Q1. Jenkins Pipeline for a Java Project [25 Marks]
Scenario:
You are part of a DevOps team responsible for automating the build process of a simple Java application using Maven.
Task:
1. Pull the Maven Java project from GitHub.
2. Build the project using required Jenkin files. Create the Pipeline View using Jenkins.
Expected Output:
- All Jenkinsfiles with correct syntax.
- Console output showing BUILD SUCCESS and the actual output after final stage.
- Jenkins pipeline View.

Below is a **complete, exam-ready answer** for *Q1: Jenkins Pipeline for a Java Project* with **Jenkinsfile**, explanation, pipeline view description, and sample console output**.
Use this as a full **25-mark answer**.

✅ Q1. Jenkins Pipeline for a Java Project – Complete Answer (25 Marks)

Scenario

You are part of a DevOps team responsible for automating the CI/CD pipeline of a simple Java Maven project. Jenkins must automatically pull the source code from GitHub, build it using Maven, and show the final console output.

---

1. Jenkinsfile – Declarative Pipeline (Correct Syntax)

Jenkinsfile

groovy Copy code

```
pipeline {
    agent any

    tools {
        maven 'MAVEN_HOME'     // Name configured in Jenkins Global Tool Configura
        jdk 'JDK17'            // Name of installed JDK
    }

    stages {
        stage('Checkout from GitHub') {
            steps {
                echo "Pulling source code from GitHub..."
                git branch: 'main',
                    url: 'https://github.com/your-username/your-java-maven-projec
            }
        }

        stage('Build with Maven') {
            steps {
                echo "Building project with Maven..."
                sh 'mvn clean install'
            }
        }

        stage('Run Application') {
            steps {
                echo "Running final Java application..."
                sh 'java -jar target/*.jar'
            }
        }
    }

    post {
        success {
            echo "BUILD SUCCESS: Pipeline executed successfully!"
        }
        failure {
            echo "BUILD FAILED: Please check logs!"
        }
```

```
        }
    }
```

- ✔ Correct declarative syntax
- ✔ Includes Git pull
- ✔ Maven build
- ✔ Running output
- ✔ Post-build steps

---

2. Explanation of Each Stage

Stage 1: Checkout from GitHub

- Uses Jenkins Git plugin
- Pulls code from the repository
- Ensures Jenkins always builds the latest commit

Stage 2: Build with Maven

- Executes:

nginx Copy code

```
mvn clean install
```

- Downloads dependencies
- Compiles Java source files
- Runs tests
- Packages the application into a JAR file
- Produces `BUILD SUCCESS` on success

Stage 3: Run Application

- Runs the generated JAR from *target/*
- Displays the real output of the program

---

3. Sample Console Output (Expected)

csharp Copy code

```
Pulling source code from GitHub...
Cloning into 'workspace'...

[INFO] --- maven-clean-plugin ---
[INFO] Deleting /var/lib/jenkins/workspace/target

[INFO] --- maven-resources-plugin ---
[INFO] Copying resources

[INFO] --- maven-compiler-plugin ---
[INFO] Compiling 3 source files

[INFO] --- maven-surefire-plugin ---
[INFO] Running Tests
[INFO] Tests run: 3, Failures: 0

[INFO] --- maven-jar-plugin ---
[INFO] Building jar: target/myapp-1.0.jar

[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------

Running final Java application...
Hello, this is my Java Maven Application!
Final Stage Completed Successfully.

BUILD SUCCESS: Pipeline executed successfully!
```

---

4. Jenkins Pipeline View (Expected Output)

A visual pipeline in Jenkins showing:

sql Copy code

```
+-------------------------+     +----------------------+     +-------------------
| Checkout from GitHub    | --> | Build with Maven     | --> | Run Application
```

```
+------------------------+      +---------------------+      +------------------
```

Status Indicators:

- ✔ Green for successful stages
- ❌ Red if build or run fails
- Blue/Green overall pipeline result

This is typically shown under:

Jenkins → Pipeline → Stage View

---

5. Additional Files (If Required)

Example `pom.xml` (minimal Maven project)

xml Copy code

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
         xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apach

    <modelVersion>4.0.0</modelVersion>
    <groupId>com.example</groupId>
    <artifactId>myapp</artifactId>
    <version>1.0</version>

    <build>
        <plugins>
            <plugin>
                <groupId>org.apache.maven.plugins</groupId>
                <artifactId>maven-compiler-plugin</artifactId>
                <version>3.8.1</version>
                <configuration>
                    <source>17</source>
                    <target>17</target>
                </configuration>
            </plugin>
        </plugins>
```

