ Works in remote areas
 - ❌ High latency, weather affects it
-

4. Mobile (4G/5G)

- **Speed:** 20 Mbps–1 Gbps
- ✅ Portable, fast with 5G
- ❌ Data caps, signal varies

5. Simulate HTTP and FTP requests using command line tools (e.g., curl).

Here's how you can **simulate HTTP and FTP requests using curl** from the command line:

1. Simulate an HTTP Request

```
curl http://example.com
```

- **What it does:** Sends a GET request to example.com and displays the HTML response.

Other useful options:

- `curl -I http://example.com` → Fetch only HTTP headers.
 - `curl -X POST -d "name=John" http://example.com/form` → Send a POST request with form data.
-

2. Simulate an FTP Request

```
curl -u username:password ftp://ftp.example.com/file.txt
```

- **What it does:** Downloads file.txt from the FTP server using your credentials.

Upload a file via FTP:

```
curl -T localfile.txt -u username:password  
ftp://ftp.example.com/
```

Note

- Replace username, password, example.com, and file names with real values.
- Ensure FTP servers are publicly accessible or that you have proper credentials.

6. Identify and explain three common application security vulnerabilities. Suggest possible solutions.

Here are **three common application security vulnerabilities** and how to address them:

1. SQL Injection (SQLi)

What it is:

An attacker injects malicious SQL code into input fields to access or manipulate databases.

Example:

```
SELECT * FROM users WHERE username = 'admin' --' AND password = '';
```

Solution:

- Use **parameterized queries** or **prepared statements**
 - Validate and sanitize user input
 - Use ORM tools (e.g., SQLAlchemy, Hibernate)
-

2. Cross-Site Scripting (XSS)

What it is:

An attacker injects malicious scripts into web pages viewed by other users.

Example:

```
<script>alert('Hacked!');</script>
```

Solution:

- Escape output in HTML, JavaScript, and attributes
 - Use frameworks with built-in XSS protection (e.g., React, Angular)
 - Implement Content Security Policy (CSP)
-

3. Broken Authentication**What it is:**

Weak or mismanaged authentication mechanisms allow attackers to impersonate users.

Example:

- Predictable session IDs
- No multi-factor authentication (MFA)

Solution:

- Use strong password policies
- Implement **multi-factor authentication**
- Secure session management (e.g., regenerate session IDs on login)

7. Identify and classify 5 applications you use daily as either system software or application software.

Here's a list of **5 common applications** and their classification as **System Software** or **Application Software**:

Application	Type	Classification
Google Chrome	Web Browser	✓ Application Software
Microsoft Word	Word Processor	✓ Application Software
Windows 10 / macOS	Operating System	✓ System Software
File Explorer / Finder	File Management	✓ System Software
Spotify	Music Streaming	✓ Application Software

◆ **System Software**

- Runs the computer and manages hardware (e.g., OS, file managers).

◆ **Application Software**

- Helps users perform specific tasks (e.g., browsing, writing, streaming).

8. Design a basic three-tier software architecture diagram for a web application.

Three-Tier Architecture

1. Presentation Tier → Web Browser (HTML/CSS/JS)
2. Application Tier → Backend Server (Business Logic/API)
3. Data Tier → Database (e.g., MySQL, MongoDB)

Each tier handles a separate concern:

◆ UI → ◆ Logic → ◆ Data

9. Create a case study on the functionality of the presentation, business logic, and data access layers of a given software system.

Here's a **brief case study** showing how the **presentation**, **business logic**, and **data access** layers function in a real-world software system.

Case Study: Online Food Ordering System

1. Presentation Layer (User Interface)

Functionality:

- Users browse menus, place orders, and track delivery using a web/mobile interface.
- Built with HTML/CSS, JavaScript, and a frontend framework like React or Flutter.

Example:

- User selects a food item and clicks “Order Now”.

2. Business Logic Layer (Application Server)

Functionality:

- Processes the order, calculates total price, applies discounts, and checks restaurant availability.
- Implements rules like “free delivery for orders over \$50”.
- Built with a backend framework (e.g., Node.js, Django, Java Spring Boot).

