Program 1: Build a maven project and migrate to gradle.

1. Open eclipse → New → Maven project (packaging - jar)
2. GroupId: com.anyname , ArtifactID: anyname
3. Skip archetype selection → checkbox.
4. Program: To check if username is valid.
5. src/main/java → Right click → New → Package.
6. Package → Right click → New → Class → Name: App.java
7. src/main/resources → Right click → New → File → Name: (config.properties)
8. In config.properties: Type:

```
username = abc
password = abc@123
```

Save it.

9. In App.java:

```
package com.projectname;
import java.util.ResourceBundle;
public class App {
 public int userlogin(String inuser, String inpwd) {
  ResourceBundle rb = ResourceBundle.getBundle("config");
 String username = rb.getString("username");
 String password = rb.getString("password");
 if (inuser.equals(username) && inpwd.equals(password))
      return 1;
 else
      return 0; } }
```

10. src / test / java → New → Package.

11. Package → Right click → New → Class → apptest.java → Finish.
(Name)

12. apptest.java:

```
package myproject;
import org.testng.Assert;
public class apptest {
    public void testlogin1() {
        App myapp = new App();   [Name you gave in main/java]
        Assert.assertEquals(0, myapp.userlogin("abc", "abc123"));
    }
    public void testlogin2() {
        App myapp = new App();
        Assert.assertEquals(1, myapp.userlogin("abc", "abc@123"));
    }
}
```

13. Go to google → mvnrepository.com → Search 'testng' → Click → latest version.

14. Copy maven code.

15. Go to POM.xml: Add this: < dependencies >
Paste code
< / dependencies >

16. Project → Right click → Maven → Update project.

[maven dependencies folder must be created].

17. Go to Help → Eclipse Market Place → TestNG → Install.

18. Project → Right click → Run as:

    i) Maven clean

    ii) Maven build : Goals: compile

    iii) Maven test

    iv) Maven Install

    v) Maven verify

19. In target folder : war file will be generated.

20. Download gradle if it isn't installed

21. Go to command prompt: gradle -v

22. cd yourproject directory

23. gradle init

24. gradle files will be generated.

25. Refresh project in eclipse: build.gradle etc will be created.

Program 2: Jenkins and Tomcat

→ java --version : (21 version)

- To install tomcat:

1. Tomcat 9x version - windows service installer.
2. Port - 8082 (or anything, dont repeat).
3. Username, password → next → Install → Finish.
4. To change port number after installation:
   → ProgramFiles → Apache Software Foundation → Tomcat → server.xml
5. To change username, password and also add this code:
   → Apache Software Foundation → Tomcat → tomcat-users.xml

- In tomcat-users.xml:

Add/change: (In the end):

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

→ Save it.

6. Chrome → localhost:8082 (tomcat page)
   ↳ manager app → enter username and password → done.

[Tomcat manager page should open].

- To install Jenkins:

1. Jenkins download and deploy
2. LTS version - windows - download.
3. jenkins.msi → click → Install.

4. java / jdk version - 21

5. port - 8080

6. Once installation is done, open localhost: 8080

7. Default username - admin

Password: ProgramData → Jenkins → .jenkins → secrets
↓
initial admin password
↓
copy it.

[To reinstall jenkins, delete app, folders in programfiles and programdata, then re-install].

8. Once logged in, install suggested plugins.

9. Dashboard → Manage Jenkins → Plugins:

Install: i) Github ii) Maven Integration iii) Deploy to container

10. Dashboard → Manage Jenkins → Tools → Maven → add maven

Steps:

1. Eclipse → New project → Maven project → groupId, artifact ID something.
→ Packaging - war format.

2. src/main → right click → file (new) → choose webapp folder
↓
name file: index.html.

3. webapp → right click → new folder: WEB-INF.

4. WEB-INF → right click → new file: web.xml

5. We have created: index.html under webapp folder (file)
WEB-INF (folder) under webapp folder.
web.xml (file) under WEB-INF folder.

6. index.html: Add a simple "hello world" code to it.

7. web.xml: Add below plugin:

```
<plugin>
<groupId> org.apache.maven.plugins </groupId>
<artifactId> maven-war-plugin </artifactId>
<configuration>
<webXml> src/main/webapp/WEB-INF/web.xml </webXml>
</configuration>
</plugin>
```

8. project → maven → update project.