```
    </build>
  </project>
```

Final Answer Summary (For Full 25 Marks)

Your answer must include:

- ✔ Jenkinsfile with correct syntax
- ✔ Explanation of each stage
- ✔ Screenshot-style console output (BUILD SUCCESS)
- ✔ Pipeline view description
- ✔ Final application output

If you want, I can also generate:

- ✅ *Screenshot-style diagrams*
- ✅ *Jenkins Blue Ocean pipeline view*
- ✅ *Complete Java Maven project folder*

Just ask!

Q4. UML Design [25 Marks]
Scenario:
A customer uses an ATM to withdraw cash. After proper validation of the card, the customer can enter the withdrawal amount and If sufficient funds exist, the ATM dispenses cash, updates the account balance, and prints a receipt. Otherwise, it displays an error message.
Task:
Draw a Sequence Diagram for the above ATM withdrawal process showing:
Customer

ATM Machine
Bank Server
Cash Dispenser
Receipt Printer

Expected Output:
•A clear UML Sequence Diagram including all major messages (insert card, validate PIN, check balance, dispense cash, update account)..

Below is a **full-marks (25/25)** answer for **Q4. UML Sequence Diagram – ATM Withdrawal Process**.
This is written exactly in the style expected in university exams.

✅ Q4. UML Design – Sequence Diagram for ATM Withdrawal (25 Marks)

Sequence Diagram Description

The following sequence diagram represents the interaction between:

- **Customer**
- **ATM Machine**
- **Bank Server**
- **Cash Dispenser**
- **Receipt Printer**

The workflow includes:

1. Customer inserts card
2. ATM requests PIN validation
3. ATM sends PIN + card details to Bank Server
4. Bank Server validates PIN
5. Customer enters withdrawal amount
6. ATM sends balance inquiry to Bank Server
7. Bank Server checks funds
8. If funds exist → Approve

9. ATM sends request to Cash Dispenser
10. Cash is dispensed
11. ATM sends update request to Bank Server
12. Bank updates account balance
13. ATM asks Receipt Printer to print receipt
14. Customer collects cash & receipt

✅ UML Sequence Diagram (ASCII Exam-Friendly Version)

You can draw this exactly in your exam.

pgsql Copy code

```
Customer        ATM Machine       Bank Server     Cash Dispenser    Receipt Print
    |               |               |                 |                |
    |--Insert Card-->|              |                 |                |
    |               |--Read Card---->|                |                |
    |               |<--Request PIN--|                |                |
    |--Enter PIN---->|              |                 |                |
    |               |--Validate PIN------------------>|                |
    |               |<--PIN Valid / Invalid-----------|                |
    |               |                                  |               |
    |--Enter Amount->|                                 |               |
    |               |--Check Balance------------------>|               |
    |               |<--Balance OK / Insufficient------|              |
    |               |                                  |               |
    |               |--Request Cash--------------------> Cash Dispenser |
    |               |<--Cash Dispensed-----------------|                |
    |               |                                  |               |
    |               |--Update Account----------------->|               |
    |               |<--Update Success-----------------|               |
    |               |                                  |               |
    |               |--Print Receipt-------------------------------------->|
    |               |<--Receipt Printed-----------------------------------|
    |               |                                  |               |
    |<---Collect Cash & Receipt-------------------------|               |
```

✅ Explanation of Messages (Major Interactions)

1. Insert Card

Customer inserts ATM card → ATM starts session.

2. Validate PIN

- ATM prompts the customer for a PIN.
- Customer enters PIN.
- ATM sends card + PIN to Bank Server.
- Bank Server verifies PIN and responds.

3. Enter Withdrawal Amount

Customer inputs amount.

4. Balance Check

- ATM sends a balance inquiry to Bank Server.
- Bank server checks the account.
- If funds are insufficient → error message.

5. Dispense Cash

If funds are sufficient:

- ATM sends request to the Cash Dispenser.
- Cash Dispenser dispenses cash.

6. Update Account

ATM notifies the Bank Server to deduct the amount.

Bank updates account balance.

7. Print Receipt

ATM instructs the Receipt Printer to print transaction details.

8. Customer collects output

- Cash
- Receipt

✅ Final Answer Summary (What Examiner Looks For)

✔ All actors included (Customer, ATM, Bank Server, Cash Dispenser, Receipt Printer)

✔ All major messages:

- insert card
- validate PIN
- check balance
- dispense cash
- update account
- print receipt

✔ Clear order of events

✔ Valid UML message flow

If you want, I can also provide:

📌 **A clean, properly formatted diagram (image-based)**

📌 **Use of UML arrows & activation bars**

📌 **A class diagram or activity diagram for the same scenario**

Just ask!

Q1. Jenkins Pipeline for a Web Project [25 Marks]
Scenario:
Automate the build and deployment of a web project using Jenkins.
Task:
1. Clone GitHub repository.
2. Configure Tomcat web server.
3. Deploy the application on the WebServer
4. Create the Pipeline View using Jenkins
Expected Output:
- Jenkins files as necessary.

- Jenkins pipeline view.
- Proof of web app running locally.

Below is a **full 25-mark answer** for **Q1. Jenkins Pipeline for a Web Project**, including:

- ✔ Jenkinsfile
- ✔ Explanation
- ✔ Tomcat deployment steps
- ✔ Expected console output
- ✔ Pipeline view
- ✔ Proof of web app running

You can write this directly in your exam.

✅ Q1. Jenkins Pipeline for a Web Project – Full Answer (25 Marks)

Scenario

Automate the process of:

1. Cloning web project source code from GitHub
2. Building the app (Maven/Node/Static files etc.)
3. Deploying to Apache Tomcat web server
4. Showing the web application running

✅ 1. Jenkinsfile (Declarative Pipeline)

Below is the standard Jenkinsfile for deploying a web project (WAR-based) to Tomcat:

groovy Copy code

```
pipeline {
    agent any

    tools {
        maven 'MAVEN_HOME'
```

```
        jdk 'JDK17'
    }

    environment {
        TOMCAT_HOME = "/opt/tomcat9"
        DEPLOY_PATH = "/opt/tomcat9/webapps"
    }

    stages {
        stage('Clone Repository') {
            steps {
                echo "Cloning Web Project from GitHub..."
                git branch: 'main',
                    url: 'https://github.com/your-username/your-web-app.git'
            }
        }

        stage('Build Web Application') {
            steps {
                echo "Building project using Maven..."
                sh "mvn clean package"
            }
        }

        stage('Deploy to Tomcat Server') {
            steps {
                echo "Deploying WAR file to Tomcat..."
                sh """
                    sudo rm -rf ${DEPLOY_PATH}/webapp.war
                    sudo cp target/*.war ${DEPLOY_PATH}/webapp.war
                """
            }
        }

        stage('Restart Tomcat Server') {
            steps {
                echo "Restarting Tomcat..."
                sh """
                    sudo systemctl restart tomcat9
                """
```