Example:

- Validates that the selected restaurant is open and food is in stock.

3. Data Access Layer (Database Interface)

Functionality:

- Interacts with the database to retrieve or update order, user, and menu information.
- Uses SQL or an ORM (like Sequelize or Hibernate) to query/update data.

Example:

- Saves the new order and updates the restaurant's available inventory.

Summary Table

Layer	Role	Technology Examples
Presentation	UI/UX interaction	React, Flutter, HTML/CSS

Layer	Role	Technology Examples
Business Logic	Processes rules & operations	Django, Node.js, Spring Boot
Data Access	Reads/writes data from/to DB	SQL, PostgreSQL, MongoDB ORM

10. Explore different types of software environments (development, testing, production). Set up a basic environment in a virtual machine.

Software Environments

- **Development:** For writing/debugging code
- **Testing:** For QA and validation
- **Production:** Live for end users

Set Up a Basic Environment in a VM

1. **Install VirtualBox**
2. **Create VM** with Ubuntu (2GB RAM, 20GB disk)
3. **Install Tools:**
4. `sudo apt update`
5. `sudo apt install git python3 pip curl`
6. (Optional) Add:
 - **Web server:** `sudo apt install nginx`

- **Code/test tools:** VS Code, pytest

11. Write and upload your first source code file to Github.

Here's a simple guide to **write and upload your first source code file to GitHub:**

Step-by-Step Instructions

1. Create a GitHub Account

- Go to <https://github.com> and sign up (if you haven't).

2. Create a New Repository

- Click "**New**" → Give it a name (e.g., hello-world)
- Choose **Public** or **Private**, then click "**Create repository**"

3. Write Your First Code

Example: hello.py

```
# hello.py
```

```
print("Hello, GitHub!")
```

4. Upload the File via GitHub Interface

- In your repo, click "**Add file**" → "**Upload files**"
- Drag and drop hello.py
- Click "**Commit changes**"

You've now uploaded your first source code file to GitHub!

12. Create a Github repository and document how to commit and push code changes.

Sure! Here's a **simple guide** to create a GitHub repository and document how to **commit and push code** using Git.

1. Create a GitHub Repository

1. Go to <https://github.com>
 2. Click “+” → “New repository”
 3. Fill in:
 - **Repository name** (e.g., my-first-repo)
 - Optional: description, README
 4. Click “Create repository”
-

2. Set Up Git Locally

```
git config --global user.name "Your Name"
```

```
git config --global user.email "you@example.com"
```

3. Clone the Repository

```
git clone https://github.com/your-username/my-first-repo.git
```

```
cd my-first-repo
```

4. Commit & Push Code

1. **Add a file** (e.g., hello.py):

2. `print("Hello, GitHub!")`
 3. **Add and commit changes:**
 4. `git add hello.py`
 5. `git commit -m "Add hello.py"`
 6. **Push to GitHub:**
 7. `git push origin main`
-

You're Done!

Your code is now uploaded to GitHub.

13. Create a student account on Github and collaborate on a small project with a classmate.

Here's a **quick guide** to create a **GitHub Student account** and collaborate on a small project:

1. Sign Up for GitHub Student Account

1. Go to [GitHub Student Developer Pack](#)
2. Click **“Get benefits”** and sign in or create a GitHub account
3. Verify with:
 - Your **school email** or
 - Upload a **student ID or proof of enrollment**

You'll get free tools (e.g., private repos, free domains, and cloud credits)

2. Collaborate on a Project with a Classmate

A. Create a New Repository

1. One student creates a repo (e.g., collab-project)
2. Click **Settings** → **Collaborators** → **Add your classmate's username**

B. Clone the Repo Locally

```
git clone https://github.com/your-username/collab-project.git
```

C. Make Changes and Push

```
echo "Hello from me!" > hello.txt
```

```
git add hello.txt
```

```
git commit -m "Add my greeting"
```

```
git push origin main
```

D. Classmate Pulls & Adds More

```
git pull origin main
```

```
# then edit or add files
```

```
git add .
```

```
git commit -m "Added my part"
```

```
git push origin main
```


14. Create a list of software you use regularly and classify them into the following categories: system, application, and utility software.