9. project → run as → maven clean
maven build (compile-goal)
maven test, install, verify [Build successful].

10. Add project to github:

→ command prompt: git --version (make sure git is installed)

→ Github account → new repository (public access).

→ Eclipse → right click project → Team → Share project.

↓

Click 'create' (new repo) → Finish

~~[All branches enabled]~~.

→ Project → right click → Team → Add to Index

→ Right click again → Team → Commit.

Message: "Initial commit", click 'commit'.

→ Copy github repository url.

→ Right click project → Team → Remote → Push.

Paste github url, select branch : source → master

Enable or click on 'all branches' → Next (i)

→ In github, settings → developer settings → personal access tokens.

→ classic → generate new → select all → done → copy.

→ When you click on Next (i) → it asks for username and password.

Username - github username

password - token created.

↓

Finish → Push → Enter credentials again → close.

→ Refresh github repository, check if all files have been pushed.

11. Building in Jenkins:

→ Dashboard → New item → Freestyle project

→ Select git and enter git repository url.

→ Build step → Add build step → Invoke top level maven targets.

→ maven - add maven

→ goals - clean install

→ Note: If pom.xml etc.. are in a subdirectory then it must be specified.

↓

Advanced → POM → program2 / pom.xml (example).

→ Post-build actions → deploy war / ear to a container:

WAR / EAR files ? → **/*.war

→ Add container: Tomcat 9.x remote

→ Credentials: Jenkins → Add username and password.

→ Choose this credential.

→ Tomcat url: copy paste localhost url.

→ Apply and save.

→ Build now → Build successful.

12. Tomcat localhost → Manager App → Enter username & password.

Under applications → program name → double click

↓

project will be displayed (webpage).

Program 3:

Part A: Deploy maven project using docker

1. Create a maven project - war packaging.
2. src / main / webapps - New file: index.html - Add simple code.

New folder: WEB-INF - New file - web.xml

↓

Same plugin.

3. Maven - update project, Run as - maven clean, build, test, install, verify.
4. Open VS code - Open this project folder.
5. Create new Dockerfile:

Add this:

```
FROM tomcat:9.0
RUN rm -rf usr/local/tomcat/webapps/ROOT.war
COPY /target/project.war /usr/local/tomcat/webapps/ROOT.war
EXPOSE 8080
```

(war filename under target)

[Save it].

6. Open terminal, make sure you are in your project directory.

7. docker build -t sample .
(any name)

8. docker run -d -p 8095:8080 sample ;
(unused port) (name you gave while building).

It will give some id.

9. Go to localhost:8095 (port you gave)
↓
It must display index.html webpage.

10. To stop and clean:
→ docker ps : It will return a container ID.
→ docker stop container ID.
→ docker rm container ID.

Part-B: Two apps - enable communication between them:

1. Open VS code: Create new Folder.
2. Create the following structure:

• Main folder:
- docker-compose.yaml (file)
- app1 (folder)
  → app.py
  → requirements.txt } (files)
  → Dockerfile
- app2 (folder)
  → app.py
  → requirements.txt } (files)
  → Dockerfile

3. In app1:

app.py

```
from flask import Flask
app = Flask(__name__)
@app.route('/')
def hello():
    return "Hello from app1"
if __name__ == "__main__":
    app.run(host='0.0.0.0', port=5000)
```

requirements.txt

```
flask
```

Dockerfile

```
FROM python:3.12-slim
WORKDIR /app
COPY requirements.txt .
RUN pip install --no-cache-dir -r requirements.txt
COPY app.py .
EXPOSE 5000
CMD ["python", "app.py"]
```

4. In app2:

app.py

```
import requests
response = requests.get("http://app1:5000/")
print("Respone from app1:", response.text)
```

requirements.txt

requests

Dockerfile - same as app1

5. docker-compose.yaml

Add:

```
services:
  app1:
    build: ./app1
    networks:
      - app-network
    ports:
      - "5000:5000"

  app2:
    build : ./app2
    networks:
      - app-network
    depends_on:
      - app1

networks:
  app-network:
    driver: bridge
```

. In terminal : (make sure project directory is correct)

i) docker-compose build

ii) docker-compose up → [It gives response from app1 and a url - localhost - webpage] → Output.

iii) To check logs:
docker-compose logs app2