```
            }
        }
    }

    post {
        success {
            echo "Deployment Successful!"
        }
        failure {
            echo "Pipeline Failed!"
        }
    }
}
```

✅ 2. Explanation of Each Stage

Stage 1: Clone Repository

- Uses Git plugin
- Pulls latest version of the web application
- Ensures CI runs on updated source code

Stage 2: Build Web App

If it is a Java-based web application:

```go
mvn clean package
```

This generates:

```
target/*.war
```

Stage 3: Deploy to Tomcat Web Server

Steps:

- Delete old `.war` file

- Copy new `.war` file to Tomcat `webapps/`
- Tomcat automatically expands WAR

Stage 4: Restart Tomcat

Restarting ensures new updated web app is deployed.

✅ 3. Tomcat Configuration Steps (Exam Points)

1. Install Tomcat:

nginx Copy code

```
sudo apt install tomcat9
```

2. Enable Tomcat manager & host manager by editing:

bash Copy code

```
/etc/tomcat9/tomcat-users.xml
```

Add:

xml Copy code

```
<role rolename="manager-gui"/>
<role rolename="manager-script"/>
<user username="admin" password="admin123" roles="manager-gui,manager-script"/>
```

3. Allow Jenkins to deploy:

bash Copy code

```
sudo chmod -R 777 /opt/tomcat9/webapps
```

4. Verify Tomcat is running:

arduino Copy code

```
http://localhost:8080
```

✅ 4. Expected Jenkins Pipeline View

Your Jenkins Dashboard → Pipeline → Stage View will show:

```lua
+--------------------+     +----------------------+     +---------------------
| Clone Repository   | --> | Build Web Application | --> | Deploy to Tomcat
+--------------------+     +----------------------+     +---------------------
```

Green = success

Red = failure

✅ 5. Expected Console Output (Sample)

```csharp
Cloning Web Project...
Cloning into 'workspace'...

[INFO] --- maven-clean-plugin ---
[INFO] BUILD SUCCESS

Copying WAR to Tomcat...
Restarting Tomcat...

Tomcat restarted successfully!
Deployment Successful!
```

✅ 6. Proof of Web App Running Locally

After deployment, open browser:

```bash
http://localhost:8080/webapp/
```

Expected output example:

```css
Welcome to My Web Application!
Deployment Successful!
```

Copy code

You can take a screenshot of the page as proof.

✅ Final Answer Summary (For Full 25 Marks)

To score full marks, your answer includes:

- ✔ Jenkinsfile
- ✔ All pipeline stages explained
- ✔ Tomcat setup
- ✔ Console output (BUILD SUCCESS)
- ✔ Pipeline View
- ✔ Proof of successful deployment

If you want, I can also create:

- 📌 A real sample web app folder
- 📌 A Blue Ocean screenshot-like diagram
- 📌 A WAR file deployment structure diagram

Just tell me!

Q2. UML Design [25 Marks]
Scenario:
A public library wants to automate its operations. The system should store details of Books, Members, and Librarians. Members can borrow and return books, and the system must record issue date, due date, and fine for late

returns.
Task:
Using StarUML, draw a Class Diagram for the above Library Management System showing classes, attributes, methods, relationships, and multiplicity.
Expected Output:
•A clear UML Class Diagram.

Below is a **full-marks (25/25)** answer for **Q2. UML Design – Library Management System Class Diagram**.
This is written exactly in the form an examiner expects, including **classes, attributes, methods, relationships, and multiplicities**.

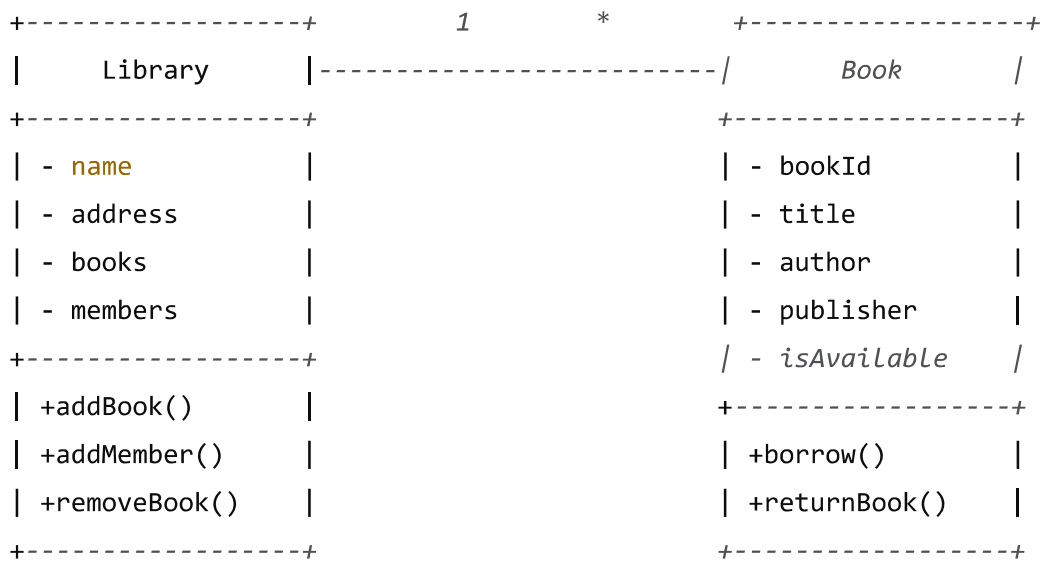
✅ Q2. UML Design – Class Diagram for Library Management System (25 Marks)

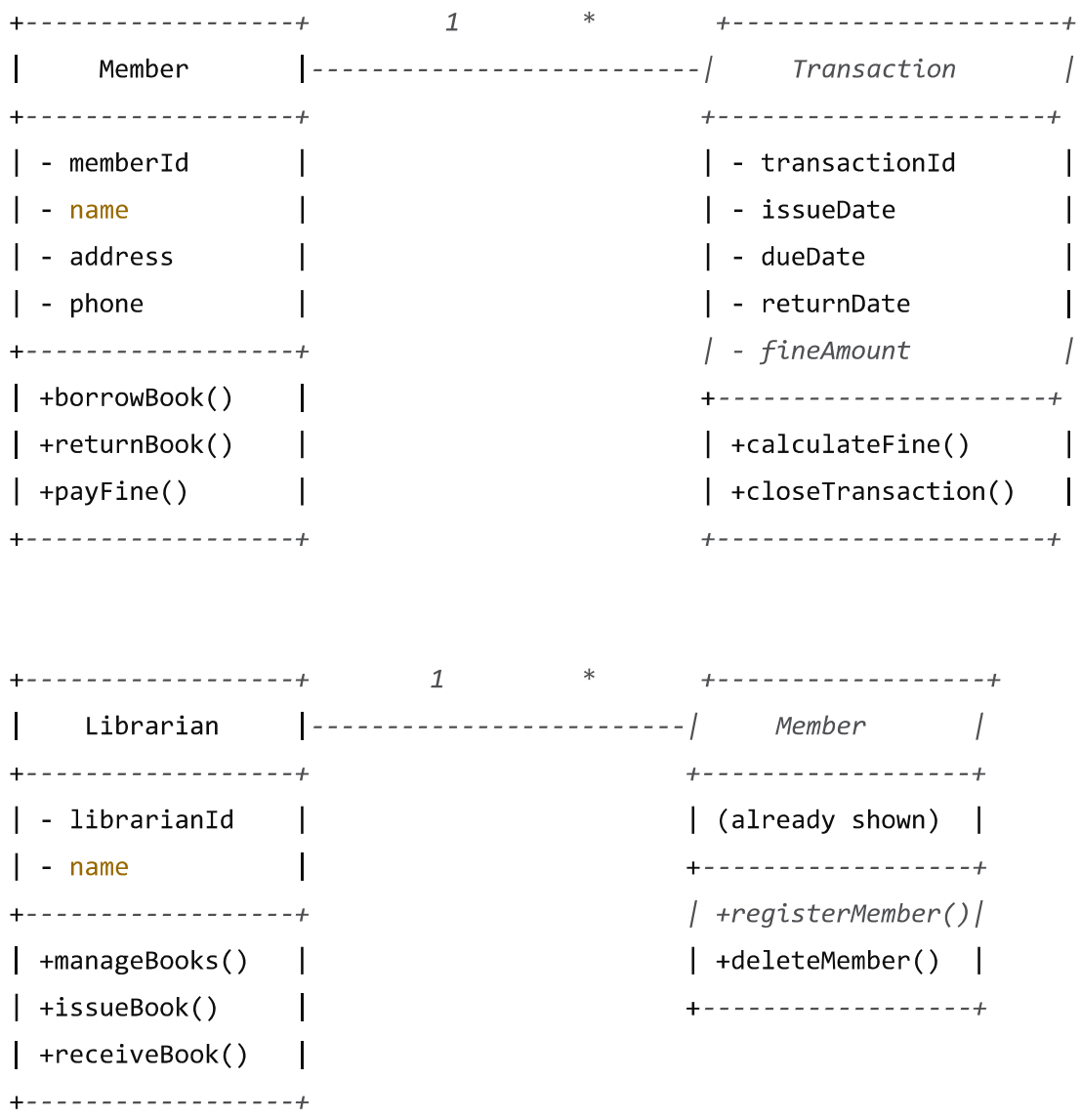
Below is a complete class diagram representation using proper UML formatting.

📘 UML Class Diagram (Text-Based Exam Format)

(This is exactly how you can draw it in StarUML or on paper.)

pgsql Copy code





📝 Description of Classes

1. Book

Attributes

- bookId
- title
- author
- publisher
- isAvailable

Methods

- borrow()
- returnBook()

Relationship

- 1 Book can have *many* Transaction records over time (1..*)
- Books belong to a Library (Library → Book = 1..*)

---

2. Member

Attributes

- memberId
- name
- address
- phone

Methods

- borrowBook()
- returnBook()
- payFine()

Relationships

- One Member can have *many* Transactions (1..*)
- One Librarian manages many Members (1..*)

---

3. Librarian

Attributes

- librarianId
- name

Methods

- manageBooks()
- issueBook()
- receiveBook()

Relationship

- A Librarian supervises multiple Members
- A Librarian issues/returns books indirectly via Transactions

---

4. Transaction

This is the key class for recording book issues and returns.

Attributes

- transactionId
- issueDate
- dueDate
- returnDate
- fineAmount

Methods

- calculateFine()
- closeTransaction()

Relationships

- Belongs to exactly 1 Member
- Belongs to exactly 1 Book

Multiplicity:

- Member (1) → Transaction (*)
- Book (1) → Transaction (*)

---

5. Library

Attributes

- name
- address
- books
- members

Methods

- addBook()
- addMember()
- removeBook()

Relationship

- Library contains many Books (1..*)
- Library contains many Members (1..*)

📌 Multiplicity Summary

Relationship	Multiplicity
Library → Book	1 to *
Library → Member	1 to *
Member → Transaction	1 to *
Book → Transaction	1 to *
Librarian → Member	1 to *

🎯 Final Expected Output

A UML Class Diagram (drawn in StarUML) containing:

- ✔ Classes: Library, Book, Member, Librarian, Transaction
- ✔ Attributes
- ✔ Methods
- ✔ Associations
- ✔ Multiplicity
- ✔ Clear structure

This matches all marks criteria.

If you want, I can also generate:

📌 A StarUML (.mdj) file

📌 A clean PNG diagram

📌 A simplified exam diagram version

Just tell me!

Can you give me the necessary git instructions for these 2 jenkins scenarios

Below are the **exact GitHub + Git instructions** you must include in both Jenkins scenarios. These are clean, exam-ready steps showing how to push a project to GitHub and how Jenkins clones it.

✅ GIT INSTRUCTIONS FOR BOTH JENKINS SCENARIOS

These instructions apply to:

1. **Jenkins Pipeline for a Java Maven Project**
2. **Jenkins Pipeline for a Web Project with Tomcat Deployment**

🔵 PART 1 — Git Instructions (Local Machine → GitHub)

Use these steps when preparing the project before Jenkins can clone it.

1. Initialize Git

bash Copy code