Sure! Here's a list of commonly used software, classified into **System**, **Application**, and **Utility** categories:

System Software

Software	Purpose
Windows 10/macOS	Operating system
Linux (Ubuntu)	Open-source operating system
Device Drivers	Hardware communication support

Application Software

Software	Purpose
Google Chrome	Web browsing
Microsoft Word	Word processing
Zoom / MS Teams	Video conferencing
Spotify / VLC	Media streaming/playback
WhatsApp Desktop	Messaging

Utility Software

Software	Purpose
WinRAR / 7-Zip	File compression
Antivirus (e.g., Windows Defender)	Security

Software

CCleaner

Disk Management

Purpose

System cleanup and optimization

Partitioning, formatting

15. Follow a GIT tutorial to practice cloning, branching, and merging repositories.

Here's a **simple Git tutorial** to practice **cloning**, **branching**, and **merging**—perfect for beginners:

Prerequisites

- Git installed: <https://git-scm.com>
- GitHub account created

Step-by-Step Git Practice

1. Clone a Repository

On GitHub:

- Create a new repository (e.g., git-practice)

In your terminal:

```
git clone https://github.com/your-username/git-practice.git
```

```
cd git-practice
```

2. Create and Switch to a New Branch

```
git checkout -b new-feature
```

This creates and switches to a branch called new-feature.

3. Make a Change and Commit

Create a file:

```
echo "Hello from new feature!" > feature.txt
```

```
git add feature.txt
```

```
git commit -m "Add feature.txt file"
```

4. Switch Back and Merge

```
git checkout main
```

```
git merge new-feature
```

This brings changes from new-feature into main.

5. Push Changes to GitHub

```
git push origin main
```

You can now see your updates on GitHub!

16. Write a report on the various types of application software and how they improve productivity.

Here's a **brief report** on the **types of application software** and how they **enhance productivity**:

Report: Types of Application Software and Their Impact on Productivity

◆ 1. Word Processing Software

- **Examples:** Microsoft Word, Google Docs
 - **Purpose:** Create, edit, and format text documents
 - **Productivity Impact:** Speeds up writing, editing, and collaboration on reports, letters, etc.
-

◆ 2. Spreadsheet Software

- **Examples:** Microsoft Excel, Google Sheets
 - **Purpose:** Organize, calculate, and analyze data
 - **Productivity Impact:** Automates financial tasks, supports data visualization, and decision-making.
-

◆ 3. Presentation Software

- **Examples:** PowerPoint, Google Slides
 - **Purpose:** Design visual presentations
 - **Productivity Impact:** Helps convey ideas clearly and professionally in meetings or lectures.
-

◆ 4. Database Management Software

- **Examples:** Microsoft Access, MySQL
- **Purpose:** Store, manage, and query large sets of data

- **Productivity Impact:** Efficient data handling and retrieval for business operations and reporting.
-

◆ 5. Communication Software

- **Examples:** Zoom, Microsoft Teams, Slack
 - **Purpose:** Enable virtual meetings and team messaging
 - **Productivity Impact:** Supports remote work, real-time collaboration, and faster decision-making.
-

◆ 6. Web Browsers

- **Examples:** Google Chrome, Mozilla Firefox
 - **Purpose:** Access online content and web applications
 - **Productivity Impact:** Facilitates research, online tools, and cloud-based workflows.
-

◆ 7. Graphic Design and Media Software

- **Examples:** Adobe Photoshop, Canva, Audacity
 - **Purpose:** Create and edit images, videos, and audio
 - **Productivity Impact:** Enhances marketing, branding, and content creation speed.
-

Conclusion

Application software streamlines specific tasks, improves accuracy, supports collaboration, and ultimately saves time—

making individuals and organizations more productive in a digital environment.

17. Create a flowchart representing the Software Development Life Cycle (SDLC).

Here's a textual representation of a **Software Development Life Cycle (SDLC)** flowchart. If you'd like a visual image or diagram, I can generate that as well.

Flowchart: Software Development Life Cycle (SDLC)

[Start]

|

v

[1. Requirement Gathering & Analysis]

|

v

[2. System Design]

|

v

[3. Implementation (Coding)]

|

v

[4. Testing]

|

v

[5. Deployment]

|

v

[6. Maintenance]

|

v

[End / Iteration (Back to Requirement Gathering)]

SDLC Stages Explained Briefly:

1. Requirement Gathering & Analysis

- Understand what the users need.
- Document functional and non-functional requirements.

2. System Design

- Create architecture and design specifications.
- Determine hardware and system requirements.

3. Implementation (Coding)

- Developers write code based on design documents.
- Version control and modular development.

4. Testing

- Verify the software meets requirements.

- Includes unit, integration, system, and acceptance testing.

5. Deployment

- Release the product to the end-users.
- Can be a full release or phased rollout.

6. Maintenance

- Fix bugs and update software over time.
- Enhance functionality based on feedback.

18. Write a requirement specification for a simple library management system.

Here's a concise **Requirement Specification Document** for a **Simple Library Management System (LMS)**:

Software Requirement Specification (SRS)

Project Title: Library Management System

Version: 1.0

Prepared by: [Your Name/Team]

Date: [Insert Date]

1. Introduction

1.1 Purpose

This document outlines the functional and non-functional requirements for a simple Library Management System (LMS)

to manage book inventory, member records, and borrowing/returning operations.

1.2 Scope

The system will be used by librarians and library members to:

- Register and manage books
 - Register and manage members
 - Issue and return books
 - Track due dates and overdue books
-

2. Functional Requirements

2.1 User Management

- Add, update, delete member records
- Assign roles: librarian, member

2.2 Book Management

- Add, update, delete book records
- Search books by title, author, ISBN
- View book availability

2.3 Lending Management

- Issue books to members
- Return books
- Check due dates and overdue status

2.4 Reports

- Generate report of borrowed books

- Generate overdue books report
-

3. Non-Functional Requirements

3.1 Usability

- Simple and intuitive user interface for all users

3.2 Performance

- The system should support up to 1000 concurrent users without performance degradation.

3.3 Security

- User authentication and role-based access
- Data protection and regular backups

3.4 Portability

- Web-based; accessible on desktops and mobile browsers
-

4. Assumptions and Constraints

- The system will be developed using web technologies (e.g., HTML/CSS, JavaScript, Python/PHP backend, SQL database).
- Deployment is on a local server or internal network.

19. Perform a functional analysis for an online shopping system.

Here's a **Functional Analysis** for an **Online Shopping System**, which breaks down the system's key functionalities and user interactions.

Functional Analysis: Online Shopping System

1. Main Actors

- **Customer (User)**
 - **Admin (System Manager)**
 - **System (Website/App)**
 - **Payment Gateway**
-

2. Functional Areas and Descriptions

2.1 User Management

- **Register/Login/Logout**
 - Users can create accounts and log in securely.
- **Manage Profile**
 - Users can update personal details, address, and payment info.
- **Authentication & Authorization**
 - Secure login; roles define access (e.g., user vs. admin).

2.2 Product Management

- **View Products**

- Customers browse products by category, search, or filter.
- **Product Details**
 - View detailed info (images, description, price, stock status).
- **Admin: Add/Edit/Delete Products**
 - Admin manages product listings and inventory.

2.3 Shopping Cart

- **Add to Cart / Remove from Cart**
 - Customers can manage items in a virtual cart.
- **Update Quantity**
 - Change item quantities before checkout.
- **View Cart Summary**
 - Display item count, subtotal, estimated delivery cost.