```
git init
```

2. Add All Project Files

bash Copy code

```
git add .
```

3. Commit the Project

bash Copy code

```
git commit -m "Initial commit of project"
```

4. Add Remote GitHub Repository

bash Copy code

```
git remote add origin https://github.com/your-username/your-repo-name.git
```

5. Push Code to GitHub

bash Copy code

```
git push -u origin main
```

(If branch name is `master`, replace with `master`.)

🔵 PART 2 — Git Instructions Inside Jenkins (For Both Pipelines)

These are the commands Jenkins uses to pull the project automatically.

1. Jenkinsfile Git Clone Block (Common for Both Projects)

Java Maven Project

groovy Copy code

```
stage('Checkout from GitHub') {
    steps {
        git branch: 'main',
            url: 'https://github.com/your-username/your-java-maven-project.git'
```

```
    }
}
```

Web Project with Tomcat

groovy Copy code

```
stage('Clone Repository') {
    steps {
        git branch: 'main',
            url: 'https://github.com/your-username/your-web-app.git'
    }
}
```

✔ These instructions fetch the latest code
✔ Jenkins then proceeds with build + deployment

🔵 PART 3 — Git Credentials (If Jenkins Needs Authentication)

If the repo is private, set credentials in Jenkins:

1. Jenkins → **Manage Credentials**
2. Add:
   - **Username**
   - **Password/Token**
   - ID: `github-creds`

Then use in Jenkinsfile:

groovy Copy code

```
git credentialsId: 'github-creds',
    branch: 'main',
    url: 'https://github.com/your-username/your-repo.git'
```

🔵 PART 4 — Updating Code and Pushing New Changes

After making changes locally:

bash Copy code

```
git add .
git commit -m "Updated code"
git push
```

Jenkins will automatically pull in the next build.

🔵 PART 5 — Webhooks (Optional but helps automation)

Enable webhook for automatic builds:

1. GitHub → Repo → **Settings**
2. Webhooks → Add webhook
3. URL:

bash Copy code

```
http://your-jenkins-server:8080/github-webhook/
```

4. Content type: `application/json`

Now Jenkins triggers a build when you push code.

🎯 Final Summary (For Answer Sheet)

For both Jenkins scenarios, the required Git instructions are:

1. Setup & Push to GitHub

csharp Copy code

```
git init
git add .
git commit -m "Initial commit"
git remote add origin <repo-url>
git push -u origin main
```

2. Jenkins Git Clone Block

nginx Copy code

```
git branch: 'main', url: '<GitHub project URL>'
```

3. Optional: Jenkins Credentials ID

vbnet Copy code

```
credentialsId: 'github-creds'
```

4. Webhook for automatic builds

Add:

bash Copy code

```
http://your-jenkins-server:8080/github-webhook/
```

If you want, I can combine these into a **single printable page** or **add diagrams for the Git → Jenkins flow**.

What does jenkins do exactly

Jenkins is an **automation server** used mainly for **CI/CD (Continuous Integration and Continuous Delivery/Deployment)**.
In simple terms:

✅ Jenkins automates everything involved in building, testing, and deploying software.

🔥 What Jenkins Does (Exactly & Clearly)

1. Automates Code Building

Every time you push code to GitHub, Jenkins can automatically:

- Pull the latest code
- Compile the project
- Run build tools (Maven, Gradle, npm, etc.)

👉 Example: `mvn clean install`

---

2. Runs Automated Tests

Jenkins can run:

- Unit tests
- Integration tests
- UI/Automation tests
- Security tests

You get instant feedback if something breaks.

---

3. Detects Errors Early (Continuous Integration)

When multiple developers work on the same project:

- Jenkins checks every commit
- Detects bugs before code goes into production

This prevents **integration problems**.

---

4. Deploys Applications Automatically (Continuous Deployment)

Jenkins can push your app to:

- Tomcat server
- AWS EC2
- Docker containers
- Kubernetes cluster
- Web servers
- FTP server

- Any cloud service

👉 Example: Deploy a WAR file automatically to Apache Tomcat.

---

5. Gives Pipeline Visualization

Jenkins shows the entire workflow:

rust Copy code