2.4 Order Management

- **Place Order**
 - Proceed to checkout, enter shipping and payment info.
- **Order Confirmation**
 - Generate confirmation receipt and order number.
- **Track Orders**
 - View status: processing, shipped, delivered.
- **Admin: View/Manage Orders**
 - Update order status, view order history, handle returns.

2.5 Payment Processing

- **Payment Gateway Integration**
 - Support for credit/debit cards, digital wallets, etc.
- **Transaction Validation**
 - Confirm payment success or failure.
- **Generate Invoice**
 - Provide digital invoice after successful payment.

2.6 Customer Support

- **Contact Support**
 - Form or chat system for customer inquiries.
- **Return & Refund Requests**
 - Allow users to initiate returns with reasons and track status.

2.7 Notifications

- **Email/SMS Alerts**
 - Order updates, payment confirmations, and promotions.

3. Summary of Core Functionalities

Function	Description
User Registration/Login	Account access and user identity management

Function	Description
Product Browsing	Search, filter, and view product catalog
Cart Management	Add/remove/update products in shopping cart
Checkout & Payment	Secure order processing and transaction handling
Order Tracking	Status updates and order history view
Admin Control Panel	Manage products, users, and orders

20. Design a basic system architecture for a food delivery app.

Here's a **basic system architecture** for a **food delivery app**, covering the key components:

1. Three-Tier Architecture Overview

Presentation Layer (Frontend)

- **Customer App:** Browse menus, place orders, track delivery
- **Restaurant App:** Manage orders and menus
- **Delivery App:** View delivery tasks, navigation
- **Admin Panel (Web):** Oversee users, restaurants, and orders

Business Logic Layer (Backend/API)

- **Authentication Service:** Login, sign-up, role management

- **Order Management:** Order creation, updates, status tracking
- **Menu & Inventory:** Manage dishes, availability
- **Payment Gateway Integration:** Handle payments securely
- **Notification Service:** SMS, push notifications for updates
- **Location Service:** Track driver/customer positions

Data Layer (Database & Storage)

- **Databases:** PostgreSQL or MongoDB
 - Users table
 - Orders table
 - Restaurants table
 - Delivery status logs
 - **Cloud Storage:** Store images, invoices, and receipts
-

Optional Integrations

- **Maps API (e.g., Google Maps):** For route optimization
- **Third-party payment gateways:** Stripe, PayPal, etc.
- **Analytics Tools:** Firebase, Mixpanel

21. Develop test cases for a simple calculator program.

Here are **test cases for a simple calculator program** that performs the four basic arithmetic operations: addition,

subtraction, multiplication, and division. The test cases cover normal, boundary, and error conditions.

Assumptions:

- Calculator functions: add(a, b), subtract(a, b), multiply(a, b), divide(a, b)
 - Input types: integers and floats
 - Error handling for division by zero
-

Functional Test Cases

Test Case ID	Description	Input	Expected Output
TC_01	Add two positive integers	add(3, 5)	8
TC_02	Add a positive and negative integer	add(10, -4)	6
TC_03	Subtract smaller from larger	subtract(10, 3)	7
TC_04	Subtract larger from smaller	subtract(3, 10)	-7
TC_05	Multiply two positive numbers	multiply(4, 5)	20
TC_06	Multiply with zero	multiply(9, 0)	0

Test Case ID	Description	Input	Expected Output
TC_07	Divide two integers	divide(8, 2)	4.0
TC_08	Divide resulting in float	divide(7, 2)	3.5
TC_09	Divide by zero	divide(5, 0)	Error / Exception

Boundary and Edge Test Cases

Test Case ID	Description	Input	Expected Output
TC_10	Add zero	add(0, 0)	0
TC_11	Subtract with zero	subtract(0, 5)	-5
TC_12	Multiply negative numbers	multiply(-3, -6)	18
TC_13	Division negative/positive	divide(-10, 2)	-5.0
TC_14	Large number operation	add(1e10, 1e10)	2e10
TC_15	Small decimal division	divide(0.1, 0.2)	0.5

Invalid Input Test Cases

Test Case ID	Description	Input	Expected Output
TC_16	Non-numeric input	add("a", 5)	TypeError / Error
TC_17	Missing argument	subtract(7)	Error / Exception
TC_18	Null input	multiply(None, 3)	TypeError / Error

22. Document a real-world case where a software application required critical maintenance.

A notable **real-world case** where a software application required **critical maintenance** is the **2018 Boeing 737 MAX MCAS software issue**. This incident is a high-impact example of how software flaws can lead to catastrophic consequences and necessitate urgent maintenance and reengineering.

Real-World Case: Boeing 737 MAX – MCAS Software Fault

Timeline

- **Initial Incident:** October 29, 2018 – Lion Air Flight 610 crash
- **Second Incident:** March 10, 2019 – Ethiopian Airlines Flight 302 crash
- **Maintenance Triggered:** March 2019 – Boeing 737 MAX grounded worldwide

Issue Summary

The Maneuvering Characteristics Augmentation System (MCAS) was a software subsystem added to the Boeing 737 MAX to improve aircraft stability. It relied on a single Angle of Attack (AoA) sensor. When the sensor fed incorrect data to the MCAS, the system erroneously activated and repeatedly pushed the aircraft's nose down.

Key Software Failures

- **Single-point sensor failure** (no redundancy)
- **Lack of pilot awareness:** MCAS activation was not documented thoroughly in pilot training manuals
- **Inadequate fail-safe mechanisms**
- **System overrides pilot input in certain cases**

Critical Maintenance Undertaken

After the crashes:

- Boeing revised the **MCAS logic** to:
 - Use input from **multiple AoA sensors**
 - Limit the number and strength of nose-down commands
 - Allow pilots to override MCAS
- Pilots received updated training on MCAS behavior
- **FAA and international aviation authorities** re-evaluated certification processes

Results and Resolution

- After 20 months of grounding, the **Boeing 737 MAX was cleared for flight in late 2020** with a thoroughly updated MCAS software.
- Massive reputational, legal, and financial consequences for Boeing (>\$20 billion in losses)
- Sparked industry-wide changes in software safety standards and oversight

Lessons Learned

- Software used in critical systems must be fail-safe, well-documented, and rigorously tested.
- Redundancy and sensor validation are crucial in embedded safety-critical systems.
- User (pilot) awareness and control should never be compromised by automation.

23. Create a DFD for a hospital management system.

Hospital Management System – Level 0 (Context Level)

The **Hospital Management System (HMS)** is a centralized software that manages core hospital functions. At Level 0, it is treated as a single process interacting with external entities.

External Entities and Their Interactions:

1. Patient

- Provides registration details.
- Requests appointments.
- Receives appointment confirmation and prescriptions.

2. Doctor

- Accesses patient records.
- Updates diagnosis and treatment information.
- Issues prescriptions.

3. Receptionist/Admin

- Handles patient registration and appointment scheduling.
- Manages billing information.

4. Pharmacy

- Receives prescription data from doctors.
- Dispenses medications to patients.

5. Laboratory

- Receives lab test requests.
- Sends back test results to the system for doctor review.

Main Functions of the System:

- Register patients
- Schedule and manage appointments

- Maintain patient records
 - Generate and manage prescriptions
 - Manage billing and payments
 - Process lab tests and results
 - Coordinate with pharmacy inventory
-

This Level 0 overview shows how external users (patients, staff) interact with a centralized hospital system that manages all data flows internally.

24. Build a simple desktop calculator application using a GUI library.

Here's a **simple desktop calculator application in Java** using the **Swing GUI library**. This calculator performs basic arithmetic operations: addition, subtraction, multiplication, and division.