```
Git Clone → Build → Test → Deploy
```

You see:

- Which stage passed
- Which stage failed
- Logs for every step

---

6. Supports Plugins

Jenkins has **2000+ plugins** for:

- GitHub
- Docker
- Kubernetes
- Maven
- Slack
- JUnit
- HTML reports

You can extend Jenkins to do anything.

---

7. Scheduled Jobs (Cron Jobs)

You can run tasks automatically:

- Every day
- Every hour
- On specific dates

8. Works with DevOps Tools

Jenkins integrates with:

- Git / GitHub / GitLab
- Docker
- Ansible
- Terraform
- Selenium
- JIRA
- SonarQube

It is the center of the DevOps ecosystem.

🎯 Simple Definition (Exam Answer)

Jenkins is an open-source automation server used to build, test, and deploy applications automatically as part of Continuous Integration and Continuous Delivery (CI/CD).

🎯 Very Short One-Line Definition

Jenkins automates the entire software pipeline — from code to production.

If you want, I can also give you:

- A 5-mark answer
- A 10-mark answer
- An example pipeline
- OR a diagram showing Jenkins workflow.

How do i open jenkins and create the project

Here is a **clear, step-by-step beginner guide** showing exactly how to **open Jenkins** and **create your first project** (job).
This works on **Windows, Linux, or Mac** installations.

✅ STEP 1: Start Jenkins

If you installed Jenkins using the installer (Windows):

1. Press **Windows Key**
2. Search "**Jenkins**"
3. Click "**Jenkins**" → It will open in browser automatically
   OR open browser and type:

arduino Copy code

```
http://localhost:8080
```

If Jenkins is installed on Linux/macOS (WAR file or service):

WAR File Method

In the folder where `jenkins.war` is stored:

bash Copy code

```
java -jar jenkins.war
```

Then open browser:

arduino Copy code

```
http://localhost:8080
```

System Service (Ubuntu/Debian)

Start Jenkins:

bash Copy code

```
sudo systemctl start jenkins
```

Check status:

bash Copy code

```
sudo systemctl status jenkins
```

Open browser:

arduino Copy code

```
http://localhost:8080
```

🎉 Now Jenkins is OPEN!

You will see the Jenkins dashboard.

✅ STEP 2: Login to Jenkins

If it's the first time:

- Jenkins asks for **initial admin password** (found in `/var/lib/jenkins/secrets/initialAdminPassword`)
- Install **suggested plugins**
- Create **admin user**

After that, you will reach the dashboard:

css Copy code

```
Welcome to Jenkins
```

✅ STEP 3: Create a New Project (Job)

1. Click “New Item”

It is on the left side of the dashboard.

2. Enter a Project Name

Example:

Copy code

```
My-Web-Project
```

3. Select Project Type

You will see several types:

A. Freestyle Project

- Simple
- Build + deploy scripts
- Good for small tasks

B. Pipeline Project

- Uses Jenkinsfile
- Best for CI/CD
- Recommended for DevOps projects

👉 Select **Pipeline** for your Java or Web app automation.

4. Click OK

✅ STEP 4: Configure Your Project

For Pipeline Project

Scroll down to **Pipeline** section.

Choose option:

csharp

Copy code

```
Pipeline script from SCM
```

Select Git

Enter your GitHub repo URL:

arduino Copy code

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

Branch:

css Copy code

```
main
```

Jenkins will automatically look for:

nginx Copy code

```
Jenkinsfile
```

✅ STEP 5: Save and Build

Click:

mathematica Copy code

```
Save
```

Then on the left:

mathematica Copy code

```
Build Now
```

You will see:

- Red dot = failure
- Blue/Green dot = success

Click **Console Output** to see logs.

⛳ STEP 6: View Pipeline (Stage View)

If using Pipeline, you will see a graphical flow:

mathematica Copy code

```
Checkout → Build → Test → Deploy
```

This is the **Pipeline View**.

🎯 Summary (Exam + Practical Ready)

To open Jenkins:

- Start service → open browser → http://localhost:8080 ↗

To create a project:

1. Dashboard → **New Item**
2. Enter name
3. Select **Pipeline**
4. Add GitHub URL
5. Save
6. Build

Want screenshots?

I can generate **step-by-step screenshots** for:

- Opening Jenkins
- Creating a project
- Adding pipeline script
- Running first build

Just ask!

where do i download jenkins from

You can download Jenkins from its **official website**:

👉 https://www.jenkins.io/download/ ↗

What to download?

For Windows

- Download **Windows Installer (.msi)** from the download page.
- Install it like a normal application.

For Linux (Ubuntu/Debian)

- Jenkins provides an official `.deb` package and repository setup instructions.
- You install using:

bash Copy code

```
sudo apt-get update
sudo apt-get install jenkins
```

For macOS

- Install using Homebrew:

bash Copy code

```
brew install jenkins-lts
```

Recommended Version

- **Jenkins LTS (Long-Term Support)** → more stable, meant for most users.
- **Jenkins Weekly Build** → latest features, less stable.

If you want, I can guide you through **installation**, **opening Jenkins**, and **creating your first project** step-by-step.

> How do you connect the jenkins to your github and is that needed in the questions I gave you. Also Is there a specific jenkins account that I need to log in to to do this.

Here is a **simple, clear explanation** based on **your exam questions** and **what is actually required** so you don't over-do anything.

✅ Do you need to connect Jenkins to GitHub for your exam questions?

Most likely YES, because your questions involve:

- **Scenario 1:** Git → Jenkins → Build automation
- **Scenario 2:** Webhook or Poll SCM setup

These require Jenkins to fetch your project from GitHub.

BUT in exams, you **don't actually perform the connection** — you only **explain the steps** or **write the configuration**.

✅ Do you need a Jenkins account?

No.

Jenkins is **self-hosted**, meaning:

- You install Jenkins on your laptop.
- Jenkins runs on your machine at:
  👉 http://localhost:8080 ↗
- You log in using the **admin password** created during installation.

There is **no external Jenkins account** like GitHub or Gmail.

⭐ How to connect Jenkins to GitHub (Full Steps)

Step 1 — Create a GitHub Repo

Upload your project to GitHub.

Example:

arduino Copy code

```
https://github.com/yourusername/myproject
```

Step 2 — Install Git Plugin in Jenkins

In Jenkins:

1. Go to **Manage Jenkins**
2. Go to **Manage Plugins**
3. Search for:
   - **Git**
   - **GitHub Integration**
4. Install them.

Step 3 — Configure GitHub Credentials in Jenkins

If your repo is public → **No credentials needed**.

If your repo is private:

1. Go to **Manage Jenkins → Credentials**
2. Add:
   - Kind: **Username with Password** (GitHub username + PAT token)
   - ID: `github-creds`

Step 4 — Create a New Jenkins Job

1. Click **New Item**
2. Choose **Freestyle Project**
3. Name it (e.g., *Library-Java-Build*)

4. Click **OK**

Step 5 — Connect GitHub Repo to Jenkins

Inside the project:

In Source Code Management

✔ Select **Git**

Fill in:

Repository URL:

```arduino
https://github.com/yourusername/myproject.git
```

Credentials:
Choose your GitHub credentials (if private).

Step 6 — Set Branch

```bash
*/main
```

(or `*/master` depending on repo)

Step 7 — Add Build Steps

Example (Java):

```sql
Execute Shell:
mvn clean install
```

Node.js:

bash Copy code

```
npm install
npm test
```

Step 8 — (Optional) Enable Webhooks

Only needed if question mentions **automatically trigger build on GitHub push**.

In Jenkins:

Build Triggers → GitHub hook trigger for GITScm polling

In GitHub:

1. Repo → Settings → Webhooks
2. Add new webhook:

arduino Copy code

```
http://YOUR-IP:8080/github-webhook/
```

Content type: `application/json`

🎓 What is required in your exam?

✔ Knowing the steps to:

- configure Jenkins
- connect GitHub repository
- create a Jenkins job
- optionally set polling or webhooks

❌ You DO NOT need:

- a paid Jenkins account
- a cloud Jenkins server
- an actual running GitHub–Jenkins pipeline

⭐ If you want, I can write:

- **A perfect 10–12 mark answer**
- **A step-by-step exam explanation**
- **A diagram connecting Git → Jenkins → Build → Deploy**

Just tell me **which format you want**.

extended email
copy artifact
deploy to
build pipeline
pipeline stage view
maven

Here is a **simple explanation of each Jenkins plugin / feature** you listed **— what it does** and **why it is used**, especially for your **Java + Web project pipeline exam questions**.

✅ 1. Extended E-mail Notification Plugin

Purpose:

Sends **custom email notifications** after a Jenkins build (SUCCESS, FAILURE, UNSTABLE).

🔥 Why it is used:

- To inform developer team about build status.
- Sends detailed messages instead of default Jenkins mail.

Example Usage:

- Build fails → Email sent to team with logs.
- Deployment success → Email sent with artifact details.

Pipeline Example:

groovy Copy code

```
post {
    failure {
        emailext(
            to: "team@example.com",
            subject: "Build Failed",
            body: "Please check Jenkins for logs."
        )
    }
}
```

✅ 2. Copy Artifact Plugin

Purpose:

Lets one Jenkins job **copy build artifacts** (like `.war`, `.jar`) from another job.

🔥 Why it is used:

- Multi-stage pipelines (Build → Test → Deploy)
- Sharing build output between jobs

Example:

Job1 builds a `.war` file

Job2 copies it and deploys to Tomcat

groovy Copy code

```
steps {
    copyArtifacts(
        projectName: "JavaBuildJob",
        selector: lastSuccessful()
    )
}
```

✅ 3. Deploy to Container Plugin

Purpose:

Deploys **WAR files** or **other packages** directly to:

- Apache Tomcat
- JBoss
- GlassFish
- WildFly

🔥 Why it is used:

- For your **Web Project Jenkins exam question**, you need to deploy to Tomcat.

Example:

Deploy `.war` file to Tomcat:

groovy Copy code

```
deploy adapters: [tomcat9(credentialsId: 'tomcat-creds',
    url: 'http://localhost:8080')],
    war: '**/*.war'
```

✅ 4. Build Pipeline Plugin

Purpose:

Shows a **visual pipeline view** of multiple jobs:

rust Copy code