Features

- GUI-based calculator using JFrame, JTextField, and JButton
 - Handles +, -, *, /, =, and C (clear)
-

Java Code: Simple Calculator with Swing

```
import javax.swing.*;  
  
import java.awt.*;
```

```
import java.awt.event.*;

public class SimpleCalculator {

    private JFrame frame;

    private JTextField textField;

    private String currentInput = "";

    private double num1 = 0, num2 = 0;

    private char operator;

    public SimpleCalculator() {

        frame = new JFrame("Calculator");

        textField = new JTextField();

        textField.setEditable(false);

        textField.setFont(new Font("Arial", Font.BOLD, 24));

        // Panel for buttons

        JPanel panel = new JPanel();

        panel.setLayout(new GridLayout(5, 4, 5, 5));

        String[] buttons = {

            "7", "8", "9", "/",

            "4", "5", "6", "*",

            "1", "2", "3", "-",

            "0", ".", "C", "+",

            "=",

        };

        for (String text : buttons) {
```

```
        JButton btn = new JButton(text);

        btn.setFont(new Font("Arial", Font.BOLD, 20));

        btn.addActionListener(e ->
buttonPressed(e.getActionCommand()));

        panel.add(btn);
    }

    // Adjust layout
    frame.setLayout(new BorderLayout());

    frame.add(textField, BorderLayout.NORTH);

    frame.add(panel, BorderLayout.CENTER);

    frame.setSize(300, 400);

frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);

    frame.setVisible(true);
}

private void buttonPressed(String text) {
    switch (text) {
        case "C":
            currentInput = "";
            num1 = num2 = 0;
            operator = '\0';
            textField.setText("");
            break;
```



```
case "=":
```

```
    try {
```

```
        num2 = Double.parseDouble(currentInput);
```

```
        double result = calculate(num1, num2, operator);
```

```
        textField.setText(String.valueOf(result));
```

```
        currentInput = String.valueOf(result);
```

```
    } catch (Exception e) {
```

```
        textField.setText("Error");
```

```
    }
```

```
    break;
```

```
case "+": case "-": case "*": case "/":
```

```
    try {
```

```
        num1 = Double.parseDouble(currentInput);
```

```
        operator = text.charAt(0);
```

```
        currentInput = "";
```

```
    } catch (Exception e) {
```

```
        textField.setText("Error");
```

```
    }
```

```
    break;
```

```
default:
```

```
    currentInput += text;
```

```
    textField.setText(currentInput);
```

```
}
```

```

    }

    private double calculate(double a, double b, char op) {
        return switch (op) {
            case '+' -> a + b;
            case '-' -> a - b;
            case '*' -> a * b;
            case '/' -> b != 0 ? a / b : Double.NaN;
            default -> 0;
        };
    }

    public static void main(String[] args) {
        SwingUtilities.invokeLater(SimpleCalculator::new);
    }
}

```

How to Run

1. Save as SimpleCalculator.java
2. Compile:
3. `javac SimpleCalculator.java`
4. Run:
5. `java SimpleCalculator`

25. Draw a flowchart representing the logic of a basic online registration system.

flowchart representing the logic of a **basic online registration system**. You can later visualize it using flowchart tools like Draw.io, Lucidchart, or pen and paper.

Flowchart Steps – Online Registration System

- 1. Start**
 - 2. Display Registration Form**
 - 3. User Fills in Details**
 - 4. User Clicks Submit**
 - 5. Validate Input Fields**
 - **If invalid:**
 - Show error message
 - Go back to step 3
 - 6. Check if Email Already Registered**
 - **If yes:**
 - Show “Email already exists”
 - Go back to step 3
 - 7. Save User Data to Database**
 - 8. Send Confirmation Email**
 - 9. Show Registration Success Message**
 - 10. End**
-

Key Decisions in the Flowchart

- Is input valid?
 - Is email already registered?
-

Flowchart Structure (Text Formatted)

Start

|

Display Registration Form

|

User Fills in Details

|

User Clicks Submit

|

Validate Input Fields

|--[Invalid?]-> Show Error Message --> Back to Form

|

Check Email Already Registered

|--[Yes]-> Show "Email Exists" --> Back to Form

|

Save Data to Database

|

Send Confirmation Email

|

Show Success Message

|

End