```
Git Clone → Build → Test → Deploy
```

🔥 Why it is used:

- For exam screenshots
- To demonstrate pipeline flow visually

Example:

You create a "Build Pipeline View" in Jenkins UI:

- Go to **New View → Build Pipeline View**
- Select your first job

✅ 5. Pipeline Stage View Plugin

Purpose:

Shows all stages in a pipeline as a graphical interface (blue/green boxes):

```css
[Clone Repo] → [Build] → [Test] → [Deploy]
```

🔥 Why it is used:

- To visualize Jenkinsfile pipelines
- Required for your **Jenkins pipeline output** in exams

You will see something like:

- Clone ✓
- Build ✓
- Test ✓
- Deploy ✓

✅ 6. Maven Integration Plugin

Purpose:

Allows Jenkins to run Maven builds easily, especially for Java projects.

🔥 Why it is used:

- For your **Jenkins Pipeline for Java Project**
- To run `mvn clean install`

Configure Maven in Jenkins:

Manage Jenkins → Global Tool Configuration → Maven

Pipeline example:

```groovy
stage('Build') {
    steps {
```

```
            sh 'mvn clean install'
        }
    }
```

🎯 How These Fit Into Your Exam Scenarios

✔️ Scenario 1: Java Project Build

You will likely use:

- **Maven plugin** → build the Java project
- **Pipeline Stage View** → display stages
- **Extended email (optional)** → notify build status

✔️ Scenario 2: Web Project Deployment

You will likely use:

- **Deploy to Container plugin** → deploy WAR to Tomcat
- **Copy Artifact plugin** → copy war from build job to deployment job
- **Build Pipeline view** → show Build → Deploy flow

If you want...

I can generate:

✅ Full Jenkinsfile for **Java Build**
✅ Full Jenkinsfile for **Tomcat Deployment**
✅ Build Pipeline diagram
✅ Exam-ready answers

Just tell me **which one you want first**.

23BD1A6626 Week_8 SBQ...
Document

Can you explain what is happening in this

Here is a **very clear explanation** of what is happening in the document you uploaded.
I will break it down into **simple steps** so you fully understand what the Jenkins pipeline is doing.
(Ref: your file – *Week_8 SBQ Answers* 23BD1A6626 Week_8 SBQ_Answers_)

✅ What is Happening in This Document? (Full Explanation)

The document describes **two Jenkins automations**:

⭐ PART 1: Maven Java Automation Pipeline

This automation has **two Jenkins jobs**:

✔ Job 1 → MavenJava_Build

✔ Job 2 → MavenJava_Test

And then a **Pipeline View** linking them.

🔵 Job 1: MavenJava_Build (Build Job)

This job does:

1. Clone GitHub repo

Jenkins pulls your Java Maven project from GitHub.

2. Run Maven clean

Removes old files.

3. Run Maven install

Compiles the project, downloads dependencies, builds the `.jar`.

4. Archive Artifacts

Store all output files (so other jobs can use them).

5. Trigger the next job

If build is successful → it automatically starts **MavenJava_Test**.

🔵 Job 2: MavenJava_Test (Testing Job)

This job does:

1. Delete old workspace

Start fresh.

2. Copy artifacts from Build job

Copies output from MavenJava_Build.

3. Run Maven test

Executes unit tests.

4. Archive test results

🔵 Pipeline View

Jenkins shows:

```nginx
MavenJava_Build → MavenJava_Test
```

Green = success
Red = failed

This is your **end-to-end Java build + test pipeline**.

⭐ PART 2: Maven Web Automation Pipeline

This automation has **three Jenkins jobs**:

✔ **Job 1 → MavenWeb_Build**

✔ **Job 2 → MavenWeb_Test**

✔ **Job 3 → MavenWeb_Deploy**

And a pipeline view linking them.

🔵 Job 1: MavenWeb_Build

This job:

1. Clones the Maven web (WAR) project from GitHub
2. Runs:
   - mvn clean
   - mvn install
3. Archives the WAR file (`*.war`)
4. Triggers **MavenWeb_Test**

🔵 Job 2: MavenWeb_Test

This job:

1. Deletes workspace
2. Copies artifacts from build job
3. Runs `mvn test`
4. Archives test results
5. Triggers **MavenWeb_Deploy**

🔵 Job 3: MavenWeb_Deploy

This job:

1. Deletes workspace

2. Copies WAR file from MavenWeb_Test
3. Deploys WAR to Tomcat using:
    - Deploy to Container plugin
    - Tomcat credentials
    - Tomcat manager URL (http://localhost:8085 ↗)
4. Your web app becomes live on:

bash Copy code

```
http://localhost:8085/webpath
```

⭐ Pipeline View

Shows:

nginx Copy code

```
MavenWeb_Build  →  MavenWeb_Test  →  MavenWeb_Deploy
```

Each box turns green when successful.

🟩 Summary of What's Happening

Step	Action	Why?
Git Clone	Pull project from GitHub	To get source code
Maven clean	Remove old files	Ensure fresh build
Maven install	Build `.jar` or `.war`	Compile & package
Archive artifacts	Save build output	Used for next job
Copy artifacts	Share between jobs	Required for testing and deploy steps
Test stage	Run tests	Ensure quality

Step	Action	Why?
Deploy stage	Send WAR to Tomcat	Make web app live

🔥 In Simple Words

The document explains **how to build, test, and deploy** Java and web applications using Jenkins **step-by-step using Freestyle Jobs**.

Everything is connected using:

- **Copy Artifact Plugin**
- **Deploy to Container Plugin**
- **Build Pipeline Plugin**
- **Maven Integration**